

**Geometric Abstractions**  
**for**  
**Information Processing in**  
**Sensor Networks**

A Dissertation Presented

by

**Rik Sarkar**

to

The Graduate School  
in Partial Fulfillment of the  
Requirements  
for the Degree of

**Doctor of Philosophy**  
in  
**Computer Science**

Stony Brook University  
May 2010

Copyright by  
**Rik Sarkar**  
2010

**Stony Brook University**

The Graduate School

**Rik Sarkar**

We, the dissertation committee for the above candidate for the  
Doctor of Philosophy Degree, hereby recommend  
acceptance of this Dissertation.

**Jie Gao – Dissertation Advisor**  
Assistant Professor, Department of Computer Science

**Joseph S. B. Mitchell – Chairperson of Defense**  
Professor, Department of Computer Science

**Samir R. Das**  
Professor, Department of Computer Science

**Alon Efrat**  
Associate Professor, Department of Computer Science  
University of Arizona, Tucson

This dissertation is accepted by the Graduate School

Lawrence Martin  
Dean of the Graduate School





Abstract of the Dissertation

**Geometric Abstractions  
for  
Information Processing in Sensor Networks**

by

**Rik Sarkar**

**Doctor of Philosophy**  
in  
**Computer Science**  
Stony Brook University  
**2010**

Computerized devices are becoming smaller and more ubiquitous. Equally importantly, they are becoming more interconnected. A *Sensor Network* is a model for such interconnected systems. Each sensor device obtains and stores information that is potentially useful to others. The challenge is to efficiently search and deliver the important information to the relevant parties. Given the large number of devices and corresponding quantities of data, this is not easy. Fortunately for us, communication is efficient and fast when addressing nearby devices. This permits us to utilize their relative locations to construct efficient methods.

The proximity and location information can be leveraged through the use of geometry. The complexity of a network and data hide simpler geometric structures that are not obvious at first sight. The objective in this dissertation is to identify such concealed structures that can be useful and can be computed in the network. An abstract structure or *abstraction* helps us to understand and represent the network and data in more convenient ways. This approach is useful in managing the data in the network, as well as in managing the network itself. Its utility is demonstrated through accompanying algorithms in each part of the dissertation.

# Contents

<b>Preface</b>	<b>ix</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Overview of sensor networks . . . . .	2
1.1.1 Sensor network model . . . . .	2
1.1.2 Common questions in distributed sensor networks . . . . .	6
1.1.3 Sensor network as a general model . . . . .	8
1.2 Geometric abstractions . . . . .	10
1.2.1 Geometry in sensor networks . . . . .	11
1.3 Overview of the chapters . . . . .	13
<b>I Metric Spaces and Spatial Distributions</b>	<b>17</b>
<b>2 Introduction to Metric Growth</b>	<b>19</b>
2.1 Review of related research . . . . .	21
2.1.1 Spatial distribution and routing . . . . .	21
2.1.2 Information processing in sensor networks . . . . .	23
2.1.3 Gossip algorithms . . . . .	24
<b>3 Routing in Metric Spaces Using Spatial Distributions</b>	<b>27</b>
3.1 Introduction . . . . .	27
3.2 Small stretch routing with approximate distances . . . . .	30
3.2.1 $(1 + \epsilon)$ -stretch forwarding region . . . . .	32
3.3 Routing table construction by spatial distribution . . . . .	34
3.4 Implementation in sensor networks . . . . .	37
3.4.1 Geographic routing table design . . . . .	37
3.4.2 Landmark-based routing table design . . . . .	38
3.4.3 Routing scheme implementation . . . . .	39
3.5 Simulations . . . . .	40
3.5.1 Geographic routing table . . . . .	41
3.5.2 Landmark-based routing table . . . . .	44
3.6 Conclusion . . . . .	44

<b>4</b>	<b>Spatial Gossip and Multi-Resolution Aggregation</b>	<b>47</b>
4.1	Introduction . . . . .	47
4.1.1	The challenge and our contribution . . . . .	49
4.2	Network setup . . . . .	50
4.3	Spatial gossip . . . . .	51
4.3.1	Hierarchical spatial gossip . . . . .	51
4.3.2	Multi-resolution representations . . . . .	54
4.3.3	Communication cost . . . . .	57
4.4	Accurate multi-resolution data . . . . .	57
4.5	Range queries . . . . .	58
4.6	Simulations . . . . .	61
4.6.1	Total communication cost . . . . .	61
4.6.2	Effectiveness of multi-resolution representation . . . . .	62
4.7	Conclusion . . . . .	64
 <b>II Virtual Coordinates: Modifying Network Geometries</b>		<b>67</b>
<b>5</b>	<b>Introduction</b>	<b>69</b>
5.1	Research review . . . . .	70
<b>6</b>	<b>Ricci Flow and Greedy Routing</b>	<b>73</b>
6.1	Introduction . . . . .	73
6.1.1	Our contribution . . . . .	74
6.2	Theory . . . . .	76
6.2.1	Riemannian metric and curvature . . . . .	76
6.2.2	Surface Ricci flow . . . . .	76
6.2.3	Discrete Ricci flow . . . . .	77
6.3	Algorithms . . . . .	80
6.3.1	Obtain a triangulation . . . . .	80
6.3.2	Other triangulation methods . . . . .	84
6.3.3	Computing the conformal map . . . . .	85
6.3.4	Routing . . . . .	88
6.4	Experimental results . . . . .	90
6.5	Conclusion . . . . .	91
<b>7</b>	<b>Data Storage on Virtual Coordinates</b>	<b>93</b>
7.1	Introduction . . . . .	93
7.2	Theoretical background . . . . .	95
7.2.1	Conformal mapping for multiply connected domain . . . . .	95
7.2.2	Möbius transform and circular reflection . . . . .	95
7.2.3	Schottky groups . . . . .	97
7.2.4	Shrinkage estimation . . . . .	98
7.3	Algorithms . . . . .	102
7.3.1	Circle estimation . . . . .	102

7.3.2	Separation modulus optimization . . . . .	103
7.3.3	Circular reflection . . . . .	104
7.3.4	Computation and communication costs . . . . .	106
7.4	Applications . . . . .	106
7.4.1	Geographic hash tables . . . . .	106
7.4.2	Double rulings . . . . .	108
7.4.3	Load balanced greedy routing . . . . .	109
7.5	Simulations . . . . .	111
7.5.1	In-network storage and sampling . . . . .	111
7.5.2	Load balanced routing . . . . .	112
7.5.3	Network dynamics . . . . .	114
7.6	Conclusion . . . . .	115

### **III Geometry and Topology of Information 117**

<b>8</b>	<b>Introduction 119</b>
8.1	Research review . . . . . 120

<b>9</b>	<b>Topology of Data and Iso-Contour Queries 123</b>
9.1	Introduction . . . . . 123
9.2	Contour trees and gradient routing . . . . . 127
9.2.1	Notations . . . . . 129
9.2.2	Sweep to identify saddle points . . . . . 130
9.2.3	Construction of the contour tree . . . . . 133
9.2.4	Information dissemination . . . . . 134
9.2.5	Handling noises . . . . . 135
9.3	Iso-contour queries . . . . . 136
9.3.1	Gradient descent routing for iso-contour queries . . . . . 136
9.3.2	Value restricted routing . . . . . 137
9.4	Simulations . . . . . 138
9.4.1	Preprocessing cost for contour tree construction . . . . . 139
9.4.2	Cost of iso-contour queries . . . . . 139
9.4.3	Load balancing . . . . . 140
9.5	Conclusion and future work . . . . . 140

<b>10</b>	<b>Differential Forms for Tracking and Searching 141</b>
10.1	Introduction . . . . . 141
10.2	Differential one-form on cell complexes . . . . . 144
10.2.1	Boundaries and boundary chains . . . . . 144
10.2.2	One-forms and tracking forms . . . . . 145
10.3	Algorithms . . . . . 148
10.3.1	Constructing a tracking form . . . . . 148
10.3.2	Containment queries . . . . . 150
10.3.3	Search queries . . . . . 151

10.3.4	Update costs . . . . .	151
10.3.5	Network holes, fault tolerance and network dynamics . . . .	152
10.3.6	Tracking without target locations . . . . .	153
10.3.7	Aggregation of signal over all nodes . . . . .	153
10.3.8	Completely mobile networks . . . . .	154
10.3.9	Contour tree computation . . . . .	155
10.4	Simulations . . . . .	156
10.4.1	Update costs . . . . .	157
10.4.2	Search costs. . . . .	158
10.4.3	Aggregate range queries. . . . .	159
10.5	Conclusions . . . . .	160
<b>11</b>	<b>Conclusion Chapter</b>	<b>161</b>
	<b>Bibliography</b>	<b>163</b>

## *Preface and Acknowledgements*

Our universe operates distributedly, with local information. People act individually to form communities. Communities interact to form civilizations. Life works distributedly at all levels, forming adaptive, resilient, intelligent systems. Most remarkably, neurons in the body sense and communicate to perform the most sophisticated computations known. It is perhaps worth investigating distributed sensing and computation just to get some hint of how such organized complexities arise. The goal of this dissertation is much more humble. We will look at artificial sensor networks as a model for distributed computation, and see how much we can do with it. In fact we restrict ourselves to identifying some properties of the distributed information that help in fast distributed response to specific questions.

Geometry is one of the ways of looking at distributed information. The concepts of *neighborhood*, *locality* and *metric* are fundamental in geometry, as they are in sensor networks and distributed information processing. Geometry has many different manifestations. We refer to it in the most general sense – spanning all the possibilities. In the course of the dissertation, we will use at our convenience the particular forms that suit our needs. They may be algebraic or differential topology, non-euclidean geometry, differential geometry or simply a metric space. At this level of generality, many intuitive ideas are geometric even if they do not appear so at first sight. Natural solutions to sensor network problems frequently adhere to continuity and locality in communication making them geometric. These observations suggested geometric abstractions as a topic for this dissertation.

Sensor networks has been an active research topic in recent years; many technologies and algorithms have been developed. Many researchers have worked in this field, and their work, books and papers have formed the foundation of the dissertation. It is difficult to cite and discuss all the relevant works in the space and time available. Any attempt at an exhaustive review is futile anyway, because the hundreds of papers to be published in the next few months are will render the list obsolete. I have made the effort to review the relevant questions in sensor network research and indicate previous works that have inspired the results presented. I hope this will help the reader to follow the discussions. My sincere apologies for any omissions.

The following chapters rely on some incredibly elegant concepts. I was fortunate to have a chance to study and spend time on them. This made the process of research much more enjoyable. In times of difficulty, I found hope in the immense possibilities of their rich structure.

I am grateful to my advisor Jie Gao for introducing me to such beautiful concepts. Her ability to spot ideas that are simple, effective yet fundamental is truly astounding. With infinite patience, she persisted through many discussions that taught me to identify interesting questions and to approach problems in interesting unconventional ways. We had a wonderful time.

Working with Joe Michell and David Gu was always a pleasure. I learned a great deal in the process of collaborating with them. Equally educational were their classes. Thanks to Joe for a lot of helpful advice and encouragement.

Alon Efrat, Samir Das and Leo Guibas have been very encouraging and have always taken interest in my research. I would like to thank them for reviewing my papers and many helpful suggestions and comments about research and presentation. I.V. Ramakrishnan, Jennifer Wong, Estie Arkin, Michael Bender and Himanshu Gupta have encouraged me in many ways at different times.

I would like to thank Xianjin Zhu for her help and support. We made an effective team. The simulations in chapters 3 and 4 are completely her work, and we collaborated on many other projects not covered here. Thanks to Xiaotian Yin and Wei Zeng for the implementation of the Ricci flow embedding algorithm in Part II.

Thanks to the Stony Brook University and the Computer Science department for making it easy in many ways for me to learn and research exciting new topics. Beyond the Computer Science department, I attended some superb classes at the departments of Mathematics, Applied Mathematics and Economics. This contributed greatly to developing my research style, and broadening my horizon of possibilities. I came to know many interesting people at Stony Brook. Thanks to Abhijit Gadde, Mithun Bhattacharya, Prasad Hegde, Prerit Jaiswal, Ritwik Mukherjee, Somnath Basu and Soumya Mohapatra for making my time at Stony Brook enjoyable and memorable; and for many discussions on topics related and unrelated to research. Special thanks to Ritwik and Somnath for helping me understand various aspects of Geometry and Topology. The algorithms in chapter 10 were inspired by a discussion with Somnath on differential forms.

Thanks to all members of my family for their encouragement. My parents have been very supportive and optimistic. Without their encouragement, it would have been impossible to complete a project of this magnitude.

*Rik Sarkar*

*Stony Brook  
May 2010*

# Chapter 1

## Introduction

A sensor network is a distinct type of computing platform. It consists of many tiny computers interconnected into a somewhat arbitrary network. Each device has its individual source of data and computing capabilities, the overall quantity of data is therefore large, and spread over many components. A sensor device does not have direct access to most of the information, therefore on its own is capable of only minimal tasks. This sets the network apart as a system where intelligent communication is the key. Unlike regular networks, communication has to be closely interleaved with computation.

Sensor networks are data oriented [60, 129, 139]. The purpose of a sensor network is to obtain and manage data, and respond to queries. These queries may sometimes require knowledge of global properties. For a simple example, consider a network where devices measure temperature. A very natural question is to ask for the average temperature. Average is a global property and requires information from all the sensors, therefore the computation requires communication all over the network. The challenge is to keep communication and energy requirements low when performing a computation.

These considerations have made sensor networks into a broad field of research with requirements for new devices, and new methods for managing devices, data and communication. Effectively, it is a new model that requires adaptation at every level of computing from hardware to applications. Accordingly, specialized devices [71, 72], operating systems [6, 35], databases [106], protocols [41, 93, 152, 161] and many other technologies [73] have been developed for sensor networks. The novelty and generality of the model have generated interest in all areas of computer science and electrical engineering.

Networked information processing have further connections to a range of general subjects. Later chapters of this dissertation utilize ideas from mathematics and social networks. Some recent and significant developments in mathematics and algorithms are seen to have applications in sensor networks. These relations are not incidental or in the manner of analogy, but in a very precise sense with rigorous correctness. In the other direction, the quest for efficient distributed algorithms for sensor networks brings to surface some of the most fundamental questions and concepts in mathematics, theoretical computer science and networks.



The first section of this chapter discusses more about sensor networks and some of the important topics in network and data management. The next section explains the close relation of sensor network issues to geometry and why a geometric viewpoint is adopted in this dissertation. The last section explains the ideas behind the organization of different parts of the dissertation.

## 1.1 Overview of sensor networks

Advances in miniaturized electronics and communication have made it possible to create smart networked sensors. These devices are capable of monitoring physical parameters of the environment such as temperature, humidity, or presence of vehicles. Modern sensors are equipped with a micro-controller and therefore capable of performing computation on acquired data. Computing ability allows more intelligent sensing. For example, if a sensor determines that the environment is changing rapidly or some other event of interest is taking place, it can decide to take samples with greater frequency and possibly issue an alarm; at times of inactivity, sensing frequency can be lowered to save energy and storage.

The networked communication capabilities open up a range of possibilities in smart sensing. Communication allows a sensor to send out notifications of significant events, while a user can query a sensor at will for data without the necessity for physical access. Most significantly, communication capabilities allow sensors to talk with each-other to collaboratively determine higher level implicit events [40]. For example, an individual sensor may only be able to detect the presence of a vehicle nearby, but by communicating with other nearby sensors that sense the vehicle at different times, they can determine the velocity of the vehicle.

Such collaborative sensing is not simple to design or implement. A sensor network has little networking capabilities – there are no routers, Internet style IP-addresses or DNS services. In addition, the lower network layers are unreliable due to wireless interference, noise and often unreliable equipments. Even a basic scheme where each sensor performs minimal processing and forwards the data to a server, poses non-trivial challenges.

Let us establish a model for sensor networks so that the important questions and topics in collaborative operation can be discussed without fear of ambiguity. A simple model is desirable over a complex one. As will be discussed later, this model is in fact sufficiently general to cover a wide range of scenarios.

### 1.1.1 Sensor network model

Sensors and networks are often designed for specific tasks. There are however certain features that can be considered general for these networks. The following is a list of these properties followed by a more detailed discussion of each. These will be assumed in the following chapters. Certain topics may require additional constraints, and those will be introduced later as needed.

1. The network consists of a large number of small devices, often called sensor *nodes*. The number may be in hundreds, thousands, or more.
2. Each device is equipped with one or more “sensors” that sense certain physical quantities.
3. Each device has a limited source of energy, equivalent to a small battery. In some cases it may be possible to harvest energy locally at a low rate from solar power or other sources.
4. Each device has a small computation power - a low power CPU and a limited memory.
5. A device is equipped with a wireless radio that it can use to communicate within a small range.
6. A sensor is placed at a location in space. The location may or may not be known, but it affects its sensor reading and its role in the network. The overall set of locations spanned by the sensors is large compared to the transmission range of the wireless radio on a node.
7. Individual devices and communication links are unreliable. Algorithms and protocols should not depend critically on unfailing long term operation of individual sensors.
8. Identity of Individual devices are not significant. Device addressing is data centric.

The characterizations above are admittedly not very precise. This is because character of a sensor network varies by its aim, construction, domain of deployment, and other parameters. Depending on these, a particular network may have different properties from another. This variation allows viewing networks from many different perspectives. The rest of this subsection elaborates these characteristics. Readers familiar with sensor networks may wish to skip ahead to subsection 1.1.2.

**Large number of devices.** This sets sensor networks apart from ad-hoc networks of fewer nodes. For small networks the scaling issues such as routing table size, or cost of accessing all the data do not play a major role, therefore it is the large networks that are worth investigating from a networking point of view. This is also a different scenario from other large networks such as the computing clusters or the Internet, which consist of powerful equipments and substantial resources. Sensor networks therefore face the scaling problems of large networks within the restrictions of ad-hoc networks. In the following, we denote the number of nodes by  $n$ .

**Each device is equipped with sensors.** Sensors may be simple equipments to measure temperature, humidity or other physical quantities. In some cases there may be more sophisticated sensors such as accelerometers or gyroscopes. In certain cases, interesting sensing may not require specialized equipment. For example, the wireless communication system itself may be used to detect presence of other wireless enabled devices nearby. In certain cases local parameters of the network itself, for example, routing load, energy or local density of nodes may serve as the “sensed” quantity. This suggests using the sensor network perspective of information processing to handle networking questions (see chapter 9).

**Small energy sources.** Connecting to regular power lines may not always be practical. Therefore, we consider a model where sensors are supplied by a small energy source. This may be a battery or local harvesting of energy. In either case, our focus will be to keep energy consumptions low to allow each sensor to operate for long durations without running out of energy.

**Computing resources.** The requirements for small size and low power consumption force the devices to be constructed with restricted resources. The large number also makes it financially impractical to use sophisticated devices. To be more precise, we assume that the memory is small compared to the size of the network. Data of size  $\Omega(n)$  is impractical at a node. Correspondingly an algorithm of  $\Omega(n)$  complexity is undesirable as a drain on energy and time. It is however possible to use some nodes with greater resources to complement the model. This topic is discussed in subsection 1.1.3.

**Wireless communication.** The wireless communication ability defines much of the characteristics of the system. While communication enables innovative uses of the network, wireless communication is also the largest drain on energy of a device. Further, since the wireless medium works on local broadcasting of data, many nodes transmitting at the same time can cause excessive interferences, slowing down all communications.

The wireless radio on a sensor is not very powerful. A powerful radio is not practical on a small energy constrained device. If many devices are transmitting within range of each-other, that can slow down communication, therefore, from an efficiency and throughput viewpoint, low power radios are desirable. These restriction gives rise to many of the issues and research questions in sensor networks.

It is assumed that all the sensors are connected into a single network. Of course, it is possible to have many networks, but our intent is to utilize the communication abilities of nodes for collaborative data processing, so we consider one connected component at a time.

**Location and distribution of nodes.** This is an important aspect of sensor networks. Location is often important with respect to sensor data, because data may be relevant only when it is coupled with the information of where the reading was

obtained. Obtaining location for a sensor is a challenge on its own and much research has been devoted to localization (see Subsection 1.1.2). Some such approaches may be suitable in certain scenarios; in other cases, GPS service may be available or a node may be able to obtain approximate locations from nearby or passing GPS enabled devices.

In many situations, knowledge of location may not be a valid assumption at all. But locations of sensor nodes still play an important role, particularly in determining the structure of the network. The low power radio on sensors means that a node typically will be able to communicate directly only with a small set of nodes. All other communication must be through *multi-hop routing* where nodes forward messages on behalf of others. These multi-hop paths are restricted to the communication graph determined by the low power sensor radios. The locations therefore determine the network.

In general, it is desirable to have nodes densely deployed - this gives better sensing resolution and redundancy. However, it is not economical to have a very high density, and a high density of active nodes may increase wireless interference. Therefore, in subsequent chapters we will assume the density to be bounded from above. There is also an implicit lower bound on a network, since a very low density network is likely to lose connectivity. In general, however, we allow network domain to have large regions without sensors – often called *holes*.

**Unreliable nodes and links.** The small energy source makes sensors likely to fail once the battery runs out. Once a device fails, its communication links are also lost. The need to have a large number of low cost devices, that are possibly of different types and from different manufacturers makes it difficult to have consistent quality and behavior. Wireless communication itself often fails transiently and individual links cannot be relied on. On the whole, it is preferable to have methods that do not rely on long term reliability of individual nodes and links.

**Data centric addressing.** Given a large number of devices monitoring some external quantity, an individual sensor's data is not useful. "What is the temperature at sensor 42673?" - is not the most interesting question. Much more useful are aggregate questions "What is the average temperature?" or "What is the maximum temperature?" or "Which sensor has the maximum temperature?" Answering these questions is a different sort of challenge.

Location also plays an important role in this respect - "What is the temperature at location (X,Y)?" may be of interest. The answer possibly lies in finding the sensor(s) nearest to that location. In fact, sensor network methods frequently rely on location as a method of addressing instead of node id. The advantage is that every node is then given a unique id as its location, also every location in the underlying continuous domain maps to a node – the one nearest to that location.

### 1.1.2 Common questions in distributed sensor networks

The combination of properties and restrictions described above have given rise to questions that are unique to sensor networks. Even conventional well studied topics like routing must be approached differently in this model.

The restrictions on communication costs force us to consider distributed processing of data. Gathering all data at a sink has the shortcoming that much resource is spent in simply forwarding data. Consider a network of 5000 nodes and one sink. If all the data is sent to the sink, then the few nodes near the sink have to forward thousands of messages in each pass when every sensor takes one reading. The numbers can be higher when we consider retransmissions and acknowledgments. Other than the loss of energy in such forwarding, this slows down data processing by slowing down the routing. In certain applications, some types of data may be more significant than others. For example, an unusually high temperature at a sensor is more significant than regular temperature readings. Again, high temperature at a close group of sensors is more significant than high temperature at a single sensor. It is faster and more efficient to be able to make such decisions locally, and inform the sink only of the aggregate decision.

Of course, not all decisions can be made locally. Certain information are inherently global, and some facts may only be deduced with a powerful central computation. It is however desirable to have distributed local processing to the extent possible, and reduce communication needed for the central operation by distributed pre-processing. Therefore it is important to know to what extent computations and deductions can be kept distributed. This will be the approach in this dissertation – to perform non-trivial tasks while keeping operations as local and distributed as possible.

The following are some of the important research topics in distributed sensor networks. These have been considered in various forms and using different tools. The diversity in possible constructions and applications of sensor networks makes it difficult to solve any of these problems completely. Therefore, all these questions remain open in some respect or other. We mention below only a few of the problems and approaches that are relevant to the core chapters of the dissertation.

**Wireless communication.** The unreliable, possibly noisy nature of wireless links coupled with signal interference make this a difficult question. Wireless communication at the physical and medium access layers has been studied extensively [41, 93, 152, 161] for sensor networks, and from the viewpoints of different objectives. Specialized radios and protocols have been built. In this dissertation we will not consider the details of wireless communication. The existing work lets us assume that hardware and protocols exist such that devices are within communication range of each-other can communicate freely.

**Localization.** This topic deals with finding the locations of the nodes from the knowledge of existing communication links. Since pairs of nodes are typically within communication range of each-other only if they are within a reasonably short

distance, the communication graph of the network is to an extent determined by the relative locations of nodes. It is natural to ask if it is possible to determine the locations, given the knowledge of the communication graph. Typically, this problem is considered in the case where all nodes are in a plane. Of course, the question of finding perfect locations is impossible to solve. Consider nodes A, B and C within a very small distance of each-other, so that they are all in communication, and in addition are in communication with exactly the same set of other nodes. There is no way to distinguish their locations purely from the communication graph.

A more reasonable question is to try to deduce a set of locations that is consistent with some knowledge of the communication links. For example, one can assume the communication links to be of the unit-disk nature, where two nodes are in communication if and only if they are within a unit distance of each-other. Another model suggests that radio signal attenuates with distance and therefore for each link the distance between communicating nodes is known from the signal strength. Both these models are far from reality, because radio signals do not attenuate uniformly at all directions and distances. The precise propagation characteristics depend on the irregularities of the transmitter as well as the surrounding environment and other transmitters. However, even under such unrealistic assumptions, finding a consistent set of locations is known to be NP-hard [39].

We will not address the question of localization in this dissertation. Much work has been done on this topic [12, 136, 137], and in most cases this has been shown to be an intractable problem [15, 39]. The improvement of GPS technologies gives us hope that in near future it may be possible to rely on it for localization. All sensors need not carry a GPS unit. Presence of some GPS positioned sensors will allow us to approximately localize the rest. Given the proliferation of GPS in cellular phones and hand held devices, it may be practical for sensors to obtain locations from nearby GPS enabled units.

**Routing.** Routing on sensor networks is different form routing in other environments. Here there is no routing infrastructure, and the nodes themselves have to take responsibility for routing. The number of nodes being large rules out flooding based methods such as AODV [125] and DSR [75].

Locations play an important role, leading to various forms of *geometric routing* [13, 77, 82, 91, 92]. Additionally, there are other forms of geometric routing that rely on virtual coordinates [16, 17, 118, 128]. In these methods, coordinates are assigned to nodes artificially, and may not be related to their true location. This allows a certain flexibility that aids geometric routing. Routing without the knowledge of locations is a particularly interesting challenge, and various methods including virtual coordinates, and landmark routing have been applied to this case. Landmark routing [43, 47, 107] is a method where certain nodes are designated beforehand, and other nodes know their respective distances to these landmarks. The routing proceeds in multiple stages by moving in the direction of different landmarks.



**Aggregation.** Aggregation may refer to either collecting all the data to one place [105], or to the process of computing aggregate functions or summary of the data such as sum, average, maximum, minimum or others. In either case, it requires information from all the sensors, and repeated aggregation can be a challenge when it is important to keep communication requirements under control. The unreliable nature of communication links can create additional difficulties [27, 117, 140].

A variety of methods have been applied to aggregation. These have used methods from distributed systems, as well as those from database, signal processing and information theory.

**Preprocessing and spatial query.** Sensor networks are expected to be able to respond to queries that relate to the monitored space. For example, an aggregate value (e.g. max or mean) in a given region  $\mathcal{R}$ , or the problem of finding regions with a reading higher than some user specified value. Ordinarily, answering this question requires aggregating the values of all sensors in  $\mathcal{R}$ . The efficiency can be improved by constructing some initial summary of the data. Such preprocessing constitute an important aspect of sensor data processing. The challenge here arises from the global nature of the question, whereas we would prefer to pre-process and respond using local distributed methods that do not require global coordination among sensors [52, 56, 62, 99]. Parts I and III in the following deal with questions of distributed uncoordinated preprocessing to efficiently answer global queries.

**Handling mobility.** Mobility of sensors introduces many more considerations. The network changes due to mobility makes it difficult to execute usual operations of communication and routing. Change in locations of sensors makes it difficult to correlate data. The final chapter of the dissertation presents an elegant method for tracking and adapting to mobility for certain types of questions.

### 1.1.3 Sensor network as a general model

Let us reconsider our view of sensor networks. The model appears to place many restrictions on the nature of a network, and one might question its validity in any foreseeable real system. In fact, it is unlikely that any real system will satisfy all the assumptions made here. It is therefore important to verify the sanity of the assumptions that they are sufficiently general to be useful and we are not developing methods that are applicable only in an unlikely or impossible universe.

The model we have set up is in general more difficult to work with than others. In most cases, the assumptions constrain the resources available to the algorithm designer. A method that works with a more restricted set of resources also works when better resources are available. This makes the model more general than the configuration of individual sensor networks, and therefore an effective baseline for the initial algorithm or protocol design.

**Heterogeneous networks.** The assumption about the sensors do not address any possibility of different nodes having different shares for resources. The subsequent chapters treat all nodes to be identical to each-other in hardware and resources. A real system is likely to be built of different types of components, particularly, some nodes (let us call them super-nodes) may be more endowed in computation, storage and communication resources than others. Treating them to be equal to the small low power nodes may look like a suboptimal use of resources.

The difficulty of trying to design algorithms directly for a heterogeneous network is that the configuration of such a network cannot be predicted beforehand. There are no specific standards for the positioning and design of super-nodes. Other conditions, for example the physical environment and financial considerations may restrict the power and distribution of available super-nodes in networks. Therefore, we need to design methods that are oblivious to the distribution of super-nodes.

Restricting design to homogeneous systems is not as wasteful as it may seem. An algorithm or protocol designed specifically with some distribution of super nodes in mind may not be able to operate at all, or work very inefficiently when the distribution of super-nodes is very different. The same problems may appear if some super-nodes fail. However, an algorithm that is efficient on the homogeneous networks oblivious to super-nodes, also works on a network with super-nodes. To better utilize the resources it is always possible to elect the super-nodes as leaders who gather data from nearby sensors and perform part of the computation on their behalf. This makes the homogeneous network model the more general one, and covers the possibilities of heterogeneous networks.

**Distributed computation.** Similar reasons as homogeneous systems suggest a more distributed computation effort compared to centralized methods. A design for distributed computation works when computation can be or must be done in a centralized fashion - the centralized entity can simply emulate the operations of the interconnected nodes. This is particularly effective for the sensor network model. The restrictions of small computation power and memory means that any algorithm designed for the model is efficient overall and can be emulated relatively easily by a centralized computer. The model therefore requires designing methods that are efficient in both distributed and centralized settings.

**Communication.** The model mentions that the communication is assumed to be wireless. But wireless communication is difficult to characterize. What we mean really by wireless communication is the following.

1. For each node, there is a set of *neighbor* nodes that it can communicate bidirectionally with at a bounded cost.
2. The number of neighbors is bounded for each node.
3. The communication graph is the graph created by connecting each node to its neighbors with unweighted edges.



4. The metric of the graph (each edge having equal weight) equals the communication costs in the multi-hop network.

Note that the assumptions do not rely on any fundamental way on the communication being wireless. A neighbor need not be one in direct wireless communication. As long as it can be reached at some bounded cost, it qualifies as a neighbor. The significance of the wireless model really is that the communication graph has bounded degree. Depending on the scenario certain other assumptions may be made. This topic will be discussed further in the next section in reference to geometric treatment of sensor networks.

The model therefore is suitable even when some other efficient infrastructures (wired, wireless or otherwise) are available for long distance communications. Even if such facilities are available, it is reasonable to assume that nodes physically nearby can communicate at a low cost. In distributed information processing, this is important. A sensor frequently needs to communicate with its physical neighbors to compare data. For example, to decide if a node is a local maximum, its value needs to be compared with others near it. Any reasonable communication infrastructure should preserve this property that nodes very close to each other can communicate at a low cost. As before, we are taking the more constrained option that only the few close links are low cost. An algorithm that is efficient under these conditions will also be able to operate when better resources are available.

The consequence of these properties is that we are considering a model that reduces to a type of graph, with certain data associated with the nodes and edges. This is very general, not only for networks, but for questions related to graphs, metrics and information processing. Thus, ideas developed on this model may well serve as a starting point for more general algorithms.

The purpose of using this model is to test the limits of distributed local computation. In using it, we may be sacrificing certain possibilities. However, every model has its shortcomings. In this case, this is the price to be paid for the generality of the model. In cases where an efficient solution is not possible by distributed computation, an investigation can still be fruitful. It may bring to light new structures and properties associated with the problem or may suggest efficient approximations - both of which could be useful in other scenarios.

## 1.2 Geometric abstractions

Geometry is visual. This gives a certain advantage in working with geometric models. It is possible to apply intuitive thinking and diagrams in solving problems and representing ideas. The solutions and ideas can then be verified for correctness with the associated formal framework. In the other direction, a formal system, when cast into a geometric form becomes easier to visualize, understand and manipulate, thereby opening new possibilities. The subject has been investigated for centuries, and many interesting structures, results and tools have been invented. Geometric interpretation of a problem makes it possible to bring this arsenal into operation.

To apply this approach to sensor networks, the elements of the network model (including nodes, communication links and data) need to be associated with the objects in a suitable model of geometry. The geometric representation forms the *Abstraction* in the given context. It is then possible to utilize different known properties associated with the geometric abstraction. Geometric models and objects that have been studied are rich in structure, which can create novel possibilities for applications and algorithms. The chapters of this dissertation are based on several such versatile structures.

Geometry comes in many different forms and flavors. For each question, the suitable geometric idea must be selected. In certain cases, it may be possible to apply different abstractions to the same problem to obtain different structures and therefore different properties and applications.

### 1.2.1 Geometry in sensor networks

There are two common aspects of sensor networks that suggest a geometric approach:

1. Node data are often associated with locations.
2. Communication is wireless.

Neither of these is necessary, but that they can be found in sensor networks has encouraged the development of geometric methods. As will be seen in later chapters of the dissertation, geometric ideas can be generalized sufficiently to provide solutions to cases even where neither of these hold.

Let us consider for now a situation where nodes sense a certain quantity and are aware of their own locations. In such a case, the data forms a sampling of a function over the domain that is being monitored. Given this data, it is possible to analyze this sampling of the function to recover information about the function implicit in the data. The precision of the answer is necessarily restricted by the resolution of the sampling, but that is a limitation of any real data. The implicit information beyond the raw data is the target of any computational method. Consider a few types of questions one may ask to be computed from raw data - for example, the number and locations of maxima or minima. or the contours at a particular signal value, or a compressed summary of the function, or aggregate in a certain region. Similar questions can be asked of derived quantities - for example, the region where the reading is changing the fastest. All these questions are geometric in nature. In particular, they depend on comparison with data items close to each other in space, or time, or both.

For multiple quantities being monitored the same ideas hold, but now at a higher dimension. Locations and time can also be treated as additional dimensions of data to get a unified view. The upshot is that when locations are available, the domain and the important questions about it are fundamentally geometric. It is therefore natural to consider geometric methods to address them.

The most natural concept of a *Location* is as cartesian coordinate  $(x,y)$  in the plane. But location need not always be in the plane. It may be in higher dimensions, and also in non-euclidean spaces. For example, it may be more practical to address a sensor location as L1="Computer Science Building, East wing, Floor 2, Server Room." Of course, designing such a location description scheme is a challenge in itself. But putting that aside, the concept here is that locations may be from more complicated spaces. The important aspect of locations is that they can still supply *proximity* information. For example, L2="Computer Science Building, East wing, Floor 2, Corridor 3" may be close to L1. This proximity can be taken into consideration the same way that nearby data is compared for locations in the euclidean plane. An ability to determine *neighborhoods* this way makes it possible to apply geometric techniques to process sensor information.

Wireless communication plays a strong role in determination of proximity. Radio signal degrades with distance. Effectively, *nearby* nodes are in direct communication with each-other. This naturally creates neighborhoods for nodes where information can be exchanged efficiently. Since such local communication is in any case necessary in many of the information processing tasks, it can be hoped that a geometry suitable for processing the data can be associated with the communication graph.

The *Unit Disk Graph (UDG)* has been considered widely as a theoretical model to abstract the geometry of the communication graph. In this model, two nodes are in communication if and only if they are within a unit distance of each-other. While unit disks can be defined on any metric, they have generally been considered as unit disks in  $\mathbb{R}^2$ . This makes it possible to apply local algorithms in determining global structures that help network operations. For example, in this model, it is easy to compute planar subgraphs such as relative neighborhood graphs and Gabriel graphs of the communication graph. The planar structure of these graphs make it possible to apply geometric techniques otherwise not applicable.

While UDG gives a simple model in the local neighborhood, the metric determined by it still does not quite represent a large network. Due to presence of large regions without nodes, commonly known as communication holes, the UDG distance may be largely disproportionate to the true euclidean distance. Equally importantly, true radio communication does not behave in a unit disk fashion. Spatial variations in radio signal propagation produces communication graphs that are much more uneven.

Note that while communication links do not quite form a UDG, they do form a metric. It is the metric of the unweighted graph created by the communication links. It is possible to apply geometric methods purely on the proximity determined by this metric without regard to real physical distances of the sensors. This is the basis of *location-free* geometric algorithms. This is an extremely general principle that can potentially be applied to a wide variety of scenarios. For example, even in domains outside of sensor networks or wireless networks.

Implicit in the discussion above is that geometry surfaces at several levels of abstraction in the question of sensor information processing.

1. **Local geometry at a node.** Location of the node, nearby nodes and local signal

characteristics determine its neighborhood in the communication graph. The combination of all such local neighborhoods creates the graph itself.

2. **Global geometry of the network.** The network has an intrinsic geometry - the hop count metric formed by the communication graph. This metric determines much of the communication possibilities in the network, such as shortest paths, number of nodes within any  $k$  hop neighborhood of a node, and costs of information dissemination.

Closely related to the intrinsic geometry of the metric is the extrinsic geometry - the *shape* of the network. This is the shape of the “cloud” of nodes embedded in a physical space. Sometimes it is useful to work on this more explicit and intuitive idea of the network shape, where it is possible to directly deal with aspects such as network boundary and holes.

3. **Geometry of the signal.** Beyond the network itself, the signal monitored by the sensors has a geometry. Imagine a signal defined over the region where the sensors are deployed - over the shape of the network. Plotting this signal in an additional dimension creates a surface over the network. It is the geometry of this surface that can be analyzed to extract hidden structures in the data and the network.

For both the geometry of the network and of the signal, an interesting question is how the local structures add up to create the large scale global properties. Since sensors are best utilized as local operators, it raises the natural question of how much of global problems we can solve through local algorithms. Much of this dissertation deals with this local to global relation. In fact much of geometry deals with questions of local to global integration. This topic will be revisited in the conclusion chapter after we have discussed geometric algorithms for sensor networks.

## 1.3 Overview of the chapters

A sensor network has two types of tasks. The first is to manage the data in the network. This implies acquiring, processing and storing the data suitably and afterwards answering queries on this data efficiently and quickly. In case of changing data, it may also be necessary to update the stored information and data structures to adapt to the changes.

The other task of the network is to manage itself. For example, localization, scheduling sleep cycles and transmissions, handling interference and routing; and other facilities that may be needed by efficient data processing algorithms.

Each part of the dissertation looks at these two types of tasks from different geometric perspectives. As will be seen, these tasks are not too different, and similar abstractions can help both. Routing will be used as the unifying network management task and will be revisited in each part. Routing can be seen as an information processing task – the search for a path in the network. In the other direction, information processing can be seen as a routing question – finding route(s) to store/retrieve

the right data. In the final part of the dissertation, routing and information processing merge to be almost indistinguishable.

**Part I: Metric Spaces and Spatial Distributions.** This part abstracts the network with its intrinsic geometry – the metric space. First we derive sufficient conditions for routing in a metric with a short stretch – a path of length at most  $(1 + \epsilon)$  times the length of the shortest path.

Next, we consider growth bounded metric spaces. This is a slightly more restricted type of metric space where the volume of a ball increases polynomially with the radius. This restriction lets the local geometry at a node be arbitrary, but restricts the global geometry to behave approximately like a euclidean metric. For this type of metric, we show that  $(1 + \epsilon)$  stretch routing can be achieved with small routing tables. The routing tables themselves can be constructed efficiently provided some approximate estimates of multi-hop distances are available. Parts of these results have previously appeared in [134].

The last chapter in this part shows that for the growth bounded model, it is possible to efficiently perform *spatial gossip*. This is a form of information exchange where nodes select exchange partners by distances. After execution of the gossip, it is possible to efficiently answer aggregate queries in sub-regions of the network. This chapter is based on [133].

The results described above that depend on a growth bounded metric actually utilize a *spatial distribution* that is known to create graphs with a *small world* structure. This structure is known to provide short stretch paths using relatively few links, which is the secret of the  $(1 + \epsilon)$  stretch routes. The short routes enable efficient the gossip method in the last chapter.

**Part II: Virtual Coordinates and Metric Deformation.** Virtual coordinates are coordinates assigned to nodes without performing localization. The availability of coordinates allows using location based geometric algorithms. Additionally, virtual coordinates permit the flexibility to modify the geometry so as to gain better properties from the geometric algorithm than when applied to raw locations.

The virtual coordinate computation in this part relies on *Ricci Flow* – a powerful technique in modern mathematics. This method can be applied distributedly. The result is a set of coordinates in the plane that allows efficient routing and data storage.

This part is based on differential geometry. The metric is modified locally to obtain global properties. The local changes to intrinsic geometry lead to a simplified shape of the network where the “holes” are circular. This simplified geometry allows us to perform routing and storage more easily and more uniformly. In fact, the simplified geometry makes it possible to eliminate the holes in a certain sense. These results have recently been published in [130, 131]

**Part III: Geometry and Topology of Information.** The last part is more information oriented than the rest. Here the emphasis is on analyzing available data to

answer queries and routing requests with respect to the data.

The first abstraction in this section is a *contour tree* that is a compact distributed summary of the data. This structure allows us to answer queries such as “What are the regions in the network where the temperature is greater than 40?” or routing queries such as “Find a path from  $A$  to  $B$  that does not go through a temperature greater than 35.” This result has been published in [135].

The second abstraction is a differential form. Using this, it is possible to track moving objects in the sensor field. The differential form essentially maintains information to be able to answer questions like “What is the total weight of objects in region  $\mathcal{R}$ ?” The solution in fact is an adaptation that lets us apply *Stokes Theorem* to sensor networks. This basic primitive allows searching and delivering messages to individual mobile objects, searching for a nearby object and several other functionalities. The method also works when the network itself is mobile and dynamic. This recent result is unpublished as yet, appearing for the first time in this dissertation.

These methods once again relate local properties to global properties, this time for the data. The contour tree is a concept from Morse theory to interconnect critical points of data in a meaningful way. While critical points are local features, the interconnections yield global properties. The differential form similarly adapts the local data in suitable ways to efficiently answer questions at larger scale.



**Part I**

**Metric Spaces and Spatial  
Distributions**





# Chapter 2

## Introduction to Metric Growth

The communication graph of a network in our model always induces a *metric*, a measure of distances, and the properties of this metric can be utilized to build efficient algorithms. An edge in the graph signifies that the end-points of the edge are within direct communication of each-other. The ‘distance’ between any two nodes is the number of communications (also called hops) that need to be executed. This distance forms a metric.

**Definition 2.0.1 (Metric).** *A metric on a set  $S$  is a function  $d : S \times S \rightarrow \mathbb{R}$ , where  $\mathbb{R}$  is the set of real numbers, such that  $d$  has the following properties for  $x, y, z \in S$ .*

1. *Positive definite:*  $d(x, y) \geq 0$ , and  $d(x, y) = 0$  if and only if  $x = y$
2. *Symmetric*  $d(x, y) = d(y, x)$
3. *Triangle inequality*  $d(x, z) \leq d(x, y) + d(y, z)$

*Function  $d$  is called the distance function, and  $S$  and  $d$  together are said to form a metric space.*

In our case, the distance function is really the communication cost between nodes, and adjacent nodes in the graph can communicate within a unit cost - by a single transmission, while nodes farther apart require more intermediate transmissions to communicate. Observe that the metric idea is not specific to the case where the graph constitutes of pairs in direct communication. It can be used in general where nodes adjacent to an edge are within a bounded communication cost or ‘distance’.

The first chapter in this part investigates the abstraction of a network as a metric space to perform efficient routing. Routing a message from a source node to a destination node is inherently a question of making decisions based on the metric. If the metric over the entire network is known, routing becomes trivial. A node with complete knowledge can always decide which of its neighbors is nearer to the destination in the given metric, and forward the message to that neighbor to advance the message further. In particular, complete knowledge makes it possible to find the shortest route from source to destination. A shortest route is generally desirable

because it delivers the message at fewest hops, therefore is in a sense most efficient and reliable. The entire metric is however a great deal of information, and not really practical to know except in very small networks. more precisely, it needs at  $\Omega(n)$  storage and processing at each node, which is impractical in our model.

In chapter 3 we present a concept that shows how a path almost as short as the shortest one can be found without always storing large amount of data for the routing. This works even when the knowledge of the metric is only approximate. Formally abstracted as an *approximate distance oracle* that takes as input the pair of nodes to return their metric distance to within constant factor. Under these conditions, we derive a set of sufficient conditions that when satisfied, will deliver the message by a path almost as short as the shortest path. The essential concept is general, and holds for any metric.

Certain network metrics have additional structure. These have been considered in different contexts to obtain better routing properties. For an unweighted graph  $G$  and we denote by  $N_r(p)$  the set of nodes within  $r$  hops from  $p$ , also called the ball of radius  $r$  centered at  $p$ . A graph is said to have  $\Delta$ -*expansion rate* if  $|N_{2r}(p)| \leq \Delta |N_r(p)|$ , for any  $p, r$  [4,76]. A graph is said to have *doubling dimension*  $\Delta$  if any ball of radius  $2r$  can be covered by at most  $2^\Delta$  balls of radius  $r$  [64]. A graph is said to have *bounded growth rate*  $\Delta$  if  $|N_r(p)| = O(r^\Delta)$  [101]. All three models try to capture that the metric growth is restrictive. For example, a binary tree does not satisfy any of the definitions above.

In the sensor network model, we impose the (upper and lower) bounded growth rate model. If we place at most a constant number of sensor nodes inside any unit disk and the holes in the sensor networks are not very fragmenting, the number of nodes at  $k$  hops from a node  $p$  will be around  $\Theta(k)$ . More precisely, we consider a graph such that the number of nodes at a distance exactly  $r$  from  $p$ , represented by  $|\partial N_r(p)|$  is bounded by  $|\partial N_r(p)| = \Theta(\rho r^{\rho-1})$ . This is equivalent to  $|N_r(p)| = \Theta(r^\rho)$ . This model is shown to imply a bounded doubling dimension, therefore it is stronger.

The significance of this model (and the implied bounded doubling dimension) is that it acts as a coarse approximation of a Euclidean metric. Imagine the number number of nodes  $|\partial N_r(p)|$  in any ball to represent the *volume* of the ball in the metric space. In Euclidean space, this volume grows in direct proportion to  $r^d$  where  $r$  is the radius and  $d$  the dimension. The growth bounded model is analogous and asymptotically show similar growth, but the  $\Theta(r^\rho)$  allows a sufficient relaxation that a discrete graph can realize the growth. Consider for example the  $\mathbb{Z} \times \mathbb{Z}$  lattice of vertices with the natural two dimensional grid as the graph. This embeds isometrically into  $\mathbb{R} \times \mathbb{R}$ . The the area of a ball of radius  $r$  in the grid graph is approximated to within constant factors by the number of lattice points in a ball of same radius in the euclidean plane. The model we employ expands this idea to more general graphs and allow for holes, occasional longer links and other irregularities to create a better representation of network graph that can still be analyzed in analogy with Euclidean plane.

The bound on metric growth allows construction of small world graphs. For any two nodes that are a distance  $r$  from each-other, a link is added between with prob-

ability proportional to  $\frac{1}{r^p}$ . J. Kleinberg [86] showed that these links produce a *small world graph* where any two nodes are at a small number of hops from each-other. This property is utilized in each of the following two chapters. In chapter 3 the property is used to create a routing scheme where the small world links coupled with the sufficient conditions of small stretch routing guarantee efficient routing with only a small amount of data stored at each node. In chapter 4 the small world property is used (following Kempe et al. [79]) to build an aggregation scheme where nodes exchange data randomly according to the distribution of the small world links, and thereby obtain aggregate information for different neighborhoods - otherwise called multiresolution aggregates.

In a sensor network, there is no freedom to change the network graph to insert additional edges, therefore it is not possible to create actual small-world links. Instead, these links act as a suggestion for useful node-pairs for communication. The actual communication is still achieved by multi-hop routing between these nodes. In the following section we review the research in routing, small world models, and information processing that are related to the chapters in this part.

## 2.1 Review of related research

In this section we survey related work and establish their connection to our results.

### 2.1.1 Spatial distribution and routing

The spatial distribution in selecting the long links in our chapter coincides with the small-world model and decentralized search proposed by Kleinberg [85, 86] to model Stanley Milgram's famous experiment [110, 151] on the small-world phenomena in social networks. The setup in the small world model is the following. Given a 2D grid (possibly of infinite size), each node chooses a long link with probability  $1/r^2$  where  $r$  is the length of the long link. Together with the four neighbors per node on the grid, a greedy routing with the location of the destination can be achieved with  $O(\log^2 n)$  jumps (on either short links between neighbors on the grid or the long links constructed) with high probability. Notice that in this setting an accurate distance oracle is actually available and greedy routing on the original grid suffices to deliver the message along the shortest paths on the grid. In the small world literature people care most about adding extra long links to create short paths between any two nodes. In our setting the long links are realized as paths in the original network. Nevertheless, our results show that if each node chooses  $O(\log^2 n)$  long links, a slightly more sophisticated but distributed routing scheme with long links has  $O(\log n)$  jumps, and also a total travel distance at most  $1 + \epsilon$  of the distance between source and destination on the grid.

The spatial distribution has been explored in a number of other data delivery and information dissemination scenarios in sensor networks, e.g., for adding long communication wires to reduce power consumption [138], or, for gossip and locality-sensitive information exchange [79, 133].

**Small state routing in sensor networks.** To deal with the problem of local minimum in geographical forwarding, various techniques have been proposed to solve the problem of ‘routing around holes’. Earlier proposals assume unit disk graph model on the communication network and propose to planarize the network and apply face routing [13, 77, 92]. Such planarization unfortunately fails badly in practice due to complex radio characteristics [83]. Improvement of the planarization process may selectively remove crossing edges [61], or use a generalized face routing on graphs with crossing edges [163], or planarize an abstracted graph to filter out the local connectivity irregularity [49]. Alternatively, one may also develop virtual coordinates to support greedy routing [16, 43, 47, 118, 119, 128]. Most of them do not guarantee small stretch routing and often require preprocessing to first discover and understand the network topologies.

We explain two protocols in more details as they are more relevant and compare with our scheme. In virtual ring routing (VRR) [17], proposed by Caesar *et al.*, the nodes are ordered by their node IDs (or any other identifiers) on a ring and the paths for nearby nodes on the ring are stored in the routing tables of the nodes on these paths. Notice that nearby nodes on the ring may be far away in the communication network. When a packet is routed to a destination, it is delivered by using the local routing table to the next hop on the pre-constructed path leading to a node closest to the destination in the ID space. VRR can be understood as building long links connecting nodes with adjacent IDs, which can be arbitrarily far apart in the network. The routing table size is roughly in the order of  $O(\sqrt{n})$  in a uniform and dense network. And there is no guarantee on the path stretch.

The small state and small stretch (S4) routing by Mao *et al.* [107] adopted the idea of compact routing schemes by Thorup and Zwick [148, 149]. The basic idea is to select about  $O(\sqrt{n})$  landmarks. These landmarks flood the network and other nodes record the hop count distance to these landmarks. In addition, a node  $p$  also maintains routing table entries to the nodes that are closer to  $p$  than their closest landmarks. The routing table size is about  $O(\sqrt{n})$  and a greedy routing scheme is guaranteed to deliver the message to the destination with maximum stretch of 3. By exploiting the geometric properties of the sensor network deployment, we are able to get  $1 + \epsilon$  stretch and reduce both the number of landmarks and the routing table size to polylogarithmic in the network size.

**Compact routing in general.** From a theoretical aspect, compact routing that minimizes the routing table size while achieving low stretch routing has been studied extensively [121]. There are two popular models in the literature, the *labeled routing model* and *name-independent routing*. In the labeled routing model [28, 37, 149], one is allowed to produce for each node a label (typically of polylogarithmic size) such that routing is done with the labels of the source and destination. In the name-independent model [4, 89], the nodes are given generic IDs that are independent of the routing scheme. Thus routing is inherently more difficult as the routing scheme needs to also find out where the node is. To understand this in the case of sensor network routing, name-independent routing works directly on the node IDs (such as in the virtual ring routing scheme). If we use geographical locations or any other

virtual coordinates, such coordinates are the ‘labels’ and to complete the solution one needs to also employ a location service (as in [98]) that maps node IDs to their geographical locations or virtual coordinates. Put in this perspective, our scheme stays in between the labeled model and the name-independent routing model. We have a label of the nodes (such as the geographical locations) naturally, but the labels only give imperfect distance information and do not guarantee delivery.

Generally speaking, the theoretical results in compact routing in a graph whose shortest path metric has a constant doubling dimension are able to obtain, with poly-logarithmic routing table size,  $1 + \epsilon$  stretch routing in the labeled routing scheme (see [20] and many others in the reference therein), and constant stretch factor routing in the name-independent routing scheme [3, 89] (getting a stretch factor of  $3 - \epsilon$  will require linear routing table size [3]). The results here are all centralized constructions and aim to get the best asymptotic bounds. Our focus in the following chapter is on a principle for distributed implementation at each node and its practical implementation in the scenario of ad-hoc sensor network routing.

### 2.1.2 Information processing in sensor networks

Existing approaches for processing information in sensor networks can be classified into two main approaches: the standard sink model and distributed indexing and storage. In the standard sink model, data is delivered to the sink for out-of-network processing. Queries are disseminated from the sink to sensor nodes who will then report their readings. Data pruning and aggregation can be undertaken when data propagates up the tree to the sink (e.g., in TinyDB) [105]. The sink model assumes little or no in-network processing and most of the intelligence stays outside the network.

The second approach uses in-network storage, builds distributed indices and stores partial aggregates to facilitate user queries. Examples of this category include DIMENSIONS [52–54], DIFS [62], DIM [99], and fractional cascading [56]. As storage devices such as flash drives become cheaper and smaller, the approach of using collective distributed storage becomes increasingly feasible. A distributed indexing structure typically involves a hierarchy to bring together data across different attribute space or spatial separations (e.g., quad-tree or *kd*-tree). Partial aggregates are computed bottom up for each node in the hierarchy. Queries take a drill-down approach and traverse the hierarchy to visit nodes holding relevant data for detailed information. Important considerations for distributed indexing and storage include how the partial aggregates are computed and who holds the aggregated data/indices. A straight-forward way is to take a hashing scheme and make certain nodes be responsible to hold aggregated data/indices on the hierarchy (e.g., in DIMENSIONS and DIFS). Special care is typically taken for nodes holding data at high levels of the tree to alleviate communication and query bottleneck [54].

The approach of fractional cascading in [56] belongs to the second category and tries to avoid the bottleneck created by higher level nodes in the hierarchy. In [56], the sensor field is recursively partitioned by a standard quad-tree. Aggregates from each quad in the tree are computed and stored at all sensor nodes in the quad.



Each node has the values of itself and aggregates of all the quads in which it resides. This improves data survivability and query efficiency as important information (e.g., the aggregates of larger regions) are naturally replicated more widely. Our multi-resolution representation can be considered as an alternative way to achieve fractional cascading. To see the difference of this method with [56], instead of a fixed quad-tree partitioning, we keep the data summarization hierarchy of each node adaptive and centered on the node itself. Thus any two nodes will have slightly different world views at each scale, as their multi-resolution ranges differ, while two leaf nodes in a fixed quad-tree may share the same data of many high-level quads. Another novelty of this chapter is to investigate gossip-based algorithm to disseminate information and construct the multi-resolution data representation. A survey of gossip algorithms and applications in sensor networks is covered in the next subsection.

### 2.1.3 Gossip algorithms

Gossip algorithm is attractive for sensor networks, due to its distributed nature, robustness to network dynamics, and good load balancing. In a gossip algorithm each node picks, according to some underlying deterministic or randomized rule, another node and exchanges information with it [69]. There are two important aspects in a gossip algorithm: the *gossip communication mechanism* that decides which node to communicate with; and the *gossip computation protocol* that decides what data to exchange.

In the literature two rules to select node to gossip with are prevailing. In uniform gossip, each node chooses to communicate with a randomly chosen node at each step [33]. In standard gossip on a graph, a node picks, according to a probabilistic distribution, one of its immediate neighbors in the graph [14, 157, 158]. Of particular relevance to our work is the spatial gossip algorithm proposed by Kempe, Kleinberg and Demers, where a node  $x$  selects a node  $y$  with probability proportional to  $1/d^\rho$ , where  $d$  is the distance between  $x$  and  $y$  and  $\rho$  is some constant parameter [79]. The intuition of the spatial distribution complies with the principle of fractional cascading and our multi-resolution data representation. Data from a sensor node should, intuitively, be disseminated more to its nearby neighbors and less to far away neighbors.

On top of the gossip communication mechanism, a gossip computation protocol specifies what information to be exchanged. In probably the simplest setting, information spreading [79], gossip is used to disseminate a piece of data from one node to the rest of the network. When two nodes communicate, the message is propagated. The protocol stops when all the nodes receive the message. More sophisticated information exchange protocols can be used to compute aggregations and global statistics among the gossip nodes. For the problem of distributed averaging [14], each node takes the average of the values of itself and its gossip partner. The algorithm converges when all nodes hold values close to the true average. Gossip-type protocols have also been developed in various settings to compute, in a distributed way, consensus [14, 113], various aggregates [78, 115], distributed linear

parameter estimation [157, 158], spectral analysis [80] or random linear projections of the data field for information compression and recovery [127].





# Chapter 3

## Routing in Metric Spaces Using Spatial Distributions

We propose a generic design principle for scalable routing in ad hoc wireless sensor networks. In our setting we assume an approximate distance oracle that estimates the graph distance (hop count distance) of any two nodes up to constant factor upper and lower bound. Such an approximate distance oracle can be constructed by using a landmark-based scheme, or obtained through the Euclidean distance of the geographic locations of the nodes. We store a small number of routing paths for selective pairs of nodes, which, when integrated with the approximate distance oracle, allows  $1 + \epsilon$  stretch routing.

In particular, we first derive a set of sufficient conditions to select the next hop of the routing path such that these conditions can be verified locally at each node and enable  $1 + \epsilon$  stretch routing on *any* metric. These conditions will serve as the ‘greedy routing’ rule. Next, to satisfy these conditions, the routing paths from each node  $u$  to  $O(\log^2 n)$  destinations are stored in the network, where the destination is selected with probability proportional to  $1/r^\alpha$ , with  $r$  as the distance to  $u$  and  $\alpha$  as an appropriate constant. For metrics of bounded growth, the routing algorithm conforming to the set of sufficient conditions guarantees with high probability  $1 + \epsilon$  stretch routing with routing table size  $O(\sqrt{n} \log^2 n)$  on average for each node. This scheme is favorable for its simplicity, generality and blindness to any global state. Global routing properties emerge from purely distributed and uncoordinated routing table design.

### 3.1 Introduction

Scalable routing is one of the most challenging problems in distributed network design — considerations include compact storage with aggressive address aggregation, efficient propagation of topology update, and most importantly, distributed and uncoordinated decisions to enable globally close to optimal routing properties.

Internet routing achieves scalability through subnetwork partitioning hierarchy and address aggregation, with one routing table entry representing routing infor-

mation to many IP addresses in a subnetwork, and extremely efficient high-end switches to quickly classify and deliver packets. For resource constrained wireless nodes used in ad hoc and sensor networks, scalable routing requires even more aggressive methods to produce compact routing information, and innovative ways to exploit the special properties of such networks.

Large-scale wireless sensor networks have strong spatial properties — they are closely related with the underlying geometric domain in which they are embedded, in terms of node distribution and the strong correlation of graph connectivity and node proximity. Various properties of the geographical embedding of the nodes have been exploited for compact routing in a sensor network — mostly in an explicit manner, as the geographical locations used in geographical routing families [13,77,92], or as in many virtual coordinate system design [16,43] that abstracts the global geometric/topological properties of the embedding.

In this chapter we use the metric properties of a wireless network graph for routing implicitly, and store selective routing paths in the network, such that the average routing table size is small, the path stretch is close to optimal (the length of the path is  $1 + \epsilon$  times the shortest path for a given  $\epsilon > 0$ ), and both the preprocessing and the routing can be achieved by the nodes making decisions on their own, blind to any global state.

We investigate the following problem:

*Given an approximate distance oracle  $\mathcal{O}$ , how to design compact routing tables to help deliver messages in a large network.*

First we remark that if we are given an *accurate* distance oracle that returns the hop distance of any two nodes in the network, then greedily selecting the next hop with smallest distance to the destination will guarantee delivery along the shortest path. Of course, the construction, maintenance and compact representation of an accurate distance oracle is not easy in a distributed setting. As shown in [148], accurate distance oracle would require about  $\Omega(n)$  storage per node in a network of  $n$  nodes. An approximate distance oracle is easier to obtain. In many cases, some approximate distance estimation is readily available.

For an example, in the sensor network setting, one can use the Euclidean distance to approximate the hop count distance of two nodes in the network. Obviously it is in most cases not an accurate distance oracle, and a message can get stuck at a local minimum if the neighbor on the shortest path to the destination estimates its distance to be larger than the distance estimation between source and destination [13,77]. Therefore we will have to augment the approximate distance oracle with additional routing information to help packets get out of or avoid the local minimum.

With any approximate distance oracle, the solution we propose for routing is to store the routing paths between some pairs of nodes that are not immediate neighbors, called *long links*. In particular, for some selected pair  $u, v$  a path between  $u, v$ ,  $P(u, v)$ , is recorded in the routing table of all nodes on this path. When a node  $p$  wants to send a message to a node  $q$ , it uses its immediate neighbors, together with the nodes with which  $p$  has long links. We define a *forwarding region* (see Fig. 3.1),

from which  $p$  selects the next hop in the path. If the selected node  $x$  is a neighbor through a long link, then the routing information stored on the path  $P(p, x)$  is used to deliver the message to  $x$ . Node  $x$  then repeats an identical procedure to advance the message. Now the question is, what long links should each node build and what is the forwarding region, without any global knowledge of the network, such that the routing table size is small, the path stretch is low, and delivery rate is high?

Our main theoretical results apply to arbitrary metrics of bounded growth. As an illustration, we describe the special case with Euclidean distance as an approximate distance oracle. That is,  $\delta_1|pq| \leq \sigma(p, q) \leq \delta_2|pq|$ , with  $\delta_1 \leq \delta_2$  as two constants,  $|pq|$  as the Euclidean distance between  $p, q$  and  $\sigma(p, q)$  as the minimum hop count between  $p, q$ . This assumption makes no unit disk graph requirement on the wireless radio communication model and uses two relaxation constants  $\delta_1$  and  $\delta_2$  incorporating both local distance disturbances due to wireless communication and global irregularities such as fat network holes<sup>1</sup>. It also allows localization errors as accurate location discovery is difficult. The routing tables are built by each node selecting its long links randomly with a spatial distribution. In particular, a node  $p$  would select a long link partner  $q$  with probability proportional to  $1/|pq|^2$ . The number of long links for each node is  $O(\log^2 n)$  with the constant depending on the stretch requirement  $1 + \varepsilon$  and the distance oracle error factors  $\delta_1, \delta_2$ . The routing algorithm using the augmented long links is able to deliver the message along a path of stretch  $1 + \varepsilon$ .

In fact, the theoretical results in this chapter address a general setting in which an approximate distance oracle is given for a metric space with bounded growth rate. A graph has bounded growth rate  $\rho$  if the number of nodes within  $r$  hops from any node  $p$  in the network is bounded by  $c_1 r^\rho$  and  $c_2 r^\rho$  from below and above respectively, with two constants  $c_1 \leq c_2$ . This model has been used to capture any physical constraints that disallow too many nodes ‘packed’ within certain distance and the graph has a bounded polynomial growth pattern instead of an exponential growth pattern (e.g., a balanced binary tree). This kind of geometric growth has been observed in many different scenarios such as VLSI design, the delay metric on the Internet overlay networks, and in our setting, wireless sensor networks. When sensor nodes are roughly uniformly deployed in a geometric region with bounded density per unit area<sup>2</sup> and when the network is not too much fragmented by deployment holes, the graph growth rate is typically 2. It is this packing property that allows us to aggressively compress the routing table entries by a simple routing table neighbor selection rule dominated by a spatial distribution.

For a metric with growth rate  $\rho$ , the long link  $p, q$  is selected by  $p$  with probability proportional to  $1/r^\rho$ , for  $r = \sigma(p, q)$  being the network hop distance between  $p, q$ . With  $O(\log^2 n)$  long links per node and on average routing table size of  $O(n^{1/\rho} \log^2 n)$  per node, the  $1 + \varepsilon$  stretch routing can be achieved with the help of the approximate distance oracle and the long links. Thus this principle of using

---

<sup>1</sup>We define a hole to be *fat* if any two nodes on the boundary of a hole has its hop count distance to be at most a constant factor of the Euclidean distance.

<sup>2</sup>If the density in a region becomes too high, it is easy to cluster neighboring nodes and operate on clusterheads so that the density of clusterheads is bounded by a constant.

spatial distribution in routing table design can be applied to applications in which a decentralized search is desired with only some approximate proximity information. For example, in an ad hoc network setting when location information is not available, one can use other distance estimation (e.g., by landmark scheme [43,87]). In the design of overlay networks on the Internet, one can estimate the distance between two peers by the round-trip delay estimation. For all these scenarios the results in this chapter show a way to achieve distributed routing along approximate shortest paths with a modest sized routing table on each node.

We also report simulation evaluations of this approach in a sensor network setting, to complement the theoretical analysis. For a connectivity network in which geographical greedy routing only achieves a delivery rate of 50% or so, with about 7 long links per node, we are able to achieve a delivery rate of 99% or higher. The routing table construction can be implemented in a completely distributed manner. Each node simply chooses its respective long links by sampling geographical locations under the spatial distribution, rounded to the nodes closest to the sampled locations, as in [133]. The routing table information for these long links is constructed in a bootstrapping manner, with the routes for nearby pairs constructed first and the routes for far away pairs constructed by using the routing tables already constructed so far, in the same manner as regular routing requests.

We have a second implementation by using landmark-based routing to show the power of the spatial distribution in routing table design. In particular, we select  $O(\log^2 n)$  landmarks that flood the entire network and each node records the landmark distance vector. The approximate distance oracle is implemented by the centered virtual distance as proposed in [43], which only requires the landmark distance vector of two nodes. We select on the paths to the landmarks long link neighbors to help improve the delivery rate. This implementation will involve some preprocessing of flooding the network from the landmarks but the routing paths of the long links are implicitly implied by the landmark distances. Thus the routing table size is improved to  $O(\log^4 n)$ , compared with  $O(n^{1/\rho} \log^2 n)$  when the routes have to be explicitly stored on the nodes of the paths.

In summary, the augmentation of long links with spatial distribution to get  $1 + \epsilon$  stretch routing on an approximate distance oracle is favorable for its simplicity, generality and ‘blindness’ to any global state. Global routing properties emerge from purely distributed and uncoordinated routing table design.

## 3.2 Small stretch routing with approximate distances

In this section we describe the idea of routing with  $1 + \epsilon$  stretch in a suitable metric space  $\mathcal{M}$ . We use  $d(p, q)$  to represent the estimate of distance between  $p$  and  $q$  supplied by the approximate oracle  $\mathcal{O}$ , and  $\sigma(p, q)$  to denote the true but possibly unknown graph distance (hop count distance) in  $\mathcal{M}$ . We assume that a node is able to get the approximate distance  $d(p, q)$  from just the names of  $p, q$ . The implementation of this distance oracle is elaborated in a later section. Here we show that when the long links are carefully chosen the routing stretch is low.

**Accurate distance oracle.** To demonstrate the basic concept, consider the case in which the oracle is in fact accurate, that is,  $d = \sigma$ . The objective is to recursively build a route from  $s$  to  $t$  with the help of the long links. Suppose  $s$  takes a long link to node  $p$ , then we want  $\sigma(s, p) + \sigma(p, t)$  to be not very large compared to  $\sigma(s, t)$ :

$$\sigma(s, p) + \sigma(p, t) \leq \gamma \cdot \sigma(s, t), \quad (3.1)$$

Where  $\gamma \geq 1$  is a parameter depending on  $\varepsilon$ . Observe that inequality (3.1) defines an ellipse in  $\mathbb{R}^2$  with  $s$  and  $t$  at foci. Now we impose an additional restriction that moving from  $s$  to  $p$  implies a certain progress in direction of  $t$ . In particular,  $p$  is closer to  $t$  by a factor of at least  $0 \leq \beta \leq 1$ :

$$\sigma(p, t) \leq \beta \cdot \sigma(s, t). \quad (3.2)$$

This describes a disk centered at  $t$ .

Next, we select  $\gamma$  and  $\beta$  such that the selection procedure enforced by inequalities (3.1) and (3.2) when applied recursively, produces a path of stretch at most  $1 + \varepsilon$ :

$$R(s, t) \leq (1 + \varepsilon) \cdot \sigma(s, t), \quad (3.3)$$

where  $R$  gives the length of the path created recursively.

A *forwarding region*  $F_\varepsilon(s, t)$  is a set of points  $p$  in  $\mathcal{M}$  from which  $s$  can select  $p$  satisfying the above relations. The following lemma gives a detailed description:

**Lemma 3.2.1.** *Values of  $\gamma$  and  $\beta$  satisfying  $\gamma + \varepsilon\beta \leq 1 + \varepsilon$  constitute the forwarding region, with the equality corresponding to the region boundary.*

**Proof:** Observe that we have:

$$\begin{aligned} R(s, t) &\leq \sigma(s, p) + R(p, t) \\ &\leq \sigma(s, p) + (1 + \varepsilon) \cdot \sigma(p, t) \\ &\leq \sigma(s, p) + \sigma(p, t) + \varepsilon\sigma(p, t) \\ &\leq \gamma\sigma(s, t) + \varepsilon\beta\sigma(s, t), \end{aligned}$$

When  $\gamma + \varepsilon\beta \leq 1 + \varepsilon$ , the right hand side is no greater than  $(1 + \varepsilon) \cdot \sigma(s, t)$ .  $\square$

It is easy to see that  $\gamma$  must lie in the interval  $[1, \frac{2+3\varepsilon}{2+\varepsilon}]$  for a given  $\varepsilon$ . For each value of  $\gamma$ , we have a region  $H_{\gamma, \varepsilon}(s, t) \subseteq \mathcal{M}$  which is the intersection of the ellipse bounded region and the disk. Thus, formally, the forwarding region is the union:  $F_\varepsilon(s, t) = \cup_\gamma H_{\gamma, \varepsilon}(s, t)$ . See Figure 3.1.

**Approximate distance oracle.** Now we look at the case in which the oracle supplies an approximate measure of the distance, with  $\delta_1$  and  $\delta_2$  as the lower and upper bounds:  $\forall p, q \in \mathcal{M}, \delta_1 d(p, q) \leq \sigma(p, q) \leq \delta_2 d(p, q)$ . Then, allowing for approximation error, it would be sufficient to guarantee the following inequalities (corresponding to relations (3.1)-(3.2) respectively):

$$\begin{aligned} \delta_2 d(s, p) + \delta_2 d(p, t) &\leq \gamma \delta_1 d(s, t) \\ \delta_2 d(p, t) &\leq \beta \delta_1 d(s, t) \end{aligned} \quad (3.4)$$

It can be verified that a sufficient relation between  $\gamma, \beta$  and  $\varepsilon$  is again given by the same inequality as lemma 3.2.1. And we can obtain again that  $R(s, t) \leq (1 + \varepsilon)\sigma(s, t)$ .

The idea of forwarding region is very general and hold in any metric space. This implies that for any metric space, including all network graphs, a routing scheme based on this concept guarantees a low stretch. What is necessary is that a node should have a long link to some node  $p$  in the forwarding region for the current destination.

**Routing Mechanism.** The analysis above suggests a natural routing scheme. Each node selects long links such that it has either an immediate neighbor or a long link to the forwarding region of any destination, and keeps corresponding routing table entries. The routes to the long link neighbors are stored on the routing table of the nodes on the path. When a node  $s$  has a message to be delivered to a destination  $t$ ,  $s$  will check its routing table to find a node  $p$  (either  $s$ 's 1-hop neighbor,  $s$ 's long link neighbor, or an endpoint of a long link whose route goes through  $s$ ), such that  $p$  lies in the forwarding region  $F_\varepsilon(s, t)$ . Node  $p$  on receiving the message will execute an identical procedure to forward the message into  $F_\varepsilon(p, t)$  and so on. Efficient randomized construction of the routing table is shown in next section.

### 3.2.1 $(1 + \varepsilon)$ -stretch forwarding region

**Geometric setting.** We first discuss the case of the Euclidean plane  $\mathbb{R}^2$ , which provides sound intuition about the geometry of the method. W.l.o.g. the coordinates of  $s$  and  $t$ , separated by a distance  $r$ , are  $(-r/2, 0)$  and  $(r/2, 0)$  respectively. We examine the forwarding region to select the long link neighbor  $p$  to realize a  $1 + \varepsilon$  stretch path to  $t$ .

With an accurate distance oracle, the relation (3.1) defines in  $\mathbb{R}^2$  a region whose boundary is given by an ellipse:

$$\frac{4x^2}{\gamma^2 r^2} + \frac{4y^2}{r^2(\gamma^2 - 1)} = 1.$$

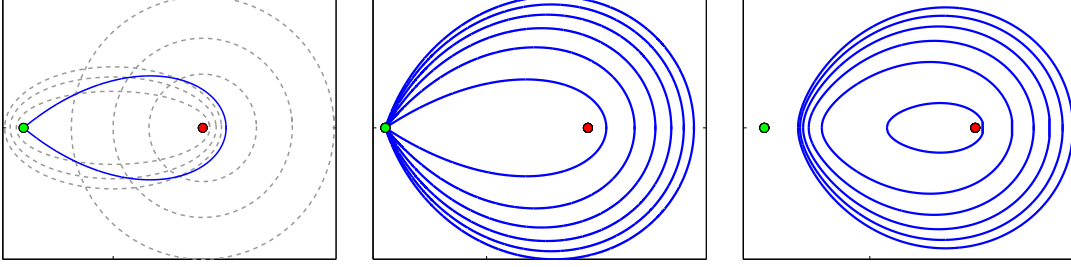
And (3.2) defines a disk whose boundary is given by a circle:

$$\left(x - \frac{r}{2}\right)^2 + y^2 = \frac{(1 + \varepsilon - \gamma)^2}{\varepsilon^2} r^2.$$

As gamma is varied, the locus of intersection of these two curves traces out the boundary of the forwarding region  $F_\varepsilon(s, t)$  (see Fig. 3.1 (i)).

For any point  $q$  on the boundary of  $F_\varepsilon(s, t)$ , the angles  $\angle qst$  and  $\angle qts$  are functions of  $\gamma$  and  $\varepsilon$  only, and are independent of  $r$ . This implies that the shape of the forwarding region is scale invariant, i.e., it does not depend on the distance between source and destination. Figure 3.1 (ii) shows the shapes of forwarding regions for different values of  $\varepsilon$ . Smaller values of  $\varepsilon$  create smaller and narrower forwarding regions.





**Figure 3.1.** (i) Boundary of  $F_\varepsilon$  as intersection of ellipses and circles. (ii) Forwarding regions for different values of  $\varepsilon$  from 0.2 to 2. (iii) Forwarding regions for different values of  $\varepsilon$  from 0.2 to 2 for approximate oracle.

With an approximate distance oracle, the corresponding ellipse and circle equations are given by:

$$\frac{\delta_2^2}{\delta_1^2} \cdot \frac{4x^2}{\gamma^2 r^2} + \frac{4y^2}{r^2 \left( \frac{\delta_1^2}{\delta_2^2} \gamma^2 - 1 \right)} = 1$$

$$\left( x - \delta_2 \frac{r}{2} \right)^2 + y^2 = \left( \frac{\delta_1}{\delta_2} \cdot \frac{1 + \varepsilon - \gamma}{\varepsilon} \cdot r \right)^2$$

The corresponding forwarding regions are shown in Fig. 3.1 (iii). Observe that in this case the forwarding regions are smaller and source  $s$  is not in the forwarding region. This is due to inaccurate distance estimates and necessitates the use of *long links* - without which  $s$  cannot access the forwarding region.

**The graph setting.** The geometric intuition needs to be realized in an ad hoc sensor network setting. In this section, we use the concept of a graph and a continuous metric space interchangeably for ease of description, but the results hold for any metric space that fits the model. A graph metric refers to the shortest path metric.

In general, we consider a graph such that the number of nodes at a distance exactly  $r$  from  $p$ , represented by  $|\partial N_r(p)|$  is bounded by  $|\partial N_r(p)| = \Theta(\rho r^{\rho-1})$ . This is equivalent to  $|N_r(p)| = \Theta(r^\rho)$ . Note that the diameter  $D$  of such a graph is bounded by  $\Theta(n^{1/\rho})$ . We have the following quick observation.

**Lemma 3.2.2.** *Given an unweighted graph  $G$  with  $|N_r(p)| = \Theta(r^\rho)$ , the graph has a doubling dimension  $\eta = O(\rho)$ .*

**Proof:** Consider a ball  $N_{2r}(p)$ , we use a greedy algorithm to select balls of radius  $r$  to cover it. In particular, we select a node  $q$  in  $N_{2r}(p)$  that is not yet covered, and cover all nodes in  $N_r(q)$ . Iterate until all nodes are covered. Now we bound how many balls are selected (denote this set as  $Q$ ). To see that, we take the selected nodes  $q \in Q$  and the balls  $N_{r/2}(q)$ . First they do not overlap as any two nodes in  $Q$  are of distance at least  $r$  away. Thus by a volume argument we have  $|Q| \leq |N_{2r}(p)| / \min(|N_{r/2}(q)|) = O\left(\frac{(2r)^\rho}{(r/2)^\rho}\right) = O(4^\rho)$ .  $\square$



**Lemma 3.2.3.** *In a metric space with doubling dimension  $\eta$ , a ball of radius  $R$  can be covered with  $O(c^\eta)$  balls of radius  $R/c$ .*

**Proof:** A ball of radius  $R$  can be covered with  $2^\eta$  balls of radius  $R/2$ . We recursively cover each such ball with balls of half the radius, until the size of balls used falls below  $R/c$ . The resultant number of balls is  $2^{\eta k}$ , where  $k = \lceil \log c \rceil$ . This is equivalent to  $O(c^\eta)$ .  $\square$

We now show the presence of a sizeable forwarding region for such a graph, when one routes from  $s$  to  $t$ :

**Lemma 3.2.4.** *There is a ball of radius  $\frac{\delta_1}{\delta_2} \left( \frac{\gamma-1}{2} \right) r$  that lies inside  $F_\epsilon(s, t)$ .*

**Proof:** Consider a point  $q$  on the shortest path between  $s$  and  $t$  separated by  $d(s, t) = r$ . Now, we take a ball of radius  $h = \frac{\delta_1}{\delta_2} \left( \frac{\gamma-1}{2} \right) r$  centered at  $q$ . One can verify that all the points inside the ball  $N_h(q)$  are inside  $F_\epsilon(s, t)$ , as they satisfy the inequalities (3.4). In particular, for any point  $p \in N_h(q)$ ,  $d(s, p) \leq d(s, q) + h$ ,  $d(p, t) \leq d(q, t) + h$ . Now we can verify that  $\delta_2(d(s, p) + d(p, t)) \leq \delta_2(r + 2h) \leq \delta_1 \gamma r$ . Also  $\delta_2 d(p, t) \leq \delta_2(d(q, t) + h) \leq \delta_1 \beta r \leq \delta_1 \left( \frac{1+\epsilon-\gamma}{\epsilon} \right) r$ .

This ball is inside a neighborhood of  $\delta_2 r - \frac{\delta_1}{\delta_2} \left( \frac{1+\epsilon-\gamma}{\epsilon} - (\gamma - 1) \right) r$  from  $s$ . The number of nodes inside this ball is at least  $\Omega \left( \left( \frac{\delta_1}{\delta_2} \left( \frac{\gamma-1}{2} \right) r \right)^\rho \right)$ .  $\square$

This lower bound on the size of forwarding region suggests that among long links chosen randomly according to a spatial distribution, at least one is likely to lie in the forwarding region with high probability. The next subsection shows that this is indeed the case.

### 3.3 Routing table construction by spatial distribution

To build the routing table, we use a spatial distribution of directed links. In particular, for nodes  $p$  and  $q$  separated by a distance  $r$ , the probability of a directed link  $pq$  being built is proportional to  $1/r^\rho$ . The rest of this section analyzes random selection of long links to make sure there is a long link in the forwarding region for every possible destination. Combined with the recursive routing in the beginning of this section, the existence of such links guarantee  $1 + \epsilon$  stretch routing.

The analysis below uses essentially balls and bins probabilistic analysis. When a long link is picked randomly with the spatial distribution, we have the following lemma.

**Lemma 3.3.1.** *For any  $\mu > 1$ , a link from  $p$  lies in the annulus  $N_r(p) - N_{r/\mu}(p)$  with probability  $\Theta \left( \frac{\ln \mu}{\ln n} \right)$ .*

**Proof:** Suppose  $C$  is the normalizing factor of the probability distribution for the given network. This means:  $C \int_1^D \frac{1}{r^\rho} |\partial N_r(p)| dr = 1$ . Integrating,  $C = \Theta\left(\frac{1}{\rho \ln n}\right)$ .

The probability that a given link lies in an annulus  $N_r(p) - N_{r/\mu}(p)$  is given by

$$\Pr(r/\mu, r) = C \int_{r/\mu}^r \frac{1}{\zeta^\rho} |\partial N_\zeta(p)| d\zeta = \Theta\left(\frac{\ln \mu}{\ln n}\right).$$

Note that this probability is independent of  $r$ . □

**Theorem 3.3.2.** *From each node it is sufficient to select  $k = O\left(\left(\frac{2}{\varepsilon}\right)^{O(\rho)} \ln^2 n\right)$  links, to guarantee a link in the forwarding region for any destination with probability at least  $1 - 1/n^2$ .*

**Proof:** Consider the forwarding region  $F_\varepsilon(s, t)$ , with  $d(s, t) = \ell$ . We choose a valid value  $\gamma$ . By lemma 3.2.4, there is a ball  $B_h$  of radius  $h' = \frac{\delta_1 \gamma - 1}{\delta_2} \ell$  within a distance of  $r = \delta_2 \ell - \frac{\delta_1}{\delta_2} \left(\frac{1+\varepsilon-\gamma}{\varepsilon} - (\gamma - 1)\right) \ell$  from  $s$ .

Choose  $\mu'$  such that  $B_{h'}$  lies in the annulus  $N_r(s) - N_{r/\mu'}(s)$ . This implies that  $\mu' = \frac{r}{r-2h'}$ . Substituting, and simplifying, we have  $\mu' = \Omega(1 + \varepsilon)$ . To show that a link lies in  $B_{h'}$ , it is sufficient to show that it lies in a smaller ball  $B_h \subseteq B_{h'}$ , which is defined below. If  $h \geq r/4$  we assign  $B_h = B_{r/4}$ , and  $\mu = 2$ , where  $B_{r/4} \subseteq B_{h'}$ , and  $B_{r/4} \subseteq N_r(s) - N_{r/2}(s)$ . If  $h < r/4$ , we assign:  $B_h = B_{h'}$  and  $\mu = \mu'$ . Thus, the width of the annulus  $N_r(s) - N_{r/\mu}(s)$  is at most  $r/2$ , and  $\mu \leq 2$ .

Now we show that with  $k = O\left(\left(\frac{2}{\varepsilon}\right)^{O(\rho)} \ln^2 n\right)$  links, there is a link to  $B_h$  (and hence to  $B_{h'}$ ) with high probability. The basic idea is the following. The annulus  $N_r(s) - N_{r/\mu}(s)$  can be covered by a small number of balls, by the constant doubling dimension property. Thus with randomly selected links, at least one will fall inside  $B_h$ .

By Lemma 3.2.3, the ball  $N_r(s)$  can be covered by at most  $A = a \left(\frac{2\mu}{\mu-1}\right)^\eta$  balls of radius  $h$  for some constant  $a$ . Restricting attention only to links from  $s$  to inside  $N_r(s) - N_{r/\mu}(s)$ , consider a covering of the annulus with balls of radius  $h$ . The ball  $B_h$  belongs to this set, and each node in  $B_h$  is selected by  $s$  with probability at least  $C \frac{1}{r^\rho}$ , where  $C = \Theta(1/(\rho \ln n))$  is the normalizing factor. Similarly, every node in the other  $A - 1$  balls is selected with a probability at most  $C \frac{\mu^\rho}{r^\rho}$ .

Thus, given that a link is in the annulus  $N_r(s) - N_{r/\mu}(s)$  the probability that it is in  $B_h$  is:

$$\Pr(B_h | (N_r(s) - N_{r/\mu}(s))) \geq \frac{(\mu - 1)^\eta}{a(2\mu)^\eta \mu^\rho + (\mu - 1)^\eta}.$$

Combining with the result of lemma 3.3.1 of the link being in the annulus, we get that the probability of a random link to  $B_h$  is  $\Pr(B_h) \geq \left(\frac{1}{K \ln n}\right)$ , where  $K = O\left(\left(\frac{2}{\varepsilon}\right)^{O(\rho)}\right)$ .

If  $2K \ln^2 n$  links are chosen from  $s$ , then the probability that none of them lie in  $B_h$  is  $\left(1 - \frac{1}{K \ln n}\right)^{(K \ln n) 2 \ln n}$ . Therefore, the probability that at least one link lies in  $B_h$  is  $(1 - 1/n^2)$ . Therefore,  $O\left(\left(\frac{2}{\varepsilon}\right)^{O(\rho)} \ln^2 n\right)$  links per node suffice to obtain the given probability.  $\square$

The theorem above describes a guarantee for a suitable link to a forwarding region to exist. In fact, the detailed proof says that a link exists to a ball  $B_{h'}$  of a radius  $h'$  inside the forwarding region. However, we still need to prove the existence of a path of  $(1 + \varepsilon)$  stretch for a given routing request, that will take us to within a small constant distance of the destination. This is done by showing the existence of a short sequence of forwarding links. First we show, that if the path exists, it only involves a few long links.

**Lemma 3.3.3.** *If a path obtained by appending the long links in the balls  $B_{h'}$  exists then it consists of  $O(\log n)$  long links and has a stretch of  $(1 + \varepsilon)$ .*

**Proof:** As in the proof of theorem 3.3.2, there is a ball  $B_{h'}$  of radius  $h' = \frac{\delta_1 \gamma' - 1}{\delta_2} l$  which by lemma 3.2.4 lies within a distance  $\frac{\delta_1}{\delta_2} \frac{1 + \varepsilon - \gamma'}{\varepsilon} l = \frac{\delta_1}{\delta_2} \beta' l$  of  $t$ .

Thus, by selecting the long link to the ball  $B_{h'}$ , we take the message to be within a constant fraction  $\beta'$  of the remaining distance to the destination at every step. Since the diameter of the network is  $n^{1/\rho}$ , this recursive forwarding will reach a constant neighborhood of  $t$  using  $O(\log n)$  hops. Given that  $B_{h'}$  is selected to be inside the forwarding region for each step, this path will have a stretch  $1 + \varepsilon$ .  $\square$

Now we combine the number of links with the probability of each link to get the final result:

**Theorem 3.3.4.** *It is sufficient to select  $O\left(\left(\frac{2}{\varepsilon}\right)^{O(\rho)} \ln^2 n\right)$  long links per node to guarantee a path of stretch at most  $1 + \varepsilon$  with probability at least  $1 - 1/n$ .*

**Proof:** Observe that by lemma 3.3.3 the path consists of  $O(\log n)$  long links, each of which exists with probability at least  $1 - 1/n^2$ , by theorem 3.3.2. Combining the two, we get that the path exists with probability  $\left(1 - 1/n^2\right)^{O(\log n)}$ , which is at least  $1 - 1/n$ .  $\square$

And the routing table size is not too large.

**Theorem 3.3.5.** *The average routing table size of the scheme is bounded by  $O\left(\left(\frac{2}{\varepsilon}\right)^{O(\rho)} n^{1/\rho} \ln^2 n\right)$ .*

**Proof:** The length of a long link is at most the diameter of the network, which is  $O(n^{1/\rho})$ . Thus a link can contribute at most  $O(n^{1/\rho})$  number of routing tables entries. By theorem 3.3.2, each node of  $n$  nodes can add  $O\left(\left(\frac{2}{\varepsilon}\right)^{O(\rho)} \ln^2 n\right)$  such links to the network. Thus, the average number of entries, when divided among  $n$  nodes, is  $O\left(\left(\frac{2}{\varepsilon}\right)^{O(\rho)} n^{1/\rho} \ln^2 n\right)$ .  $\square$

In the case of sensor networks in a plane ( $\rho \approx 2$ ), for a given stretch  $\varepsilon$ , this amounts to a table size of  $O(\sqrt{n} \ln^2 n)$  per node. In the next section we describe an implementation that implicitly stores the long links with substantially smaller routing table sizes of  $O(\ln^4 n)$ .

### 3.4 Implementation in sensor networks

Here we describe the implementation of the routing table design in a distributed setting. In particular, how to implement the approximate distance oracle, how to choose the long links with the spatial distribution and how to build routes representing the long links. We give two different approaches to implement the distributed routing table, one with the geographical locations, one with landmarks and landmark-based distances.

Note that the implementation of approximate distance oracle is really independent of our routing table design and the implementations can be entirely decoupled. Any method that provides reasonably good distance estimate can be used as a distance oracle.

#### 3.4.1 Geographic routing table design

In this part we describe using the spatial distribution principle to augment standard geographical forwarding with additional routing information to increase the delivery rate.

**Approximate distance oracles.** As mentioned in the introduction, the geographical locations often serve as a good approximate distance oracle to the minimum hop count distance metric on the communication network. To formulate this notion rigorously, we assume that the sensor field is deployed in an environment with *fat* (not necessarily convex) obstacles. That is, for any two points  $p, q$  on the boundary of a hole, the geodesic distance<sup>3</sup>  $g(p, q)$  is at most  $\tau$  times the Euclidean distance  $d(p, q)$  for a constant  $\tau > 1$ , as shown in Figure 3.2. Given this, we can show that for

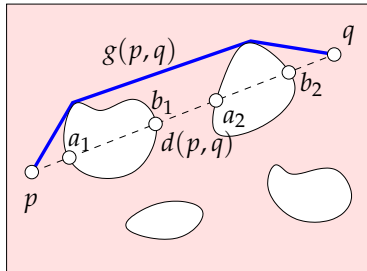


Figure 3.2. The geodesic distance  $g(p, q)$  is at most  $\tau \cdot d(p, q)$  with fat holes.

<sup>3</sup>The geodesic distance between two points in a geometric domain is defined as the Euclidean length of the shortest path connecting the two points in the domain, avoiding obstacles.

any two points  $p, q$  in the underlying geometric domain, we have  $g(p, q) \leq \tau d(p, q)$ . In addition, we assume that the sensor nodes are deployed in the environment approximately uniformly such that the minimum hop count distance is at most  $\tau'$  the geodesic distance. Thus we have  $d(p, q) \leq \sigma(p, q) \leq \delta \cdot d(p, q)$ , for a constant  $\delta = \tau \cdot \tau' > 1$ .

**Geographic spatial sampling.** We include the routing paths between pairs of nodes chosen with a spatial distribution. With geographical locations, we will implement the spatial sampling of a partner  $q$  of  $p$  by choosing with probability proportional to  $1/r^2$  a *geographical location*  $q^*$  and round it to the nearest node  $q$ . That is, the node  $q$  whose Voronoi cell contains the sampled location  $q^*$  is taken as the long link partner of  $p$ . If the nodes are not uniformly distributed, the Voronoi cells have different areas and the nodes are selected with a biased probability. Thus we use von Neumann's rejection sampling to 'smooth out' the non-uniformity introduced by the variation of Voronoi cell area. This idea is originally proposed and used in taking a uniform random sampling of sensor nodes [11, 33] and later adapted to get a similar spatial sampling [133].

**Incremental routing table construction.** The last implementation problem is to discover and store the routes of the long links selected by the spatial distribution for each node. Notice that here we have a seemingly chicken-and-egg problem, as route discovery requires a routing algorithm, while the routing table construction is to supply such a routing scheme. We actually find the routes with bootstrapping and incrementally construct the routes for the long links with increasing lengths. Specifically, every node first selects their long link partners (in fact, the geographical locations). The routes for the pairs with shorter distances are constructed first, and the routes for the pairs with length  $k$  are discovered with the current routing table information, that is, with the help of the long links with lengths smaller than  $k$ .

The route for a long link  $pq$  is stored on the routing table of the nodes on this path. Specifically, each routing table entry is a tuple  $(p, q, N_q)$ , where  $N_q$  is the next hop neighbor leading to  $q$ . Thus a node maintains the routes to its long link partners as well as the routes that pass through it.

The simplicity of this scheme also suggests an 'on-demand' implementation to improve the basic routing. That is, when a packet is stuck at a local minimum we will select long links according to the spatial distribution. Thus routing delivery rate might be low or the delay can be long initially but as the routes for the long links are constructed and recorded the network gradually 'learns' and 'repairs' the imperfect distance oracle.

### 3.4.2 Landmark-based routing table design

When the location information is not available or when the sensor field is deployed in an environment so that the Euclidean distance does not provide a good approximate distance oracle, we propose a second scheme with landmark-based distances.

Specifically, we select  $m = O(\log^2 n)$  landmarks  $\ell_i$  uniformly randomly in the sensor network. For example, each node proposes to be a landmark with probability  $\log^2 n/n$ . The landmarks then flood the network and every other node records the hop count distance to these landmarks. The communication cost for the preprocessing is  $O(n \log^2 n)$ .

**Landmark-based distance oracles.** Each node  $p$  is given a landmark-based distance vector, represented by the vector of minimum hop count distance to all  $m$  landmarks,  $(\sigma(p, \ell_1), \sigma(p, \ell_2), \dots, \sigma(p, \ell_m))$ . We would like to use the landmark distances to estimate the hop count distance of any two nodes. In the simulations we used the centered distance measure proposed in [43], which is a  $\ell_2$  norm of the centered landmark-based distance vector  $(\sigma(p, \ell_1)^2 - M, \sigma(p, \ell_2)^2 - M, \dots, \sigma(p, \ell_m)^2 - M)$ , where  $M = \sum_i \sigma(p, \ell_i)^2 / m$ .

**Landmark-based sampling.** To build the long links for a node  $p$ , we will use the landmarks to help with sampling. In particular, we select first randomly  $k$  out of the  $m$  landmarks. For each landmark  $\ell_i$ , we select from the distribution  $1/(r \ln D)$  ( $D$  is the network diameter) a distance  $\xi$ . If  $\xi \leq \sigma(p, \ell_i)$ , we take the node  $q$  along the path from  $p$  to  $\ell_i$  with distance  $\xi$  from  $p$  as the long link partner. Otherwise we drop landmark  $\ell_i$ . Intuitively, we select along the path from  $p$  to  $\ell_i$  a node  $q$  with the spatial distribution restricted on this path. Since the landmarks are randomly selected, the probability that a landmark  $\ell_i$  is at distance  $r$  from  $p$  is proportional to  $r$ . Now the probability that for each landmark  $\ell_i$  we can obtain a valid long link is

$$\text{Prob}\{\xi < \sigma(p, \ell_i)\} = \int_0^D \int_1^\xi \frac{1}{\xi \ln D} d\xi \frac{2\xi}{D^2} d\zeta = 1 - \frac{1}{2 \ln D}.$$

Thus in expectation we obtain  $k(1 - \frac{1}{2 \ln D})$  long links for each node. This means that choosing  $m = O(\log^2 n)$  landmarks suffices to get enough long links for each node. At last we remark that although different nodes use the same set of landmarks to create their long links, the theoretical analysis in the previous section still holds – as the only requirement is that we have a sufficient number of independent long links for each individual node.

**Landmark-based routing tables.** With the long links constructed by the landmarks, the routing table size can be further reduced. In fact, a node  $p$  remembers in its routing table the long link partners and their landmark-based addresses. Different from the geographical case, the routes for the long links are implicitly implied by the landmark distances. The size of the routing table is therefore  $O(\log^4 n)$ , for  $O(\log^2 n)$  landmarks/long link neighbors, and a storage of  $O(\log^2 n)$  for storing the address of each long link neighbor.

### 3.4.3 Routing scheme implementation

We implemented our routing algorithm for simulations, using both the Euclidean distance oracle and the landmark based oracle. Each node keeps the routing table



entries for its immediate neighbors, as well as the long link neighbors it has selected. The routes to the long link neighbors are stored on the routing tables of the nodes on the path. When a node  $s$  has a message to be delivered to a destination  $t$ ,  $s$  will check its routing table to find a next hop node  $p$ . The node  $p$  was selected randomly from the set of feasible nodes in the forwarding region. Other than this stretch guaranteed strategy, we also simulated the effects of selecting a long link greedily from the routing table, where the  $p$  is the node in the routing table that is nearest to  $t$  according to the oracle. The message may not travel the entire long link if on a node in the the middle the message finds a closer neighbor to the destination.

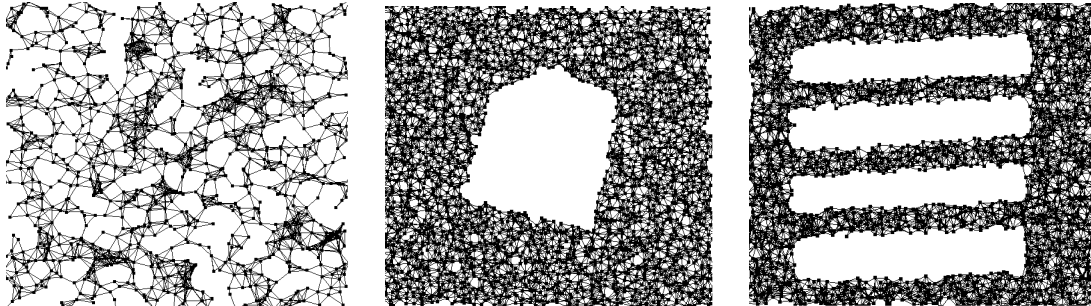
The simulations (Figure 3.4) show that the greedy heuristic performs well in practice. Both schemes achieve high delivery rate and low stretch. The greedy routing may sometimes have lower delivery rate, but has better stretch. These results are understandable in the light of the fact that the forwarding region contains the destination, and a large region in between the source and the destination. Thus, the link in routing table that reaches closest to the destination is likely to be one in the forwarding region. Which means, in many cases, this heuristic satisfies the conditions of the algorithm, and because greedy choice is more likely to be nearer the destination than a random choice, it results in a low stretch. Thus, in simulations, we consider the greedy strategy to be comparable to the theoretical strategy. This also suggests further study and analysis of the spatial distribution and routing table constructions along these lines.

## 3.5 Simulations

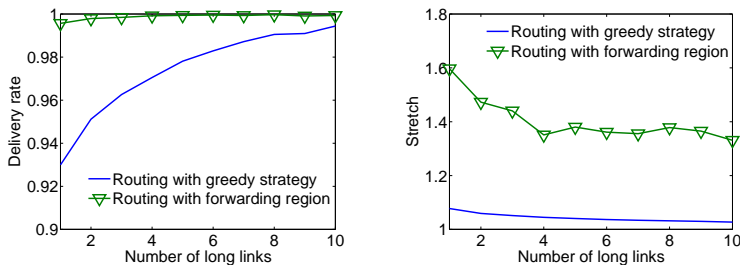
In this section, we present simulation results to show the performance of the proposed schemes in practice. We mainly focus on geographic routing table to show the tradeoff of the routing table size v.s. routing stretch. We also evaluate the performance of landmark-based scheme on a network of complex topology, for which landmark-based approximate distance oracle captures the underlying network connectivity more accurately. We compare our approach with two recently proposed routing protocols, S4 [107] and VRR [17], on three important criteria, i.e., delivery rate, the size of routing table and routing stretch. We also discuss the preprocessing cost of each scheme. In summary, our approach achieves high delivery rate (above 99%) and small stretch (about 1.03) with only a small number of long links, and a small routing table with modest preprocessing.

**Simulation setup.** We focus on evaluating the performance of all approaches at the routing layer, and assume the underlying details (i.e., packet loss and interference) have been handled at MAC and link layers. This is sufficient for our purpose of verifying the validity of the proposed ideas. Respecting reality, we adopt a lossy radio model used in the standard simulator TOSSIM [97] to determine direct communication links between nodes. The lossy radio model is generated based on empirical data and specifies the loss rate on the link between a pair of nodes. We only consider links with sufficient low loss rate and the resulted network is not necessarily unit disk graph, and could have directional links. We run simulations on three typical

topologies, i.e., a sparse network with 1000 random distributed nodes, a network with a large hole in the center and a network with multiple holes (see Figure 3.3). Each simulation run is repeated 10 times. In each round, we randomly selected 10000 pairs of source and destination. All results are averaged on all pairs.



**Figure 3.3.** Network topologies used in simulations. (i) Topology 1. Random network: 1000 nodes, avg. degree 7.2; (ii) Topology 2. Network with one hole: 2400 nodes, avg. degree 9.5; (iii) Topology 3. Network with multiple holes: 2000 nodes, avg. degree 10.6.



**Figure 3.4.** (i) Delivery rate for Topology 2. (ii) Stretch for Topology 2.

### 3.5.1 Geographic routing table

We evaluate the performance of our approach with geographic routing table, as explained in Section 3.4.1.

**Delivery rate.** To show the effect of long links on the delivery rate, we vary the number of long links each node maintains from 0 to 16. When the number of long links is set to 0, the routing protocol is essentially the geographical greedy routing based on the location information within one-hop neighborhood. Figure 3.6 (i) shows that greedy routing performs very poorly without long links. The delivery rate is only around 50%, 65% and 44% in Topology 1, 2 and 3 respectively. When the number of long links increases, the delivery rate reaches 99% with 6, 8, 7 long links per node in three different topologies, respectively. The results confirm that a small number of long links are sufficient and can significantly improve the delivery rate in most of typical network topologies. Since our scheme behaves similarly in various topologies, in the rest of this subsection, unless mentioned otherwise, we only present results on Topology 2 due to space limitation.



We show the preprocessing cost of our scheme with varying number of long links in Figure 3.6 (iv). More long links results in higher preprocessing cost and increased delivery rate.

**Routing table size.** The size of routing table is measured by the number of entries in the table. We compare the average routing table size of our scheme with VRR and S4. For VRR, each node maintains routes to a set of virtual neighbors on the ID ring. Those virtual neighbors can be viewed as “long links”. Thus, we show the change of routing table size as the number of long links changes for both our scheme and VRR in Figure 3.6(ii). It is easy to see that the size of routing table is proportional to the number of long links. But our scheme uses much smaller routing table than VRR when maintaining the same number of long links. Our scheme saves routing table size by taking long links with probability  $1/r^2$  rather than the uniform distribution used in VRR. Thus, our scheme favors relatively shorter links. Figure 3.5 shows the distribution of the lengths of the long links in terms of hop counts. In our scheme there are fewer long links, while the distribution in VRR is more uniform.

Size of routing table	Our scheme	S4	VRR
Topology 1	26.08	68.83	41.52
Topology 2	39.02	105.85	62.48
Topology 3	37.28	90.62	63.82

Table 3.1. Average size of routing table.

Table 3.1 shows the routing table size of three schemes with a set of fixed parameters. For comparisons, we use 50 landmarks for S4 and each node maintains routes to 4 virtual neighbors in VRR. We select those parameters since they give the best performance of S4 and VRR in terms of both routing table size and stretch. For our scheme, we use 6, 8, 7 long links in three topologies respectively to get above 99% delivery rate. We use the same set of parameters in other Tables. From Table 3.1, S4 requires the largest routing table, since each node needs to maintain routes to roughly  $O(\sqrt{n})$  landmarks and  $O(\sqrt{n})$  nodes within its local cluster. Our scheme has the smallest routing table size, but achieves comparable delivery rate.

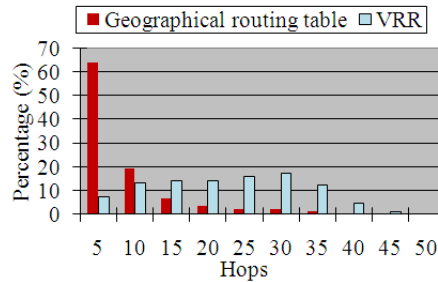
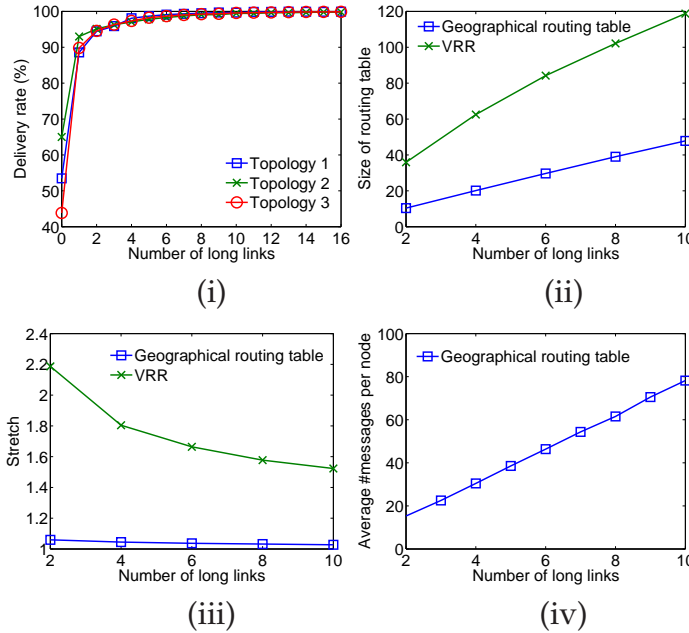


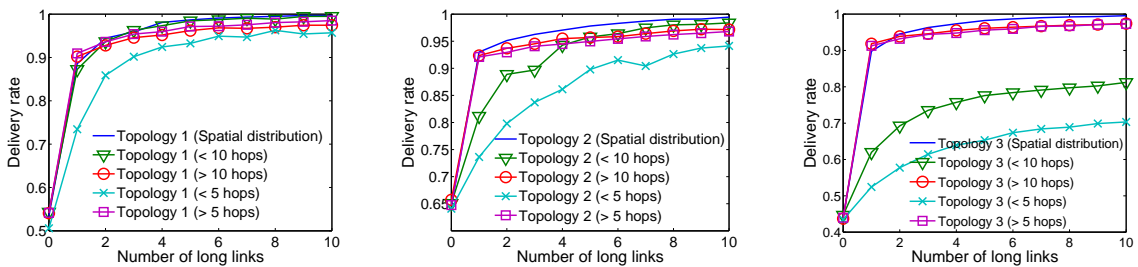
Figure 3.5. The distribution of long links w.r.t their lengths in hops.

**Stretch.** Figure 3.6(iii) shows the average stretch of our scheme and VRR with varying number of long links. The stretch of our scheme is always below 1.1 and decreases when the number of long links increases. With 6 long links, the stretch is

only about 1.03. With more long links, each node has more choices when choosing the next hop and can switch to the best direction as soon as it finds a neighbor or long link closer to the destination. Table 3.2 compares the average stretch of three schemes. It shows that our scheme achieves similar stretch as S4 (but with smaller routing table) and is much better than VRR.



**Figure 3.6.** (i) Delivery rate of geographical routing table with varying number of long links in different network topologies. (ii)-(iv) Performance of our scheme and VRR in Topology 2. (ii) The average size of routing table. (iii) Average stretch. (iv) Communication cost in preprocessing stage.



**Figure 3.7.** Delivery rate for different topologies. (i) Topology 1. (ii) Topology 2. (iii) Topology 3.

**Diversity of inaccuracy.** The inaccuracy of distance oracle is due to diverse disturbances of the network, like low density of node distribution or holes and obstacles. Here, we study the impact of different types of links (relatively short links and long links) on different types of network topologies. We compare spatial-distribution link selection scheme with other four schemes, i.e., schemes that only select nodes

Average stretch	Our scheme	S4	VRR
Topology 1	1.03	1.03	1.73
Topology 2	1.03	1.03	1.80
Topology 3	1.04	1.02	1.75

**Table 3.2.** Average stretch.

within 5 hops ( $< 5$ ), within 10 hops ( $< 10$ ), at least 5 hops apart ( $> 5$ ) and at least 10 hops apart ( $> 10$ ). From all three figures (Figure 3.7), we can see that spatial distribution with a mixture of short and long links (blue line) achieves the highest delivery rate for all topologies. Relatively short links ( $< 5$  hops) works best for Topology 1 compared to the other two topologies, and the scheme with only links shorter than 10 hops even performs better than other schemes with relatively longer links, because the local disturbance due to sparsity can be resolved by short links to close nodes. Longer links ( $> 10$  hops) performs significantly better than pure short links in Topology 3, since global disturbance (big holes) requires longer links to compensate the inaccurate distance measure. Different network topologies may require different types of links, but the spatial distribution with a mixed set of short and long links gives a generic solution and hides the diversity of distance inaccuracy, with high delivery rate, small routing table size, low stretch and cost.

### 3.5.2 Landmark-based routing table

We evaluate the performance of the landmark-based routing table (in 3.4.2) on three topologies, compared with S4, as both use a set of landmarks. The benefits of our scheme are that it incurs much cheaper preprocessing cost with smaller routing table size than S4. Our scheme needs fewer landmarks ( $O(\log^2 n)$  rather than  $O(\sqrt{n})$  landmarks). Each node only needs to remember the next hop to each landmark and the sample along the path to that landmark, and does not construct any additional local routing tables. So the size of the routing table is exactly the number of landmarks. The total preprocessing cost is just the message flooding from the landmarks. After that, routes to all long links are built up automatically.

Simulation results show that 30 landmarks are sufficient to achieve good delivery rate (above 94%) and small stretch (about 1.04) in our scheme. In S4, we use 50 landmarks with an average routing table size of 90.62 to achieve the best stretch and routing table size tradeoff. The routing table size in our scheme is 30, with the total preprocessing cost only about 1/3 that of S4 on Topology 3.

## 3.6 Conclusion

We presented in this chapter a geometric theory to build a small number of routing links in very general domains. The method is distributed and uncoordinated, but guarantees global properties such as routing with low stretch and compact routing tables. The use of spatial distribution ensures that the routing works well at all scales and distances.

We have presented here implementation details and simulation results for sensor networks, but the core results are expected to be useful in a wide variety of graphs.



# Chapter 4

## Spatial Gossip and Multi-Resolution Aggregation

In this chapter we propose a lightweight algorithm for constructing multi-resolution data representations for sensor networks. We compute, at each sensor node  $u$ ,  $O(\log n)$  aggregates about exponentially enlarging neighborhoods centered at  $u$ . The  $i$ th aggregate is the aggregated data among nodes approximately within  $2^i$  hops of  $u$ . We present a scheme, named the hierarchical spatial gossip algorithm, to extract and construct these aggregates, for *all* sensors simultaneously, with a total communication cost of  $O(n \text{ polylog } n)$ . The hierarchical gossip algorithm adopts atomic communication steps with each node choosing to exchange information with a node distance  $d$  away with probability  $1/d^3$ . The attractiveness of the algorithm attributes to its simplicity, low communication cost, distributed nature and robustness to node failures and link failures. We show in addition that computing multi-resolution aggregates precisely requires a communication cost of  $\Omega(n\sqrt{n})$ , which does not scale well with network size. The approximation in aggregate computation like that introduced by the gossip mechanism is therefore necessary in a scalable efficient algorithm. Besides the natural applications of multi-resolution data summaries in data validation and information mining, we also demonstrate the application of the pre-computed spatial multi-resolution data summaries in answering range queries efficiently.

### 4.1 Introduction

Distributed wireless sensor networks provide revolutionary ways to attain large scale, dense data collection and long-term environment monitoring. The immediate challenge is to develop efficient methods to extract, encode, and distribute information gathered by sensors, for both the robustness and survivability of data, as well as the flexibility and efficiency to answer user queries. In this chapter we study the problem of constructing multi-resolution data representation in a sensor network to facilitate routing and answering multi-dimensional range queries. Our approach of processing data in a multi-resolution format follows the principle of

*fractional cascading* that states: “a sensor knows a fraction of the information from distant parts of the network, in an exponentially decaying fashion by distance” [56]. This multi-resolution, locality-preserving representation is motivated by observations that sensors are typically monitoring a physical phenomena, which exhibits high correlation in both the spatial and temporal domain. Naturally information relevant to each node is decaying with the distance to this node.

In the setup of this chapter we have  $n$  sensors deployed uniformly and densely inside a region monitoring a continuous data field. We compute, at each sensor node  $u$ ,  $O(\log n)$  aggregates about exponentially enlarging neighborhoods centered at  $u$ . The  $i$ th aggregate is the aggregated data among nodes approximately within  $2^i$  hops of  $u$ . The specifics of aggregation techniques will be application dependent. For example, the aggregates can be the MAX/MIN or AVG, or more involved aggregates such as histogram [140], parameter estimations [157], or random linear projections used for compressed sensing and information recovery [127]. This multi-resolution scheme is inherently load-balanced. The storage requirement at each node is bounded by  $O(\log n)$ . We present a scheme to extract and construct these aggregates, for *all* sensors simultaneously, by a hierarchical spatial gossip algorithm. The total communication cost is  $O(n \text{ polylog } n)$ , only a small polylogarithmic factor of the cost for flooding or information aggregation at a sink, yet we obtain multi-resolution aggregation for each and every sensor node in the network.

The multi-resolution data summaries provide a basis for information mining, data validation and efficient range queries. One of the major challenges in a sensor network is that nodes start with no idea of the big picture over the data field. Thus it is difficult for a node to assess whether its sensor reading is valid or not since detection of outlier or abnormality usually requires comparison with other sensor readings. In certain applications, the sensor field is deployed to detect events of interest to the owner. A sensor node often needs to decide, by itself, whether it holds interesting data or not. In some cases it is trivial, e.g., an unusually high reading by an acoustic sensor typically means activities in its vicinity. Sometimes this requires comparison with the average of sensor readings in an appropriate neighborhood. For example, the temperature threshold considered as ‘high’ in winter is different from that in summer. With the summarized data from each of its exponentially enlarging neighborhoods, a node has a basis against which its own reading can be compared, in order to spot local spikes which indicate data significance [155]. In addition, these partial aggregates can be used to support range queries injected from any node in the network. Queries for the aggregated value inside a geographical region can be answered by combining the pre-computed partial aggregates, without the necessity of examining each and every node in the geographical range. Thus both communication cost and query delay can be improved.

The major contribution of this chapter is the development of a light-weight algorithm for constructing multi-resolution data representations for sensor networks, as well as the application of multi-resolution data for range queries.

### 4.1.1 The challenge and our contribution

To construct the multi-resolution data representation, we first note that simple flooding and aggregation from each node will incur too high communication cost –  $O(n^2)$  since each node incurs a cost of  $O(n)$  to flood the network. In this chapter we investigate gossip algorithms with almost linear communication cost.

In our setting the metric we care most is the total communication cost of the gossip algorithm, which depends on two factors: the cost of communication for each iteration step, and the number of iterations for it to converge. Existing gossip protocols either assume that every two nodes can communicate with a unit cost (e.g., in peer-to-peer networks and distributed systems), or allow only immediate neighbors to gossip (e.g., in the standard gossip model). In our setting, we allow far away nodes to be chosen as gossip partners, and communication between them is performed by multi-hop routing. Thus the cost of each gossip step may involve any two nodes and have a higher cost if the nodes are far apart. This idea is also adopted in geographical gossip to reduce the communication cost of distributed averaging in a random geometric graph [33].

Under the objective of minimizing the total communication cost, the selection of gossip communication mechanism needs to balance two important factors. First, the fast convergence of a gossip protocol depends critically on the selection of gossip partners. Intuitively fast convergence requires information to be well mixed — one of the best is to select a random node in the network as the gossip partner. On the other hand, if we choose a random node to gossip in each iteration, the cost of communication with multi-hop routing is proportional to the distance to a random node in the network, which is roughly  $O(\sqrt{n})$  in a grid-like network with uniformly deployed sensors. To reduce the communication per each iteration, the best is to simply gossip with its immediate neighbors. But analysis of standard gossip on a random geometric graph or a 2-dimensional grid shows a slow convergence of roughly  $O(n^2)$  gossip steps<sup>1</sup> [14, 156], which is asymptotically the same order with that of naive flooding.

The second challenge of the gossip algorithm in this chapter, different from all the other gossip protocols, is on its multi-resolution nature. We would like information to be exchanged and mixed for fast convergence but also want to make sure that information does not travel too far and pollute the aggregates at other nodes. Thus the two conflicting considerations – fast convergence and restricted propagation range – also need to be carefully balanced.

We propose to use a hierarchical spatial gossip algorithm that automatically takes care of all the issues we worried about above. Our hierarchical gossip algorithm proceeds in  $O(\log n)$  phases. In phase  $i$ , we compute, for *all* sensor nodes, their respective aggregates in a roughly  $2^i$  neighborhood. This is achieved by a spatial gossip algorithm in a restricted range, where each node  $x$  picks, from nodes within distance  $2^i$ , a node  $y$  with probability proportional to  $1/d^3$ , where  $d$  is the distance between  $x$  and  $y$ ,  $1 \leq d \leq 2^i$ . Each phase stops after  $O(\text{poly}(i))$  itera-

---

<sup>1</sup>Here we use ‘gossip step’ to refer to the atomic operation of one node gossiping with its partner.



tions,  $i \leq \log n$ . At the end of phase  $i$ , we compute for each node  $u$  the aggregate of a subset of nodes  $S_i(u)$  that contains all the nodes within distance  $2^i$  from  $u$  with high probability, and does not contain any node more than distance  $\text{poly}(i)2^i$  apart. The total communication cost over all phases is bounded by  $O(n \text{polylog } n)$ . Notice that this achieves a substantial improvement in terms of communication cost to the naive flooding approach and is only at most a polylogarithmic factor away from an obvious lower bound of  $\Omega(n \log n)$  for constructing the multi-resolution data representation<sup>2</sup>.

In this chapter, we use the spatial distribution in the form  $1/d^3$  to simplify discussion. This is really equivalent to a distribution  $1/d^{\rho+1}$  on a growth bounded graph, and this distribution also produces a small world. All the analysis here can accordingly be adapted to the graph scenario. Our aim here is really to summarize a continuous signal sampled from a physical space. This is typically useful when the sensors are aware of their locations. Therefore we assume here that the sensors know their coordinates, and work in terms of an explicit embedding in the two dimensional euclidean plane.

What is critical to the success of our hierarchical gossip algorithm is that we use order and duplicate insensitive synopsis (ODI-synopsis) [27, 117] to compute and represent the partial aggregates. The idea of an ODI-synopsis is that the same data can be aggregated multiple times but it is counted only once. Certain aggregates such as MAX/MIN are naturally ODI-synopses. ODI-synopsis for other aggregates such as COUNT and SUM/AVG are available, by implementation through MAX/MIN or boolean OR computations [26, 27, 55, 117]. ODI-synopsis combined with gossip algorithm removes trouble caused by the same data disseminated and aggregated multiple times. In addition, ODI-synopsis is helpful for range queries as we do not need to worry about over-counting resulting from partial aggregates from overlapping regions.

One last note is that our gossip-based method is randomized. The multi-resolution aggregation covers roughly the  $2^i$  neighborhood, for  $i = 0, \dots, \log n$ . The question of computing an accurate set of multi-resolution aggregates, i.e., the aggregate of all the nodes precisely within  $2^i$  hops, is considered in section 4.4. We describe a deterministic algorithm to achieve this. This algorithm has a communication cost of  $\Theta(n\sqrt{n})$ . We show that this is in fact asymptotically optimal, and there is a lower bound of  $\Omega(n\sqrt{n})$  for the message complexity. Accurate multi-resolution computation therefore does not scale well with network size. This makes it necessary to introduce approximate neighborhoods, as considered in our spatial gossip method.

## 4.2 Network setup

In this chapter we consider a network of  $n$  sensor nodes in a square. The sensor nodes are deployed with sufficient sensing coverage such that any unit disk cen-

---

<sup>2</sup>For each sensor node simply reading in their  $\log n$  data summaries it requires a communication cost of  $\Omega(n \log n)$ .

tered inside the region contains at least 1 sensor node. Each sensor node knows its own location and generates a reading which is the sample of an underlying continuous data field at the location of this sensor.

Notice that the above assumption guarantees sufficient coverage but does not prevent regions with dense node distribution. We can further improve the uniformity of the sensor sampling by clustering. We compute a set of clusterheads such that every two clusterheads are of distance at least 1 away and every node is within distance 1 of at least one clusterhead. The clustering can be easily implemented by a greedy and distributed algorithm. Each node checks its nearby nodes to see if there is a clusterhead within distance 1. Otherwise it will promote itself as a clusterhead. By local communication the nodes can select a subset of nodes as clusterheads as desired above.

The set of clusterheads has bounded density. Every two clusterheads are at least distance 1 apart, as specified by the algorithm. Further, inside any disc of radius 2, denoted by  $D_2$ , there are at least 1 clusterhead — this is because any clusterhead outside this disc cannot cover the unit-radius disk  $D_1$  co-centric with  $D_2$ . Thus by the sampling assumption there is at least one node inside the unit disk  $D_1$ , whose clusterhead must be within  $D_2$ .

Thus, without loss of generality we will assume in the following of the chapter that: (i) any two sensor nodes are of distance at least 1 apart; (ii) any disk of radius 2 contains at least one sensor node.

We assume that two sensor nodes can communicate with each other directly if they lie within a small distance of each other. However, we do not enforce that the connectivity corresponds to a unit disk graph or any specific model. We assume that the deployment permits the existence of a multi-hop routing algorithm that can carry a message from node  $x$  to node  $y$  using at most  $O(d_{x,y})$  hops, where  $d_{x,y}$  is the Euclidean distance between the two nodes. For sensors uniformly deployed, simple geographical forwarding would suffice to find a path with length proportional to the Euclidean distance between them.

## 4.3 Spatial gossip

In this section we describe the hierarchical spatial gossip algorithm to compute multi-resolution data summaries for every sensor node.

### 4.3.1 Hierarchical spatial gossip

We use a gossip mechanism where each node selects from a restricted neighborhood a node to gossip with and sends a message to it. The algorithm proceeds in *phases*. The phase  $i$  calculates for each node the aggregate of values inside a roughly  $2^i$  neighborhood centered at itself. The phases are completely independent so that phase  $i + 1$  starts fresh. Since we have a network of  $n$  nodes, with a lower bound on density,  $O(\log n)$  phases are sufficient for the phase with the largest neighborhood to cover the entire network.

For phase  $i$ , we adopt a restricted spatial gossip algorithm. We implement the selection of gossip partner with geographical routing. At each round, a node  $x$  chooses a location  $y^*$  in the sensor field with probability:

$$p_i(x, y^*) = \begin{cases} \frac{1}{\pi} \cdot \frac{1}{(|xy^*|+1)^3}, & |xy^*| \leq 2^i; \\ 0, & |xy^*| > 2^i. \end{cases}$$

where  $|xy^*|$  is the Euclidean distance between nodes  $x$  and  $y^*$ . Notice that  $y^*$  is not necessarily the location of a sensor.  $x$  will send the information towards  $y^*$  and eventually reach the node  $y$  whose location is closest to  $y^*$ . Then  $y$  is  $x$ 's gossip partner and takes the information delivered by  $x$ .

With the above gossip algorithm and the uniformity of sensors, the probability that a node  $x$  chooses a sensor node located at  $y$  (also denoted by  $y$  by slightly abusing the notation) is also proportional to  $1/|xy|^3$ .

**Lemma 4.3.1.** *At phase  $i$ , let the probability that a node  $x$  gossips with a node  $y$  be  $q_i(x, y)$ . Then if  $2 \leq |xy| \leq 2^i + 2$ ,*

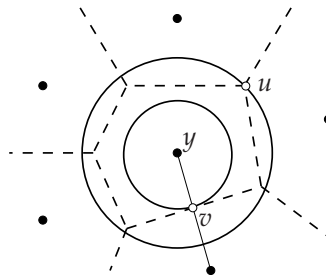
$$q_i(x, y) \leq \frac{4}{(|xy| - 1)^3};$$

and if  $|xy| \leq 2^i - 2$ ,

$$q_i(x, y) \geq \frac{1}{16(|xy| + 3/2)^3};$$

if  $|xy| \geq 2^i + 2$ ,  $q_i(x, y) = 0$ .

**Proof:** We compute the Voronoi diagram of all the sensor nodes (a partitioning of the region into cells such that all the points inside one cell are closest to the same sensor node) and only inspect the part inside the bounding square. In order for  $x$  to choose node  $y$  as its gossip partner,  $x$  must have chosen a location  $y^*$  that falls inside the Voronoi cell of sensor node  $y$ . Denote by  $V(y)$  the Voronoi cell of  $y$ , then we have  $q_i(x, y) = \int_{y^* \in V(y)} p_i(x, y^*)$ .



**Figure 4.1.** The Voronoi cell of a sensor node  $y$  is enclosed inside a disk of radius 2 and contains a disk of radius 1/2.

We now upper and lower bound the Voronoi region for  $y$ .  $V(y)$  is a convex region. The point on the boundary of  $V(y)$  furthest from  $y$  is realized at a Voronoi

vertex ( $u$  in Figure 4.1), which has three sensors (including  $y$ ) as its closest nodes. Thus the disk centered at  $u$  with radius  $|yu|$  has no other sensor nodes inside. Since any disk of radius 2 has at least one sensor node inside,  $|yu| < 2$ . Thus  $V(y)$  is enclosed by a disk centered at  $y$  with radius 2, denoted by  $D_2(y)$ . On the other hand, the point on the boundary of  $V(y)$  closest to  $y$ , say  $v$ , is realized as the midpoint connecting  $y$  and one of its Delaunay neighbors (the sensors whose Voronoi cells are adjacent to that of  $y$ 's). Thus  $|yv| \geq 1/2$ . Consider that  $y$  can be placed at the corner of the sensor bounding square.  $V(y)$  includes at least  $1/4$  of a disk of radius  $1/2$  centered at  $y$ .

With the upper and lower bound of  $V(y)$ , we will bound the probability  $q_i(x, y)$ . Take the point in  $V(y)$  closest to  $x$ , denoted by  $w$ .  $|xw| \geq |xy| - 2$ . Therefore  $q_i(x, y) = \int_{y^* \in V(y)} p_i(x, y^*) \leq p_i(x, w) \cdot \pi 2^2 = 4/(|xw| + 1)^3 \leq \frac{4}{(|xy| - 1)^3}$ . Similarly, we have  $q_i(x, y) \geq \frac{1}{16(|xy| + 3/2)^3}$ .

The above bound is valid when  $V(y)$  is completely within distance  $2^i$  from  $x$ , which is true if  $|xy| \leq 2^i - 2$ . If  $|xy| \geq 2^i + 2$ , then all points in  $V(y)$  are of distance  $2^i$  away. Thus  $y$  will never be chosen as  $x$ 's partner.  $q_i(x, y) = 0$ .  $\square$

We assume that all the nodes gossip in a synchronous way. At each clock tick, every node selects and shoots its information to its respective gossip partner. We consider each clock tick as a *round*. Once a node  $x$  chooses another node, say  $y$ , with distance at most  $2^i$  from it,  $x$  sends its current synopsis to  $y$ .  $y$  will incorporate the information it receives from  $x$  and maintain the aggregation of synopsis of its old value with the synopsis from  $x$ . Note that this is asymmetric as only node  $y$  updates its synopsis and node  $x$  keeps its current synopsis value. The asymmetry is an attractive feature as reliable round-trip multi-hop routing adds communication overhead and implementation difficulty. Denote by  $s_{i,j}(x)$  the synopsis at any node  $x$  after round  $j$  of phase  $i$ . The original value at  $x$  is thus given by  $s_{0,0}(x)$ . After round  $j$ , each node updates its synopsis to be the aggregation of its synopsis at round  $j - 1$  and all the values it received in this round. The value computed at node  $x$  at completion of phase  $i$  is denoted by  $s_i(x)$ .

All the aggregates in our scheme are order and duplicate insensitive synopsis. In particular, given a set of values  $S$ , an ODI-synopsis is an aggregate computed for values in  $S$  that remains the same no matter how many times one duplicates some values in  $S$  or what order the aggregation was performed. For example, MAX/MIN are naturally ODI-synopsis. ODI-synopsis for other aggregates such as SUM or AVG are available [26, 27, 55, 117] by essentially implementing them by MAX/MIN or Boolean operations. We remark that some of the ODI-synopsis are probabilistic in nature. In this chapter we often use MIN as the example, but the algorithm works with any ODI-synopsis.

The use of ODI-synopsis is key to the success of the spatial gossip algorithm for constructing multi-resolution data representation. The insight is that aggregation by ODI-synopsis tremendously simplifies gossip computation protocol. Each node  $u$  keeps only a value  $s(v)$  which is the ODI-synopsis of the set of values it has received so far and does not keep the set of values in its original form. When one node

$u$  chooses to gossip with  $v$ ,  $u$  sends to  $v$  its aggregate  $s(u)$  and  $v$  computes and keeps the ODI-aggregation of the synopsis of both  $u$  and  $v$ .  $s(v) \leftarrow s(u) \oplus s(v)$ , where  $\oplus$  represents the aggregation function of the ODI-synopsis. This not only reduces the cost of transmission as only one aggregated value is delivered each step, but also guarantees that over-counting is eliminated although the same value may potentially be received multiple times. In short, with ODI-synopsis the model of gossip computation is the same as alarm spreading — each node starts with its own value and in each gossip step one node will send all the values it has received so far — but with reduced communication cost since only the aggregate (not the whole set of values) is delivered. When the algorithm stops, a node keeps the aggregate of all the values it has received.

To make the analysis easier, we also denote by  $S_{i,j}(x)$  the set of nodes whose values  $x$  should have received if we deliver all the original values instead of a synopsis in the gossip algorithm. In other words,  $s_{i,j}(x)$  is the aggregation of the values in the set  $S_{i,j}(x)$ . The set corresponding to the value  $s_i(x)$  at node  $x$  at the completion of phase  $i$  is denoted by  $S_i(x)$ .

To summarize, there are at most  $O(\log n)$  phases in the hierarchical spatial gossip algorithm. In phase  $i$ , every node executes  $O(i^{3.4})$  synchronous rounds. Each round consists of a single gossip operation performed by every node, and each phase consists of sufficient number of rounds so that nodes  $x$  and  $y$  that lie within a distance  $2^i$  of each-other obtain each-other's values with high probability. Thus, at the end of phase  $i$ , any node has considerable information about values within a distance  $2^i$  from it. Thus the synopsis aggregate at each node has incorporated sufficiently many nodes within its  $2^i$  neighborhood.

The gossip algorithm for each phase is very similar to the spatial gossip protocol proposed by Kempe *et al.* [79], except that we restrict the maximum range of gossip partners. This modification is to reduce the level of pollution such that a node does not receive information from nodes too far away, as will be made clear later.

### 4.3.2 Multi-resolution representations

In this section, we analyze the multi-resolution information computed by the algorithm described above. We show that if we stop the algorithm at phase  $i$  after  $j^* = O(i^{3.4})$  rounds, the synopsis kept at node  $x$ , i.e., the aggregated value of a set of nodes  $S_{i,j^*}(x)$ , captures the information in a roughly  $2^i$  neighborhood around  $x$ . Without loss of generality we denote by  $S_i(x)$  and  $s_i(x)$  the respective values when  $j = j^*$ .

Specifically, we show upper and lower bounds for the set of nodes in  $S_i(x)$ . Theorem 4.3.4 says that  $S_i(x)$  includes with high probability each node within distance  $2^i$  from  $x$ . Theorem 4.3.5 says that  $S_i(x)$  does not include nodes  $O(2^i i^{3.4})$  away for sure and with high probability does not include nodes with distance  $O(2^i i^{2.4})$  or more away.

Before we prove our main theorems, we first observe that when we run more iterations of the gossip algorithm, the amount of information each node gets is monotonically non-decreasing within a phase. Recall that  $S_{i,j}(x)$  is the set of nodes whose

values have reached  $x$ , after  $j$  rounds at phase  $i$ . Thus,

**Observation 4.3.2.**  $S_{i,j}(x) \subseteq S_{i,j+1}(x)$ , for any  $i, j, x$ .

### Lower bound

We first show a lemma that bounds the rate of information propagation by the restricted spatial gossip algorithm. Intuitively the lemma says that after  $O(\text{polylog } d)$  rounds information at one node reaches a node at distance  $d$  with high probability.

**Lemma 4.3.3.** *In phase  $i$ , if the distance between nodes  $x$  and  $y$  is  $d \leq 2^i$  then  $S_{i,j}(x) \subseteq S_{i,j+\alpha}(y)$  within  $\alpha = O(\log^{3.4} d)$  rounds of iterations with probability at least  $1 - O(\frac{1}{d})$ .*

The proof is an adaptation of the proof in [79] that bounds the information spread rate in spatial gossip, with necessary modification that additionally takes care of the restricted range. While the essential proof is the same, the adjustment to get the specific result is not entirely trivial. We therefore include the complete proof in the appendix. This lemma shows that information propagates pretty fast in the network. Thus we can stop the algorithm in  $O(\text{poly}(i))$  rounds for phase  $i$ ,  $i \leq \log n$ , in order to collect information from almost all nodes inside the desired range  $2^i$ .

**Theorem 4.3.4.** *With probability at least  $1 - O(1/2^i)$ , the set  $S_{i,w}(x)$  includes node  $y$  with  $|xy| \leq 2^i$  in phase  $i$  consisting of  $w = O(i^{3.4})$  rounds.*

**Proof:** Obviously  $x \in S_{i,0}(x)$ . We apply Lemma 4.3.3 with  $d = 2^i$  to obtain the theorem. This implies that in round  $i$ , any node collects information from each node in its  $2^i$  neighborhood with probability at least  $1 - O(1/2^i)$ .  $\square$

### Upper bound

The subsection above shows that in phase  $i$ , any node receives the information within a distance  $2^i$  with high probability if we run the algorithm for  $O(i^{3.4})$  rounds. Now we show an upper bound that a node does not get information from nodes too far away. Thus the ‘pollution’ from far away nodes is under control.

**Theorem 4.3.5.** *After  $k$  rounds of phase  $i$ ,*

1.  $S_{i,k}(x)$  does not include nodes with distance  $d > k2^i$  away from  $x$ , for sure.
2.  $S_{i,k}(x)$  does not include nodes with distance  $d > \frac{3k2^i}{i+1}$  away from  $x$  with probability at least  $1 - o(1/2^k)$ , when  $i$  is greater than a sufficiently large constant.

**Proof:** To make the analysis easier, we assume that we actually propagate, by the hierarchical spatial gossip algorithm, the list of values together with their source nodes. Initially each node has only its own value. Then they propagate to other nodes. We examine, for the value of a node  $u \in S_{i,k}(x)$ , the path it may take to get



from  $u$  to  $x$ , denoted by  $P = \{u, u_1, \dots, u_\ell = x\}$ .  $\ell \leq k$ . In round  $j$ ,  $u_j$  selects  $u_{j+1}$  as its gossip partner.

**Claim 1.** The value of  $u$  cannot travel further than  $k2^i$  because in any iteration, a gossip step can go as far as  $2^i$  at most and there are total  $k$  rounds.

**Claim 2.** Intuitively, for the value of  $u$  to reach a node  $x$  that is distance  $d = \frac{(3+\varepsilon)k2^i}{i+1}$  away ( $\varepsilon$  is very small) via a path of length at most  $k$ , it must make enough number of long jumps. We argue that the probability for this to happen is small. The following analysis is to make this intuition rigorous.

First observe that the probability of a node  $u_j$  choosing a node  $u_{j+1}$  of distance  $d' > 2^i/(i+1) - 2$  away is at most

$$\int_{2^i/(i+1)}^{2^i} \frac{2r}{(r+1)^3} dr \leq \frac{i+1}{2^{i-1}}.$$

Now consider a path  $P$  of at most  $k$  hops that starts from  $u$  and ends at  $x$ . Let  $k'$  be the minimum number of steps of length  $2^i/(i+1)$  or more in  $P$ . Then the minimum value of  $k'$  satisfies the relation

$$(k - k')\left(\frac{2^i}{i+1} - 2\right) + k'(2^i + 2) \geq d = 2^i \frac{(3 + \varepsilon)k}{i+1}.$$

When  $i$  is sufficiently large,  $k' \geq (2 + \varepsilon/2)\frac{k}{i}$ . Therefore, for a  $k$ -hop path to reach node  $x$ , it needs to have at least  $k'$  long jumps, the probability of which is at most  $\binom{k}{k'} \left(\frac{i+1}{2^{i-1}}\right)^{k'}$ . Thus, the probability that a  $k$ -hop path  $P$  does not have  $k'$  or more links of length  $2^i/(i+1)$  or more is at least  $\left(1 - \binom{k}{k'} \left(\frac{i+1}{2^{i-1}}\right)^{k'}\right)$ .

In each round, a node that has a data sends a copy of it to another node. Thus, every existing copy gets replicated at a new node. At the end of  $k$  rounds, the total number of copies in the network is at most  $2^k$ . We bound the probability that none of these  $2^k$  paths reach  $x$ . This is at least

$$\begin{aligned} & \left(1 - \binom{k}{k'} \left(\frac{i+1}{2^{i-1}}\right)^{k'}\right)^{2^k} \\ & \approx 1 - \binom{k}{k'} \left(\frac{i+1}{2^{i-1}}\right)^{k'} 2^k \geq 1 - 2^{2k} \left(\frac{i+1}{2^{i-1}}\right)^{k'} \\ & \geq 1 - \left(\frac{(2(i+1))^{2/i}}{2^{\varepsilon/2}}\right)^k \geq 1 - 1/2^k. \end{aligned}$$

The last step is true when  $i$  is greater than a sufficiently large constant.  $\square$

For a phase  $i$ , with  $k = i^{3.4}$  rounds, the probability that the value at a node does not spread beyond a distance  $2^i \frac{3k}{i+1}$  is at least  $1 - o(1/2^{i^{3.4}})$ . Thus with high probability  $S_i(x)$  does not include nodes with distance  $O(2^i i^{2.4})$  away.

### 4.3.3 Communication cost

In this section we show that the communication cost of constructing the multi-resolution data representation is almost linear.

**Lemma 4.3.6.** *The communication cost incurred by any node in a single round of phase  $i$  is  $O(i)$ .*

**Proof:** The expected distance to the gossip partner chosen by a node  $x$  is at most

$$2 + \int_0^{2^i} 2r \frac{r}{(r+1)^3} dr \simeq O(i).$$

Since the cost of routing to a node distance  $d$  away is  $O(d)$ , the communication cost by any node in a round of phase  $i$  is  $O(i)$ .  $\square$

**Theorem 4.3.7.** *The algorithm creates multi-resolution data as described above at every node using  $O(\log^{4.4} n)$  rounds and total communication cost  $O(n \log^{5.4} n)$ . The storage requirement at each sensor node is  $O(\log n)$ .*

**Proof:** In an  $n$  node network, with a constant lower bound on density, the maximum distance between any two nodes is  $O(n)$ . Thus, the number of phases required by the algorithm is  $O(\log n)$ . Each phase  $i$  consists of  $O(i^{3.4})$  rounds. Thus, the number of rounds is  $\sum_{i=1}^{\log n} O(i^{3.4}) = O(\log^{4.4} n)$ . In phase  $i$ , at each round, a node uses a single message with an expected communication cost of  $O(i)$ . Thus, the communication cost per node for the algorithm is:  $\sum_{i=1}^{\log n} O(i \cdot i^{3.4}) = O(\log^{5.4} n)$ . The total communication cost is thus  $O(n \log^{5.4} n)$ . Notice that during the spatial gossip algorithm for phase  $i$ , each node at any time only keeps one value. The total storage requirement for each node is  $O(\log n)$ .  $\square$

## 4.4 Accurate multi-resolution data

The gossip based algorithm is randomized, and therefore has some inaccuracy associated with the aggregates it computes. In this section, we discuss a deterministic algorithm to compute multi-resolution aggregates and show a communication lower bound of  $\Omega(n\sqrt{n})$  messages on computing multi-resolution data. These results show that approximation is necessary in order to achieve near linear communication cost.

For the ease of description we use the  $\ell_\infty$  metric, and assume that the  $n$  nodes are placed on a unit grid in a square. A disk in this metric looks like a square. Suppose the aggregate minimum is being computed. The algorithm works as follows:

At step  $i$ , every node  $p$  finds the aggregate of the  $\ell_\infty$  disk of radius  $2^i$  centered at itself. This is done as follows:  $p$  collects the aggregates of step  $i-1$  from each node  $q$  at distance  $2^{i-1}$  from  $p$ , and computes the minimum to find the aggregate



minimum of its  $2^i$  neighborhood. Each node  $q$  needs to send its  $(i - 1)^{th}$  average to nodes at a distance  $2^{i-1}$  from it. This is done by traversing the boundary of the disk of radius  $2^{i-1}$ , at a cost of  $O(2^{i-1})$ .

The total cost per node is therefore  $\sum_{i=0}^{\log \sqrt{n}} O(2^{i-1}) = O(\sqrt{n})$ .

The following example shows that this is in fact a lower bound on the asymptotic complexity of computing multi-resolution data.

Suppose that the left topmost corner of the grid has position  $(0, 0)$ . The node at row  $i$  and column  $j$  has position  $(i, j)$  and value  $v_{ij} = i\sqrt{n} + j$ . More importantly, this is also the rank of the value. Now consider the quadrant with  $i, j \in [\sqrt{n}/2, \sqrt{n}]$  and in particular the node at  $(\sqrt{n}/2, \sqrt{n}/2)$ . The minimum of its  $\sqrt{n}/2$  neighborhood is given by  $v_{00}$ . The corresponding aggregate of any node in the quadrant at  $(\sqrt{n}/2 + i, \sqrt{n}/2 + j)$  is given by  $v_{i,j}$ . Therefore, each such value has to be transmitted a distance  $\Omega(\sqrt{n})$ . Since at least a constant fraction of the values have to be transmitted this distance, the lower bound on the message cost is  $\Omega(n\sqrt{n})$ .

## 4.5 Range queries

The pre-computed data summaries by the hierarchical spatial gossip algorithm can be useful in answering user queries about aggregates in large regions of the network with reduced cost. For example, the aggregate for the entire network is available at any single node. Similarly, it is possible to obtain probabilistic information about a large region of radius  $2^i$  by visiting a single node at its center. If the query requires better estimates of the aggregate, then it can be answered by making use of the different ODI synopses computed at different phases of the algorithm. Thus, the query response mechanism can adapt to the quality of estimate and restriction on pollution desired by the user.

In the rest of this section we discuss a case where the user wishes to obtain with high probability the correct ODI synopsis of a rectangular region, without any pollution. The query consists of an  $a \times b$  axis aligned rectangular area  $A$ , and a small probability  $\delta$ . The response to the query is the ODI synopsis  $s$  corresponding to a set  $S$ , such that, for any node  $x$ , if  $x \in A$  then  $x \in S$  with probability at least  $1 - \delta$ , and if  $x \notin A$  then  $x \notin S$ . That is, no node outside the region  $A$  should be included in set  $S$ , and no node inside  $A$  should be excluded with a probability more than  $\delta$ . Without loss of generality, we can assume that  $a \leq b$ .

By Theorem 4.3.4, after phase  $i$ , the ODI synopsis at any node  $x$  includes the value at any other node inside a disk of radius  $2^i$  with a high probability. For distances measured in the  $L_\infty$  metric, this disk corresponds to a square of side  $2 \cdot 2^i$ . We refer to such a square as a square of radius  $2^i$  (analogous to a disk of same radius), and use a set of such squares to *cover* the given query region.

We denote by  $B_i(x)$  a square of radius  $2^i$  centered at node  $x$ . For a node  $y \in B_i(x)$ , by Theorem 4.3.4,  $y \notin S_i(x)$  with probability  $O(1/2^i)$ . Corresponding to any square  $B_i(x)$ , there is a square  $G_i(x)$  of radius  $\eta i^{3.42^i}$ , for a proper constant  $\eta$ , such that

for any node  $y \notin G_i(x)$ ,  $y \notin S_i(x)$ , by Theorem 4.3.5. If the user query requires no pollution from outside the query region, the bigger square  $G_i(x)$  must be completely inside the query range.

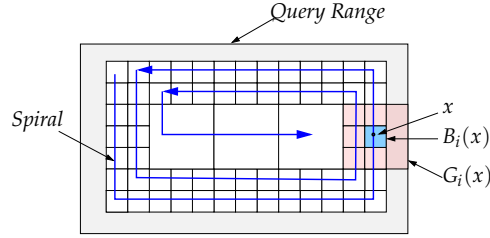
We refer to a square  $B_i(x)$  as a *maximal piece* if  $G_i(x) \subseteq A$  and  $G_{i+1}(x) \not\subseteq A$ , and  $i$  as the maximal level of node  $x$ . Let  $B_p(x)$  be the largest maximal piece in  $A$ , then formally

$$p = \max_{x \in A} \{i : B_i(x) \text{ is a maximal piece}\}.$$

Then, we have

$$\eta p^{3.4} 2^p \leq \frac{a}{2} < \eta(p+1)^{3.4} 2^{(p+1)}.$$

This implies that  $p = O(\log a)$ . Now we can collect the partial aggregates from these maximal pieces to answer the query. This can be done in a manner similar to that in [56] by starting at the boundary and spiraling inward accumulating synopsis for maximal pieces that together cover the entire region. Additionally, we must ensure that the probability of any node being excluded in the synopsis is small.



**Figure 4.2.** The spiral used for response for a given query region. Nodes are visited individually in the shaded region at the perimeter. The figure also shows the maximal square  $B_i(x)$  for a node  $x$  of maximal level  $i$ , and the corresponding pollution region  $G_i(x)$ .

We use a spiral path that guarantees the required probability for every node in the query region. By Theorem 4.3.4, if a node is covered by a maximal piece  $B_i(\cdot)$ , the probability of it being included in the corresponding synopsis set  $S_i(\cdot)$  increases with the size of  $B_i(\cdot)$ . This implies that given a  $\delta$ , nodes more than a certain distance (depending on delta) away from the boundary are covered by one or more maximal pieces that provide the required probability. Thus, our spiral starting at the boundary accumulates synopsis from all individual nodes up to this distance, and makes use of maximal pieces to obtain the synopsis for the rest of the region. Figure 4.2 shows a schematic representation of this idea. The following theorem gives the cost for such a computation.

**Lemma 4.5.1.** *Given a query  $(A, \delta)$  where  $A$  is an  $a \times b$  rectangular axis aligned query region, the query can be answered at a cost of  $O(\max(a, b) \log^{4.4} \min(a, b) + \max(a, b)(1/\delta) \log^{3.4}(1/\delta))$ .*

**Proof:** For a node  $x$ , let  $d_x$  be the distance of node  $x$  from the perimeter of the region  $A$ . If  $i$  is the maximal level of  $x$ , then  $\eta i^{3.4} 2^i \leq d_x < \eta(i+1)^{3.4} 2^{(i+1)}$ . This implies that all nodes of maximal level  $i$  occur in an annular rectangular region of inner

boundary  $(b - 2\eta(i + 1)^{3.4}2^{(i+1)}) \times (a - 2\eta(i + 1)^{3.4}2^{(i+1)})$  and outer boundary  $(b - 2\eta i^{3.4}2^i) \times (a - 2\eta i^{3.4}2^i)$ . The thickness of this annular rectangle is  $O(i^{3.4}2^i)$ .

To obtain the result with parameter  $\delta$ , we start at the perimeter of region  $A$ , and spiral inward accumulating the synopsis  $s$ . At a distance  $\eta d \log^{3.4} d$  from the boundary, a maximal piece of level  $\log d$  can be used, and the probability of a node at this level being missed by a maximal piece is  $O(1/d)$ . By the requirements of the query, it has to be ensured that  $\delta = O(1/d)$ . Thus, the spiral visits each individual node until the distance to the boundary reaches  $O((1/\delta) \log^{3.4}(1/\delta))$ . At every node  $x$ , the synopsis is updated as  $s = s \oplus s_0(x)$ . The cost of such a path is  $O((a + b)^{\frac{1}{\delta}} \log^{3.4}(1/\delta))$ .

After this point, the synopsis are updated according to maximal levels. At a node  $x$  of maximal level  $i$ , we set  $s = s \oplus s_i(x)$ , which is equivalent to the operation  $S = S \cup S_i(x)$ . The lowest maximal level that we can use for the given query is  $\gamma = O(\log(1/\delta))$ . The cost incurred to process nodes at any maximal level  $i \geq \gamma$  is  $O((a - i^{3.4}2^i)i^{3.4} + (b - i^{3.4}2^i)i^{3.4}) = O(b \cdot i^{3.4})$ .

The cost for the spiral covering all maximal levels  $i$  for  $\gamma \leq i \leq p$  is given by

$$\sum_{i=\log(1/\delta)}^p O(b \cdot i^{3.4}) = O(bp^{4.4}) = O(b \log^{4.4} a).$$

Thus, the total communication cost for answering the query is  $O(\max(a, b) \log^{4.4} \min(a, b) + \max(a, b)(1/\delta) \log^{3.4}(1/\delta))$ .  $\square$

**Spatial gossip with no maximum range restriction.** We note that the hierarchical spatial gossip for phase  $i$  makes only one change to the spatial gossip algorithm as in [79]. Essentially a node chooses its gossip partner with a maximum distance range  $2^i$ . This way we are able to restrict the amount of pollution from distant nodes. In the above range query, we make use of the fact that the data summaries do not include information beyond a certain distance threshold (claim 1 in Theorem 4.3.5), to answer queries with no false positive errors.

For applications in which small false positive errors are not a problem, we can propose to use the single-phase spatial gossip algorithm to construct the multi-resolution data representations. Essentially, we just run the standard spatial gossip algorithm where each node chooses another node with distance  $d$  away with probability roughly  $1/d^3$ . We run the algorithm for  $O(\log^{3.4} n)$  rounds. During the algorithm, we keep the current aggregation value after round  $O(i^{3.4})$ , as the data summary of the  $2^i$ -hop neighborhood. Notice that the probabilistic upper bound on pollution as the second claim in Theorem 4.3.5 still holds. Thus the  $i$ th data summary we compute has a large probability to include every value inside a  $2^i$ -hop neighborhood and not include values outside  $2^i i^{2.4}$  neighborhood. This alternative solution saves a factor of  $O(\log n)$  in the total communication cost, at the cost of more pollution from far away nodes. For range query, a probabilistic solution with both small false positives and small false negatives can be obtained. In practice

either variation can be adopted, dependent on application requirements. We evaluated and compared the gain of each variation in the simulation section.

**Error introduced by ODI synopsis.** The analysis above considers the probabilistic error introduced by the gossip. ODI-synopses for aggregates such as SUM, AVG are probabilistic with small probabilities of error. Thus, the overall system error may incorporate this factor, which will depend on the actual ODI synopsis used.

## 4.6 Simulations

In this section, we show that simulation results confirm our expectations on the properties of the hierarchical spatial gossip. We compare our approach with the naive flooding and standard spatial gossip (single phase, with no restriction maximum range) for the total communication cost and the effectiveness of multi-resolution representation. We focus on evaluating the performance of our approaches at the algorithm level, and do not consider the underlying details, such as collisions and packet loss, at MAC and link layers. We use geographical routing in the simulations. Each packet transmitted only contains necessary location information and a piece of aggregate data of the source node. All simulations are done in C++ on a unit-disk graph model. For the simplicity of explanation, we denote the set of nodes within  $2^i$  distance from node  $x$  as  $D_i(x)$ . The aggregate of  $D_i(x)$  is referred as the aggregate of resolution level  $i$  at node  $x$ . We compute the aggregate MIN as an example in the following simulations, other ODI-synopsis can be evaluated in the same way. All simulation results are averaged on 10 runs.

### 4.6.1 Total communication cost

We simulated a grid network where the sensor nodes have a fixed transmission range 2. Nodes can communicate directly if they are within the transmission range of each other. Keeping the density of the network constant, we vary the number of nodes from 256 to 4900, and vary the size of the sensor field from  $32 \times 32$  to  $140 \times 140$ .

In the hierarchical spatial gossip, each phase  $i$  was terminated when at least  $(1 - 0.5/i)$  fraction of nodes in the entire sensor field correctly computed the minimum value of resolution level  $i$ . This condition was found to provide a reasonable balance between fast information propagation and low pollution rates.

Figure 4.3 shows the total communication cost in grid networks with various sizes. Naive flooding incurs dramatically higher cost, because in a network with  $n$  nodes, it requires  $O(n)$  transmissions for propagating one piece of data, and  $O(n^2)$  transmissions in total. As expected, the hierarchical spatial gossip costs slightly more than the spatial gossip, but is still almost linear in the network size.

## 4.6.2 Effectiveness of multi-resolution representation

The approach of flooding the network is communication expensive. But with flooding we can compute the accurate multi-resolution data summaries by labeling each flood message with the location of its starting point. Thus, we only compare the effectiveness of multi-resolution representation of our approach with the single phase spatial gossip here.

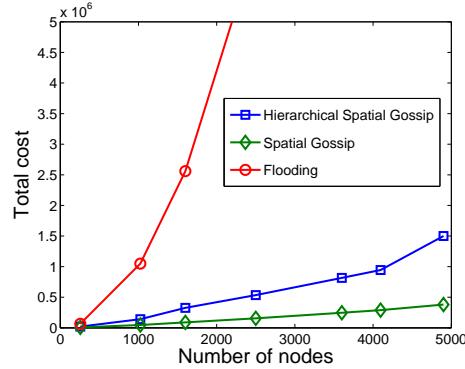


Figure 4.3. Total communication cost in grid networks with various size.

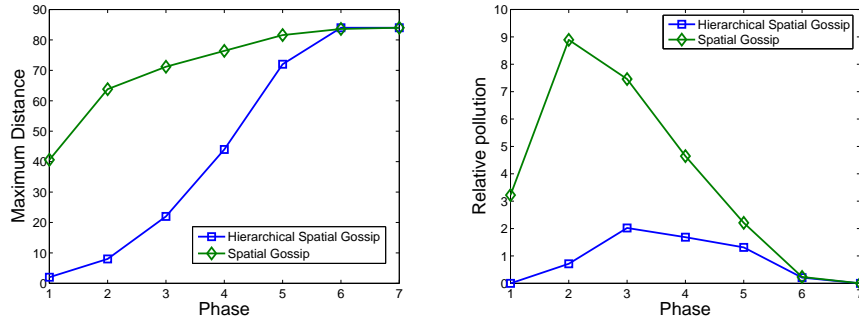


Figure 4.4. (i) Maximum distance reached in each phase. (ii) Relative pollution in each phase.

We compare the standard spatial gossip with hierarchical spatial gossip when they reach roughly the same state. For example, if round 15 of spatial gossip is the first round at which at least a fraction of  $(1 - 0.5/3)$  nodes correctly compute the aggregates of resolution level 3, then the states of the 15<sup>th</sup> round is comparable to phase 3 in hierarchical spatial gossip. The following simulations are conducted in a  $128 \times 128$  grid network with 4096 sensor nodes. We take one piece of data  $s$  of the node located in the center of the network as a representative, and evaluate the entire process of its propagation. All other data is propagated in the same way. Intuitively, an ideal multi-resolution representation should compute aggregates at level  $i$  of almost all nodes belonging to  $D_i$ , and little or no pollution beyond  $D_i$ .

The example of one execution in Figure 4.6 shows different phases of the propagation of  $s$  in the hierarchical spatial gossip. We can see that the information  $s$  is propagated within a restricted range in each phase and pollutes very few nodes beyond a certain distance. In the following, we use three measurements, viz., per-

centage of coverage, maximum distance and relative pollution, to compare the performance of our approach with the spatial gossip.

**Coverage.** We define the *percentage of coverage* at distance  $d$  as the percentage of the number of nodes receiving  $s$  at distance  $d$  from the origin of  $s$ . In Figure 4.5, we show the percentage of coverage in an intermediate phase (phase 4) for both standard spatial gossip and hierarchical spatial gossip. The result confirms that there is a disk such that nodes within it receive the value with high probability. And the probability of a node outside this disk receiving the values falls sharply with the distance from the origin.

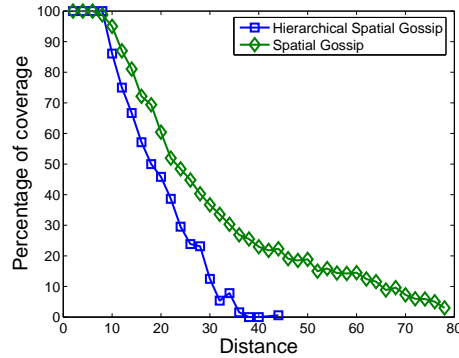


Figure 4.5. Coverage of phase 4.

In the hierarchical spatial gossip, all nodes within a disk with radius 8 from the center receive  $s$ . The percentage of coverage decreases quickly as the distance increases, and goes below 10% beyond distance 30. The propagation quickly stops at distance 44.6. In standard spatial gossip, all nodes within a disk with radius 6 from the center receive  $s$ , but it pollutes the information at distant nodes up to a distance of 78, almost to the boundary of the network.

**Pollution.** The small coverage in hierarchical spatial gossip implies low pollution rates. This is visible in figure 4.5. The single-phase spatial gossip always selects nodes from the entire network, thus it cannot guarantee a comparable restriction on pollution. We characterize and compare the pollution caused by the two approaches using two more criteria - maximum distance and relative pollution. We define the *maximum distance* of phase  $i$  as the distance between the center and the furthest node receiving  $s$  in phase  $i$ . The *relative pollution* of phase  $i$  is defined as the ratio of the number of nodes receiving  $s$  beyond  $D_i(\text{center})$  and the number of nodes receiving  $s$  within  $D_i(\text{center})$ .

Figure 4.4(i) shows the maximum distance reached at the end of each phase in both approaches. Since we simulate in a  $128 \times 128$  grid network, the farthest point from the center is at a distance of about 90 units. In the hierarchical spatial gossip, the maximum distance increases relatively slowly with phases, while in the single-phase spatial gossip, the data often reaches distant nodes within the first few rounds. From Figure 4.4(ii), we can see that there is a big gap between the single-phase spatial gossip and the hierarchical spatial gossip in terms of relative



pollution. The peaks are 9 and 2 respectively. Since we compare the states of the standard spatial gossip at the point of reaching the same state in the hierarchical spatial gossip, the number of nodes getting  $s$  within  $D_i$  is roughly the same in both approaches. However, to build up the same level resolution, the single-phase spatial gossip would pollute data at about 4 times as many nodes beyond that level than the hierarchical spatial gossip.

**Conclusion.** Efficient communication and sharp multi-resolution representation are two conflicting goals. The naive flooding can obtain exact accurate aggregates but with high communication cost. The standard single phase spatial gossip is communication efficient, but it is possible that the information propagates to distant nodes before a sufficient number of nearby nodes gets the data. The hierarchical spatial gossip balances the above two goals by restricting the range of information propagation. Compared with the single-phase spatial gossip, it achieves a sharper multi-resolution representation with only a slightly higher communication cost.

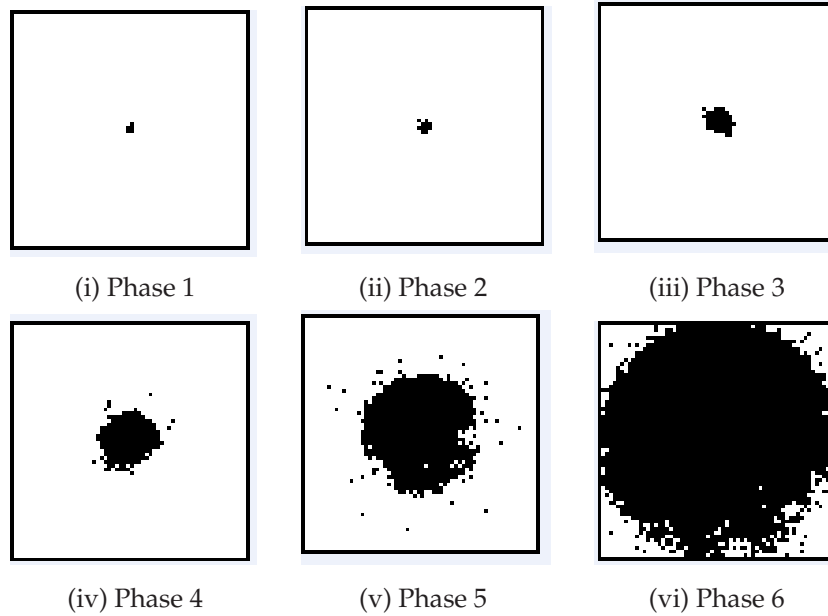


Figure 4.6. Propagation of one piece of data of the node located in the center of the field.

## 4.7 Conclusion

In this chapter, we propose an efficient algorithm with a total communication cost of  $O(n \text{ polylog } n)$  to extract and construct sharp multi-resolution data representations for sensor networks. We believe that the multi-resolution data summary is a fundamental data storage paradigm to equip each node with compact sketches of the global picture of the data field. As the future work we will explore more applications



of multi-resolution data summaries for advanced data processing and validation, as well as efficient query evaluations.

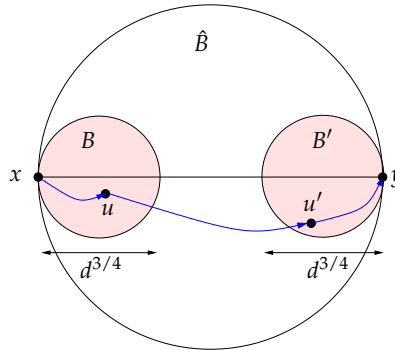
## Appendix

**Proof (Lemma 4.3.3):** From the setup of the network described in section 4.2, observe that the density of the node deployment has lower and upper bounds in any region of the network. In particular, for the following analysis we assume that there are constants  $\beta_1, \beta_2$  ( $\beta_1 < \beta_2$ ) such that the number of nodes in any disk of radius  $r \geq 1$  lies between  $\beta_1 r^2$  and  $\beta_2 r^2$ .

The probability  $1 - O(\frac{1}{d})$  can be rewritten as  $1 - \gamma g(d)$  for  $\gamma = O(\log^{-2.4 - \frac{\log d}{\log \log d}} d)$  and a suitable function  $g(d) = O(\log^{2.4} d)$ . And the number of rounds  $O(\log^{3.4} n)$  can be written as  $\tau g(d)$  for a suitable  $\tau = O(\log d)$ .

Note that, if in the  $j$ th round of phase  $i$  a node  $x$  selects a node  $y$  to gossip, then  $S_{i,j}(x) \subseteq S_{i,k}(y), \forall k > j$ . And this property holds transitively. So, all we need to prove is that there would be a sequence of gossip selections taking the message from  $x$  to  $y$  within  $g(d) = O(\log^{2.4} d)$  rounds with probability at least  $1 - \gamma g(d)$ . Our induction hypothesis is that the result holds for distances upto  $d^{3/4}$ .

First note that for the base case of  $r$  equal to some constant, any constant probability  $1 - \gamma g(r)$  of the value from  $x$  reaching  $y$  can be obtained with constant  $k$  number of selections by  $x$ . This constant will depend on  $\beta_2$ , the upper bound on density since there can be  $\beta_2 d^2$  nodes that are nearer to  $x$  than  $y$ .



**Figure 4.7.** In a time interval  $\tau$ , the long link  $uu'$  exists with high probability, and links  $xu$  and  $u'y$  exist with corresponding high probability.

Consider the disk  $\hat{B}$  of diameter  $d$  containing both  $x$  and  $y$ . Inside  $\hat{B}$ , we take two disks  $B$  and  $B'$  of diameter  $d^{3/4}$  containing  $x$  and  $y$  respectively. Our induction hypothesis is that the result holds for pairs of nodes  $d^{3/4}$  apart. Thus,  $g$  is the recursive function  $g(r) = 1 + 2g(r^{3/4})$ . It can be shown that  $g(r) = O(\log^{2.4} r)$ .

We divide the time interval  $\tau g(d)$  into intervals  $\tau g(d^{3/4}), \tau$  and  $\tau g(d^{3/4})$ . We need to show, that with high probability, some node  $u$  from  $B$  selects some node  $u'$  from  $B'$  to gossip with in a time interval of length  $\tau$ . The rest follows by induction.

The probability that in  $\tau$  rounds, no node from  $B$  selects any node from  $B'$  to gossip with, is given by

$$\left(1 - c\beta_1 \frac{k^{3/2}}{4k^3}\right)^{\tau\beta_1 k^{3/2}} \leq \left(\frac{1}{e}\right)^{c\tau\beta_1^2/4}.$$

We select  $\tau$  such that  $\left(\frac{1}{e}\right)^{c\tau\beta_1^2/4} = \gamma$ . Then the probability that some node  $u$  in  $B$  selects some node  $u'$  in  $B'$  in an interval of  $\tau$  rounds is at least  $1 - \gamma$ . In other words, assuming that  $u$  has the message at the end of  $\tau g(d^{3/4})$  rounds, the probability that some  $u' \in B'$  receives the message in the  $\tau$  is at least  $1 - \gamma$ .

By induction hypothesis,  $u$  receives the message from  $x$  with probability  $1 - \gamma g(d^{3/4})$  in the first  $\tau g(d^{3/4})$  rounds. And  $y$  receives the message from  $u'$  with probability  $1 - \gamma g(d^{3/4})$  in another  $\tau g(d^{3/4})$  rounds after  $u'$ . Thus, the probability that in  $\tau g(d)$  rounds the message propagates from  $x$  to  $y$  is at least  $1 - \gamma - 2\gamma g(d^{3/4}) = 1 - \gamma g(d)$ .

Since  $\gamma = O(\log^{-2.4 - \frac{\log d}{\log \log d}} d)$  and  $\tau = O(\frac{-1}{\beta_1^2 c} \log \gamma)$ , we have  $\tau = O(\log d)$ .

Therefore, in  $\tau g(d) = O(\log^{3.4} d)$  rounds the message travels from  $x$  to  $y$  with a probability of  $1 - O(\frac{1}{d})$ .  $\square$

## **Part II**

# **Virtual Coordinates: Modifying Network Geometries**



# Chapter 5

## Introduction

This second part handles sensor networks through virtual coordinates. Instead of relying on the default geometric properties of the graph metric, the approach will be to modify the metric to suit our needs. By modifying the metric, we simplify the shape of the network to have useful geometric properties. Of course, the shape or the metric of a network itself cannot be modified without extensive re-wiring or physically moving the nodes. The method therefore is to use artificial *virtual* locations unrelated to a node's physical placement, and modify that at will. We apply virtual coordinates to perform easier routing, and for in-network storage.

Ricci flow is a powerful technique in differential geometry. It can be used to deform the metric and shape of a surface. We use it for our modification of the network in virtual coordinates. The flow acts locally at every node as an iterative algorithm, interleaved with communication with neighbors. In this respect, its pattern of operation is very much like a distributed gossip algorithm. As in the first part, routing is a reference application. Along with introducing Ricci flow, we demonstrate its utility in directly enabling greedy routing, which is perhaps the simplest possible routing scheme.

Greedy routing has received a lot of attention since it was proposed for routing in ad hoc wireless networks [13,77], due to its simplicity and efficiency. A node forwards the packet to its neighbor whose distance to the destination is the smallest. Thus routing decision is made with only local knowledge. On a dense network without holes greedy routing typically works very well and gives close to optimal routing paths.

A well-known problem with geographical forwarding is that packets may get stuck at nodes with no neighbor closer to the destination. There have been many ways to handle this problem, the most well-known one is face routing [13,77,92]. In this paper we take the approach of constructing a map of the network and finding *virtual coordinates* for the sensor nodes such that simple greedy routing with virtual coordinates always succeeds.

The property of Ricci flow that makes greedy routing possible is that the flow causes the network holes to converge to circular holes in the virtual coordinate. As will be seen in the following chapter, this suffices for successful greedy routing.

In chapter 7 we extend the virtual coordinate system for better routing and data

storage. The data storage question can be seen as a map from the data space to the physical network space - each piece of data is stored somewhere in the network. We store the data on a *covering space* of the network. The covering space can be imagined as the virtual coordinate system of chapter 6 from which the holes have been practically removed. This provides better load balancing in routing and data storage. In general, the practical absence of holes goes further in simplifying geometry of the network, and methods conceived for a flat uniform plane are more readily adaptable to a space without holes.

The mapping from the logical data space to the physical sensor field, done in a straightforward manner, often leads to high data concentration on nodes near network boundaries, simply for the reason that they are adjacent to empty or low density regions. In GHT, for example, all the data hashed inside a hole will eventually be mapped to the nodes on the hole boundary, that get a higher storage load. When GHT exploits the nodes on the boundary of a planar face to also store the data (for robustness to node failures), the load imbalance is even higher. Similarly for DIM, the node near a zone empty of nodes will substitute to store the data. Nodes near hole boundaries store more data and carry more traffic.

When the sensor field is irregular, many geometry based data storage and retrieval schemes run into problems. As another example, double rulings, or quorum based schemes, store data on a curve and retrieve data along another curve. As long as the data retrieval curves intersect with the data storage curve, one can successfully discover the desired data. When a sensor field has a regular shape (a square region or a disk, for example), one can design the storage/retrieval curves as the horizontal/vertical lines [104,145,159], or proper circles (great circles through a stereographic mapping) [132]. Both of them may get stuck at network boundaries. Of course one can use various hole bypassing techniques to get around the holes [42,77]. This may also lead to higher storage and traffic load on the hole boundaries – the same problem encountered earlier.

The imbalance of storage or traffic load adversely affects the system performance. On one hand, the nodes with high load are bottleneck nodes. They carry out more tasks than average. If the nodes are battery powered, this means the highly loaded nodes would run out of battery sooner. When these heavily used nodes are on hole boundaries the problem is worse, as holes are enlarged and the network may be disconnected prematurely. In addition, the nodes with high traffic load denote the bottleneck of communication. Spatial diversity is not best utilized to avoid wireless interference, leading to lower network throughput.

## 5.1 Research review

Using virtual coordinates for greedy routing was first used in NoGeo routing proposed by Rao *et al.* [128]. In this method the network boundary is pinned on a convex planar curve (such as a square or a circle) and the interior nodes are embedded by using the rubberband representation [102]. The rubberband representation is obtained by each node (that is not fixed) running an iterative algorithm of putting

itself at the center of mass of the neighbors' current locations until convergence.

Motivated by NoGeo, a few theoretical works ask under what conditions embedding of a given graph in the plane exists such that greedy routing always works [21, 120]. Such an embedding is called a *greedy embedding*. It is known that not every graph admits a greedy embedding, for example a star with 7 leaves [120]. Some graphs are known to have greedy embeddings, for example, any graph with a Hamiltonian path, any complete graph, any 4-connected planar graph (since they are Hamiltonian [147]), and Delaunay triangulations. It still remains open to fully characterize the class of graphs that admit greedy embeddings.

Papadimitriou and Ratajczak [120] made the conjecture that any planar 3-connected graph<sup>1</sup> has a greedy embedding in the plane. Dhandapani discovered that any planar triangulation (without holes) admits a greedy embedding in the plane for which greedy forwarding always succeeds [32]. Recently the 3-connected graph conjecture was proved to be true by Leighton and Moitra [96], and independently by Angelini *et al.* [7]. Later the algorithm in [96] was improved such that the coordinates use  $O(\log n)$  bits for a graph with  $n$  vertices [59].

A recent observation by Kleinberg [88] shows that if we use hyperbolic space then greedy routing becomes easy. He showed that any connected graph has an embedding in the hyperbolic space such that by using the hyperbolic distance greedy routing from any node to any node always succeeds. The intuition is to embed a tree in a hyperbolic space such that greedy routing works on the tree. Since any connected graph has a spanning tree, greedy routing works at all times. A similar idea was used in [38].

For all the virtual coordinates schemes, one needs to have a location service such that any node can inquire the virtual coordinate of any other node in the network. Efficient location services for sensor networks have been developed [98, 129]. Such location services can be used in routing with virtual coordinates developed in this paper. The virtual coordinate addresses in this case, as in [128], are simple euclidean coordinates of the form  $(x, y)$ . The service simply needs to have tables of such coordinates for the nodes.

Prior research on sensor networks have proposed the 'data-centric' notion [73, 129] for sensor network design. The generation, collection, processing, storage and retrieval of sensor data are the most critical functions around which the network protocols should be designed. As the state of the art, networks in the size of thousands of sensor nodes are deployed [1, 112] with the target size of hundreds of thousands in the next few years. As networks grow large in size, centralized data collection has a fundamental bottleneck at nodes near the sink. Distributed in-network data storage in which there is no single sink is more desirable for its robustness.

For distributed in-network storage, data is mapped to rendezvous sensors for storage and processing. Such a mapping is often obtained by considering data in a logical space with indices mapped to geographical locations. For example, in geographical hash table (GHT) [129], data keys are hashed to a random geographical location in the sensor field and the sensor node closest to the hashed location

---

<sup>1</sup>A graph is 3-connected if it remains connected after the removal of any 2 nodes.



is denoted as the home node and stores the data. In DIM [99], data in a multi-dimensional attribute space is mapped to the sensor field by using a quad-tree, such that data with nearby indices are mapped to physically nearby zones, in order to support range queries. Variations of quadtrees have also been used in other schemes to organize data and the corresponding storage [52,56,62].

# Chapter 6

## Ricci Flow and Greedy Routing

Greedy forwarding with geographical locations in a wireless sensor network may fail at a local minimum. In this chapter we propose to use *conformal mapping* to compute a new embedding of the sensor nodes in the plane such that greedy forwarding with the virtual coordinates guarantees delivery. In particular, we extract a planar triangulation of the sensor network with non-triangular faces as holes, by either using the nodes' location or using a landmark-based scheme without node location. The conformal map is computed with *Ricci flow* such that all the non-triangular faces are mapped to perfect circles. Thus greedy forwarding will never get stuck at an intermediate node. The computation of the conformal map and the virtual coordinates is performed at a preprocessing phase and can be implemented by local gossip-style computation. The method applies to both unit disk graph models and quasi-unit disk graph models. Simulation results are presented for these scenarios.

### 6.1 Introduction

To find virtual coordinates for the sensor nodes in a network, we look at a more fundamental problem of studying maps between *spaces*. This is motivated by the fact that most practical applications of sensor networks require sufficient sensor density, for both sensing coverage and system robustness/redundancy to cope with node failure. Taking the view point of maps of spaces also introduces some insensitivity to link dynamics and local disturbances. In a wireless network, communication links are volatile and may go up and down. Nodes may also die or be replaced periodically. The shape of the geometric region for which the sensors are deployed and aim to monitor, is much more stable, provided that the sensor network still has sufficient coverage for its normal functioning.

The question we ask is, given a domain surface  $\mathcal{R} \subseteq \mathbb{R}^2$ , is there a continuous map  $f : \mathcal{R} \rightarrow \mathcal{D}$  such that greedy routing on  $\mathcal{D}$  always succeeds? If the domain  $\mathcal{R}$  is a simply connected convex region (i.e., with no holes), then the identity map is good, since greedy routing is essentially straight line routing in  $\mathcal{R}$  and always successfully delivers a message. If the domain  $\mathcal{R}$  has holes, especially some concave ones, then we need a non-trivial map to prevent greedy routing from getting stuck at hole

boundaries. In fact, we would like to map the domain  $\mathcal{R}$  to a domain  $D$  such that all the holes in  $D$  are circular — since greedy routing never gets stuck at circular hole boundaries. Thus the map that achieves this will produce virtual coordinate system for the nodes with guaranteed delivery for greedy routing.

The map we will use is a *conformal map*, as shown in Figure 6.1. A conformal map between two surfaces preserves angles. For any two arbitrary curves  $\gamma_1, \gamma_2$  on the surface  $S$ , a conformal map  $\phi$  maps them to  $\phi(\gamma_1), \phi(\gamma_2)$  with the same intersection angle as that of  $\gamma_1, \gamma_2$ . According to conformal geometry theory, a genus zero surface with multiple boundaries (a topological multi-holed annulus) can be conformally mapped to the unit disk with circular holes, as shown in Figure 6.1. Such a conformal mapping is unique up to a Möbius transformation in each homotopy class of degree one mappings. Recent advances in differential geometry, in particular, on *Ricci flow* lead to computationally efficient algorithms to construct such kind of conformal mapping.

Ricci flow was introduced by Richard Hamilton for Riemannian manifolds of any dimension in his seminal work [65] in 1982. Intuitively, a surface Ricci flow is the process to deform the Riemannian metric of the surface. The deformation is proportional to Gaussian curvatures, such that the curvature evolves like the heat diffusion. It has been considered a powerful tool for finding a Riemannian metric satisfying the prescribed Gaussian curvature in mathematics and has been applied in the proof of the Poincaré conjecture on 3-manifolds [122–124]. Chow and Luo [24] proved a general existence and convergence theorem for the discrete Ricci flow on surfaces, and proved that the Ricci energy is convex. Jin *et al.* provided a computer algorithm in [74].

### 6.1.1 Our contribution

We investigate in this paper algorithms for computing a conformal map of a sensor field and the application in enabling greedy routing. We first extract from the communication network a planar graph  $H$  such that all non-triangular faces map to network holes that will be later mapped to circular holes in the embedded domain  $D$ . Ideally these holes in  $H$  are also real holes in the network/environment. Thus the triangulated mesh  $H$  is a discrete representation and approximation of the underlying domain  $\mathcal{R}$ . We show that when the sensors are deployed densely in  $\mathcal{R}$  such that any disk with diameter 1 has at least one sensor inside, the unit disk graph on the nodes contains such a triangulation  $H$  of  $\mathcal{R}$  that can be computed locally. Note that the density requirement here is simply a condition for detecting the topology faithfully and is not necessary for successful routing. We present a method to construct a suitable triangulation from any unit disk graph. Similar results can be proved as well for certain quasi-unit disk graphs (when there must be a link between two nodes within distance  $1/\alpha < 1$  and no link when distance is greater than 1). When node locations are not known, we use a landmark-based scheme as in [43, 49] to locally select landmarks of constant bounded density (i.e., the Voronoi diagram of each landmark has  $O(1)$  nodes) and compute a planar triangulation on the landmarks. The conformal map is computed on this triangulation. The planar

triangulation algorithm for a sensor network may be of interest by itself since many surface processing algorithms assume a nice triangular mesh and now can be applied in the network setting.

The triangulation and conformal map computation are performed as a preprocessing phase during network setup. There are two major advantages of using Ricci flow method for computing the virtual coordinates: distributed nature and conformality.

1. Ricci flow algorithm is intrinsically distributed, each node only requires the information from its one-hop neighbors, and the curvature at that node can be calculated, the metric deformation is proportional to the curvature, therefore the flow can be performed completely in parallel.

2. Asymptotically, Ricci flow leads to a conformal mapping. We set the virtual edge lengths of the triangulation to be one, therefore each triangle is an equilateral one in the original triangulation.

The virtual coordinates are disseminated to every node with which they can use greedy routing for point-to-point message delivery. In our simulation section we demonstrate the efficiency of conformal map computation, in terms of the number of messages used per node, different network topologies and node density.

On a last note, we remark that the rubberband representation used in NoGeo algorithm [128] is essentially the discrete version of finding a harmonic map between the simply connected domain defined by the sensor field outer boundary and a convex planar domain. **However, harmonic maps for multi-holed annulus can not be guaranteed to be a diffeomorphism.** Therefore holes in the network are not handled properly. In particular, near a non-convex hole, the network may be folded over itself, causing the routing to fail. Our algorithm can be considered as an extension of Dhandapani's result that any triangulation (without holes) admits a greedy embedding. Our method considers triangulation of a domain with possible holes and converges to an embedding of the network with holes mapped to circles as long as such embedding exists [68]. The existence of embedding can be verified by the combinatorics of the network as explained in [5]. An embedding can always be ensured by appropriate local refinements to the triangulation. For conventional methods, it is more challenging to handle dense networks. Our method is especially good at handling dense networks. In fact, our discrete mapping converges to smooth conformal mappings with the increase of density. The proof can be found in [15].

In the next section we will briefly introduce the theory behind the Ricci flow algorithm to compute a conformal map. Readers may also choose to read the algorithm section first in which we introduce the implementation of the algorithm in a network setting and the entire pipeline for computing virtual coordinates for greedy routing.



(a) Face surface (b) Conformal Mapping (c) Möbius transformed (d) Texture Mapping

**Figure 6.1. Conformal Mapping.** Genus zero surface (a) with multiple boundaries can be mapped to the unit disk with circular holes conformally. Two such conformal mappings differ by a Möbius transformation are shown in (b) and (c). The checker board image is applied for texture mapping in (d), where all the right angles of the checkers are well preserved, therefore, the mapping is conformal.

## 6.2 Theory

In this section, we introduce several important concepts in differential geometry and the theory of the Ricci flow.

### 6.2.1 Riemannian metric and curvature

Suppose a surface  $S$  is embedded in  $\mathbb{R}^3$ , then it has a Riemannian metric, induced from the Euclidean metric of  $\mathbb{R}^3$ . The metric tensor is denoted by  $\mathbf{g} = (g_{ij})$ .

Riemannian metric determines the Gaussian curvature  $K$  and the geodesic curvature  $k$ . Gauss-Bonnet theorem states that the total curvature is a topological invariant

$$\int_S K dA + \int_{\partial S} k ds = 2\pi\chi(S), \quad (6.1)$$

where  $\partial S$  represents the boundary of  $S$ ,  $\chi(S)$  is the Euler characteristic number of  $S$ .

Suppose  $u : S \rightarrow \mathbb{R}$  is a scalar function defined on  $S$ , then it can be verified that  $e^{2u}\mathbf{g}$  is another Riemannian metric on  $S$ , denoted by  $\bar{\mathbf{g}}$ . It can be proven that angles measured by  $\mathbf{g}$  are equal to those measured by  $\bar{\mathbf{g}}$ . Therefore,  $\bar{\mathbf{g}}$  is *conformal* to  $\mathbf{g}$  and now  $e^{2u}$  is called the conformal factor. The Gaussian curvatures are related by

$$\bar{K} = e^{-2u}(-\Delta u + K), \quad (6.2)$$

where  $\Delta$  is the Laplace-Beltrami operator under the original metric  $\mathbf{g}$ . Similarly, the geodesic curvatures satisfy

$$\bar{k} = e^{-u}(\partial_{\mathbf{n}}u + k), \quad (6.3)$$

where  $\mathbf{n}$  is the tangent vector orthogonal to the boundary. According to Gauss-Bonnet theorem (equation 6.1), the total curvature doesn't change.

### 6.2.2 Surface Ricci flow

Suppose  $S$  is a smooth surface with Riemannian metric  $\mathbf{g}$ . The Ricci flow is the process to deform the metric  $\mathbf{g}(t)$  according to its induced Gaussian curvature  $K(t)$ ,

where  $t$  is the time parameter

$$\frac{dg_{ij}(t)}{dt} = -2K(t)g_{ij}(t). \quad (6.4)$$

Suppose  $T(t)$  is a temperature field on the surface. The heat diffusion equation is  $dT(t)/dt = -\Delta T(t)$ , where  $\Delta$  is the Laplace-Beltrami operator induced by the surface metric. The temperature field becomes more and more uniform with the increase of  $t$ , and it will become constant eventually.

In a sense, the curvature evolution induced by the Ricci flow is exactly the same as heat diffusion on the surface, as follows:

$$\frac{K(t)}{dt} = -\Delta_{\mathbf{g}(t)}K(t), \quad (6.5)$$

where  $\Delta_{\mathbf{g}(t)}$  is the Laplace-Beltrami operator induced by the metric  $\mathbf{g}(t)$ . We can simplify the Ricci flow equation 6.4. Let  $g(t) = e^{2u(t)}g(0)$ , then Ricci flow is

$$\frac{du(t)}{dt} = -2K(t). \quad (6.6)$$

The following theorems postulate that the Ricci flow defined in 6.4 is convergent and leads to the conformal uniformization metric.

**Theorem 6.2.1 (Hamilton 1982).** *For a closed surface of non-positive Euler characteristic, if the total area of the surface is preserved during the flow, the Ricci flow will converge to a metric such that the Gaussian curvature is constant everywhere.*

**Theorem 6.2.2 (Chow [23]).** *For a closed surface of positive Euler characteristic, if the total area of the surface is preserved during the flow, the Ricci flow will converge to a metric such that the Gaussian curvature is constant everywhere.*

The corresponding metric  $\mathbf{g}(\infty)$  is the *uniformization metric*. Moreover, at any time  $t$ , the metric  $\mathbf{g}(t)$  is conformal to the original metric  $\mathbf{g}(0)$ .

The Ricci flow can be easily modified to compute a metric with a *prescribed* curvature  $\bar{K}$ , and then the flow becomes

$$\frac{dg_{ij}(t)}{dt} = 2(\bar{K} - K)g_{ij}(t). \quad (6.7)$$

With this modification, any target curvatures  $\bar{K}$ , which are admissible with the Gauss-Bonnet theorem, can be induced from the solution metric  $\mathbf{g}(\infty)$ .

### 6.2.3 Discrete Ricci flow

Smooth surfaces are often approximated by simplicial complexes (triangle meshes). We consider such a triangle mesh  $\Sigma$  with vertex set  $V$ , edge set  $E$  and face set  $F$ .

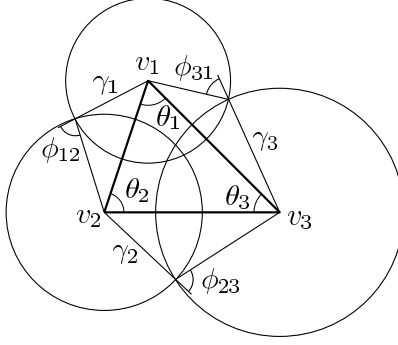


Figure 6.2. The circle packing metric.

**Discrete Riemannian Metric** In the discrete setting, the edge lengths on a mesh  $\Sigma$  simply define the Riemannian metric on  $\Sigma$ ,

$$l : E \rightarrow \mathbb{R}^+,$$

such that for a face  $f_{ijk}$  the edge lengths satisfy the triangle inequality:  $l_{ij} + l_{jk} > l_{ki}$ .

The discrete metric determines the angles. Suppose we have a triangle  $f_{ijk}$  with edge lengths  $\{l_{ij}, l_{jk}, l_{ki}\}$ , and the angles against the corresponding edges are  $\{\theta_k, \theta_i, \theta_j\}$  (see figure 6.2). By the cosine law,

$$l_{ij}^2 = l_{jk}^2 + l_{ki}^2 - 2l_{jk}l_{ki} \cos \theta_k, \quad (6.8)$$

The discrete Gaussian curvature is defined as the angle deficit on a mesh,

$$K_i = \begin{cases} 2\pi - \sum_{f_{ijk} \in F} \theta_i^{jk}, & \text{interior vertex} \\ \pi - \sum_{f_{ijk} \in F} \theta_i^{jk}, & \text{boundary vertex} \end{cases} \quad (6.9)$$

where  $\theta_i^{jk}$  represents the corner angle attached to vertex  $v_i$  in the face  $f_{ijk}$ .

In the discrete setting, the Gauss-Bonnet theorem (equation 6.1) still holds on meshes with the discrete Gaussian curvatures, as follows.

$$\sum_{v_i \in V} K_i = 2\pi\chi(M).$$

The circle packing metric was introduced [144,150] to approximate the conformal deformation of metrics. Let us denote by  $\Gamma$  a function which assigns a radius  $\gamma_i$  to each vertex  $v_i$ .

$$\Gamma : V \rightarrow \mathbb{R}^+$$

We also define a *weight* function:

$$\Phi : E \rightarrow [0, \frac{\pi}{2}].$$



by assigning a positive number  $\Phi(e_{ij})$  to each edge  $e_{ij}$ . The pair of vertex radii and edge weight functions on a mesh  $\Sigma$ ,  $(\Gamma, \Phi)$ , is called a *circle packing metric* of  $\Sigma$ . Figure 6.2 illustrates the circle packing metric. Each vertex  $v_i$  has a circle whose radius is  $r_i$ . On each edge  $e_{ij}$ , an intersection angle  $\phi_{ij}$  is defined by two circles of  $v_i$  and  $v_j$ , which intersect with or are tangent to each other. Two circle packing metrics  $(\Gamma_1, \Phi_1)$  and  $(\Gamma_2, \Phi_2)$  on a same mesh are *conformal equivalent*, if  $\Phi_1 \equiv \Phi_2$ . Therefore, a conformal deformation of a circle packing metric only modifies the vertex radii.

For a given mesh, its circle packing metric and the edge lengths on the mesh can be converted to each other by using cosine law.

$$l_{ij}^2 = \gamma_i^2 + \gamma_j^2 + 2\gamma_i\gamma_j \cos \phi_{ij} \quad (6.10)$$

Let  $u_i$  to be  $\log \gamma_i$  for each vertex. Then, the discrete Ricci flow is defined as follows.

$$\frac{du_i(t)}{dt} = (\bar{K}_i - K_i) \quad (6.11)$$

Discrete Ricci flow can be formulated in the variational setting, namely, it is a negative gradient flow of some special energy form.

$$f(\mathbf{u}) = \int_{\mathbf{u}_0}^{\mathbf{u}} \sum_{i=1}^n (\bar{K}_i - K_i) du_i, \quad (6.12)$$

where  $\mathbf{u}_0$  is an arbitrary initial metric. The integration above is well defined, and called the *Ricci energy*. The discrete Ricci flow is the negative gradient flow of the discrete Ricci energy. The discrete metric which induces  $\bar{k}$  is the minimizer of the energy.

Computing desired metric with prescribed curvature  $\bar{K}$  is equivalent to minimizing the discrete Ricci energy. The discrete Ricci energy is strictly convex (namely, its Hessian is positive definite). The global minimum uniquely exists, corresponding to the metric  $\bar{\mathbf{u}}$ , which induces  $\bar{\mathbf{k}}$ . The discrete Ricci flow converges to this global minimum [24].

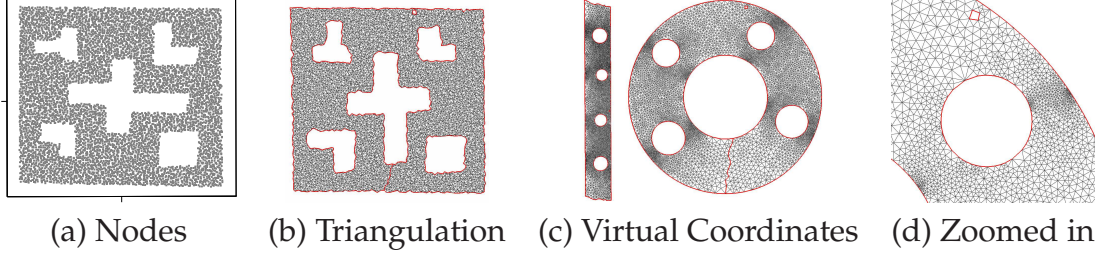
**Theorem 6.2.3 (Chow & Luo: Euclidean Ricci Energy).** *The Euclidean Ricci energy  $f(\mathbf{u})$  on the space of normalized metric  $\sum u_i = 0$  is strictly convex.*

The convergence rate of the discrete Ricci flow using equation 6.11 is governed by the following theorem

**Theorem 6.2.4 (Chow & Luo).** *The Ricci flow 6.11 converges exponentially fast,*

$$|\bar{K}_i - K_i(t)| < c_1 e^{-c_2 t}, \quad (6.13)$$

where  $c_1, c_2$  are two positive constants.



**Figure 6.3. Algorithm pipeline.** Given a network in (a), a triangulation is constructed in (b). The boundaries are traced as  $\gamma_k$ 's in (b). A cut between  $\gamma_1$  and  $\gamma_2$  is located as  $\eta$ . The triangular mesh is sliced open along  $\eta$ . By using Ricci flow the sliced mesh is flattened to a parallelogram in (c), then mapped to the unit disk with circular holes in (c). Local region is zoomed in as shown in (d), which shows all the triangles are with acute angles.

## 6.3 Algorithms

This section describes the distributed algorithm to compute the virtual coordinates corresponding to the conformal map. The first step is to obtain a triangulation from the network that is a compact manifold with boundaries and is homeomorphic to a bounded subset of  $\mathbb{R}^2$ . Later, we describe the algorithm to compute the conformal map of this triangulation.

### 6.3.1 Obtain a triangulation

We describe methods for obtaining a triangulation in cases where the nodes are aware of their locations as well as in cases where locations are not available. The methods apply to quasi-unit disk graphs with suitable parameter. For unit disk graphs in the plane, the only requirement is that the graph should be connected. We denote the set of neighbors of a node  $u$  as  $N(u)$ .

#### Location-based triangulation

A method for computing a triangulation of a unit disk graph through local computations is described in detail in [57]. This graph is called the *restricted delaunay graph* (RDG). The essential method is as follows:

1. At each node  $u$ , compute the delaunay triangulation of  $N(u) \cup \{u\}$ , denote this triangulation as  $T(u)$ .
2. For each node  $u$ , if an edge  $(u, v)$  is in  $T(u)$ , then it is valid if and only if  $(u, v) \in T(x)$ , for all common neighbors  $x$  of  $u, v$ , i.e.  $\forall x \in (N(u) \cup \{u\}) \cap (N(v) \cup \{v\})$ .
3. All invalid edges are removed.

It is proved in [57] that RDG is connected and planar. The essential reason being that for a pair of crossing edges of a unit disk graph, at least one of the four nodes

that lie at the end points of the edges is aware of both the edges, and hence can force the crossing to be removed. Our goal here is to obtain virtual coordinates for routing, but we would like to approximate the true topology of the network as closely as possible. The following theorem suggests that at least for dense sensor networks, the method above preserves the topology:

**Lemma 6.3.1.** *If a bounded set  $\mathcal{R} \subseteq \mathbb{R}^2$  is covered by sensors  $P$  such that any disk with unit diameter centered inside  $\mathcal{R}$  has at least one node inside, any non-triangular face in the RDG computed above is of distance at most  $1/2$  away from the boundary of  $\mathcal{R}$ .*

**Proof:** Denote by  $D(P)$  the Delaunay triangulation on  $P$  and  $D'(P)$  the graph with all the Delaunay edges on  $P$  with lengths smaller than 1. It has been shown in [57] that the restricted Delaunay graph  $RDG$  computed is a planar graph that includes  $D'(P)$ . Therefore, any non-triangular face in the  $RDG$  must map to a non-triangular face in  $D'(P)$ . In the following we argue that there can not be a non-triangular face ‘deeply inside’ the region  $\mathcal{R}$ . Consider any Delaunay triangle  $\Delta uvw$  in  $D(P)$ , if its circumcircle  $C$  is centered inside  $\mathcal{R}$ , then the circumcircle has radius at most  $1/2$  — otherwise the sensor density requirement will put one node inside  $C$ , which contradicts with the property that the circumcircle of a Delaunay triangle is empty of any other nodes. This shows that all the three Delaunay edges of  $\Delta uvw$  have length at most 1 and are then inside  $D'(P)$ . Therefore any Delaunay triangle  $\Delta uvw$  not in  $D'(P)$  must have its circumcircle centered outside  $\mathcal{R}$ . The circumcircle has radius greater than  $1/2$  — otherwise the three edges are no longer than 1 hence the triangle  $\Delta uvw$  is in  $D'(P)$ . Now we argue that all three nodes  $u, v, w$  are within distance  $1/2$  from the boundary of  $\mathcal{R}$ . Otherwise, we can shrink the circumcircle  $C$  of  $\Delta uvw$  to a unit diameter circle, completely inside  $C$ , with its center within  $\mathcal{R}$ . By the density requirement there must be a node inside the shrunk circle, thus inside  $C$ . This again contradicts the Delaunay triangulation property.  $\square$

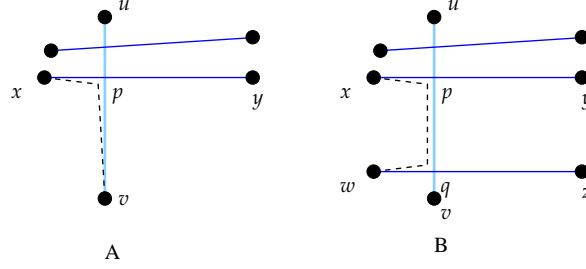
The above Lemma shows that the  $RDG$  we get indeed provides a triangular mesh that covers the region  $\mathcal{R}' = \{p | p \in \mathcal{R}, d(p, \partial\mathcal{R}) \leq 1/2\}$ , which does not include the points within  $1/2$  distance from the boundary  $\partial\mathcal{R}$ .

This distributed algorithm can be adapted to produce a planarization algorithm for connected quasi unit disk graphs (quasi-UDG)<sup>1</sup> of parameter  $\alpha \leq \sqrt{2}$ . This is done as follows:

1. Compute the  $RDG$  with  $1/\alpha$  sized disks for neighborhoods instead of unit disk neighborhoods. The  $RDG$  algorithm applies without modification, hence produces a planar graph. The planar graph produced may not be connected. It is possible that connectivity of the quasi-UDG relied on some edges longer than  $1/\alpha$ .

---

<sup>1</sup>In a *quasi unit disk graph* with parameter  $\alpha \geq 1$ , if two nodes are within distance  $1/\alpha$ , an edge between the two exists, if they are at a distance more than 1, the edge does not exist; while for other distances, the existence of the edge is uncertain.



**Figure 6.4.** Restoring connectivity. Dark edges are RDG edges, light blue edge is a quasi-UDG edge not in RDG. Dotted lines are virtual edges. (A) Crossing edge belonging to connected component of  $u$  (B) Crossing edges belonging to different connected components.

## 2. Restore connectivity using edges of quasi-UDG.

The following method shows that connectivity can be restored without sacrificing planarity.

**Restoring connectivity.** First, observe that any two nodes that are within distance  $1/\alpha$  must be in the same connected component of the RDG. Therefore, if RDG has more than one connected component, then somewhere there must be a quasi-UDG edge  $(u, v)$  longer than  $1/\alpha$  where  $u$  and  $v$  belong to different components. If no edge of the current RDG crosses  $(u, v)$ , we can simply add the the edge, and connect  $u$  and  $v$ .

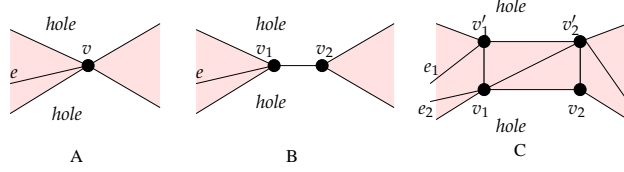
If an existing edge  $(x, y)$  in RDG crosses  $(u, v)$  at point  $p$ , then one of the nodes  $x, y$  must be within a distance  $1/\alpha$  to one of the nodes  $u, v$ , and therefore neighbors in quasi-UDG [90]. Without loss of generality, we assume that  $u$  and  $x$  are within distance  $1/\alpha$ . Then these nodes are in the same connected component of RDG. Note that by this argument, for any RDG edge crossing  $(u, v)$  its endpoints must belong to the connected component of  $u$  or  $v$ .

Now, consider the scenario of 6.4(A), where the crossing edges belong to the connected component of  $u$ . If there are more than one such edges, we consider the edge whose intersection with  $(u, v)$  is nearest to  $v$ . Observe that by construction, no edge crossing  $(x, y)$  can exist in RDG. To restore connectivity, we insert the edge  $(x, v)$ . This can be done without compromising planarity, since no RDG edge intersects  $(p, v)$  or  $(x, p)$ , we can lay down an edge that is infinitesimally close to the path  $(x, p, v)$ , that does not intersect any RDG edge. Note that we only need a combinatorial planar graph and do not require a straight-line planar embedding under the original coordinates.

If another crossing edge such as  $(w, z)$  exists, in the connected component of  $v$ , then we similarly choose to insert the virtual edge  $(x, w)$  via the corresponding path  $(x, p, q, w)$ .

These virtual edges do not correspond to real quasi-UDG edges, communication between their endpoints are achieved in the network by routing through the quasi-UDG edge  $(u, v)$ .

**Orientation and degeneracy removal.** We need an oriented planar triangulation to



**Figure 6.5.** Handling degeneracy. (A) Holes sharing a boundary vertex, (B) Holes sharing a boundary edge and (C) triangulation using virtual nodes.

obtain a manifold on which the conformal map can be applied. Therefore, we start with detecting all the triangles ( $K_3$ ) in  $\mathcal{G}$ . This is done simply by gathering 2-hop information at each node. Based on this, we find *boundary* edges:

**Definition 6.3.2. Boundary Edge.** *Any edge that does not belong to exactly 2 different triangles.*

The boundary cycles can now be determined by traversing connected components of boundary edges. The boundary cycles bound the *holes* in the network.

Then we start with any triangle in  $\mathcal{G}$ , and assign an arbitrary orientation to it. This determines orientations of all faces of  $\mathcal{G}$ , and are computed distributedly as follows:

1. Once a triangle is oriented, any triangle adjacent to it can compute its own orientation, by orienting the shared edge in the opposite direction.
2. Once an edge on a particular boundary cycle has been identified, that determines the orientation of the hole, and is propagated along the boundary cycle.

Observe that given any vertex, the orientation of edges and faces incident on it, determine a cyclic (clockwise or counterclockwise) order of incident edges.

The planar graph consists of two types of oriented faces - triangles and holes. For our purposes, it is necessary that the union of the triangles form a 2-manifold. At this point, however, the planar graph may not contain such a triangulation. In particular, it is possible that an edge may belong to the boundary cycle of two different holes - see Figure 6.5(B).

We handle such a case by creating triangulation of virtual nodes (Figure 6.5(C)). First, the degenerate edge  $(v_1, v_2)$  is copied to a new edge  $(v'_1, v'_2)$ . Then we add in edges  $(v_1, v'_1)$ ,  $(v_2, v'_2)$  and diagonal  $(v_1, v'_2)$ . Note that to maintain a triangulation, one of the edges incident on each of  $v_1$  and  $v_2$  must be duplicated. For example, in Figure 6.5(C), the edge  $e$  has been duplicated. Each other edge incident on an original vertex is assigned to one of the copies, the assignment being uniquely determined by the cyclic order defined by the orientation, and the choice of the duplicated edge. Orientation of all new triangles and edges are also determined uniquely by the orientation of existing faces.

A degeneracy of the form of Figure 6.5(A) is also possible, where two holes share a common boundary vertex. In this case, we duplicate the degenerate vertex  $v$  into  $v_1, v_2$  and connect by an edge. Then we repeat the method for the previous case.

Note that we need not assign coordinates to the new virtual nodes, since we are working purely on the graph structure, and our mapping algorithm does not require coordinates.

This method extends to chains of degenerate edges, dangling edges and to multiple holes sharing a vertex. We omit the details at this point, and skip ahead to the next topic.

### Landmark-based triangulation

In a network where location information is not available, we have to rely only on the hop-count distance metric to obtain a triangulation. This is achieved by means of the landmark Voronoi diagram. From the landmarks, we flood messages, that measure distance of other nodes from the landmarks. This creates a set of Voronoi cells in the graph. The adjacency of these cells give rise to a dual *combinatorial delaunay complex* (CDC).

Landmark based Voronoi and delaunay graphs have been used to do routing in [43,49].

We follow the approach of [49,50], to select landmarks. The idea is to choose landmarks such that:

- Any two landmarks are  $k$  hops apart (for a small  $k = 5$  or  $6$ ).
- Any non-landmark node is within  $k$  hops of some landmark.

This method requires a flood from a landmark to last only  $k$  hops, hence the overall cost is linear for a network of bounded density. This produces a dense set of landmarks. But the CDC obtained from the adjacency is not planar. The following method for obtaining a planar graph from the CDC is described in detail in [50]:

*An edge of CDC is valid if there is a path between the corresponding landmarks  $a$  and  $b$  such that no node on the path has a neighbor that belongs to the Voronoi cell of any landmark other than  $a$  and  $b$ . The graph formed by the set of valid edges is called the Combinatorial delaunay map (CDM).*

This method is proved to produce a planar graph from a quasi unit disk graph ( $\alpha \leq \sqrt{2}$ ). At this point we can apply the methods from the previous discussion to obtain a triangulation. While this landmark based method is more natural and intuitive for large scale networks of high density, it applies in principle to any network corresponding to a suitable quasi unit disk graph. The theory suggests that in certain cases it may be necessary to refine the triangulation further. However, this was not necessary in any of the networks we have tried.

### 6.3.2 Other triangulation methods

The algorithms above both require a quasi-UDG model and thus does not work when a sensor network does not follow the quasi-UDG assumption. The following algorithms compute planar graphs without such assumptions.



Kim *et al.* [61,82] addressed the problem that planarization techniques using relative neighborhood graph or Gabriel graph fail when the communication model does not comply to the unit disk graph assumption. They developed a cross link detection protocol to probe each link, detect and remove possible crossings with other links. The resulting graph is combinatorially planar.

Zhang *et al.* [162] developed a location-free algorithm to extract a planar sub-graph from the connectivity graph. The main idea is to planarize adjacent layers of a shortest path tree. Again this method does not require a unit disk graph model or quasi-UDG model.

All these algorithms above can be used in our method. In our implementation, we have used both the restricted Delaunay graph approach and the landmark based triangulation approach [50]. But all the other schemes can also work well with our framework. In the worst case when a triangulation is not available, for example, when crossing edges are introduced, the result of the Ricci flow algorithm is theoretically unpredictable.

### 6.3.3 Computing the conformal map

Figure 6.3 shows the algorithm pipeline for computing the conformal mapping.

**Algorithm description** The conformal mapping can be achieved by using the discrete Ricci flow algorithm. The algorithm only requires the connectivity information. The locations of nodes are irrelevant, and all edges are assumed to be of length 1.

**Discrete Ricci flow** Each node  $v_i$  is associated with a disk, with radius  $e^{u_i}$ . For simplicity, the length of each link connecting  $v_i$  and  $v_j$  equals to  $e^{u_i} + e^{u_j}$ . The corner angles of each triangle can be estimated using cosine law by each node locally. The curvature can be computed by each node directly. Then  $u_i$  is modified proportionally to the difference between the target curvature and the current curvature. Once the curvature error is less than a given threshold, the process stops. The details can be found in the algorithm 1.

**Flattening** The virtual coordinates can be estimated by flattening triangle by triangle using the resulting metric (edge length) from the Ricci flow algorithm. First, the root face is chosen. Given three edge lengths of the root triangle  $[v_0, v_1, v_2]$ , the node coordinates can be constructed directly. Then the neighboring triangle of the root, e.g.  $[v_1, v_0, v_i]$ , can be flattened, the virtual coordinates of  $v_i$  is the intersection of two circles, one is centered at  $p_0$  with radius  $l_{0i}$ , the other is centered at  $p_1$  with radius  $l_{1i}$ . Furthermore, the normal of the triangle  $[v_1, v_0, v_i]$  is consistent with the root triangle. In similar way, the neighbors of the newly flattened triangles can be further embedded. The whole network can be flattened by the flooding process. The details can be found in the algorithm 2.



---

**Algorithm 1** Discrete Ricci Flow

---

**Require:** Triangular Mesh  $M$ , Target curvature for each vertex  $\bar{k}_i$ . Error threshold  $\epsilon$ .  
Step length  $\delta$ .

**Ensure:** Discrete metric (edge lengths) satisfying the target curvature.

1: for all vertex  $v_i$ ,  $u_i \Leftarrow 0$

2: **while** true **do**

3:   Compute edge length  $l_{ij}$  for edge  $[v_i, v_j]$ ,

$$l_{ij} = e^{u_i} + e^{u_j}$$

4:   Compute the corner angle  $\theta_i^{jk}$  in triangle  $[v_i, v_j, v_k]$ ,

$$\theta_i^{jk} = \cos^{-1} \frac{l_{ij}^2 + l_{ki}^2 - l_{jk}^2}{2l_{ij}l_{ki}}$$

5:   Compute the curvature  $k_i$  at  $v_i$

$$k_i = \begin{cases} 2\pi - \sum_{jk} \theta_i^{jk}, & v_i \notin \partial M \\ \pi - \sum_{jk} \theta_i^{jk}, & v_i \in \partial M \end{cases}$$

6:   **if**  $\max |\bar{k}_i - k_i| < \epsilon$  **then**

7:     **return** The discrete metric  $\{l_{ij}\}$ .

8:   **end if**

9:   Update  $u_i$

$$u_i \Leftarrow u_i + \delta(\bar{k}_i - k_i)$$

10: **end while**

---

**Conformal Mapping** Given a triangular mesh  $M$ , which is a multi-holed annulus (genus zero surface with multiple boundaries), using above algorithm it can be mapped to a canonical unit disk with circular holes. Furthermore, the mapping is an approximation of a conformal mapping. In smooth case, the conformal mapping is unique upto a Möbius transformation as shown in Figure 6.1. In the following algorithm, the Möbius ambiguity is removed.

First, the boundary loops of  $M$  are traced, and sorted by their lengths decreasingly using hop distance, denoted as  $\gamma_k$ , where  $k$  is from 1 to  $n$ , as shown in Figure 6.3 frame (b). Second, the target curvature is set, such that all interior nodes have zero curvatures, nodes on  $\gamma_1$  and  $\gamma_2$  are of zero curvatures also. Nodes on  $\gamma_k$ ,  $k > 2$  has the target curvature  $\frac{-2\pi}{|\gamma_k|}$ , where  $|\gamma_k|$  denotes the length of  $\gamma_k$ . The edge lengths satisfying the target curvatures are computed using the Ricci flow algorithm.

---

**Algorithm 2** Flattening

---

**Require:** Triangular Mesh  $M$ , Discrete metric  $\{l_{ij}\}$  with zero curvatures on all interior vertices.

**Ensure:** Virtual coordinates for each vertex.

- 1: for each node  $v_i$ , label  $v_i$  as *un-accessed*
- 2: flatten the first triangle  $[v_0, v_1, v_2]$ , such that

$$p_0 \leftarrow (0, 0), p_1 \leftarrow (l_{01}, 0), p_2 \leftarrow l_{20}(\cos \theta_0^{12}, \sin \theta_0^{12}),$$

label  $v_0, v_1, v_2$  as *accessed*.

- 3: for each un-accessed node  $v_i$ , check all its neighboring faces  $[v_i, v_j, v_k]$ , if  $v_j, v_k$  have been accessed, then  $p_i$  is the intersection point of two circles

$$|p_i - p_j| = l_{ij}, |p_i - p_k| = l_{ki},$$

furthermore  $(p_j - p_i) \times (p_k - p_i) > 0$ . Label  $v_i$  as *accessed*.

---

Third, a shortest path  $\eta$  from  $\gamma_1$  to  $\gamma_2$  is traced, suppose  $\eta$  intersects  $\gamma_1$  and  $\gamma_2$  at  $v_1, v_2$  respectively.  $M$  is sliced along  $\eta$  to form another mesh  $\tilde{M}$ ,  $v_k$  is split to  $v_k^1, v_k^2 \in \tilde{M}, k = 1, 2$ . The edge lengths are copied from  $M$  to  $\tilde{M}$ ,  $\tilde{M}$  is flattened to the plane. The virtual coordinates of  $v_k^1, v_k^2, k = 1, 2$  form a parallelogram. Without loss of generality,  $v_1^1, v_1^2$  are mapped to the  $y$ -axis and their distance is  $h$ , as shown in Figure 6.3 frame (c). Then by the following conformal map  $e^{\frac{2\pi z}{h}}$ ,  $\tilde{M}$  is mapped to the canonical unit disk with circular holes, as shown in Figure 6.3 frame (c). Then the virtual coordinates of nodes are copied from  $\tilde{M}$  to  $M$ . This algorithm guarantees to map  $\gamma_1$  and  $\gamma_2$  to concentric circles, and the only ambiguity left is a rotation. Detailed description can be found in algorithm 3.

In practice, in order to make the inner holes more circular, the target curvatures of  $v_j \in \gamma_k, k > 2$  can be updated and more iterations of Ricci flow algorithm are performed. The target curvatures can be updated using the following formula

$$\bar{k}_j = -2\pi \frac{l_{j-1} + l_j}{\sum_{e_i \in \gamma_k} l_i}$$

where  $l_j$  and  $l_{j-1}$  are the current edge lengths of edges adjacent to  $v_j$ ,  $l_i$  is the current edge length of  $e_i$  which is in the boundary  $\gamma_k$ . In general, 4 or 5 iterations are good enough. Figure 6.3 frame (c) and (d) show the computational result of this algorithm. In (c), all the boundaries become circular, in (d), all the triangle corners are acute, the triangulation is with good quality.

---

**Algorithm 3** Conformal Mapping

---

**Require:** Triangular Mesh  $M$ , genus zero with multiple holes.

**Ensure:** Virtual coordinates  $p_j$  for each node  $v_j$ , all the boundaries are circular.

- 1: locate all the boundaries  $\gamma_1, \gamma_2, \dots, \gamma_n$ , sorted increasingly according to their lengths using the hop distance.
- 2: set target curvature,

$$\bar{k}_i = \begin{cases} 0 & v_i \notin \partial M \\ 0 & v_i \in \gamma_1 \cup \gamma_2 \\ \frac{-2\pi}{|\gamma_k|} & v_i \in \gamma_k, k > 2 \end{cases}$$

- 3: Compute the metric using Ricci flow algorithm 1.
- 4: compute the shortest cut from  $\gamma_1$  to  $\gamma_2$ , denoted as  $\eta$ ,  $\eta$  intersects  $\gamma_1, \gamma_2$  at  $v_1, v_2$  respectively.
- 5: slice  $M$  along  $\eta$  to get another mesh  $\tilde{M}$ ,  $v_k$  is split to two nodes  $v_k^1, v_k^2$ , where  $k = 1, 2$ .
- 6: compute the virtual coordinates of  $\tilde{M}$  using algorithm 2, such that  $p_1^1, p_1^2, p_2^1, p_2^2$  form a parallelogram.  $p_1^1, p_1^2$  are along  $y$ -axis. The distance between them is  $h$ .
- 7: for each node  $v_j \in M$ , there exists a corresponding node  $\tilde{v}_j \in \tilde{M}$ ,  $p_j = (x_j, y_j)$ ,

$$p_j \leftarrow e^{\frac{2\pi}{h}(x_j + iy_j)}$$

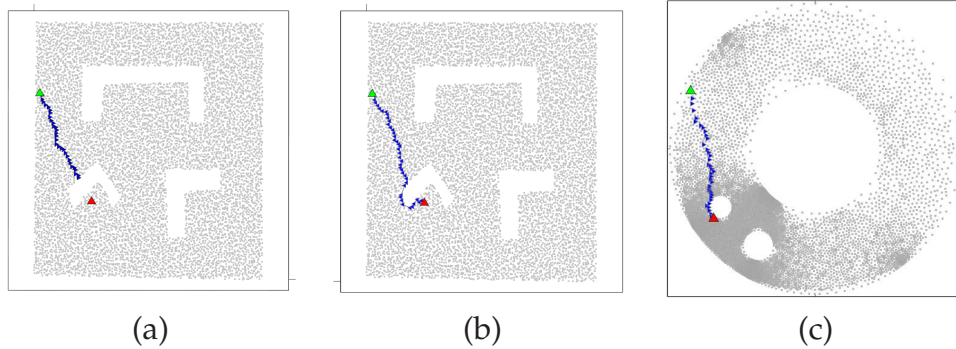
---

**Robustness** The accuracy of the computation is mainly controlled by the curvature error bound  $\epsilon$  in the Ricci flow algorithm 1. According to theorem 6.2.4, the curvature error decreases exponentially fast. Therefore, the number of steps to reach the desired error bound is given by  $O(-\frac{\log \epsilon}{\delta})$ , where  $\delta$  is the step size in the Ricci flow algorithm. In practice, if the number of triangles in the network is about tens of thousands and the error bound is about  $1e - 8$ , the algorithm is stable.

### 6.3.4 Routing

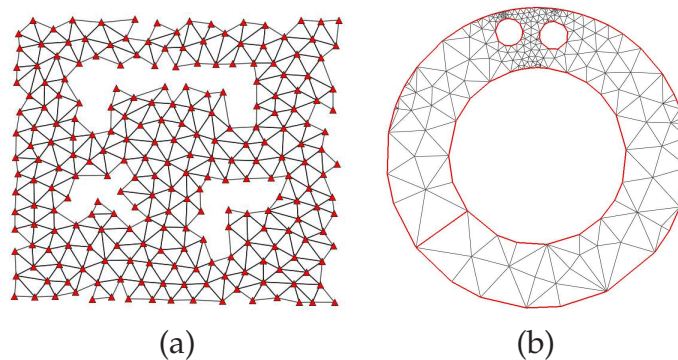
Having obtained the virtual coordinates, greedy routing is straight forward. As mentioned earlier, a message cannot get stuck on the boundary of a circular hole. It has been shown in [42] that greedy routing cannot get stuck at vertices with an angle less than  $2\pi/3$ . In the mappings we obtained, all angles were acute. However, in a case of a large angled triangle appearing in the final embedding, the routing can be handled by routing on *edges* as follows. Suppose in  $\triangle ABC$  the angle  $\angle BAC > 2\pi/3$ , and the current routing request to destination  $D$  arrives at  $A$ , and has no nearer neighbor. We can create a virtual node  $E$  (representing the edge  $BC$ ) and the message is delivered from  $A$  to  $BC$  (or  $E$ ) in the sense that the edge  $BC$  is closer to the destination. As all the non-hole faces are triangles, the triangle adjacent to  $\triangle ABC$  sharing the edge  $BC$  must have another edge closer to the destination. Thus greedy routing will guarantee delivery.

Figure 6.6 shows the effect of greedy routing on the virtual coordinates. The routing cannot be successful under normal greedy routing (Figure 6.6(a)). The path under the virtual coordinates, however, easily gets past the hole to the other side (Figure 6.6(b)).



**Figure 6.6.** Routing. Network of about 8700 nodes, average degree of about 20 in quasi-UDG setting, and nodes in a perturbed grid distribution. (a) Geedy routing gets stuck at a hole boundary. (b) Routing based on virtual coordinates successfully goes around the hole. (c) The routing path in the virtual coordinate space.

The domain can also be triangulated using landmarks as described in section 6.3.1. Figure 6.7 shows the CDM triangulation of the domain of Figure 6.6, and the corresponding virtual coordinates.



**Figure 6.7.** (a) Landmark based triangulation of domain of fig 6.6. (b) The corresponding virtual coordinate map.

Routing in the landmark based scheme is achieved in the usual way. At every stage, the next Voronoi tile to visit is decided based on the virtual coordinates of the landmarks of neighboring tiles. Then a local routing scheme is applied to reach the chosen neighboring tile. This local routing can be executed in different ways. For example, since the size of the tiles are constant in a bounded density network, it is possible to store a routing table for the entire tile. Alternatively, it is possible to flood a tile from the boundary with each neighbor, and thus obtain paths to neighboring tiles from each node.

In theory, and in practice, the method also works for very fragmented networks with many holes. Figure 6.8 shows an example.

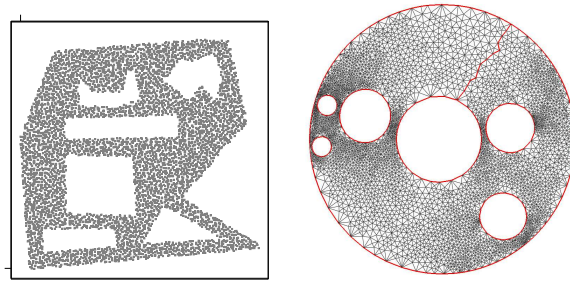


Figure 6.8. (a) Network of 7000 nodes with many holes (b) Virtual coordinates

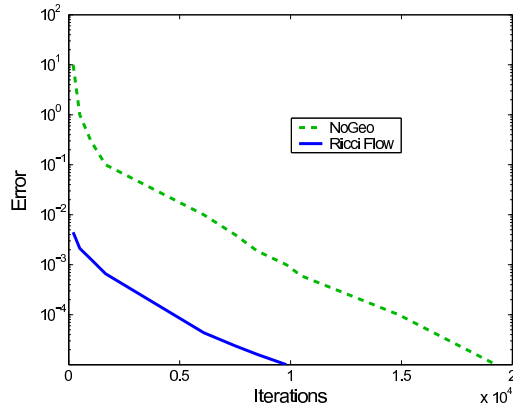
## 6.4 Experimental results

We conducted extensive experimentation on UDG or quasi-UDG based network of Figure 6.6 with about 8700 nodes, and on networks of similar topology but different number of nodes. From a routing performance point of view, the following are important observations from the experiments and simulations:

- **100% Routing guarantee.** We selected 10,000 random source-destination pairs in the network, and performed greedy routing based on the real coordinates and using our virtual coordinates. With the real coordinates, the success rate of routing is only about 52.29%, while with the virtual coordinate greedy routing, we achieve 100% success rate.
- **Small routing stretch.** The path length of the virtual coordinate routing was compared with the shortest path in the graph for 5000 random source-destination pairs. The average stretch (ratio of routing path length to shortest path) was 1.59, while the maximum stretch was 3.21.

Since our mapping algorithm uses a numerical method, we carried out some tests to estimate the convergence time of the algorithm, and compared the results with the convergence of NoGeo [128]. While NoGeo does not guarantee delivery even on full convergence, the comparison is interesting, as described in the introduction. The results are shown in Figure 6.9. Note that NoGeo iterates on the actual node coordinates, whereas the Ricci-flow reduces the error in the curvature. The result shows that in this case, the Ricci-flow method converges faster than NoGeo, and guarantees delivery.

We further compared our method with NoGeo on routing stretch and delivery guarantee on the network of Figure 6.6, over 5000 source-destination pairs. The results are shown in table 6.1. Our algorithm incurs a larger stretch (ratio of routing path length to shortest path length) than NoGeo, but guarantees delivery.



**Figure 6.9.** The solid blue curve shows the error in curvature after a number of iterations by the Ricci flow method, the dashed green curve shows the error in location after a number of iterations by NoGeo.

**Table 6.1.** Stretch and Delivery Comparison

Method	Delivery rate	Avg Stretch	Max Stretch
Our Method	100%	1.59	3.21
NoGeo	83.66%*	1.17	1.54

\*NoGeo performs extremely well in simply connected networks and networks with convex holes, as shown in [128]. But as discussed earlier, in this case the presence of concave holes and holes of large aspect ratio affects its performance adversely.

The algorithm was executed on networks of several different sizes but of same essential topology. We measured the number of iterations required to obtain a small enough error on curvature to get a successful embedding and routing. The resulting plot is shown in Figure 6.10. The curvature error bound was selected as  $1e - 6$ .

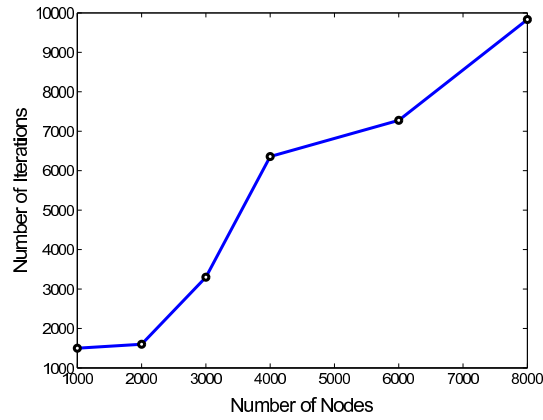
Table 6.2 lists some statistics about the triangulations of different networks shown in the chapter.

**Table 6.2.** Experimental Statistics

Case	Nodes	Faces	Edges	Holes
Graph of Figure 6.7	250	378	630	3
Network of Figure 6.6	8714	17091	25807	3
Network of Figure 6.3	5299	10179	15482	6

## 6.5 Conclusion

This work proposes a novel method for greedy routing with guaranteed delivery based on Ricci flow algorithm. The method has solid theoretic background and



**Figure 6.10.** The number of iterations required to obtain a given error bound on the curvature.

competitive performance, compared with prior algorithms under the metric of pre-processing cost and routing quality. The distributed nature of the Ricci flow method makes it valuable for the practical applications on wireless sensor networks. Network dynamics such as node failures and resultant topology changes can be handled efficiently by incrementally recomputing the map starting from the previously computed one. Analysis and experiments related to this aspect are under investigation.



# Chapter 7

## Data Storage on Virtual Coordinates

For in-network storage schemes, one maps data, indexed in a logical space, to the distributed sensor locations. When the physical sensor network has an irregular shape and possibly holes, the mapping of data to sensors often creates unbalanced storage load with high data concentration on nodes near network boundaries. In this chapter we propose to map data to a *covering space*, which is a tiling of the plane with copies of the sensor network, such that the sensors receive uniform storage load and traffic. We propose distributed algorithms to construct the covering space with Ricci flow and Möbius transforms. The use of the covering space improves the performance of many in-network storage and retrieval schemes such as geographical hash tables (GHTs), and the double rulings (or quorum based schemes), and provides better load balanced routing.

### 7.1 Introduction

We propose to solve the imbalance of storage and traffic load in an irregular sensor network by ‘uniformizing’ the sensor field shape. As the logical data space is often regular, we make the sensor field regular as well — irregular shape is turned into circular, and holes are filled up. We propose to create a *covering space* of the sensor network, which is a tiling of the space with transformed copies of the network itself. Data hashed to a geographical location inside a hole is actually mapped to another copy of the sensor field. Similarly, with a regular shape, previously proposed double rulings scheme can be applied to irregular network with almost zero modification. Thus our *network regulation* technique provides a generic solution for data storage problems in an irregular network, and greatly extends the application scope of existing schemes. See Figure 7.3 for an example of the original sensor field and the covering space.

We achieve this by using Ricci flow and reflections in boundary circles. We first extract a triangulation of the sensor network. And then apply Ricci flow to make all holes circular. The new idea in this chapter is to use the embedding obtained from the Ricci flow algorithm and *fill up the holes*. Suppose the network has  $k$  (circular) holes. For each interior hole  $C_i$ , we take a Möbius transform that essentially ‘re-

flects' the network inward with respect to  $C_i$ . This Möbius transform is conformal and maps circles to circles. Thus,  $C_i$  becomes the outer boundary of the reflected network with all the nodes mapped inside it. This partially fills up the hole  $C_i$ , except that there are  $k$  smaller circular holes. Now we can continue such transforms so that all the holes are eventually filled up, with infinitely many transformed copies of the original sensor field. The collection of Möbius transforms used to generate these mappings is captured in the *Schottky group*. Thus, one does not need to precalculate any of these mapping and is able to generate the reflections on the fly when necessary.

The generation of the covering space as described above asks for infinitely many transformed copies to completely cover the space. We show that for any practical applications only  $O(\log 1/\varepsilon)$  copies are necessary, where  $\varepsilon$  is the threshold of the size of a hole. Indeed, we prove that the total area of the holes shrinks by a fraction after each Möbius transform, and is reduced exponentially fast. When the holes are tiny, the chance that data is hashed to be inside a hole is very small and can be omitted. Similarly, the chance that a double ruling curve hits the boundary of a tiny hole is negligible. When it does happen, we can get around the hole by following the greedy routes along the circular hole boundaries. In our simulations only 5 reflections are necessary and for some applications 2 levels of reflections give the best result.

With the regulation of the network shape by conformal Möbius transforms, we can improve the performance of various data storage schemes.

- *GHT*. When a piece of data is hashed to a geographical location  $p$  inside a hole, in the original GHT scheme, it is allocated to the sensor node whose Voronoi cell contains  $p$ . Nodes on the boundary have larger Voronoi cells and share higher load. With the covering space, the area inside the hole is shared by the *entire* network, eliminating fundamentally the storage and traffic overhead on the holes boundaries.
- *Double rulings*. Double rulings design can be directly applied on the covering space. When a curve hits a hole boundary  $C_i$ , it then enters another copy of the network mapped to the interior of  $C_i$ . Equivalently, in the original embedding, the curve 'reflects' on the hole boundary. The intersection properties are still maintained with the conformal Möbius transforms.
- *Load balanced greedy routing*. The embedding generated by the Ricci flow algorithm allows greedy routing to work with delivery guarantee, as greedy routing can not get stuck at circular holes. However, such greedy routes still tend to hug the hole tightly causing high traffic load on the boundary nodes. Instead, we can execute the greedy routing in the covering space. Instead of getting around the hole by following the circular hole boundary, one can 'enter' the hole to route in another copy of the network, effectively reflect on the hole boundary. Thus the boundary nodes are not used as often, improving the load balancing. The greedy routing can be used in combination with the GHT scheme to deliver and retrieve data from the hashed location.

In summary, the covering space universally improves the load imbalance in data storage and routing in an irregular network. Essentially, in the covering space the holes are filled up so there are no ‘boundaries’. The nodes on the boundary are now treated in the same way as the other nodes with respect to data storage or relay routing.

In the following of the chapter we first present the mathematics of the conformal Möbius transform, the Scottky group, and the covering space. We then present the use of the covering space in applications such as GHT, double rulings and greedy routing with simulation results.

## 7.2 Theoretical background

This section focuses on the theoretic background of Möbius transformation and reflections to generate the covering space. We refer readers to [34] and [126] for further details.

### 7.2.1 Conformal mapping for multiply connected domain

Let  $(S_1, g_1)$  and  $(S_2, g_2)$  be two surfaces with Riemannian metrics  $g_1, g_2$ . A mapping  $\phi : S_1 \rightarrow S_2$  is called a *conformal map (angle preserving map)*, if the intersection angle of any two curves are preserved.

A planar domain  $D$  of connectivity  $n$  is called a circular domain, if all its  $n$  boundaries are circles. It is known from conformal geometry that any genus zero multiply connected planar domain can be mapped to a circular domain by conformal maps. The different circular mappings of a given planar domain differ by Möbius transforms [31, 126].

One way to compute the conformal mapping from a surface to a circular domain is to use Ricci flow, as introduced in [74, 130]. Given a multiply connected domain  $\Omega$  with  $m$  interior holes, denoted as  $\partial\Omega = \{\gamma_1, \gamma_2 \cdots, \gamma_m\}$ , by Ricci flow, we can construct a conformal map  $\phi : \Omega \rightarrow \mathbb{C} \cup \{\infty\}$  to a circle domain, such that each  $\phi(\gamma_j)$  is a circle,

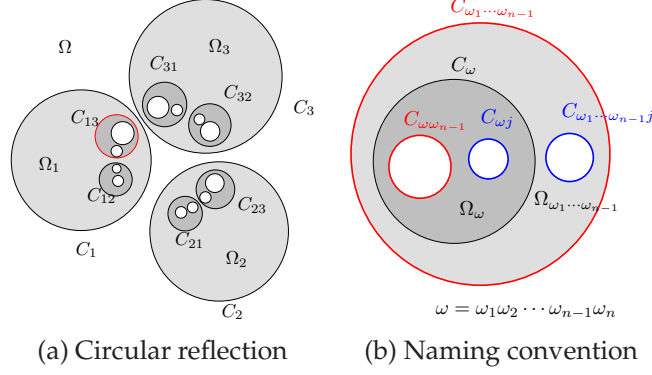
$$\partial[\phi(\Omega)] = \{C_1, C_2, \dots, C_m\}, C_j = \{z : |z - \mathbf{c}_j| = r_j\}.$$

where  $j = 1, 2 \cdots, m$ . Examples can be found in Figure 7.3, 7.8 and 7.9.

### 7.2.2 Möbius transform and circular reflection

A *Möbius transformation* is a map that maps a complex plane to itself, represented by  $f(z) = \frac{az+b}{cz+d}$ , where  $a, b, c, d$  are four complex numbers satisfying  $ad - bc = 1$ . A Möbius transformation is a conformal map and maps circles to circles. A special case of Möbius transformation

$$\rho_C(z) := \mathbf{c} + \frac{r^2}{\bar{z} - \bar{\mathbf{c}}}. \quad (7.1)$$



**Figure 7.1.** Circular Reflection and naming convention.

is a *circular reflection* that maps the points inside a circle  $C$  with center  $c$  and radius  $r$  to the points outside  $C$ , and vice versa. For a circular domain  $\Omega$  with interior circular holes  $C_1, C_2, \dots, C_m$ , we denote  $\rho_{C_j}$  by  $\rho_j$  for  $C = C_j$ .  $\rho_j$  essentially fills up the hole  $C_j$  by reflecting the points out of  $C_j$ . We use such circular reflections to fill up the holes in a sensor network.

**An example.** Figure 7.1 (a) shows an example of a triply connected circular domain  $\Omega$  with boundary  $\partial\Omega = \{C_1, C_2, C_3\}$ . Reflect  $\Omega$  through  $C_k$  to get

$$\Omega_k = \rho_k(\Omega).$$

The circle  $C_j$  is reflected by  $\rho_i$  to be circle  $C_{ij}$ ,

$$C_{ij} = \rho_i(C_j), i \neq j.$$

$\Omega_k$  is a bounded domain with outer boundary  $C_k$  and 2 circular inner boundaries. The boundary of  $\Omega_k$ 's in Figure 7.1 (a) are  $\partial\Omega_1 = \{C_1, C_{12}, C_{13}\}$ ,  $\partial\Omega_2 = \{C_2, C_{21}, C_{23}\}$ ,  $\partial\Omega_3 = \{C_3, C_{31}, C_{32}\}$ .

Now  $\Omega_1$  has two small holes  $C_{12}$  and  $C_{13}$ . We reflect  $\Omega_1$  with respect to each of the interior hole. Thus we have the reflected domains in the next level. In general,

$$\Omega_{ij} = \rho_{ij}(\Omega_i), 1 \leq i, j \leq 3, i \neq j.$$

The new boundary circles are

$$C_{121} = \rho_{12}(C_1), C_{123} = \rho_{12}(C_{13}), \partial\Omega_{12} = \{C_{12}, C_{121}, C_{123}\}$$

$$C_{131} = \rho_{13}(C_1), C_{132} = \rho_{13}(C_{12}), \partial\Omega_{13} = \{C_{13}, C_{131}, C_{132}\}$$

The boundaries of  $\Omega_{21}, \Omega_{23}, \Omega_{31}$  and  $\Omega_{32}$  are similar.

In general, for a circular domain with  $m$  interior holes, the reflected regions and circles are labeled with multi-indices

$$\omega = \omega_1 \omega_2 \dots \omega_q, 1 \leq \omega_j \leq m, \omega_k \neq \omega_{k+1}, 1 \leq k \leq q-1.$$

A reflected domain is defined by the reflections following the indices.

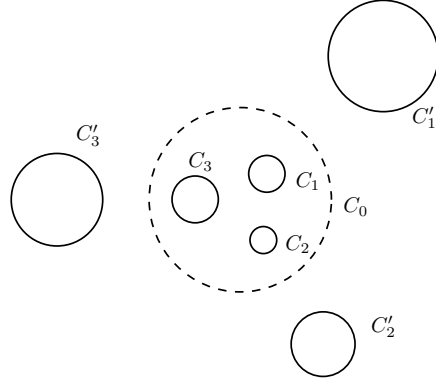


Figure 7.2. Schottky Group generators.

**Definition 7.2.1.** The set of multi-indices of length  $q$  ( $q > 0$ ) is denoted

$$\sigma_q = \{\omega_1\omega_2\cdots\omega_q : 1 \leq \omega_j \leq m, \omega_k \neq \omega_{k+1}, 1 \leq k \leq q-1\},$$

and  $\omega_0 = \emptyset$ .

As shown in Figure 7.1 (b), if  $\omega \in \sigma_q, q > 1$ , the circular domain

$$\Omega_\omega = \rho_\omega(\Omega_{\omega_1\omega_2\cdots\omega_{q-1}})$$

has exterior boundary  $C_\omega$  and  $m-1$  interior boundary circles

$$C_{\omega\omega_{q-1}} = \rho_\omega(C_{\omega_1\omega_2\cdots\omega_{q-1}}),$$

and

$$C_{\omega j} = \rho_\omega(C_{\omega_1\omega_2\cdots\omega_{q-1}j}), j \neq \omega_{q-1}, \omega_q.$$

There are  $m(m-1)^{q-1}$  elements in  $\sigma_q$ , at the level  $q$ , and  $m(m-1)^{q-1}$  circles.

### 7.2.3 Schottky groups

Taking reflections of a circular domain with respect to the circular holes to the limit will eventually have all the holes 'filled up'. This is shown by using the Schottky groups, as described below.

Suppose  $C_j$  is the circle  $|z - \mathbf{c}_j| = r_j^2$ ,  $C_0$  is the unit circle  $|z| = 1$ , denote  $\rho_0(C_j)$  as  $C'_j$  (see Figure 7.2). The Möbius map

$$\theta_j(z) = \mathbf{c}_j + \frac{r_j^2 z}{1 - \bar{\mathbf{c}}_j z}$$

maps the exterior of  $C'_j$  to the interior of  $C_j$  (and  $C'_j$  to  $C_j$ ).

The Schottky group  $\Theta$  is defined to be the infinite free group of Möbius mappings generated by compositions of the  $2m$  basic Möbius maps  $\{\theta_j | j = 1, \dots, m\}$  and their

inverses  $\{\theta_j^{-1} | j = 1, \dots, m\}$ . Consider the unbounded region  $\Omega$  of the plane exterior to the  $2m$  circles  $\{C_j | j = 1, \dots, m\}$  and  $\{C'_j | j = 1, \dots, m\}$ . The union of copies of  $\Omega$  generated by the  $\theta \in \Theta$  is denoted as

$$\Theta(\Omega) := \bigcup_{\theta \in \Theta} \theta(\Omega).$$

This work is based on the following fundamental theorem of Schottky groups.

**Theorem 7.2.2.** *The complement set of  $\Theta(\Omega)$*

$$\Theta(\Omega)^c := \mathbb{C} - \Theta(\Omega)$$

*is a Cantor set of zero measure.*

The detailed discussion can be found in [34], [126] and [31].

$\theta_j$  can be generated by two circular reflections: first the exterior of  $C'_j$  is reflected through the unit circle  $C_0$ , then the interior of  $C_0$  is reflected through  $C_j$ . The composition of these two reflections is  $\theta_j$ . In this work, we use circular reflections instead of explicitly using Schottky group.

## 7.2.4 Shrinkage estimation

Asymptotically the whole plane can be covered by the copies of the network using Schottky transformation. But in practice, only a finite number of reflections can be used. Therefore, we need a precise estimation of the size of holes after  $n$  levels of reflections. In the following, we give the estimation of the area shrinkage of the holes. We follow the method in [126] and [31].

As shown in figure 7.4,  $\Omega$  is a bounded double connected domain on the complex plane  $\mathbb{C}$ , with exterior boundary  $\Gamma_0$  and interior boundary  $\Gamma_1$ ,  $\partial\Omega = \Gamma_0 - \Gamma_1$ . There exists a conformal map  $\phi : \Omega \rightarrow \mathbb{D}$ , where  $\mathbb{D}$  is a circular domain, with inner radius  $\mu_{01}$  and outer radius 1,

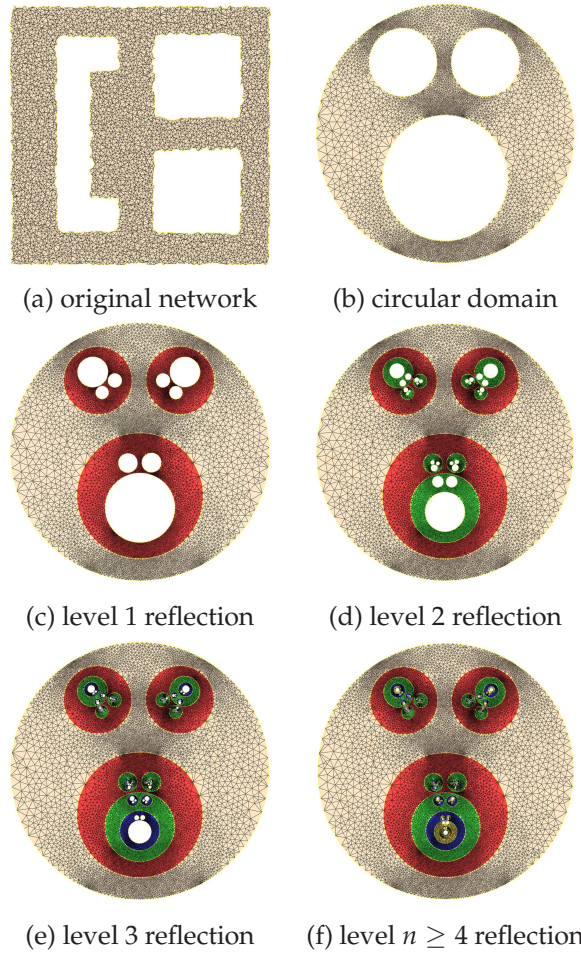
$$\mathbb{D} = \{z \in \mathbb{C} : \mu_{01} < |z| < 1, \}.$$

We call  $\mu_{01}$  the conformal modulus of the original domain  $\Omega$ .

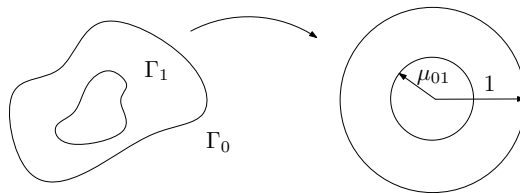
**Definition 7.2.3.** *The separation modulus for two circles  $C_j, C_k$  is defined as*

$$\tilde{\mu}_{jk} := \frac{\gamma_j + \gamma_k}{d_{jk}} < 1, j \neq k, 1 \leq j, k \leq m, \quad (7.2)$$

where  $\gamma_j$  and  $\gamma_k$  are the radii of  $C_j, C_k$  respectively, and  $d_{jk}$  is the distance between the centers of  $C_j, C_k$ .

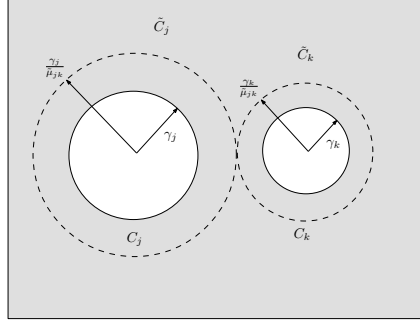


**Figure 7.3.** 4-level circular reflections for a 3-hole sensor network with 5492 nodes. The initial network (a) is conformally mapped to the circular domain (b). The level 1 reflection is in red color in (c), level 2 reflection is in green in (d), level 3 reflection is in blue in (e), level 4 reflection is in yellow in (e).



**Figure 7.4.** Conformal modulus  $\mu_{01}$  of a doubly connected domain.





**Figure 7.5.** Separation modulus of a doubly connected circular domain.

The separation modulus of the region is given by

$$\Delta := \max_{i,j,i \neq j} \tilde{\mu}_{ij}.$$

As shown in Figure 7.5, suppose  $\tilde{C}_j$  is the circle with the center  $\mathbf{c}_j$  and radius  $\frac{\gamma_j}{\Delta}$ , then  $\frac{1}{\Delta}$  is the smallest magnification of the  $m$  circles, such that at least two  $\tilde{C}_j$ 's just touch.

The following lemma shows that the separation modulus is bounded by conformal modulus, the proof can be found in the Appendix.

**Lemma 7.2.4.** *The conformal modulus is the lower bound of the separation modulus:*

$$\mu_{jk} < (\tilde{\mu}_{jk})^2 \leq \Delta^2.$$

**Proof:** Let  $\mu$  denote  $\mu_{jk}$  and  $\tilde{\mu}$  denote  $\tilde{\mu}_{jk}$ . The conformal module can be calculated directly for circular domain as

$$\mu = \frac{d_{jk}^2 - \gamma_j^2 - \gamma_k^2 - \sqrt{[(d_{jk} - \gamma_j)^2 - \gamma_k^2][(d_{jk} + \gamma_j)^2 - \gamma_k^2]}}{2\gamma_j\gamma_k}.$$

Then

$$\alpha = \frac{1}{2} \left( \mu + \frac{1}{\mu} \right) = \frac{d^2 - (\gamma_j^2 + \gamma_k^2)}{2\gamma_j\gamma_k}$$

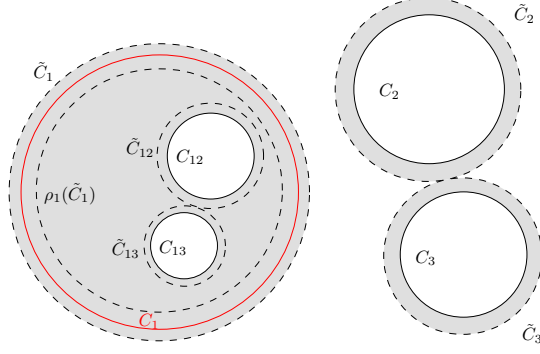
$$(\tilde{\mu}d)^2 = (\gamma_j + \gamma_k)^2,$$

It follows

$$\alpha \tilde{\mu}^2 - 1 = (1 - \tilde{\mu}^2) \frac{\gamma_j^2 + \gamma_k^2}{2\gamma_j\gamma_k} \geq 1 - \tilde{\mu}^2.$$

Then

$$\frac{1}{\tilde{\mu}^2} \leq \frac{\alpha + 1}{2} = \left[ \frac{1}{2} \left( \sqrt{\mu} + \frac{1}{\sqrt{\mu}} \right) \right]^2.$$



**Figure 7.6.** Area Shrinkage.

Because  $\mu < 1$ ,

$$\sqrt{\mu} < \frac{2\sqrt{\mu}}{\mu + 1} \leq \tilde{\mu} \leq \Delta.$$

□

**Theorem 7.2.5.** *At level  $q + 1$ , the total area of holes is*

$$\sum_{\omega \in \sigma_{q+1}} S(\tilde{C}_\omega) \leq \Delta^{4q} \sum_{i=1}^m S(\tilde{C}_i), \quad (7.3)$$

where  $S(C_i)$  is the area inside the circle  $C_i$ .

The proof depends on the following lemma. As shown in Figure 7.4,  $\Omega$  is a bounded double connected domain on the complex plane  $\mathbb{C}$ , with exterior boundary  $\Gamma_0$  and interior boundary  $\Gamma_1$ ,  $\mu_{01}$  is the conformal modulus.

**Lemma 7.2.6.** *Suppose  $S(\Gamma_k)$  is the area bounded by  $\Gamma_k$ ,  $k = 0, 1$ , then*

$$S(\Gamma_1) \leq \mu_{01}^2 S(\Gamma_0).$$

Detailed proof can be found in [126], Lemma 17.7c(a), P.503. The proof for theorem 7.2.5 is as follows:

**Proof:** As shown in Figure 7.6.  $\Omega$  has three boundary circles  $C_1, C_2, C_3$ . Magnify each circle by factor  $\frac{1}{\Delta}$ , we get (dashed) circles  $\tilde{C}_1, \tilde{C}_2, \tilde{C}_3$ . By definition of separation modulus,  $\tilde{C}_1, \tilde{C}_2, \tilde{C}_3$  may touch, but have no overlaps. Then  $\tilde{C}_2, \tilde{C}_3$  are in the exterior of  $\tilde{C}_1$ .

Reflect  $\tilde{C}_j$  through circle  $C_1$  (the red circle). Denote

$$\tilde{C}_{ij} := \rho_i(\tilde{C}_j), i \neq j, 1 \leq i, j \leq 3.$$

Then  $\tilde{C}_{12}$  and  $\tilde{C}_{13}$  are contained in  $\rho_1(\tilde{C}_1)$ ,

$$S(\tilde{C}_{12}) + S(\tilde{C}_{13}) < S(\rho_1(\tilde{C}_1)).$$

The annulus bounded by  $C_1$  and  $\tilde{C}_1$  has conformal modulus  $\Delta$ . After reflection, the image is the annulus bounded by  $C_1$  and  $\rho_1(\tilde{C}_1)$ . By Lemma 7.2.6,

$$S(\rho_1(\tilde{C}_1)) \leq \Delta^2 S(C_1) \leq \Delta^4 S(\tilde{C}_1).$$

Similarly

$$S(\tilde{C}_{23}) + S(\tilde{C}_{21}) < \Delta^4 S(\tilde{C}_2),$$

$$S(\tilde{C}_{32}) + S(\tilde{C}_{31}) < \Delta^4 S(\tilde{C}_3).$$

By induction, we can prove Equation 7.3.  $\square$

This theorem shows that the total area of the holes is reduced exponentially fast. Thus, after  $-\log \varepsilon$  number of levels, each hole has a maximum area of  $\varepsilon$ . This shows that only a small number of levels is needed in practice. The proof of the theorem is put in the Appendix.

## 7.3 Algorithms

For a sensor network, we compute the covering space up to level  $q$  (for a constant  $q$  typically) in the following steps.

1. Extract a triangulation of the network.
2. Apply a distributed Ricci flow algorithm from the previous chapter to embed the triangulation  $T$  such that  $T$  is a circular domain — it is embedded in the plane with each hole (a non-triangular face) embedded on a circle.
3. With the circular domain  $T$  we apply circular reflections to compute the covering space.

Figure 7.3, 7.8 and 7.9 demonstrate the pipeline of the algorithm.

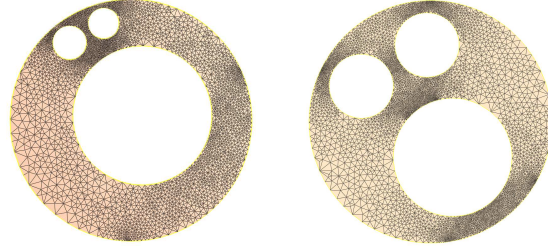
### 7.3.1 Circle estimation

For a circular domain  $T$ , for each boundary  $\gamma_k$ , we need to estimate the circle  $C_k(c_k, r_k)$ . We take three consecutive nodes  $\{z_1, z_2, z_3\}$  on the hole boundary to form a triangle, the circle  $C_k(c_k, r_k)$  is the circumcircle of the triangle. Its center is

$$c = \frac{|z_1|^2(z_2 - z_3) + |z_2|^2(z_3 - z_1) + |z_3|^2(z_1 - z_2)}{z_1(\bar{z}_3 - \bar{z}_2) + z_2(\bar{z}_1 - \bar{z}_3) + z_3(\bar{z}_2 - \bar{z}_1)} \quad (7.4)$$

and its radius is  $r = |z_1 - c|$ . The derivation of the equation above can be found in [34].

Since the circle can be computed by any three adjacent boundary nodes, the computation of the circular hole equation can be done locally at each boundary node. The computation only involves a constant number of algebraic operations.



(a) Original mapping; (b) Optimized mapping

Figure 7.7. Improve separation modulus by Möbius transformation.

### 7.3.2 Separation modulus optimization

From the theoretical result, we can see that the total area of circular holes shrink to zero exponentially fast. The convergence rate is governed by the separation modulus  $\Delta$ . In order to make the holes as small as possible, we can find an optimal Möbius transformation, that minimizes the separation modulus. In some sense, this transformation will map the holes to be as ‘well-separated’ as possible. We remark that this optimization step is optional.

A Möbius transformation preserving the unit disk is given by

$$\phi_{\theta, z_0}(z) = e^{i\theta} \frac{z - z_0}{1 - \bar{z}_0 z}, \quad |z_0| < 1,$$

which maps circles to circles.  $\phi_{\theta, z_0}(C_k)$  is still a circle, whose center and radius can be computed from

$$\{\phi_{\theta, z_0}(A), \phi_{\theta, z_0}(B), \phi_{\theta, z_0}(C)\}$$

using formula 7.4, where  $A, B$  and  $C$  are three points on the original circle. The rotation part  $e^{i\theta}$  doesn’t affect the separation modulus, in practice, we always set the rotation angle  $\theta$  to 0. Let  $\mu(C_j, C_k) = \tilde{\mu}_{jk}$  be the separation modulus between circles  $C_j, C_k$  as in formula 7.2. Define

$$\Delta(z_0) := \max_{j \neq k} \mu(\phi_{0, z_0}(C_j), \phi_{0, z_0}(C_k)).$$

The optimization problem is formulated as:

$$\min_{|z_0| < 1} \Delta(z_0).$$

In practice, we use gradient descent method to solve this non-linear optimization. See Figure 7.7 for an example.

To run this optimization step in a sensor network, we only need the knowledge of the center and radii of the circular holes. One node can pull the information of all circular holes together and run the optimization to get the Möbius transformation, which is then disseminated to all nodes in the network. The nodes apply the Möbius transformation to obtain the new mapping.

### 7.3.3 Circular reflection

In a circular domain we perform circular reflection of the network. In our implementation, we use a Hermitian matrix  $H$ ,  $H^T = \overline{H}$  and  $\det H < 0$  to represent a circle with center  $\mathbf{c}$  and radius  $r$ . Suppose

$$H(\mathbf{c}, r) = \begin{bmatrix} a & b \\ \bar{b} & c \end{bmatrix}$$

Then the circle equation represented by  $H(\mathbf{c}, r)$  is given by

$$0 = [\bar{z} \ 1] \begin{bmatrix} a & b \\ \bar{b} & c \end{bmatrix} \begin{bmatrix} z \\ 1 \end{bmatrix}, \quad (7.5)$$

where the center is  $\mathbf{c} = -\frac{b}{a}$  and the radius  $r = \frac{\sqrt{|b|^2 - ac}}{|a|}$ . For a circle  $|z - c| = r$ , the corresponding Hermitian matrix format is

$$0 = [\bar{z} \ 1] \begin{bmatrix} 1 & -c \\ -\bar{c} & |c|^2 - r^2 \end{bmatrix} \begin{bmatrix} z \\ 1 \end{bmatrix}. \quad (7.6)$$

Orientation preserving Möbius transformation on  $\mathbb{C} \cup \{\infty\}$

$$m(z) = \frac{\alpha z + \beta}{\gamma z + \delta}$$

is represented as a matrix

$$M = \begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix}, \alpha, \beta, \gamma, \delta \in \mathbb{C}, \quad (7.7)$$

its inverse is given by

$$M^{-1} = \begin{bmatrix} \delta & -\beta \\ -\gamma & \alpha \end{bmatrix}.$$

The reflection through a circle  $C$  is represented as

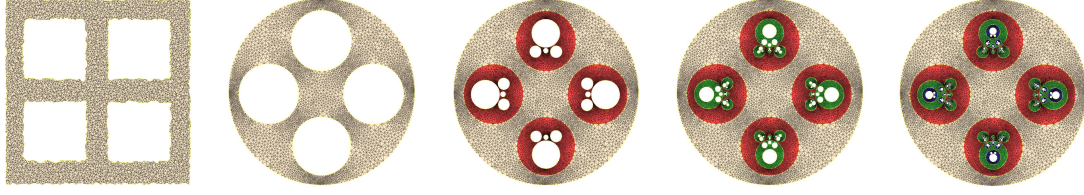
$$(\rho_C) = \begin{bmatrix} c & r^2 - |c|^2 \\ 1 & -\bar{c} \end{bmatrix}. \quad (7.8)$$

Thus, the composition of Möbius transformations are represented as matrix multiplications. A Möbius transformation  $m$  maps a circle  $H$  to a circle. The Hermitian matrix representation of the image circle is given directly by

$$\overline{M^{-1}}^T H M^{-1}.$$

For orientation reverse Möbius transformation

$$m(z) = \frac{\alpha \bar{z} + \beta}{\gamma \bar{z} + \delta},$$



**Figure 7.8.** 3-level circular reflections for a 4-hole network with 4764 nodes.

the matrix representation of the transformed circle is

$$M^{-T}H\overline{M^{-1}}.$$

Therefore, all the computations are carried out using complex matrix multiplication. Notice that the matrices  $H$  and  $M$  are only two by two matrices. So the matrix multiplication and the Möbius transformation can both be done with a constant number of algebraic operations.

The reflection  $\rho_C$  through a circle  $C$  is given by the analytic formula 7.1.

The reflection process is recursive. The naming conventions for all the circles  $C_\omega$  and all the reflections  $C_\omega$  have been explained in details in subsection 7.2.2, where  $\omega$  is a word in the multi-index set  $\sigma_n$  in definition 7.2.1. Suppose we are given a multi-index word  $\omega = \omega_1\omega_2 \cdots \omega_n$ , the recursive algorithms for computing the reflection  $\rho_\omega$  and the circle  $C_\omega$  (represented as matrices) are as follows.

---

**Algorithm 4** Matrix circle(word  $\omega$ )

---

```

if( $|\omega| = 1$ ) then
return  $H(C_{\omega_1})$ ; (Eqn.7.6)
end if
Matrix  $M = \text{reflection}(\omega_1\omega_2 \cdots \omega_{n-1})$ ;
if( $\omega_n = \omega_{n-2}$ )then
Matrix  $H = \text{circle}(\omega_1\omega_2 \cdots \omega_{n-2})$ ;
else
Matrix  $H = \text{circle}(\omega_1 \cdots \omega_{n-2}\omega_n)$ ;
end if
return  $m^{-T}H\overline{m^{-1}}$ ;

```

---



---

**Algorithm 5** Matrix reflection(word  $\omega$ )

---

```

Matrix  $H = \text{circle}(\omega)$ ;
Compute center and radius  $(\mathbf{c}, r)$  from  $H$ ;
(Eqn.7.5)
return  $M(\rho(\mathbf{c}, r))$ ; (Eqn.7.8)

```

---

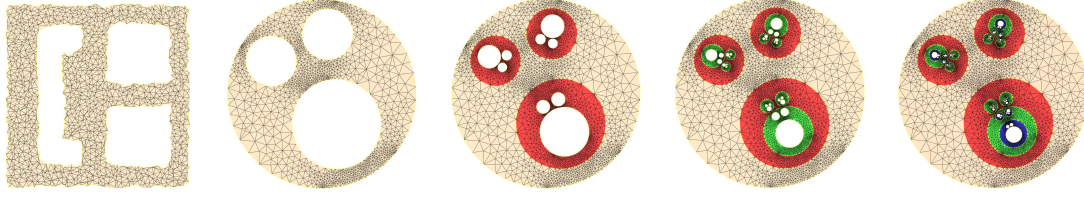


Figure 7.9. 3-level circular reflections for a 3-hole network with 1376 nodes.

### 7.3.4 Computation and communication costs

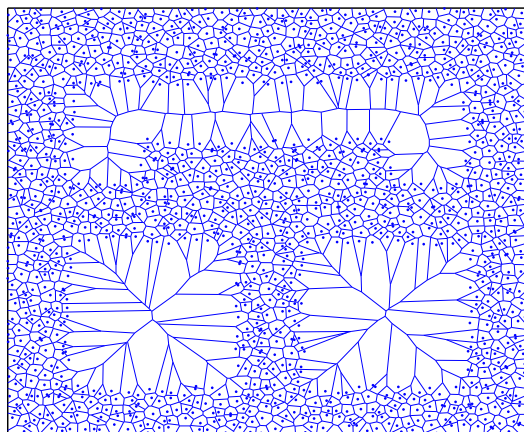
The communication cost of the scheme is analyzed for each component. In the first step of triangulation extraction, the algorithms used to extract a triangulation are localized algorithms. Each node only requires the knowledge of nodes in a constant size neighborhood. For most practical networks with constant average node degree, the total number of messages required is  $O(n)$ . The Ricci flow algorithm is an iterative algorithm with all nodes adjusting local metrics and local curvatures. The curvature error decreases exponentially fast. Therefore, the number of steps to reach the desired error bound is given by  $O(-\frac{\log \epsilon}{\delta})$ , where  $\delta$  is the step size in the Ricci flow algorithm. The total communication cost is thus  $O(-\frac{n \log \epsilon}{\delta})$ . The computation of network reflections are done locally in an on-demand manner. When a message hits a node on the boundary, depending on the number of levels of reflections required, the node can locally compute the reflection transform. The indices of the reflections are attached to the message. No additional communication is needed. Thus the communication cost in theory is linear in the number of nodes. In practice, the Ricci flow algorithm is the dominating factor of the communication cost.

## 7.4 Applications

### 7.4.1 Geographic hash tables

We apply the covering space with geographical hash table (GHT) [129] for storing data in the network. Data is indexed with a key. Each data item  $x$  is hashed to a geographical location by using a random hash function  $h(x) = g'$ . The producer of the data item delivers the data towards the location  $g'$  using geographical routing (GPSR [77] in particular). If GPSR can not find a node right at the hashed location, it will eventually enter the face routing mode, by following the edges of a planar face  $f$  enclosing  $g'$ . Such face routing will necessarily fail to find the destination  $g'$  and return to the first node when the message enters this face. At this point the algorithm stops. The node on the face  $f$  closest to  $g'$  is denoted as the home node. The face  $f$  is denoted as the home perimeter. A perimeter refresh protocol is used to maintain the home perimeter when there are node or link failures. Except the home node, the other nodes on the home perimeter are called replica nodes. The home node stores the data. The replica nodes may also hold data, to improve the system robustness to failures. GHT also has a hierarchically structured replication scheme





**Figure 7.10.** Voronoi diagram for the network in Figure 7.9 .

where multiple hash images are used. In our case as we examine the influence of the network irregularity to the storage balancing, we use the basic scheme only.

A node  $p$  is the home node for all the hashed locations inside its Voronoi cell (which contains all the points closest to  $p$  than all other nodes). Thus the storage load of a node is directly proportional to the area of its Voronoi cell. It can be seen that the nodes near a hole has a large Voronoi cell and thus are allocated more data compared with the average load. An example is shown in Figure 7.10. When the replica nodes also store data, the storage load on boundary nodes is even higher, as the hole creates a large perimeter face such that all the data on the nodes of this face are shared.

One way to deal with the high concentration of data on the nodes in sparse region or near network holes is to use rejection sampling [154]. In particular, we select a random geographical location  $g'$  and round to the closest sensor node  $g$ . But we only accept the node  $g$  with probability  $A/(cnA(g))$ , where  $A(g)$  is the Voronoi cell area of node  $g$ ,  $A$  is the total area of the domain from which  $g'$  is selected,  $n$  is the number of sensors, and  $c$  is a sufficiently large universal constant to make sure  $A/(cnA(g)) \leq 1$ . In the case of a sample  $g$  being rejected, another random location is selected and so on, until a sample node is accepted. This procedure will produce a sensor node uniformly randomly chosen from the network after about  $c$  trials. With the presence of holes, the smallest area of a Voronoi cell might be far smaller than  $A/n$ . Thus  $c$  needs to be large, leading to the waste of communication messages. To ensure a data-centric query eventually finds the data, we assume a source of randomness common to all nodes. Thus, if a node  $g$  rejects a data, the data-centric query for this piece of data will be re-directed to the same node and eventually either finds the data or claims failure (when the data is not rejected and the current node is not holding the data). With rejection sampling the storage load can be made uniform, but the rejections can increase the network traffic and energy consumption.

With the reflections and the covering space, the holes are filled by transformed copies of the network. If we use  $k$  levels of reflections for a  $m$ -hole circular domain, each node has  $m^{k-1} + 1$  images (including itself) in the covering space. We calcu-

late the Voronoi diagram of the sensor nodes and their images in the covering space. The chance that a random location is rounded to a sensor node is proportional to the total sum of the area of the Voronoi cell of  $p$  and all its images. Since the holes are now covered up, the area of the holes are then distributed to the entire network. All nodes are then selected with similar probability. Thus the maximum storage load of any node is substantially reduced. When rejection sampling is used, the maximum traffic load caused by delivering data to the hashed locations is reduced dramatically, as not only the number of rejections is smaller but also the traffic of the greedy routes are also spread more evenly in the network. For the improvement of traffic load balancing for greedy routing with the covering space, please see Section 7.4.3.

## 7.4.2 Double rulings

We also apply double rulings, or quorum-based data storage and retrieval scheme, with the covering space. In a double rulings scheme, the producer stores data or data pointers along a storage curve and the consumer (the user who would retrieve the data) routes the request along a retrieval curve. As long as any storage curve intersects with any retrieval curve, it is guaranteed for successful discovery of data. In fact, any retrieval curve can discover *all* the data stored in the network making it more efficient to retrieval data. These curves are often designed as some nice geometric curves. When we route data or request packets, we use greedy routing to select the next hope as the one near the geometric curve and making progress along the curve (e.g., the projection onto the geometric curve is further away) [116]. This leads to routing paths that approximate the geometric double ruling curves. For example, when a network has a rectangular shape, one can use the horizontal lines as storage curves and vertical lines as retrieval curves [104, 145, 159], denoted as rectilinear double rulings. The data packets are routed to the neighbor furthest in the vertical directions, and the retrieval packets are routed to the neighbor furthest in the horizontal directions.

Existing double rulings schemes all assume some nice regular shape (rectangular or circular). When there are holes, many geometric curves may hit a hole. Thus the greedy routing paths approximating the double ruling curves may either get stuck at a local minimum, or, when advanced hole bypassing techniques [42, 77] are used, cause high traffic load on the hole boundaries. Another problem with an irregular shape is that it is not easy to design the geometric double ruling curves that guarantee intersection without using perimeter mode face routing. One can easily construct examples such that rectilinear double rulings, or many other elegant geometric curves fail to intersect inside the sensor domain.

With the covering space, the network is turned into a circular disk  $D$  with radius  $R$ . Designing geometric double ruling curves is trivial. For example, one can use co-centric circles and rays emitting from an origin as storage and retrieval curves respectively. In our simulation, we use spherical double rulings [132]. We map the covering space to the bottom hemisphere tangent to the center of  $D$  with the stereographic projection [29]. We put a sphere with radius  $r < R/2$  tangent to the plane at the origin. Denote this tangent point as the south pole and its antipodal point as the

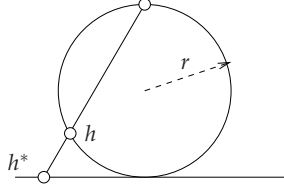


Figure 7.11. Stereographic projection.

north pole. A point  $h^*$  in the plane is mapped to the intersection of the line through  $h^*$  and the north pole with the sphere. See Figure 7.11 for a cross intersection. We choose  $r < R/2$ . Thus the bottom hemisphere is covered by image of the disk into which the network is embedded. For a data item  $x$ , we use a hash function  $f$  to select a random hash location  $g^*$  in the plane and store the data along the storage curves, defined as the great circle through the producer and  $g$ . The use of hashing is to allow multiple data items of the same type to be possibly stored and aggregated at the hashed node. The retrieval curve is any great circle through the consumer. It can be shown that any storage curve and retrieval curve have a common intersection in the bottom hemisphere. In the implementation, we find the routing path traversing through the triangles that intersect with a storage (or retrieval) curve. With a sufficiently high  $k$  (the number of levels of reflections), the holes are mostly filled up with only tiny holes left in the domain. Thus the chance that a double ruling curve hits a tiny hole is very small. For a retrieval curve to discover the data, we only need the intersection of the retrieval curve and the storage curve to be not in a tiny hole. The chance for this to happen is small as well.

### 7.4.3 Load balanced greedy routing

Greedy routing selects the next hop as the neighbor whose distance to the destination is minimum, with the distance defined in some coordinate system. It is an extremely simple and completely local algorithm but may not always work if all the neighbors have greater distances to the destination. In our previous work [130], we embed a sensor network in the plane such that all the holes are circular. Thus greedy routing can not get stuck at any hole boundaries. This leads to guaranteed delivery of greedy routing in the generated virtual coordinates. Nevertheless, greedy routes that hit a node on the boundary of a hole  $C$  will lead to a path that follows the hole boundary until the tangent point of the line through the destination and  $C$  (i.e, when the packet is able to ‘see’ the destination). This effect causes higher traffic load on the hole boundary.

By using the covering space, we can mitigate the imbalance of traffic load caused by the greedy routing method. When a packet reaches a node on the hole boundary, greedy routing directs it to enter another copy (Figure 7.12), effectively reflect on the hole boundary and take a detour to get around the hole in the real network (Figure 7.13). Therefore, not all the packets that arrive at a hole boundary will necessarily route along the hole. The detours they take are spread out in the network,

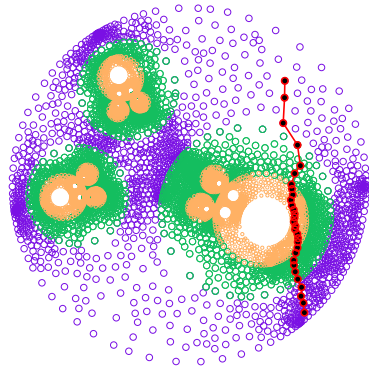


Figure 7.12. Path in covering space.

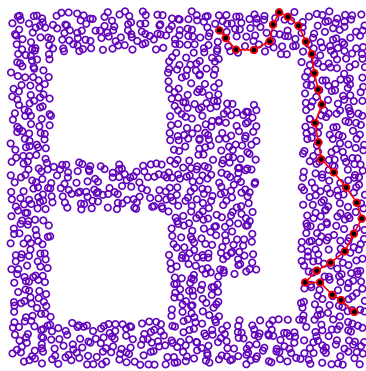


Figure 7.13. Path in real network.

reducing the traffic pressure on the hole boundaries.

The routes, reflected on the holes, are possibly longer than before. But this is in some sense necessary in order to improve load balancing. Minimizing the path length and minimizing the maximum traffic load are two contradicting objectives that cannot be achieved at the same time [58]. There is also an interesting trade-off as longer paths increase the total message cost and the average traffic load. In our simulations we demonstrate that a small number of reflections suffice to strike a good balance of path stretch and load balancing.

## 7.5 Simulations

We carried out extensive simulation tests on several different networks to verify the utility of this method. The data presented in this section are based on the network represented in figure 7.9. This network has about 1400 nodes in a perturbed grid distribution, spread over a  $200 \times 200$  region, and a maximum communication radius of 12 units. The graph is a quasi-unit disk graph of inner radius  $12/\sqrt{2}$ .

The following are the important observations we obtained from this set of simulations:

1. The reflection based covering space reduces the maximum storage load at any sensor to almost half, compared with the original embedding, if we uniformly sample geographical locations and then round to the closest sensors.
2. Greedy routing on the covering space has better load distribution than GPSR on the original network.
3. Using the reflections, double ruling schemes can be extended to networks of non-trivial topology. The storage cost is higher but the retrieval cost is much lower.

### 7.5.1 In-network storage and sampling

As described in the previous section, when using a GHT type storage scheme, the storage load at any node is proportional to the area of its Voronoi cell. The nodes at the boundary of a hole tend to get higher load because their voronoi cells together cover the area of the entire hole. We compared the maximum Voronoi areas of the original embedding with those of covering spaces obtained by one or more reflections. Reflections create multiple images of points for each node, the load on the node is taken to be the sum of the Voronoi cell area of all images.

The results are shown in Figure 7.14 and Table 7.1. The total load is normalized to be 1. It is shown that the covering space reduces the max load to almost half that of the original embedding. This is achieved with only 2 or 3 reflections, after which the load does not change too much. This can be understood as follows. The uneven loads are caused by big Voronoi cells, which can be created by the presence of holes, and/or by distortions introduced by the conformal map. After a few reflections the

area of the holes are shared by nodes of the entire network and do not contribute large areas to any node in particular. Thus the larger cells created by the distortion of the conformal map are the major reason for the uneven load. This does not get smaller with more reflections.

We carried out rejection sampling on these embeddings, and found that the covering spaces created by 2 or more reflections require about 15% fewer trials on average. Further, when the communication costs are considered, the communication loads are much better balanced in the covering space scheme. This is essentially because greedy routing in generally is better load balanced on the covering space than GPSR on the original network. We describe the routing results in the next subsection.

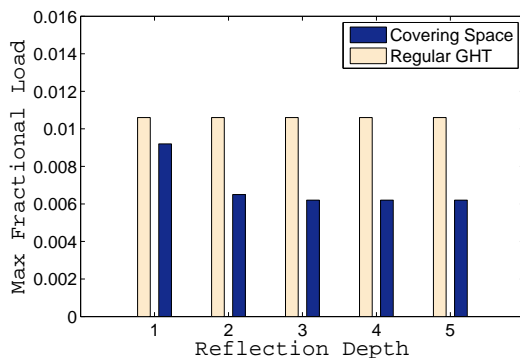


Figure 7.14. The largest fraction of the storage load at any sensor.

Table 7.1. Maximum storage load for GHT.

Scheme	1-ref	2-ref	3-ref	4-ref	5-ref
Covering Space	0.0092	0.0065	0.0062	0.0062	0.0062
Regular GHT	0.0106	0.0106	0.0106	0.0106	0.0106

<sup>+</sup> $n$ -ref -  $n$  depth reflection

## 7.5.2 Load balanced routing

We selected 2000 random source-destination pairs and computed routes. On the original network we used GPSR. On the covering spaces we used simple greedy routing, since greedy routing guarantees delivery in these networks. The number of reflections only shows the maximum number of reflections we permit. The covering space is not pre-computed to that depth, reflections are computed locally as a route progresses and the relevant data is attached to the message.

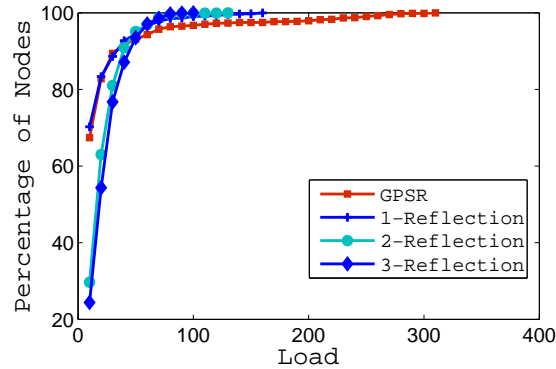
The results are shown in Table 7.2. Clearly, 2-3 reflections give the best results. Beyond this point, the route lengths tend to increase as some paths go through several reflections. However, the load balancing is always good, since the covering space method does not hug the boundary when going past a hole. The path bounces off the boundary and spreads the load more evenly (see the plot in Figure 7.15).



**Table 7.2.** Traffic load and path length for greedy routing.

Scheme	Avg. load	Max load	Avg. length	Max length
GPSR	33.6840	620.0	24.1915	92
1-ref	24.0682	319.0	17.571	42
2-ref	35.4960	190.0	25.439	117
3-ref	39.1742	241.0	27.9715	159
4-ref	43.9143	199.0	31.235	196
5-ref	46.3129	216.0	32.8865	228

<sup>+</sup>*n*-ref - *n* depth reflection



**Figure 7.15.** Cumulative distribution of load. Showing that the covering space method has fewer nodes with high load, and also fewer nodes with very low load. GPSR on the other hand has 5% to 10% nodes with substantially high load.

**Node lifetime experiments** We carried out experiments on how the load balancing properties affect the longevity of nodes. Each node is assumed to have the energy to transmit 200 messages, after which it is considered dead. We count how many nodes die in the process of delivering 4000 messages. As nodes die, the network loses the property of circular holes and guaranteed delivery. We count the number of messages that are delivered successfully under these conditions. GPSR guarantees delivery, but with dying nodes, the network eventually gets partitioned and then some messages fail. The results are shown in 7.3.

**Table 7.3.** Death and message delivery rates for 4000 routing attempts.

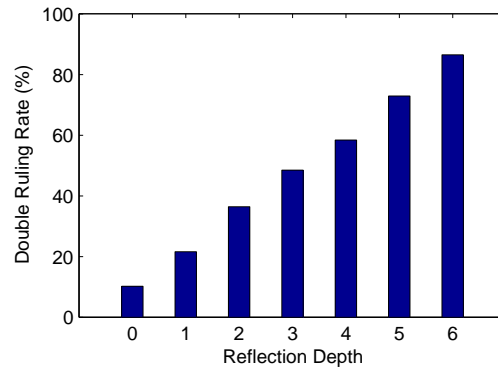
Scheme	GPSR	0-ref	1-ref	2-ref	3-ref	4-ref
Deaths	286	33	29	50	122	154
Deliveries	3164	3361	3790	3849	3886	3842

<sup>+</sup>*n*-ref - *n* depth reflection

**Double rulings** *Double ruling* is a general method that extends GHT. The intuition being that storing data on a path makes it easier for consumers to find that data. Existing double ruling schemes such as [132] design the storage and retrieval paths with simple networks in mind. The covering space, by *almost* eliminating the holes in the network can be expected to make double ruling applicable to more general



networks. In particular, as the holes get smaller in size, it can be expected that fewer of the storage and retrieval paths hit them. We carried out experiments to test this and other properties on covering spaces of different depths. The radius of the sphere was taken to be one-third the radius of the embedded network in virtual coordinates. The results below are for 2000 random producer, consumer and hash location triples.



**Figure 7.16.** Percentage of successful double ruling retrievals for different depths of covering space.

Figure 7.16 shows the success rate of double ruling with increasing depth of covering space. For no reflections, the percentage of successes are very small, about 10%, but as holes get smaller, success rate climbs to 86% for 5 reflections. In the following, We augmented the process at the final level by perimeter mode traversal of boundaries to obtain full success rate.

We carried out path length and load measurements. The results are shown in figures 7.17 and 7.18 respectively. The storage costs can be seen to be relatively high, as producers may sometime select a curve that goes through many reflections. In comparison, consumer retrieval costs are lower. This is because the consumer path stops as soon as it intersects a producer path. While this is also true for double ruling in the original embedding, the covering space introduces additional properties. there are many copies of a node  $x$  in the covering space. Imagine that the storage path passes through a copy  $x'$ . Now it is possible that the retrieval path hits a different copy  $x''$  before it intersects the storage curve in the coring space. In such a case, the consumer has hit a storage node in the real network, and can stop searching. This possibility reduces retrieval costs even further. In a sense, the higher storage costs are compensated by a smaller retrieval cost.

### 7.5.3 Network dynamics

Finally, we tested some effects of network dynamics. In a general sensor network nodes may fail, and occasionally some nodes added. These cause changes in the triangulation. For example, failure of a node creates a 'hole' in the triangulation. Many of these changes can be handled by simple adjustments. Failure of a node in a dense network can be handled by adding edges between its neighbors to fill up

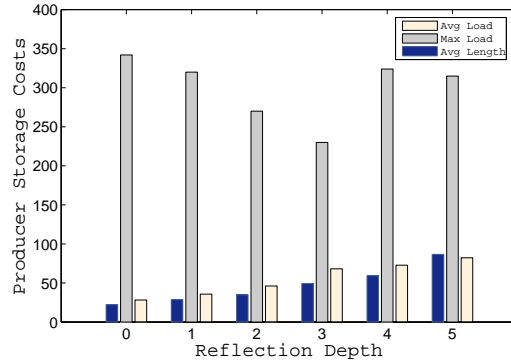


Figure 7.17. Producer storage costs.

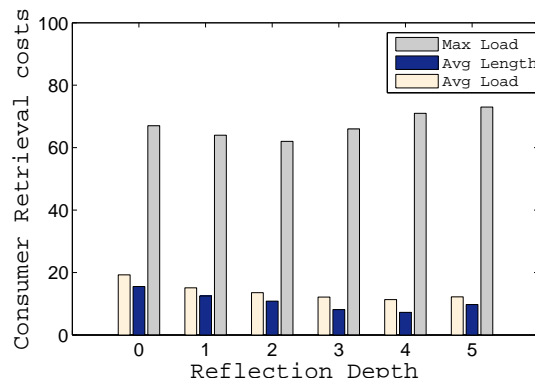


Figure 7.18. Consumer retrieval costs.

the hole. In certain cases, say after several failures, such simple local adjustments may no longer suffice and then the embedding needs to be recomputed. However, instead of computing the embedding from scratch, we can use the existing configuration as a starting point to speed up the process. We found that for small changes to the triangulation, the reconvergence is quite fast.

Table 7.4. Reconvergence Time.

Error bound	1e-1	1e-2	1e-3	1e-4	1e-5
Iterations	129	274	697	1392	2221

We made some small changes to the triangulation, such as edge swaps, and measured the time taken for the Ricci flow to converge to some desired error. The results are shown in table 7.4.

## 7.6 Conclusion

The network metric and embedding are crucial to sensor network operations. In this chapter we presented ideas of using Möbius transforms and Ricci Flow algorithms

to regulate a sensor network. All networks can be made to be circular with the interior holes filled up. The created covering space embedding is universally useful to even out sensor storage and traffic load for in-network data storage schemes. We plan to investigate further applications of the embedding in sensor network and sensor data management.

**Part III**

**Geometry and Topology of  
Information**



# Chapter 8

## Introduction

Geometry and topology can be applied to analyze functions, and therefore to understand and summarize data. Suitable geometric structures associated with data can reflect fundamental properties hidden in the raw information. In our goal of information processing in sensor networks, recovering structural properties of data is essential. The approach in this chapter will be to construct geometric abstractions that represent attributes of data that is not evident from the direct sensor readings. These abstractions help us intelligently answer queries about the data in a distributed manner from within the network.

The next chapter is based on a structure from differential topology called the contour tree. This tree represents the nesting relations between contours of a function. We present an algorithm to compute the contour tree and the various applications it has in sensor networks. The computation and representation of the tree are both distributed, and do not rely on availability of locations. A contour tree can be constructed as long as the domain is simply connected. Accordingly, the our algorithm is for dense sensor networks that do not have holes.

Contours are fundamental aspects of real valued functions, and their analysis enables us to answer questions related to ranges of values occurring in different regions of the network. In particular, we concentrate on the following problems:

- *Iso-contour query*: from a query node  $q$ , find the iso-contours at value  $x$ , or count/report iso-contour components at given value/range.
- *Value-restricted routing*: find a path from a source node  $s$  to a destination node  $t$  with all values on the path within a user-specified range. This can be used for navigation of packets in the network (e.g., avoiding sensor nodes with low energy level), or navigation of objects in the physical environment (e.g., avoiding traffic jam).

The second functionality can be modified to answer queries of the type “Find a path on which the maximum value is minimized.” We show a labeling scheme that makes it possible to perform the value restricted routing with distributed information at the nodes, so that nodes do not need to know the complete contour tree, and can operate with local information about the tree.

The last chapter relies on a very basic geometric construct called a differential form. It is used here to compute aggregate information about data. In particular, given a region  $\mathcal{R}$  in the network, we compute the sum of sensor values inside it by simply summing the differential form along the boundary of  $\mathcal{R}$ . This is the sensor network application of *Stokes theorem*.

The differential forms approach works for any function, but our main focus will be on a function that tracks mobile targets detected by the network. This is because in the case of mobile targets that move along continuous paths, the form can be updated very efficiently to adjust to target movement. The method has enormous flexibility – it adjusts automatically and naturally to node failures, network holes, rapid movements of targets, even addition and mobility of the sensor nodes themselves.

Differential forms are fundamental constructs in geometry and analysis, and the ability to compute general sums in arbitrary regions opens up many other possibilities. For example, in a network where a planar graph can be computed, contour trees can be computed distributedly, with an algorithm that is different from the sweep paradigm of existing methods. Further, this approach can handle domains with holes, unlike the direct computation of contour tree. This method is described briefly in chapter 10.

## 8.1 Research review

There are a lot of previous works on tracking mobile targets and on queries of sensor data. We briefly review these work.

**Range queries.** For a typical range query, we are given a query sensor region plus possibly a range of the sensor data, and then ask for all the the sensors in the query region whether any sensor data is within the data range. This is a problem that has been studied a lot in computational geometry. Centralized data structures for geometric range query on static points [5] or motion data [?], have been developed. But they are obviously not a good fit for a distributed sensor network setting. Various distributed schemes have been proposed. In the case of a scalar field, one solution is to partition the information about large geographic regions into subsets according to smaller ranges of the field value, and store these subsets in different nodes. This is the approach taken in the DIFS system [62], where each node in a quad-tree has multiple parents, according to a finer partition based on smaller field ranges. Thus the wider the spatial extent an index node knows about, the more constrained the value range it covers. In the DIM system [99] a locality preserving hash function is used to map portions of a multidimensional attribute space to sensors so that all data needed to answer a range searching query can be located conveniently. In the fractional cascading approach [56], information is stored so that more detailed information is available about data obtained in the spatio-temporal locality of the sensor where the query is injected—but without sacrificing the ability to query distant regions or times as well. All of these schemes are designed to support range queries for static sensor data. For mobile data such as a swarm of mobile targets, constant



updates in a global manner make these schemes too costly. In addition, these prior schemes are mainly for rectangular ranges only. Ranges of other shape must be first partitioned into smaller rectangular ranges, which are queried separately.

**Location services.** Existing solutions for tracking and searching for mobile targets, termed as *location services*, focus on the tracking and searching of a single target. The earliest work is by Awerbuch and Peleg [9] and followed up in [2, 46, 98] to fine tune the system. The location of a mobile target is updated to a carefully selected set of nodes, called the location servers, whose spatial density cascades exponentially as we move away from the target. This allows ‘locality-sensitive’ queries, i.e., the cost of a query is proportional to the distance to the target. When a target moves, information is updated on a location server, with the frequency inversely proportional to the distance to the target. Note that this method requires tracking data to be sent and stored at far away nodes. Thus, even if targets are concentrated only one region of the network, other nodes have to stay awake for storage and communication of the tracking data. Thus the information of a nearby location server is more up-to-date. Forwarding pointers are left at the old position pointing to the current position of the target. A query far away from the target may first obtain outdated information pointing to a past location, from where the query can be delivered to the current position by following the forwarding pointers. This family of schemes focused on the tracking and searching of an individual, identifiable target. Location services have amortized update cost of  $O(d \log d)$  when a target moves a distance  $d$ , and a query cost of  $O(d')$  if the query node is of distance  $d'$  away from the target’s current location. In comparison, we have better asymptotic bounds. Our update cost is worst case  $O(d)$  and query cost is no more than  $O(d')$ . In addition, location services do not support range queries very well. If there are multiple targets, they are handled separately. For range queries or aggregated queries (such as density) one has to search for location servers for all potential targets within the range, which can be highly inefficient. In this chapter we evaluate the performance of using location services and using our method for range queries in the simulation section. We show that for both update cost and the query cost, our method is substantially better.

**Information gradients.** The third approach is to define a potential field centered at the target. Such information potential fields can be either the natural gradients of physical phenomena, since the spatial distribution of many physical quantities, e.g., temperature measurements for heat, follows a natural diffusion law [25, 44, 45, 103], or built explicitly on a mobile target. One scheme in this family uses harmonic function to build such information strength field [100], which satisfies the Laplace’s equation  $\nabla^2 \Phi(x) = 0$  with proper Dirichlet boundary condition (1 at the target location and 0 at the network boundary). Such an information field is guaranteed to be free from local minima. Thus every node can follow the local *information gradient* to arrive at the target. This works for both identifiable (information fields are maintained separately) and non-identifiable targets (a single information field is maintained for all targets). In addition, the divergence-free property of harmonic gradients and Faraday’s law of induction imply an easy solution for counting range queries — touring the boundary of a given range and summing up the difference of

the potential values on the edges across the region boundary provide the number of targets in the interior of the range. When a target moves, the information field needs to be updated to ensure the harmonic function property. The limitation of the scheme is that updating the potential field for mobile target is costly by the global nature – nodes far away from the target have to update their information strength, while ideally we hope to restrict the updates to be within a small neighborhood of the target. If we ‘rotate’ the gradient vectors by  $90^\circ$ , the result is a differential harmonic one-form. In our scheme we do not require the differential one-form to be harmonic – thus one can not easily navigate towards the target as in the scheme in [100]. However, the benefit of using a relaxation as simply a differential one-form is to allow quick maintenance of the one-form under target motion. As we have shown, the update is completely restricted to the target neighborhood.

# Chapter 9

## Topology of Data and Iso-Contour Queries

We study the problem of data-driven routing and navigation in a distributed sensor network over a continuous scalar field. Specifically, in this chapter we address the problem of searching for the collection of sensors with readings within a specified range. This is named the *iso-contour query* problem. We develop a gradient based routing scheme such that from any query node, the query message follows the signal field gradient or derived quantities and successfully discovers *all* iso-contours of interest. Due to the existence of local maxima and minima, the guaranteed delivery requires preprocessing of the signal field and the construction of a *contour tree* in a distributed fashion. Our approach has the following properties: (i) the gradient routing uses only local node information and its message complexity is close to optimal, as shown by simulations; (ii) the preprocessing message complexity is linear in the number of nodes and the storage requirement for each node is a small constant. The same preprocessing also facilitates route computation between any pair of nodes where the route lies within any user supplied range of values.

### 9.1 Introduction

Wireless sensor networks have shown great potential for providing dense monitoring and sensing capabilities with modest cost and management effort. In many typical sensor network applications, sensors are densely deployed in a physical environment to provide good coverage at fine sensing resolutions. Existing work has established many fundamental mechanisms for sensor deployment to ensure coverage [10, 94, 108, 164] as well as energy efficient networking functions to collect data from these nodes.

There are two fundamental aspects of sensor networks that differentiate them from other types of wireless networks. First, it is the data from the sensor nodes, rather than the network nodes themselves, that is of most interest to the users. While many wireless networks, such as wireless LANs, cellular networks, and ad hoc mobile networks, focus on supporting low-latency end-to-end communications and

maximizing the system throughput, sensor network designs are often tailored towards their target application and are bound tightly to the physical environment that they are supposed to monitor/sense. In the most prevailing applications of environmental monitoring, sensors measure readings of the physical space, such as temperature, pressure, chemical concentration, and many others. Such physical quantities often exhibit continuity properties over space and/or time. Thus the smoothness of the physical signal field, and the spatial correlation of discrete sensor data, naturally suggest possibilities for data compression and exploitation for efficient system design.

A second unique property of sensor networks resides in their great potential in allowing seamless interaction between users and the physical world. In many civilian and military applications, the users operate in the same space in which the sensors are embedded. This allows novel applications in which real-time sensor data is quickly delivered to users of interest for appropriate response and actions. All of this eventually leads to a smart environment that could revolutionize the way we observe, interact with and influence the physical world.

In this chapter we look at the iso-contours of a scalar signal field represented by sensor data, together with a local gradient descent routing scheme, with which the users can navigate in this signal field with guaranteed success.

**Iso-contour related queries.** For a continuous field, an iso-contour at an isovalue  $x$  is the collection of points with value equal to  $x$ . In a discrete sensor network, this is often approximated by the collection of sensors with readings sufficiently close to  $x$ . The iso-contours encode spatial structures of the signal field, such as boundaries of the ‘hot’ regions that indicate overheating or a fire, or pollution dissemination that may require special treatment. The signal field can also be the energy map or traffic load on the networked sensors, and thus the iso-contours are related closely and provide information about the general health of the network or its traffic bottlenecks.

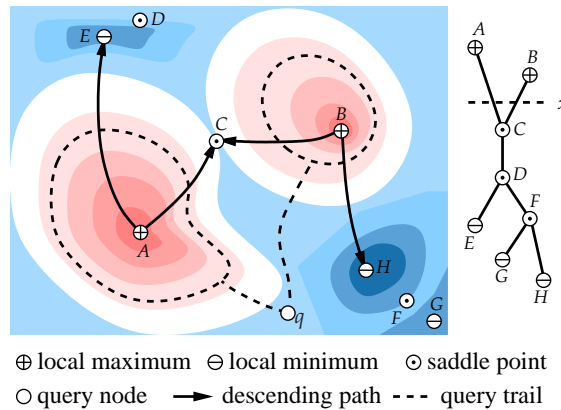
A few papers have studied compression, approximation and aggregation of iso-contours with space-efficient data structures, when sensors report their data along an aggregation structure to the base station [51,70,109]. In this chapter we are interested in in-network data processing and the usage of iso-contours for navigation in the signal field. Consider a scenario in which sensors and users (such as rescuers or patrol officers) are embedded in the same physical space. Users with hand-held devices communicate with nearby sensors to obtain directions to places that require attention or service, indicated by the sensor data being within a specified range. We consider the following two routing and navigation functions:

- *Iso-contour query*: from a query node  $q$ , find the iso-contours at value  $x$ , or count/report iso-contour components at given value/range.
- *Value-restricted routing*: find a path from a source node  $s$  to a destination node  $t$  with all values on the path within a user-specified range. This can be used for navigation of packets in the network (e.g., avoiding sensor nodes with low en-

ergy level), or navigation of objects in the physical environment (e.g., avoiding traffic jam).

For both problems, we are looking for efficient solutions without flooding all the nodes. The chapter is tailored to the iso-contour query as it best demonstrates the basic idea, with which the value-restricted routing can be answered easily.

**Gradient descent routing.** The most intuitive solution for iso-contour queries is to use gradient descent, by exploiting the natural continuity of the signal field. Starting from the query node  $q$ , the query message can be greedily guided either downhill or uphill, depending on the comparison of the value at  $q$  and the target value  $x$ . This greedy descent routing is simple and requires only local knowledge. Thus it has been explored in a number of settings for low-cost data-centric routing [25,44,45, 103,160]. Greedy descending/ascending can typically lead the query message to one iso-contour, unless the query message reaches a local minimum or local maximum, in which case the query gets stuck. Indeed, using simple gradient descent for an iso-contour query has a serious defect: the signal field may have multiple peaks and valleys, and greedy descending discovers at most one iso-contour, and is not able to discover all of the iso-contours due to the existence of local optima.



**Figure 9.1.** The level sets of a signal field and the contour tree spanning all the critical points (in the right). The figure also shows *some* descending paths connecting the critical points.

Figure 9.1 shows an example of a potential field by drawing its level sets. Red colors mean hot and blue colors mean cold. We also show all the local maxima, minima and saddle points. A greedy gradient routing from a query node  $q$  looking for a desired level contour will follow the local gradient and climb up the mountain. Once the query reaches the desired level it can locally trace out one contour, e.g, the contour on the left peak in the figure. However with only local information the query does not know whether there are other peaks and if so where they are.

The difficulty here is that the greedy gradient routing is completely local, while iso-contours reflect the global topology of the signal field. This is a general problem in navigation with a potential field, as has also been studied in robotics: with only information about the local potential one lacks the big picture of the signal field which is important for guaranteed success. In particular, the collection of critical

points (local maxima, minima and saddle points) represents the global topology of the signal field. Thus, in order to make the local greedy descend algorithm always work, one needs to augment it with a compact representation of the critical points and their relationships.

**Our contribution.** We propose to investigate distributed algorithms to pre-process the iso-contour structures of the signal field by what is called the *contour tree* [153], using which a gradient routing scheme can successfully discover *all* iso-contours. In short, a contour tree is a tree on all the critical points of the signal field and captures the topology of the iso-contours. It is a special case of the *Reeb graph* in Morse theory [111]. Take Figure 9.1 as an example, the right figure shows the topological contour tree consisting of eight vertices, corresponding to two local maxima, three local minima, and three saddle points. A contour tree captures how the connected components of the iso-contours merge/split as we increase/decrease the isovalue.

We propose an algorithm for the construction of the contour tree in a fully distributed manner. The basic idea is similar to the centralized construction [18,30,146,153]. But we need to account for numerous robustness issues due to local noise and degeneracies, and lack of global coordination. We use distributed sweeps [143], initiated at local maxima and minima to identify the saddle points and nodes on the saddle contour. Next an information dissemination phase following the contour tree structure distributes necessary information for gradient descent routing. The pre-processing involves all together four rounds of sweeps of the signal field and has a linear message complexity.

The invariant we maintain on a node  $p$  is the max/min value in the interior and exterior of the iso-contour component through each point  $p$ . This represents only a small constant storage requirement at each node. For iso-contour queries, the gradient descent routing alternates between two operations (i) at a node on some saddle contour, it checks the split/merged contours and send one or two (if necessary) messages to the new connected components. (ii) at other nodes, the query message either follows an iso-level or follows gradient ascending/descending path to reach the desired contour. The gradient routing only uses information stored at a node itself and every routing step is justified, in the sense that there will definitely be a contour discovered for each query message. Thus no effort is wasted. Our simulations show that the gradient routing achieves comparable message complexity, when compared with the minimum spanning tree covering the iso-contour components, which is at most twice the length of the minimum Steiner tree, the optimal solution if the global knowledge about the entire signal field were available.

At the same time, the same contour tree permits a scheme for restricted value routing, and a labeling scheme such that validity of a restricted value route request can be determined simply from the labels of the source and destination nodes. Intuitively, the spatial structures of the signal field are entirely captured by the contour tree, and low values paths in the field can be mapped to a low value path on the tree.

Lastly we note here that in this chapter we only consider a static signal field, be-



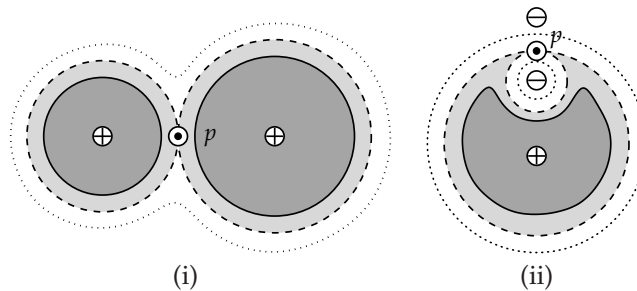
cause the problem for a static signal field is already quite challenging. In practice, as the signal evolves over time we can periodically execute the contour tree construction phase. The maintenance of the contour tree for a time-varying signal field will be future work.

## 9.2 Contour trees and gradient routing

Given a continuous signal field  $\mathcal{F}$ , the *iso-contour* (aka. *level sets*) at an *isovalue*  $x$  is the collection of points  $p$  with value  $\mathcal{F}(p) = x$ , and may have multiple connected components. We denote by  $C$  one connected component of an iso-contour and by  $C(p)$  the connected component containing node  $p$ .

As we decrease the isovalues from the global maximum to the global minimum, the connected components on the iso-contours may merge together, split, emerge, or disappear. These changes happen at *critical points*, such as *local minima*, *local maxima* and *saddle points*. The *contour tree* captures such topological changes of the iso-contours. In a contour tree, each node corresponds to a critical point, and an arc in the contour tree connects two critical points. In particular, as we start from  $+\infty$  and decrease the isovalue,

- at a local maximum, a contour component emerges;
- at a local minimum, a contour component vanishes;
- at a saddle point, two contour components merge into one or one contour component splits into two (see Figure 9.2). In the first case, there are two branches of the iso-contour emanating from the saddle point, representing the two components. Such a saddle point is called a *merge saddle* (with respect to decreasing isovalues). The second case with respect to decreasing isovalues is called a *split saddle*. (For increasing isovalues, split saddles and merge saddles are interchanged.)



**Figure 9.2.**  $\oplus$  indicates a local maximum.  $\ominus$  indicates a local minimum.  $\odot$  indicates a saddle point. Dark colors mean larger values. When we start from  $\infty$  and decrease the isovalue, at a saddle point, (i) two contour components merge into one; (ii) one contour component splits into two.

It has been proved that the merging and splitting of contour components are indeed represented by a tree. Further, without degeneracy (no two saddle points have the same values), a local maximum or a local minimum has degree 1; a critical



point has degree 3. To visualize, we place the vertices of a contour tree, i.e., the critical points, at the height levels of their values. A merge saddle has a ‘Y’ shape and a split saddle has an inverted ‘Y’ shape. Then we can map contour components in an iso-contour at value  $x$  to the points obtained by cutting the tree at level  $x$ . The contour component through a saddle is mapped to the saddle vertex on the tree. Thus, at a point  $p$  in the signal field, if its contour component  $C(p)$  is mapped to a point on arc  $\alpha$  in the contour tree, then we say  $p$  is on arc  $\alpha$ . In Figure 9.1, the iso-contour at the query value  $x$  has two components, the left contour stays on the arc  $AC$  and the right one stays on the arc  $BC$ . If we embed the contour tree in the domain by representing each edge with a monotonic path connecting the corresponding critical points, it can be verified that this mapping is continuous and the contour tree is a retract of the original domain under this mapping.

In a sensor network the continuous signal field is sampled by discrete sensors. To compute the contour tree in the discrete setting, we have the following challenges:

**Local identification of critical points.** In a continuous signal field, a critical point  $p$  is a point with all partial derivatives vanishing at  $p$ . In a sensor network we can easily identify the local maxima and local minima. A local maximum (minimum) has all the neighboring values no greater (smaller) than itself. However, it is not easy to identify saddle points, which have larger and smaller values in its neighborhood in an alternating way. When we do not have sensor locations or do not have accurate locations (say, the neighbors may switch their angular ordering), identifying a saddle point robustly is not straight-forward. In our algorithm, the saddle points are discovered along with the construction of the contour tree, as the nodes where the contour components merge.

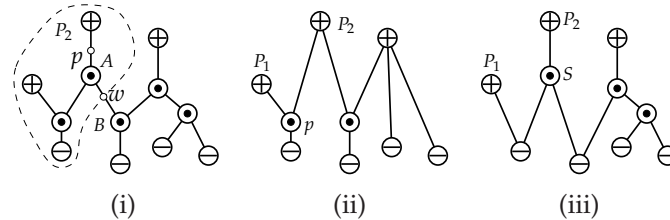
**Distributed construction of the contour tree.** The construction of the contour tree of a piecewise linear mesh has been studied before [18, 22, 30, 146, 153]. The best algorithm achieves a running time of  $O(n \log n)$  on a piece-wise linear surface with  $n$  vertices and can even be made to be output sensitive [18, 22]. However, these algorithms are centralized and are not appropriate for low-cost in-network processing in a distributed sensor network. We propose a distributed algorithm that involves four passes of sweeps, to be explained in details in subsection 9.2.2. Thus the construction costs roughly  $4n$  message transmissions. After the preprocessing phase gradient-based routing with guaranteed success for iso-contour queries can be performed at *any* node in the network.

**Handling noises and plateau regions.** An important practical issue regarding contour trees for a sensor network is that the sensor data is a noisy approximation of the underlying smooth signal field, due to sensor inaccuracy, hardware noise, etc. Thus there could be many more local maxima and minima in the sensor data than the the original (unknown, smooth) signal field. We propose two methods to handle this. First, we will locally simplify a contour tree by using topological persistence [19]. Small bumps will be chopped off. Second, we will not keep the entire

contour tree at each node but rather only keep enough information for gradient routing. Thus, local optima due to noises in the measurement only influence a small neighborhood and are ‘invisible’ to queries from far away regions.

### 9.2.1 Notations

Before we describe the algorithm, we first state conceptually what we want to achieve with the contour tree construction and what we want to store at each node. An example of a contour tree is given in Figure 9.3 (i). A node  $w$  on an arc  $AB$  has a contour component  $C(w)$  in between  $C(A)$  and  $C(B)$ . The contour component  $C(w)$  decomposes the entire signal field into two components, the *interior* and the *exterior*, corresponding to the two subtrees when  $w$  is removed. The interior contains the critical point  $A$ , which is reachable from  $C(w)$  via a gradient ascending path. We call  $A$  the *ascending saddle*. The exterior contains the critical point  $B$ , which is reachable by a gradient descending path.  $B$  is called the *descending saddle*. Not every node has both ascending/descending saddles. Now we will state what is needed to store



**Figure 9.3.** (i) A contour tree and the interior of  $C(w)$  shown in the bounded region; (ii) merge tree; (iii) split tree.

at each node for gradient descent routing with success.

At a node  $w$  (not on the contour component of a saddle), we will store four values:

- $I^+(w), I^-(w)$  correspond to the maximum,  $I^-$  and minimum value in the interior of  $C(w)$ ;
- $E^+(w), E^-(w)$  correspond to the maximum and minimum value in the exterior of  $C(w)$ .

This information is to guarantee that when we send a query message either uphill or downhill, we know for certain that there exist some contours for which we are looking.

For the consideration of easy navigation with the contour tree, each node also keeps information about the contours that split off/merge together at their *ascending merge saddle* or *descending split saddle*. Take point  $p$  on the arc  $P_2A$  and its descending split saddle  $A$  in Figure 9.3 (i) as an example. The contour component  $C(A)$  is the union of two contours  $C_1(A)$  and  $C_2(A)$  splitting up soon. Thus we keep at each node  $u \in C(p)$ ,

- the maximum/minimum values of the interior/exterior of both  $C_1(A)$  and  $C_2(A)$ ;

- gradient descending pointers leading to  $C_1(A)$  and  $C_2(A)$ .

This information helps us decide before we reach a saddle contour, whether it is worth visiting one or two of the contour components that split off of it and if so, how to get there.

To summarize, each node only keeps a small constant amount of information. Next we will explain how to get this information. In the rest of this chapter we assume a dense deployment of sensors in which each node  $u$  has an value  $\mathcal{F}(u)$ . The nodes have a communication graph  $G$  that models the pairs of nodes who can directly communicate with each other. We assume no two sensors have the same values, if they do, ties are broken by their IDs (the one with higher ID is considered larger).

## 9.2.2 Sweep to identify saddle points

The construction of the contour tree and the spread of information about the peaks/valleys of the signal field are conducted by a sweep algorithm, similar to the one in [143]. Without loss of generality, we explain the details with the sweep top down. A node has its *higher (lower) neighbors* as the subset of neighbors with value strictly higher (lower) than itself.

Each sweep is initiated and labeled by a critical node (a maximum, minimum or a saddle node). A node identifies itself as a local maximum if it discovers that all its 1-hop neighbors have value no greater than itself. It then initiates a sweep top down. The sweep algorithm runs in a distributed fashion on all the nodes. A node has two possible states, *swept* and *not swept*. Each local maximum node initializes itself as a swept node. When a node has all of its higher neighbors in the swept state, it changes itself to be swept. The nodes who participate in the sweep do not need to be synchronized and advance the sweep frontier with their local knowledge.

In the sweep initiated by a local maximum  $p$ , the sweep message carries the tuple  $(p, \mathcal{F}(p))$ , i.e., the node ID and value of  $p$ . Each node being swept will keep this information, as well as from which nodes it received this information. We define a *descending path* as a path in which each node has a value no greater than its precedent. During the sweep the information about a local maximum  $p$  is propagated along descending paths from  $p$ . In addition, each node swept learns ascending pointers which eventually lead to the local maximum.

If a node gets two sweep messages from different local maxima, this indicates that two contour components start to merge. Thus a saddle should be identified. Since the nodes advance the sweep frontier in a distributed fashion, it may happen that two nodes at the same time both receive the sweep messages from two peaks. Thus we will need to define a saddle rigorously and resolve the ambiguity.

**Definition 9.2.1.** *We define a node to be a merge saddle node if it is the one with highest iso-value with two descending paths from different critical points (other merge saddles or local maxima), i.e., it receives two sweep messages from different critical points.*

Notice that this definition is recursive in nature. A merge saddle is precisely the first node when two contour components merge, as shown below.

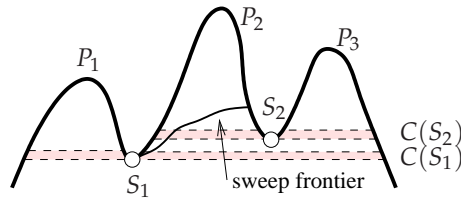
**Lemma 9.2.2.** *For a merge saddle  $q$  of two critical nodes  $p_1, p_2$ , if we remove the sensors with values strictly smaller than  $\mathcal{F}(q)$ , and obtain a subgraph  $G'$ , then  $q$  is the cut node of  $G'$  (removing  $q$  will result in two or more disconnected components.).*

**Proof:** First, define  $L_1$  ( $L_2$ ) as the set of nodes in  $G'$  with ascending paths to  $p_1$  ( $p_2$ ). We claim that  $L_1$  and  $L_2$  has only node  $q$  in common. If otherwise,  $q \neq q' \in L_1 \cap L_2$ . Since  $q'$  is a node in  $G'$ ,  $q'$  has a higher value than  $q$ . Now this contradicts with the definition that  $q$  is the saddle node.

Now we argue that once  $q$  is removed from  $G'$ , then the set of nodes  $L_1$  is disconnected from  $L_2$ . Suppose otherwise, that there are two nodes  $x_1 \in L_1, x_2 \in L_2$  and a path  $\mathcal{P}$  connecting them that does not go through  $q$ . This path  $\mathcal{P}$  must use nodes other than those in  $L_1 \cup L_2$ . Now take the first node on  $\mathcal{P}$  coming out of  $x_1$  that is not in  $L_1 \cup L_2$ , denoted by  $y_1$ . Without loss of generality we can also assume that  $y_1$  is just next to  $x_1$  (otherwise take  $x_1$  to be the preceding node of  $y_1$ ). Now we must have  $\mathcal{F}(y_1) > \mathcal{F}(x_1)$ , since  $y_1$  is not in  $L_1$ . Now take an ascending path from  $y_1$ , it will lead eventually to either a local maximum or a saddle, denoted by  $p_3$ . Thus the node  $q$  cannot be a merge saddle with  $p_1, p_2$ , since there will be another saddle of  $p_1$  and  $p_3$ , which is at least higher than node  $y_1$  and  $q$ . This shows a contradiction.  $\square$

We also remark that with a top-down sweep we do not identify the saddle when one contour component splits into two – the split saddles will be discovered when we do a bottom-up sweep from local minima, in a completely symmetric fashion.

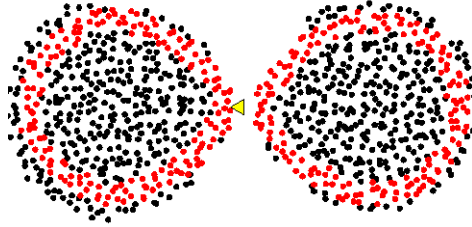
Now we show how the merge saddle node is identified in a robust and efficient way. A node who is not a local minimum and first receives two sweep messages from different peaks  $P_1, P_2$  will promote itself to be a *potential merge saddle*  $S(P_1, P_2)$ . In a distributed setting we need to worry about two issues: (i) two nodes  $u, v$  (or more) may become potential merge saddles  $S(P_1, P_2)$  for the same two peaks. In this case only the real saddle node (the one with highest isovalue) should survive. (ii) it may happen, if the sweep frontier does not proceed in the same speed, that the lower saddle may be discovered before the higher saddle, as shown in Figure 9.4.



**Figure 9.4.** If the sweep from  $P_2$  proceeds faster and reaches  $S_1$  before it reaches  $S_2$ , then  $S_1$  will notice it is a potential saddle for  $S(P_1, P_2)$ . The correct contour tree should have the saddle  $S_1$  to be the merge saddle for  $P_1, S_2$ .

The two problems will be resolved by the traversal of contour component, described below. Once a node  $u$  becomes a potential saddle for two peaks  $p_1, p_2$ , it starts to traverse the contour component  $C(u)$ , defined as,

**Definition 9.2.3.** The contour component  $C(u)$  for a node  $u$  is defined as the set of nodes that have values above  $\mathcal{F}(u)$  and have a lower neighbor lower than  $\mathcal{F}(u)$ . If  $u$  is a merge saddle for two critical points  $p_1, p_2$ , then  $C(u)$  is partitioned into two components  $C_1(u), C_2(u)$  (sharing only node  $u$ ) that have ascending paths to  $p_1, p_2$  respectively.



**Figure 9.5.** A merge saddle (shown as the triangle) and its contour component (in red).

Thus a potential saddle node  $u$  sends the tuple  $(p_1, p_2)$  to the nodes in  $C(u)$ . To resolve the first issue that several potential saddle nodes compete for the real saddle, the traversal message from  $u$  is suppressed if it hits a node  $x$  with a traversal message from a winning potential saddle  $u'$  (with higher value) for the same two peaks.  $x$  stops forwarding the message from  $u$ . Thus the traversal from  $u$  will stop because it either visits all the nodes in  $C(u)$  or if  $u$  loses to some other potential saddle.

During the message dissemination, a backward pointer is cached at each node in the traversal. Thus a tree rooted at  $u$ , named  $T(u)$ , is established and used for information aggregation and for  $u$  to learn about whether it wins and becomes the real saddle, or whether it loses the competition. In particular, a leaf  $w$  in this aggregation tree will return to its parent ‘loser’ if  $w$  has another winning traversal message, otherwise return the sweep message it has received. If a node is not yet swept, it waits for its sweep message before it reports back. An interior node in the aggregation tree returns to its parent the union of the messages from its children. Now the potential saddle node  $u$  becomes the real saddle for  $p_1, p_2$ , if (i) it does not get the ‘loser’ message from its aggregation tree; (ii) all nodes in  $C(u)$  are swept by  $p_1$  or  $p_2$ .

The new saddle  $q$  will start with a new top-down sweep and they propagate the tuple  $(q, \mathcal{F}(q), M(p_1, p_2))$ , where  $M(p_1, p_2)$  indicates that  $q$  is the merge saddle of two critical points  $p_1, p_2$ . All the nodes in  $C(q)$  are considered swept by  $q$  and the new sweep moves forward.

Notice that the sweep from a merge saddle  $q$  is distinct from the sweeps from  $p_1, p_2$ . In fact, the merge saddle  $q$  and all the nodes who receive the traversal message from  $q$  do not forward the sweep from  $p_1$  or  $p_2$  anymore. In the case when a node  $w$  has already forwarded the sweep from  $p_1$  or  $p_2$  by the time it gets the traversal message, it simply participates in the new sweep of  $q$ . Notice that again we do not require synchronization. The old sweeps from  $p_1$  and  $p_2$  cannot propagate very far from  $C(q)$ , since  $q$  stopped participating; thus,  $q$ ’s lower neighbors cannot possibly be swept, and so on and so forth.

If the merge saddle  $q$  also happens to be a local minimum (in a setting with low discrete resolution),  $q$  is in fact a merge saddle, a split saddle, and a local minimum all by itself. One trouble this may potentially cause is that the old sweeps from  $p_1$



and  $p_2$  may propagate without being dragged behind by  $q$ , since  $q$  does not have lower neighbors. The system however, will eventually arrive at the correct state, since the sweep from saddle  $q$  will overwrite the old sweeps from  $p_1, p_2$ .

The traversal also resolves the second issue mentioned above. In particular,  $S_1$  cannot win before the saddle  $S_2$  successfully identifies itself and proceeds with its sweep — this is because  $S_1$  will only get its aggregated message when all the nodes in  $C(S_1)$  have been swept, and  $S_2$  and its descendants cannot be possibly swept before the saddle  $S_2$  is done. During the aggregation phase for  $S_1$ ,  $S_1$  will learn about the sweep messages on  $C(S_1)$ . A subtle issue is that some nodes in  $C(S_1)$  may consider them swept by  $P_2$  and report  $P_2$  back to  $S_1$ . Thus  $S_1$  learns that some of the nodes are swept by  $S_2 = M(P_2, P_3)$  and some nodes are swept by  $P_2$  or  $P_3$  alone. Now  $S_1$  un-sweeps the nodes only swept by  $P_2$  or  $P_3$  and will only proceed to be the real saddle for  $P_1, S_2$  when all the nodes are swept by  $P_1, S_2$ . In this case  $S_1$  is initially proposed to be a saddle for  $P_1, P_2$  but eventually becomes a saddle of  $P_1, S_2$  when it wins.

To summarize, If there are two nodes both identifying themselves as a merge saddle, then the one with lower value will be swept and corrected (i.e., removed) eventually.

**Lemma 9.2.4.** *With the algorithm above, there cannot be two nodes both identifying themselves as the merge saddle of two critical nodes. Thus the algorithm defines a unique contour tree structure.*

After the top-down sweep, we have identified all merge saddles. By symmetry, we perform another sweep bottom-up initiated by local minima. Thus, after both sweeps we identify all saddle points and all nodes on the contour components of these saddles, thereby obtaining inherently the entire contour tree structure.

### 9.2.3 Construction of the contour tree

In this section we will extract the combinatorial contour tree after the saddles are identified. Notice that during top-down and bottom-up sweeps we have identified the merge tree (on all local maxima/minima and merge saddles) and the split tree (on all local minima/minima and split saddles). We will combine them to the contour tree such that each critical node learns its parent/child on the tree. Figure 9.3 (ii) (iii) shows the merge tree and split tree, respectively.

We use descending and ascending paths to discover the contour tree. Starting from a merge saddle  $p = M(P_1, P_2)$ , we follow ascending paths towards  $P_1, P_2$  respectively. If the ascending path towards  $P_1$  reaches  $P_1$  before it hits any other critical contour level, then  $p$  will consider  $P_1$  its parent in the contour tree. If the ascending path towards  $P_2$  hits a split saddle contour  $S$ , then  $p$  will consider  $S$  as its other parent in the contour tree. Similarly  $p$  also sends a descending path and identify its child in the contour tree. The operations for a split saddle, maximum/minimum are very similar and not repeated.

The operations require that an ascending path does not cross a split saddle contour without noticing it. This is guaranteed by the definition of a contour component. Suppose that in an ascending path  $x$  has value  $\mathcal{F}(x) < \mathcal{F}(q)$ , with  $q$  as a split saddle node, and the next node on the path  $y$  has value  $\mathcal{F}(y) \geq \mathcal{F}(q)$ . Thus  $x$  must be on the saddle component  $C(q)$ , because  $x$  has a value below  $\mathcal{F}(q)$  and has a neighbor  $y$  above it. This guarantees that the contour tree will be detected precisely as the combination of the merge tree and the split tree.

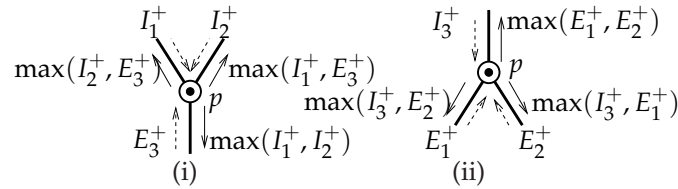
## 9.2.4 Information dissemination

With the contour tree constructed, we will need to disseminate information such that each node  $w$  learns

1. the maximum/minimum value,  $I^+(w), I^-(w)$ , inside the *interior* of its contour component  $C(w)$ ;
2. the maximum/minimum value,  $E^+(w), E^-(w)$ , inside the *exterior* of  $C(w)$ .

This is done by information dissemination along the contour tree. By symmetry, we first explain how a node  $w$  learns about the maximum value inside the interior/exterior of its contour component. Suppose that  $w$  is on an arc  $AB$ . Recall that the interior of  $C(w)$  corresponds to the subtree containing the ascending neighbor  $A$ , when  $C(w)$  is removed. Thus the maximum of the exterior (interior) of  $C(p)$  for a local minimum (maximum)  $p$  is its own value.

We explain the basic operation by using the contour tree. For an arc  $e$ , the removal of  $w \in e$  leaves two subtrees  $T_1$  and  $T_2$ , the maximum value in  $T_1$  is sent through the arc, by a sweep, to  $T_2$ , and vice versa. In particular, we specify the dissemination rules at saddle points. See Figure 9.6. First, examine a merge saddle  $p$



**Figure 9.6.** Information dissemination on (i) merge saddle; (ii) split saddle.

with two incoming arcs  $e_1, e_2$  and one outgoing arc  $e_3$ . Suppose by induction that the maximum is already learned and propagated along the arcs  $e_1, e_2, e_3$  to saddle  $p$ , as shown by  $I_1^+, I_2^+, E_3^+$  in the figure. The contour component  $C(p)$  has two components  $C_1(p), C_2(p)$ , corresponding to the nodes with ascending paths along  $e_1$  and  $e_2$  respectively. Now for a node  $w$ ,

- if  $w \in C_1(p)$ ,  $w$  sends  $E_1^+ = \max(I_2^+, E_3^+, \mathcal{F}(w))$  along the bottom-up sweep of  $e_1$ ;
- if  $w \in C_2(p)$ ,  $w$  sends  $E_2^+ = \max(I_1^+, E_3^+, \mathcal{F}(w))$  along the bottom-up sweep of  $e_2$ ;
- if  $w \in C(p)$ ,  $w$  sends  $I_3^+ = \max(I_1^+, I_2^+)$  along the top-down sweep of  $e_3$ .



This says that the nodes on  $C(q)$  will initiate a sweep bottom-up along  $e_1$  and  $e_2$ , and a top-down sweep along  $e_3$  and propagate information as shown in Figure 9.6(i) accordingly. At a split saddle, information propagates in a similar way as shown in Figure 9.6(ii). We do not repeat here.

Notice that we do not need close synchronization among these sweeps. In particular, the bottom-up sweep on  $e_2$  in Figure 9.6(i) can start when both  $I_1^+$  and  $E_3^+$  are done, even if the sweep  $I_2^+$  is not finished yet.

The information dissemination phase is initiated by the local minima and local maxima. A local minimum  $p$  initiates a bottom-up sweep with value  $E^+(p) = \mathcal{F}(p)$ . A local maximum  $p$  initiates a top-down sweep with value  $I^+(p) = \mathcal{F}(p)$ . Along each arc there are at most two sweeps in different directions. In addition, the dissemination of both the minimum and the maximum can be integrated in the same sweep so that the total cost for this phase is roughly  $2n$ .

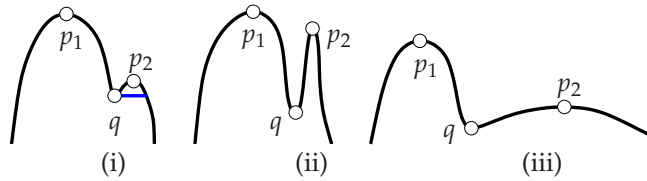
For navigation purposes, we will also disseminate information such that a node traveling in a gradient ascend path can easily find ways to each of the two peaks that will split up on the upcoming merge saddle  $p$  (so that we do not need to reach the saddle to decide). Specifically, each node on  $C_1(p)$  records its hop count within  $C_1(p)$  from the saddle  $p$ . This is called its index. For a node  $w$  with  $p$  as its ascending merge saddle, if  $w$  has higher neighbors with ascending pointers to  $p_1$ , then  $w$  has an ascending pointer to  $p_1$  and its index is the minimum of the indices of those higher neighbors. A node may have ascending pointers to both  $p_1, p_2$ , for example, the saddle node  $p$  itself and all the nodes with ascending paths to  $p$ . Similarly we disseminate the descending pointers along the ascending paths from a split saddle until the next critical contour. This information sweep can be combined with the previous sweep thus it does not incur extra cost.

To summarize, the total communication cost is bounded by the cost of sweeps, and the cost of traversing the saddle contours. In the ideal case when the saddle contours do not severely overlap and the sweeps are stopped in time by the saddle contour traversal, both the sweep cost and the saddle contour traversal cost are a constant factor of the network size. The construction cost in practice is evaluated in simulations.

## 9.2.5 Handing noises

With real sensor data, the signal field may have noises, causing lots of local optima. In practice we will de-noise the signal field by simplifying the contour tree during construction, to improve the construction efficiency. At a saddle node  $q$ , we will check the values of the two peaks  $p_1, p_2$ . Say  $\mathcal{F}(p_1) > \mathcal{F}(p_2)$ . If  $\mathcal{F}(p_2) - \mathcal{F}(q) < \varepsilon$  and  $q$  is at least  $\gamma$  hops away from  $p_2$ , with  $\varepsilon$  and  $\gamma$  as upper bounds on the height and size of a bump to be considered as noises, we consider  $p_2$  insignificant and chop it off. See Figure 9.7 (i). At the saddle  $q$ ,  $q$  will detect that  $p_2$  is too small, thus it will be chopped at the value of  $\mathcal{F}(q)$  and the sweep of  $p_1$  will take over.

The above operations effectively ‘smooth out’ the signal field, guided by local geometric measures. This can substantially simplify the contour tree in a noisy data



**Figure 9.7.** (i) a bump considered as noise and flattened; (ii) too high to be a noisy bump; (iii) too wide to be a noisy bump;

field. The gradient routing for iso-contour queries will miss at most some small components, whose sizes are controlled by  $\varepsilon$  and  $\gamma$ .

## 9.3 Iso-contour queries

### 9.3.1 Gradient descent routing for iso-contour queries

The invariant we constructed so far enables an efficient gradient routing for iso-contour queries with guaranteed success. The gradient descent algorithm uses only the information stored at a node and its immediate neighbors.

Starting at  $q$  we first check whether  $x$  is beyond the range of the signal field, in which case we do not travel even one step and immediately return  $\emptyset$ . Effectively, this is by checking whether  $I^+(q) < x$  and  $E^+(q) < x$ , or  $I^-(q) > x$  and  $E^-(q) > x$ . If not, we know that there must be some non-empty iso-contours at level  $x$  and we use a greedy gradient algorithm to find them. The main idea is to send the query message along the contour tree, possibly splitting at internal branches, and discover all components of the iso-contour of interest. At the query node  $q$ ,

- If  $I^+(q) \geq x \geq I^-(q)$ , then  $q$  initiates a query message to follow the gradient uphill.
- If  $E^+(q) \geq x \geq E^-(q)$ , then  $q$  initiates a query message to follow the gradient downhill.

We first explain the ascending query message from  $q$ . If a query message hits a node  $w$  with isovalue  $x$ , it will then start a traversal along the contour component  $C(w)$ . This is done by the same algorithm as explained earlier. At the same time, we also need to check at  $w$  whether it is worth getting even higher up — it is possible that at the interior of  $C(w)$  there are still contours of value  $x$ . Again this is done by checking a higher neighbor of  $w$ , say  $v$ , whether  $I^+(v) \geq x \geq I^-(v)$ .

For an ascending query message at a node  $w$ , suppose  $w$  stays on an arc with  $p$  being an ascending merge saddle. Then we will check for two parents of  $p$ , denoted by  $p_1, p_2$ , whether we will need to ascend on one peak or both of them. Luckily this information has been disseminated for all the nodes on this arc. Thus  $w$  will check the value range within the interior of  $C_1(p), C_2(p)$  respectively. If the query value  $x$  falls in the range,  $w$  will initiate an ascending query message for it. See the red query in Figure 9.8 as an example of two query messages, one for each peak.

For an ascending query message towards say peak  $p_2$ , if  $w$  has ascending pointers to  $p_2$ , this query message is simply delivered by gradient ascent routing, as the

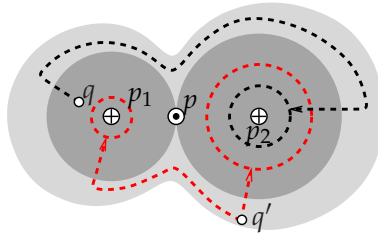


Figure 9.8. Examples of two queries.

query from  $q'$  shown in Figure 9.8. If not, then the query message will follow a contour at a random value (below  $\mathcal{F}(p)$  and above  $\mathcal{F}(w)$ ) and follow the index-decreasing path, in order to cross the ridge and discover some ascending paths to  $p_1$ . The descending query message is delivered in a symmetric manner looking for contour components at  $x$ . We may go a random number of hops further after ascending pointers are discovered, in order to avoid always using the nodes on the ridge. To summarize,

- The gradient routing algorithm is completely local and distributed and successfully finds *all* contour components at a given query level.
- Every step of the routing algorithm is *justified*, we send a query message only when we are sure there is something to be found. So no message will end up in vain.
- The routing scheme does not have to go through the saddles or follow critical contours, thus does not overload those nodes.

We note that this iso-contour query is the most basic query of a family of queries on iso-contours. Other iso-contour queries include: reporting the number of contours at value  $x$ , in particular, is there a single contour component? Range-limited queries (count/report contours within a value range)? These can be handled with either the iso-contour query as a subroutine, or by using a similar gradient routing algorithm. We omit the details here as the extension is relatively straight-forward.

### 9.3.2 Value restricted routing

The contour tree can be used for *value restricted routing*: given a source  $s$  and destination  $t$ , find a path  $\mathcal{P}$  from  $s$  to  $t$  such that at every node  $x$  on  $\mathcal{P}$ ,  $a \leq \mathcal{F}(x) \leq b$ , abbreviated as  $a \leq \mathcal{F}(\mathcal{P}) \leq b$ . Recall that the contour tree is produced by a retraction  $\mathcal{R}$  which maps every point on a contour component to a point on the arc in the contour tree. Thus we have:

**Lemma 9.3.1.** *For any path  $\mathcal{P}$  between points  $s$  and  $t$ , the image  $\mathcal{R}(\mathcal{P})$  in the tree contains the unique path  $\mathcal{P}'$  in the tree between  $\mathcal{R}(s)$  and  $\mathcal{R}(t)$ .*

**Theorem 9.3.2.** *A value restricted path exists in the network if and only if a value restricted path exists in the contour tree.*

**Proof:** In the following, we assume  $\mathcal{F}(s), \mathcal{F}(t) \in [a, b]$ , since otherwise the request is clearly invalid. First, a path  $\mathcal{P}'$  in the contour tree implies a path in the network. Since the tree is a retract of the domain, a path  $\mathcal{P}'$  in the tree is also a path in the network. Also it is possible to traverse from  $s$  to  $\mathcal{R}(s)$  and from  $t$  to  $\mathcal{R}(t)$  along  $C(s)$  and  $C(t)$  respectively. Appending these to  $\mathcal{P}'$  gives the required path.

For the other direction, a path  $\mathcal{P}$  in the network implies a path in the tree. Let  $\mathcal{P}'$  be the unique path between  $\mathcal{R}(s)$  and  $\mathcal{R}(t)$ . Then by lemma 9.3.1,  $\mathcal{P}' \subseteq \mathcal{R}(\mathcal{P})$ . Since  $\forall p, \mathcal{F}(\mathcal{R}(p)) = \mathcal{F}(p)$ , we have  $\max(\mathcal{P}') \leq \max(\mathcal{R}(\mathcal{P})) = \max(\mathcal{P})$  and  $\min(\mathcal{P}') \geq \min(\mathcal{R}(\mathcal{P})) = \min(\mathcal{P})$ .  $\square$

The results have a number of implications. The contour components on path  $\mathcal{P}'$  on the contour tree are ones that any path from  $s$  to  $t$  in the network must intersect. In moving from  $s$  to  $t$  along any path, we can keep record of number of times each component appears, or simply push and pop them on a stack. The ones remaining in the stack at the end constitute the path  $\mathcal{P}'$ . Thus, a value restricted path can be obtained by deforming any path connecting source and destination.

To answer the value restricted routing problem in a sensor network, if we disseminate the entire contour tree to every node, then a route in the network can be found in a greedy manner by following the contour components connecting the source and destination. If we do not store the entire tree at every sensor, we can develop a node labeling scheme, such that by using the labels of source and destination we can tell whether a path exists or not.

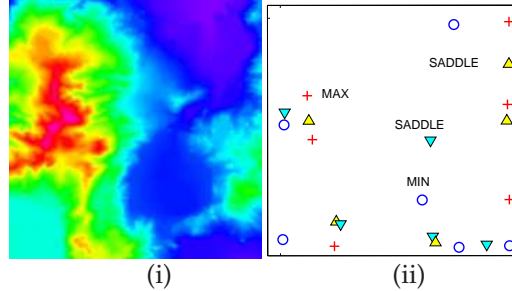
Given a contour tree with  $m$  vertices, we first do a balanced decomposition of the tree. For any tree there is a cut node, whose removal will leave subtrees each of size no more than  $2/3$  of the total vertices. Repeatedly partition each subtree to get a balanced decomposition of depth  $\log m$ . The label of a node  $u$  will be the concatenation of the IDs of all the cut nodes along the path from  $u$  to the root of the decomposition tree, as well as the max/min value of the paths from  $u$  to these cut nodes. Thus the size of the label is  $O(\log m)$ . For any two nodes  $s$  and  $t$ , by their labels, we can immediately find the lowest level cut node  $w$  shared by them. The path between them will necessarily go through  $w$ . Thus by taking the union of the range of the paths from  $s, t$  to  $w$ , we get the value range of the path connecting  $s, t$  in the contour tree.

With the labels pre-computed, each node  $p$  in the contour component will store the labels of its ascending and descending critical points. Thus one can use the labels of source and destinations to answer the value restricted routing requests.

## 9.4 Simulations

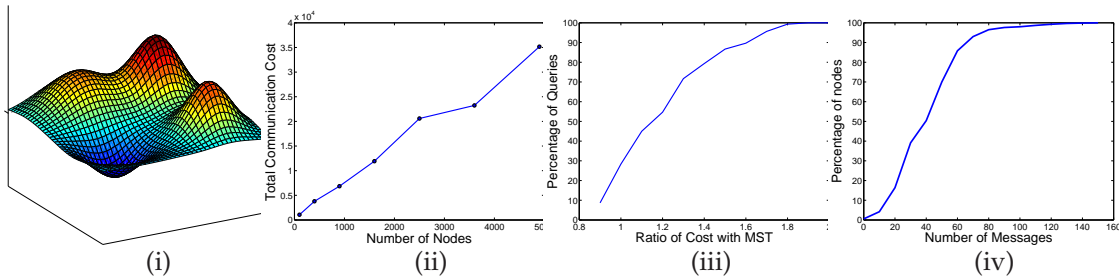
We implemented the algorithm for constructing contour tree and for answering iso-contour queries with gradient routing. Our simulations do not take into consideration many important networking details, e.g., packet loss, delay and channel contention. This set of simulations is a proof of concept and aims to verify the correctness of the algorithm and evaluate the feasibility of the approach on the algorithmic

level.



**Figure 9.9.** (i) Elevation map of West Reno (obtained from usgs.gov). (ii) The critical points discovered by our contour tree algorithm with a 2500 node sampling.

Unless specified otherwise, the simulation setup consists of 1600 nodes, deployed in a 16 units by 16 units square region with unit disk graph as the communication model. Nodes are deployed in a perturbed-grid distribution, where each node is assigned a random position within its grid square. The average number of neighbors per node is about 21. The sensors sample from a continuous signal field shown as in Figure 9.10 (i).



**Figure 9.10.** (i) The continuous signal field sampled by the distributed sensors; (ii) The message complexity of contour tree construction; (iii) The CDF of the ratio of our query cost v.s. the cost of MST; (iii) The CDF of the node load distribution.

### 9.4.1 Preprocessing cost for contour tree construction

We first evaluate the cost of contour tree construction. We vary the number of nodes with the same signal field and count the total number of messages, assuming a broadcast medium. In our implementation, a random node on the sweep frontier is selected to become swept. The number of messages grows linearly in the number of nodes as shown in Figure 9.10 (ii). The constant factor is about  $6 \sim 7$ .

### 9.4.2 Cost of iso-contour queries

We compare the cost of gradient routing versus a global solution of using the minimum spanning tree to connect the query node  $q$  and all the nodes on the iso-contour at value  $x$ , which is a 2-approximation of the minimum Steiner tree, the optimal (minimum cost) solution if the full knowledge of the signal field is available. We take 300 random queries with  $q$  randomly selected within the field of deployment

and the query value  $x$  randomly chosen between the global minimum and global maximum values. For each query, we take the ratio of our query cost versus the cost of MST (both in terms of number of hops). We calculated the cumulative distribution, i.e., the percentage of queries for which the ratio is below  $x$ , in Figure 9.10 (iii). Roughly all cases have a ratio below 2 and 80% of the queries have a ratio below 1.4.

### 9.4.3 Load balancing

In the same setup as the previous section, we plot the load on every node involved in the query procedure. A node is involved if it is on the routing path or is on the iso-contour to be queried. The maximum message load on any node is 148, the average message load is about 48.7. The load distribution is shown in Figure 9.10 (iv), in which 90% of the nodes have a message load of below 70.

## 9.5 Conclusion and future work

In this chapter we presented the distributed construction of a contour tree and its application in iso-contour queries by gradient routing with guaranteed delivery. Our future work is to update and maintain the contour tree for a time-varying signal field [36].

# Chapter 10

## Differential Forms for Tracking and Searching

Consider a static sensor field used to track and monitor a swarm of mobile targets, we develop distributed algorithms for in-network storage and range queries for aggregated data. For example, to return the number of targets within any user given region. Our scheme stores the target detection information locally in the network, and answers a query by examining the perimeter of the given range  $R$ . The cost of updating data about mobile targets is proportional to the target displacement. The key insight is to maintain in the sensor network a function with respect to the target detection data on the graph edges that is a *co-vector* field, also called a *differential one-form*. The integral of this one-form along any closed curve  $C$  gives the integral within the region bounded by  $C$ .

The differential one-form has great flexibility. The basic range query can be used to find a closest target or any given identifiable target with cost  $O(d)$  where  $d$  is the distance to the target in question. Dynamic insertion, deletion, coverage holes and mobility of sensor nodes can be handled with only local operations, making the scheme suitable for a highly dynamic network. Although we illustrate the major application of the differential forms for tracking swarms of targets, the same routine can be applied for organizing streaming scalar sensor data (such as temperature data field), to support efficient range queries. We demonstrate through analysis and simulations that this scheme compares favorably with existing schemes using location services for answering aggregated range queries of target detection data.

### 10.1 Introduction

Tracking of mobile targets is a major motivating application for sensor networks [63, 81, 142, 165]. Target tracking algorithms and architectures have been extensively investigated in the past few years. A few tracking systems have also been deployed and evaluated on real testbeds (such as [8, 67, 141]). A closely related problem is the design of the interface with which the target trajectories are accessed by the users. Arguably, the most adopted approach so far has the sensors that detect a target



record the detection event in the data logger or report it to a base station, where the target trajectories are assembled from the individual detections for post-experiment analysis. As sensor networks mature and become large scale, they intrude into the space where people live and work, the detection data needs to be processed in a timely manner, and accessed by users residing in the same physical space, sometimes with stringent delay requirements.

Consider the following scenario of wide-area deployment of sensors along major roads to track and monitor moving vehicles. A sensor can detect the position and velocity of a target within its sensing range [95]. A target may either have a distinct signal signature (e.g., a visual signature), or, for privacy protection, a non-identifiable signature (e.g., acoustic ones). The moving vehicles come in swarm as in the typical case of medium to heavy traffic situation. A user may use hand-held devices (smartphones, PDAs, etc) to communicate with nearby sensors or other portals and inquire for the target distribution. Of particular interest to us are *range* queries for *aggregated* data, for example, the level of traffic congestion in a specified neighborhood and its evolution over time. Formally, we ask a counting range query: what is the number of targets in any user-specified region  $R$ ? The topic for this chapter is to develop an efficient data processing and query scheme for such applications. A desirable solution should have low query delay, low communication costs, as well as low maintenance cost as the targets move.

In many practical scenarios, movements of targets are relevant only in the local region and for a short period of time. For example, some cars turning on a particular by-road is a relevant traffic information only while they are in the neighborhood. The communication and storage costs of updating a remote server are hard to justify for such fleeting pieces of data. A centralized solution also represents a single point of failure, is not resilient to attacks and is not efficient when handling many such updates. Very often, users might be in a neighborhood of where the data is generated. A centralized solution would require both the data and query from the users to be delivered to a (possibly) remote server. This leads to unacceptable delay and unnecessary network traffic.

Alternatively, the sensor in the proximity of a target can detect the target and can locally cache the detection event. This scheme has low maintenance cost as data is stored locally and only local updates are needed when target moves. But with such raw detection data stored directly in the network it is not easy to answer range queries. One has to flood all the nodes inside the range  $R$  to find out the total number, the communication cost of which is proportional to the area of  $R$ ,  $A(R)$ .

Both central and local data storage methods suffer from issues with sensing holes. Targets may enter and leave these holes at arbitrary times. It becomes difficult to track possible presence of targets in a hole unless entry and exit data are stored for long durations and carefully matched. This is even harder in the local caching scheme since the entry and exit may happen at different points on the boundary of the hole.

The solution we propose in this chapter uses local maintenance, but instead of storing raw detection data, stores target movements implicitly. It has a query cost proportional to the perimeter of  $R$ ,  $P(R) \ll A(R)$ . For this we use a novel notion

of differential one-form on the network. The key insight is to maintain in the sensor network a function on the edges that is the *co-vector* field with respect to target detection data. Thus integral along any closed curve  $C$  gives the integral of the region bounded by  $C$ . The details are introduced below.

**Differential one-form.** The tangent bundle on a manifold is the collection of all tangent vectors, along with the information of the point to which they are tangent. The co-tangent bundle is the dual of the tangent bundle, i.e., the vector space of linear functions on the tangent vectors. A co-vector field or differential one form that we consider is a *section* of the co-tangent bundle, that is, a co-vector defined at every point. In the discrete case as in the sensor network setting, a differential form can be defined on a cell complex, for example, a decomposition of the plane into non-overlapping faces by a planar graph. This particular idea of differential forms, while not common, can be found in mathematics literature [48,66,114]. The vertices, edges, and faces are called 0, 1, 2-cells respectively. Consider the simplest case, one target is located within a face  $f_0$  and has a weight of  $w$ . The differential one-form is to define a value  $\zeta$  for each directed edge. The value for  $ab$  is the negation of the value for  $ba$ . We maintain the property that for the face  $f_0$ , the summation of all the values of the edges on its boundary, in clockwise order, is  $w$ , and the summation of all the values of the boundary edges of all other faces is 0. This ensures that any cycle containing the face  $f_0$  will have a total summation of  $w$ , and any cycle not containing  $f_0$  will have a sum of 0. In other words, one is able to answer range queries by simply integrating the differential one-forms along the range boundary. The basic definition for one target can be generalized to multiple non-identifiable targets – such that the integral of a face is the total weight of the targets within the face. This way range query can be done for a swarm of targets with the same query cost. Using range queries we can implement the query for locating a closest target or a given identifiable target. The idea is to use exponentially enlarging range around the query node and once the range includes the target, reduce the range by using divide and conquer. The cost for such is bounded by  $O(d)$ , where  $d$  is the distance to the target in question, representing locality sensitivity.

The differential one-form has great flexibility that allows low maintenance cost under both network dynamics and target movements. When a target moves from one face  $f_0$  to an adjacent face  $f_1$ , we only need to update the differential one-form on the edge  $ab$  common to  $f_0, f_1$ . In particular,  $f(ab) \leftarrow f(ab) - w$ , for a target of weight  $w$ . This ensures that the property of the one-form is maintained. The cost for the update is a constant and can be done locally. Network dynamics such as link addition and removal, or node insertion and removal, can be handled in constant time. We also show that the differential one-form can be initialized in linear communication cost, i.e., constant cost per node. Further, this aids in energy management. Sensors only need to be active if there are targets nearby. A region of the network where there are no targets need not perform any communications to maintain tracking data, and can sleep or go to low power mode for extended periods.

Further, the method automatically handles sensing holes. A target’s movement in and out of a hole are recorded implicitly and gets processed automatically at the

time of query at no additional computation or storage. In fact, the network need not even be aware of the hole.

Although we present as the major application of the differential forms the tracking of targets in swarm, the same routine can be applied for organizing streaming scalar sensor data (such as temperature data field), to support efficient range queries.

## 10.2 Differential one-form on cell complexes

The network we consider consists of the nodes embedded in a region in the plane, and has an associated communication graph  $G$ . As a first step, we obtain a planar subgraph  $P \subseteq G$  that contains all the nodes, but is drawn in the plane without crossing edges. We can apply planarization techniques to extract a planar graph from the network connectivity graph. Such methods have been developed in the past [50,57,130,162]. Note that any such algorithm can be used for our purpose.

The vertices, edges and faces of the planar graph are the 0, 1 and 2 dimensional elements created by the planar graph. We refer to these as the 0-cells, 1-cells and 2-cells respectively. See Figure 10.1 for examples. The composition of the different dimensional cells covering the deployment region is called a cell complex. A more detailed treatment of cell complexes can be found in [66].

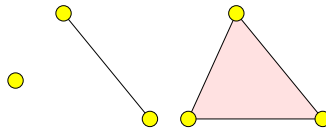


Figure 10.1. 0, 1, 2-cells.

Our goal is to track targets in the plane that move from face to face of the planar graph. We assume that all nodes know their locations and a sensor node can detect and locate a target in its sensing range. Various target detection schemes and signal processing primitives have been developed in the literature [95]. Our strategy assigns values to edges of the planar graph, and changes these values as the target moves. We introduce the following definitions and notations to represent the related faces, edges and values.

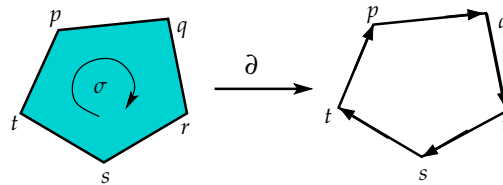
### 10.2.1 Boundaries and boundary chains

A face is demarcated by the edges or 1-cells that surround it. Such a set of edges form the *boundary* of the cell. For an edge  $pq$ , we use the ordered pair  $(p, q)$  to represent a directed edge whose direction or orientation is from  $p$  to  $q$ . Further, we use  $-(p, q)$  to represent the same edge with orientation  $(q, p)$ . For brevity, we can represent  $(p, q)$  and  $(q, p)$  as  $e$  and  $-e$  respectively. In a diagram, when an edge is labeled simply as  $e$ , an arrowhead is used to represent the intended orientation. The opposite orientation will naturally correspond to  $-e$ .

**Definition 10.2.1. Edge chain or 1-chain.** Suppose  $a, b, c \dots$  are oriented edges or 1-cells, then a chain on these edges is a formal sum  $\lambda_1 a + \lambda_2 b + \lambda_3 c + \dots$ , where each  $\lambda_i$  is an integer.

This chain simply signifies  $\lambda_1$  occurrences of  $a$ ,  $\lambda_2$  occurrences of  $b$  etc. The advantage of the summation notation will be clear in a short while. Note that in many cases we consider, the edges will be adjacent to each other and form a connected path. But this is not necessary in general, and the edges in an edge chain can in fact be any set of edges from the complex.

We can also associate orientations with 2-cells or faces. These correspond to traversing the boundary cycle of a face in some direction, clockwise or counter-clockwise. In this chapter we assume that all faces are oriented in the clockwise direction. Such a consistent orientation of cells is made possible by the fact that the 2-dimensional plane is *orientable* [84]. Thus, given a cell  $\sigma$  represented as an ordered tuple  $\sigma = (p, q, r, s, t)$ , as shown in Figure 10.2, we understand that the order corresponds to a clockwise traversal of edges  $(p, q)$ ,  $(q, r)$ ,  $(r, s)$ ,  $(s, t)$  and  $(t, p)$ . Correspondingly,  $-\sigma$  is the same cell with the opposite orientation,  $-\sigma = (t, s, r, q, p)$ . Observe that the orientation of a cell implies a specific orientation for each edge on its boundary.



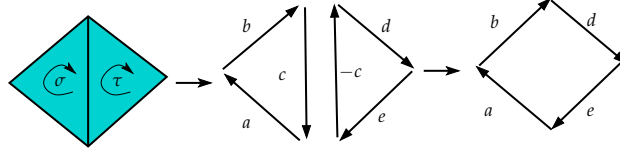
**Figure 10.2.** Action of boundary operator on a face  $\sigma$  will give a chain of its boundary edges with orientations inherited from the orientation  $\sigma$ .

**Definition 10.2.2. Boundary operator  $\partial$ .** The boundary operator  $\partial$  acts on a 2-cell or a face  $\sigma$  to produce a chain  $\partial(\sigma) = a + b + c \dots$  where  $a, b, c \dots$  are the edges on the boundary of  $\sigma$ , with orientations inherited from the clockwise orientation of  $\sigma$ . For a set of faces  $U = \{\sigma, \tau \dots\}$ , we extend  $\partial$  to operate on it as  $\partial U = \sum_{\sigma \in U} \partial \sigma$ .

The idea behind this definition is shown in Figure 10.3. The two neighboring faces  $\sigma$  and  $\tau$  have boundaries  $\partial \sigma = a + b + c$  and  $\partial \tau = d + e + (-c)$ , respectively. Note that a shared edge like  $c$  must always appear with opposite orientation, and therefore have opposite signs for the two faces. Thus the resultant boundary  $\partial\{\sigma, \tau\} = a + b + d + e$  is exactly the boundary of the union of two faces. This applies more generally to any set of faces. We refer the reader to [84] for more details on the algebra of chains.

## 10.2.2 One-forms and tracking forms

In this subsection we define functions over edge chains and show how they help in tracking a target.



**Figure 10.3.** Action of the boundary operator  $\partial$  on faces  $\sigma$  and  $\tau$  produces the boundary of the union of the two.

We consider a function  $f$  that assigns a value to each directed edge in the planar graph  $P$ . The function is defined to have the property that  $f(-e) = -f(e)$ . We extend this function to edge chains by making it distributive over summation:  $f(a + b + c + \dots) = f(a) + f(b) + f(c) + \dots$ . Let us refer to such functions as *one-forms* or *edge forms*. A one-form can also make sense on the faces of the planar graph, if we let it take the value on the boundary of that face, that is,  $f(\sigma) = f(\partial\sigma)$ .

Now suppose there is a single target  $T$  of weight  $w$  in the domain. Then at any given time this target resides in single unique face of the planar graph  $P$ <sup>1</sup>. Then we define a one-form on the faces and edges such that it is non-zero on this face and is zero on every other face:

**Definition 10.2.3. Tracking form  $\zeta$ .** A tracking form  $\zeta$  for a target  $T$  of weight  $w$  is a one-form such that

$$\zeta(\sigma) = \begin{cases} w & \text{if } \sigma \text{ contains } T \\ 0 & \text{otherwise} \end{cases}$$

Remember that on the face  $\sigma$  the form is defined to take a value equal to its sum on the boundary edges,  $\zeta(\sigma) = \zeta(\partial\sigma)$ . We can extend the form to a set  $U$  of faces by simple summation:  $\zeta(U) = \sum_{\sigma \in U} \zeta(\sigma)$ .

As a direct consequence of this definition, we know that to evaluate the presence of the target within a subset  $U$  of faces, it suffices to add the tracking-form  $\zeta$  on the faces in  $U$ . If a face in  $U$  contains the target  $T$ , then  $\zeta(U)$  sums to  $w$ , else it sums to zero. The following lemma implies that it is sufficient to sum the form  $\zeta$  only on the edges that form the boundary of the set  $U$  to obtain  $\zeta(U)$ .

**Lemma 10.2.4.** The sum of the form on the faces in a set  $U$  equals its sum applied only to the boundary of  $U$ , that is:  $\zeta(U) = \zeta(\partial U)$ .

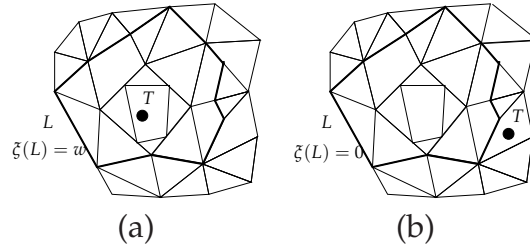
**Proof:** This follows directly from the definitions that

$$\begin{aligned} \zeta(U) &= \sum_{\sigma \in U} \zeta(\sigma) && \text{from definition 10.2.3} \\ &= \sum_{\sigma \in U} \zeta(\partial\sigma) && \text{from extension of } \zeta \text{ to faces} \\ &= \zeta\left(\sum_{\sigma \in U} (\partial\sigma)\right) && \text{by distributivity of } \zeta \text{ over } + \\ &= \zeta(\partial U) && \text{by definition 10.2.2} \end{aligned}$$

<sup>1</sup>The degenerate cases of the target being on an edge or a vertex can be resolved locally by a predetermined policy between the local nodes to assign the target to a face. Therefore, we ignore these cases to keep our discussion simple

□

The significance of this lemma becomes clear in Figure 10.4. Given any cycle  $L$  in  $P$ , it is possible to detect if the target  $T$  is inside the loop or not, by simply adding the tracking form along  $L$ . If  $T$  is in the interior, then  $\zeta(L) = w$ , and if  $T$  is not in the interior, then  $\zeta(L) = 0$ . In either case, the query does not need to visit the nodes in



**Figure 10.4.** Query for a target  $T$  inside  $L$ . (a)  $T$  is inside  $L$ , therefore  $\zeta(L) = w$ . (b)  $T$  is not inside  $L$ , therefore  $\zeta(L) = 0$ .

the interior of  $L$ . A simple walk on the loop suffices to find the answer. Further, this works exactly the same way for any arbitrary loop  $L$  and position of the target  $T$ .

**Multiple Targets.** This idea extends to any number of targets in the domain. Suppose targets  $T_1, T_2, \dots, T_k$  of weights  $w_1, w_2, \dots, w_k$ , individually give rise to tracking forms  $\zeta_1, \zeta_2, \dots, \zeta_k$ . Then we can construct a combined tracking form as the sum of these  $\zeta = \zeta_1 + \zeta_2 + \dots + \zeta_k$  on each edge. Given any loop  $L$ , the sum  $\zeta(L)$  will provide the total weight of targets inside  $L$ .

The weights assigned to targets can be adjusted to suit the needs of the system. For example, if all weights are equal, then  $\zeta(L)$  provides the count of targets inside. If each individual target  $T_i$  is given weight  $2^i$ , then from  $\zeta(L)$  it is possible to deduce exactly which ones are located inside  $L$ . This is equivalent to maintaining a form for each individual target. It is possible to imagine other scenarios where targets are assigned different weights according to their importance, for example, objects can be classified according to needs and weights assigned according to their types.

Note that given the weights and target locations, it is always possible to create a suitable tracking form. In the next section we will describe an efficient algorithm.

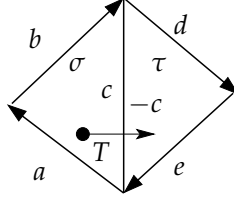
**Updating one-form for mobile targets.** When a target moves from one face to another, we need to update the tracking form by changing its value on the directed edges. Without loss of generality, we consider the example in Figure 10.5, where  $T$  moves from face  $\sigma$  to an adjacent face  $\tau$ . Let us say, the shared edge that was crossed by  $T$  appears as  $c$  in  $\partial\sigma$ , and as  $-c$  in  $\partial\tau$ . In the initial configuration, we had  $\zeta(\sigma) = w$  and  $\zeta(\tau) = 0$ . After the move, we need to have a final configuration with  $\zeta(\sigma) = 0$ , and  $\zeta(\tau) = w$ . This is achieved by the following simple modification to the form on the shared edge:

$$\zeta(c) := \zeta(c) - w \tag{10.1}$$

The same assignment can alternately be written from the point of view of  $\tau$  as:

$$\zeta(-c) := \zeta(-c) + w \tag{10.2}$$





**Figure 10.5.** Target  $T$  of weight  $w$  moves from face  $\sigma$  to face  $\tau$ . Modify  $\zeta(c) \leftarrow \zeta(c) - w$  to obtain the new form.

Evidently, these two are the same operation, since  $\zeta(-c) = -\zeta(c)$ .

This operation works for a system with any number of targets and any preexisting weights on the faces and edges. For a system with a single target, the final values are  $\zeta(\sigma) = 0$  and  $\zeta(\tau) = w$ , as required. In general, the weight of  $T$  is removed from the weight of  $\sigma$  and added to the weight of  $\tau$ .

## 10.3 Algorithms

In this section, we describe the algorithms for constructing the one-form, and for supporting range queries or other queries. First we compute the planar graph. The correctness of the basic scheme does not depend on the exact graph chosen. We only assume that the edges of the graph are short enough that they are ‘covered’ by their end-points, so that when a target crosses the edge, the event is detected by at least one sensor. In section 10.3.6 we remove this requirement, and in fact describe a tracking scheme for cases where sensors only detect the presence of a target, but do not know their locations.

### 10.3.1 Constructing a tracking form

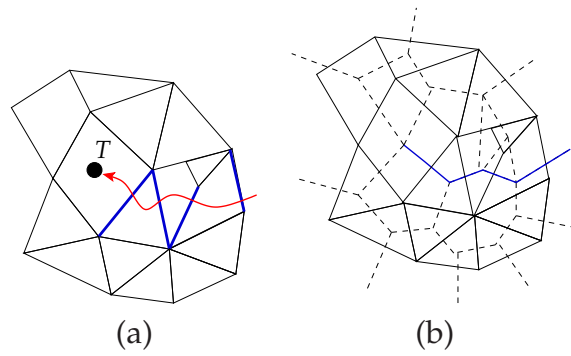
In this subsection, we show how to initialize a tracking one-form in the network. First, we describe the simple case where the network is empty of targets to start with, and all targets enter through the outer boundary. Next we will see that the ideas from this case provide a mechanism for initializing the more general case where targets may be present at the time of initialization.

**Starting with an empty field.** In this case, we initialize all edges to zero, that is for every edge  $e \in P$ ,  $\zeta(e) = 0$ . Now, suppose that a target  $T$  of weight  $w$  enters the network. It crosses the edge  $c \in \partial\tau$  to enter the face  $\tau$ . Then we modify  $\zeta(c) := \zeta(c) + w$ . Clearly, after this modification,  $\zeta(\tau) = w$ . As  $T$  moves, we can adaptively modify the form according to equation (10.1) or (10.2).

The process is shown in Figure 10.6(a). As the target moves from face to face, it modifies  $\zeta$  on the shared edges between adjacent faces. Creating a trail of edges with non-zero values.

Now, let us look a complex  $\bar{P}$  that is the dual complex of  $P$ . A vertex (say  $\bar{\sigma}$ ) in  $\bar{P}$  corresponds to a face ( $\sigma$ ) in  $P$ . An edge  $\bar{e}$  between vertices in  $\bar{P}$  represents the





**Figure 10.6.** The entry of a target  $T$  into the network. (a) As it moves from face to face, it leaves a trail of edges that it modified - shown in bold blue. (b) The trail in the dual graph. The edges of the dual graph are shown as dotted lines, and the dual trail of the target as a solid blue path.

shared edge  $e$  between corresponding faces of  $P$ . The trail of edges in  $P$  thus results in a dual trail, which is a path in  $\bar{P}$ , shown in Figure 10.6(b). For a more complete picture, we can regard the region outside of the planar graph as a *face at infinity*, and then the dual trail of  $T$  is a path from this face to the current position of  $T$ .

**Initializing a field with targets.** The idea of the dual trail directly leads to a simple algorithm to initialize targets in the field. We simply take a dual path to the face at infinity and add the suitable weight to edges of  $P$  whose dual are on the path.

More formally, for a target  $T$ , we select any simple directed path  $\alpha$  in  $\bar{P}$  from the current face of  $T$  to the face at infinity. If  $\bar{e} = (\bar{\sigma}, \bar{\tau})$  is on  $\alpha$ , and  $e \in \partial\sigma$ , then we do the following modification:

$$\bar{\zeta}(e) := \bar{\zeta}(e) + w, \quad (10.3)$$

where  $w$  is the weight of  $T$ . Quite clearly, any simple directed clockwise loop that contains  $T$  passes through one such edge. In cases where the loop has more than one such edges, the additional edges appear in oppositely oriented pairs and the values on them cancel out each other.

The following theorem shows that the algorithm above creates a correct tracking form.

**Theorem 10.3.1.** *Suppose we had  $\bar{\zeta}(\sigma) = u$ , then after the algorithm above is executed,*

1. *If a face  $\sigma$  contains target  $T$ , then  $\bar{\zeta}(\sigma) = u + w$*
2. *Else  $\bar{\zeta}(\sigma) = u$ .*

**Proof:** Suppose  $T \in \sigma$ , then  $\bar{\sigma} \in \alpha$  and has an outgoing edge  $\bar{e}$ . Therefore, after the algorithm is executed,  $\bar{\zeta}$  changes on  $e \in \partial\sigma$  by  $\bar{\zeta}(e) := \bar{\zeta}(e) + w$ . All other edges on  $\partial\sigma$  remain unchanged. Therefore, after the modification,  $\bar{\zeta}(\sigma) = u + w$ . This proves the first claim.

Suppose  $T \notin \sigma$ , if  $\bar{\sigma}$  is not on the trail  $\alpha$ , then of course nothing changes, and  $\bar{\zeta}(\sigma) = u$ . So, the only case we need to consider is when  $\bar{\sigma}$  is on the path  $\alpha$ . We know that  $\alpha$  is a path from the current face of  $T$  to the face at infinity, and  $\sigma$  is neither of these. Therefore,  $\bar{\sigma}$  has degree exactly 2 in  $\alpha$ . Suppose the incoming and outgoing

edges are  $\bar{e}_1$  and  $\bar{e}_2$  respectively. Then the algorithm will have made the following modifications :  $\zeta(-e_1) = \zeta(-e_1) + w$  and  $\zeta(e_2) = \zeta(e_2) + w$ . Therefore, the original sum  $\zeta(\sigma) = a + \dots + \zeta(e_1) + \zeta(e_2) + \dots = u$  remains unchanged :  $\zeta(\sigma) = a + \dots + (\zeta(e_1) - w) + (\zeta(e_2) + w) + \dots = u$ . This proves the second claim.  $\square$

Once again, the proof works for domains with multiple targets. We execute this once for each target in the domain or for each face containing targets with the total weight of these targets. Thus producing the correct form for initialization. The same procedure can be executed in case a target appears in the middle of the network at any time during the operation.

In cases where there are many targets in the field, creating a trail to the boundary for each can be expensive. In such cases, we perform the initialization as a sweep on the network. We discuss this further in section 10.3.7.

### 10.3.2 Containment queries

Given a one-form on the planar graph, we can query the number of targets inside any loop on the planar graph. In this subsection we extend it to queries of a geometric range. In the following we use the example of user specified squares. Other geometric ranges can be handled in a similar manner.

For now, we assume that the network is sufficiently dense so that every point within it is covered (sensed) by one or more sensors, in particular that every point in a face is within a small constant distance  $\delta$  of some vertex of the face. We also assume that the density is bounded, that is inside any disk of radius 1 the number of nodes is bounded by some constant  $k$ . This is not a very restrictive assumption. In a very dense network, we can select a sample of bounded density that still covers the region. We assume geographic face routing [77] is used to follow the faces that intersect a given geometric curve.

Let us use the notation  $S_p(r)$  to denote the square of side length  $2r$ , centered at point  $p$ . We sometimes use  $p$  to denote both a node and its location. We call the size of  $S_p(r)$  to be  $r$ . The goal is to compute the weight of targets inside this box, or equivalently, compute the sum of the tracking form on the boundary  $\partial[S_p(r)]$ .

Consider the faces of  $P$  that intersect this boundary. By the assumptions above, there are at most a constant number of these within a unit distance of any point on  $\partial S_p(r)$ . Therefore, the number of faces intersected by the boundary is  $O(|\partial S_p(r)|)$  or  $O(r)$ .

Let  $Q$  represent this set of faces at the boundary. For a sufficiently large box queried,  $Q$  is an annulus and  $\partial Q$  has 2 different connected components — say  $\partial Q = \beta + \gamma$  where each is a connected edge chain, in fact a cycle. One of these, say  $\gamma$  lies outside  $S_p(r)$  and  $\beta$  lies inside. We say that  $\gamma$  and  $-\beta$  respectively form the outer and inner approximations of  $\partial S_p(r)$ . The reason for taking  $-\beta$  is that  $\beta$  by default is oriented counter clockwise, therefore we reverse the orientation to match our conventions.  $\zeta(-\beta)$  gives a lower bound on the weight of targets inside the box, while  $\zeta(\gamma)$  gives an upper bound.

We can now find the answer to our query. First, we find  $\zeta(-\beta)$ . Next, for every face  $\sigma \in Q$ , we manually check the total weight of targets inside  $\sigma \cap S_p(r)$ . The sum of these values with  $\zeta(-\beta)$  gives the answer.

Note that this entire computation can be done in a distributed manner by a single walk along the cycle  $\partial S_p(r)$ . The size of the sub-complex induced by  $Q$  and therefore the cost of this computation is  $O(r)$ .

### 10.3.3 Search queries

In this section, we build an algorithm to answer queries of the type “Find the target nearest to node  $p$ .” The same method applies to searching for an identifiable target.

We search in two stages. First, we find the smallest box  $S_p(2^i)$  that contains a non-zero weight of targets. This is done by successively checking  $S_p(2^i)$  for  $i = 0, 1, 2, 3, \dots$ . Suppose the nearest target is at a distance  $d$ , then the size of the largest box tested in this process is  $2^{\lceil \lg(d) \rceil}$ . Now suppose the cost of checking a box of size  $r$  is bounded by  $ar$  for some constant  $a$ . Then the total cost of the test above is

$$a \sum_{i=0}^{\lceil \lg(d) \rceil} 2^i = O(d).$$

In the second step, we search this box recursively for the actual location of the target. We partition the box  $B_p(r)$  into four quads, each of size  $r/2$ , and check each of these for the presence of a target. Each test costs  $ar/2$ , therefore, the total test for 4 quads costs  $2ar$ . This is done recursively until we arrive at a node that ‘sees’ the target. Clearly, the cost of this recursive search is  $4ar(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots) = O(r)$ . Since  $r$  is at most  $2^{\lceil \lg(d) \rceil}$ , we have that the total cost of finding the nearest target is  $O(d)$ , that is of the order of the distance to the target.

Our query cost is sensitive to the distance to the target. Notice that whether we simply want to deliver a message to the nearest target or obtain the identity and location of it, the cost is  $\Omega(d)$ . Thus our query cost is asymptotically optimal.

### 10.3.4 Update costs

The network incurs a certain cost in updating the tracking form as a target moves. To be precise, every time the target moves from one face of  $P$  to another, the form on that edge has to be updated. Therefore, the total cost of the update equals the number of faces traveled by the target. By the arguments in section 10.3.2 as a target moves along a straight line segment of length  $d$ , the system requires  $O(d)$  updates at nodes. If updating an edge requires communication between the endpoints, then the communication cost is also  $O(d)$ . Note that in some cases this may not be necessary. If both the sensors can detect a target entering a face, which can happen for example if the sensing range covers the entire edge, then the target is sensed by both these sensors, and each can update their view of the edge without any mutual communication. In such cases, the update is carried out without any communication at all.

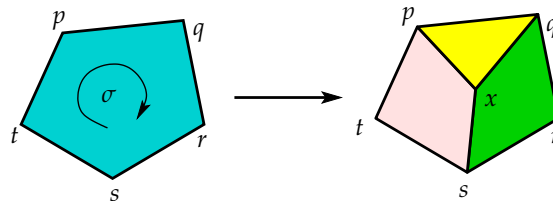
One can consider adversarial behavior, for example where a target repeatedly crosses an edge back and forth to induce many updates in the sensors. However, this sort of behavior is easy to detect, and can be handled simply. In such a case, the system stops updating that edge for some time. That is, the edge is assumed not to exist in  $P$  for that duration. Note that this ‘hole’ in the graph does not affect anything in the rest of the network at all. Updates and queries can proceed as usual. Later, the edge can be reinstated when appropriate.

In general, when a part of the network is very active with many and frequent movements, it is not very useful to track all such changes. Our scheme is sufficiently flexible and robust that tracking can be turned off in such regions without any loss to other parts or any overhead. Alternatively, it is possible to reduce the tracking resolution in that region by selectively removing nodes and edges so that the faces are larger and therefore incur fewer updates.

### 10.3.5 Network holes, fault tolerance and network dynamics

If a network has coverage holes, that does not affect the correctness of the tracking form. The ‘hole’ is treated as just another face, and the target entering that face does not induce any extra storage or communication. When trying to detect the weight of targets inside a box  $S_p(r)$ , precise estimates are impossible with any method if the boundary of the box possibly intersects an uncovered region, and it is not clear if a target is just inside or outside. We can however get upper and lower bounds (such as  $\zeta(\gamma)$  and  $\zeta(-\beta)$  in section 10.3.2) by computing the weights inside such uncovered faces. When initializing a network with large holes, these are simply disregarded, that is, the corresponding vertex does not exist in the dual. The dual trail for the initialization therefore never goes through the hole.

The scheme is also fault tolerant and adaptive to network dynamics. If some nodes fail, or all nodes in a region fail even including those near the target, that does not affect the correctness of the tracking form. Thus, this permits dynamic networks where nodes can be turned off without any overhead. Nodes can also be inserted into the network. This only requires refining the planar graph and the tracking form locally. See Figure 10.7 for an example.



**Figure 10.7.** Suppose a node  $x$  is inserted inside a face  $\{p, q, r, s, t\}$  of total weight  $w$  and the face is partitioned into three faces  $\{p, q, x\}$ ,  $\{q, r, s, x\}$ ,  $\{p, x, s, t\}$ , where the total weights within these faces are  $w_1, w_2, w_3$  respectively,  $w_1 + w_2 + w_3 = w$ . We simply set the values of the edges  $f(x, p) = 0$ ,  $f(x, q) = f(p, q) - w_1$ ,  $f(x, s) = f(p, q) + f(q, r) + f(r, s) - w_1 - w_2$ . One can verify easily that these values conform to the definition of a tracking form.

The effect of sensing noise is extremely local. Suppose an edge gets updated

incorrectly due to sensing or communication failure. This only affects the evaluation of loops that actually pass through that edge. All other loops still produce the correct results.

### 10.3.6 Tracking without target locations

Up to this point, we have assumed that the location of the target can be sensed by the nearby sensors. We now show how to modify the tracking scheme so that it can work without localization.

Suppose the target  $T$  is detected by exactly one sensor at a time. We initialize this scenario as follows. Suppose  $s$  is the sensor detecting  $T$ . Remove  $s$  (and all incident edges) from  $P$  to get a new planar graph  $P'$ . Then in  $P'$ ,  $T$  is assumed to reside in the new face with the neighbors of  $s$  on the boundary. Now, we can initialize the form as usual on the dual of  $P'$ . When the target moves from  $s$  to a neighboring node  $t$ , we first remove  $t$  from  $P'$  and then reinstate  $s$  and its edges using the method for inserting vertices.

The method naturally extends to cases where a target is detected by a set of sensors. In this case, we just remove all the detecting nodes, and when the target moves, we reinstate those that no longer detect it.

### 10.3.7 Aggregation of signal over all nodes

Beyond tracking moving targets, differential forms can also be used to compute aggregates of arbitrary functions sampled by sensor network. Suppose that  $h$  is an arbitrary function sampled by the network. Since we have a method for computing sums of values defined over faces of  $P$ , we adapt to make use of that existing method. For any node  $s$ , we apply small perturbation to the location. That is, the value  $h(s)$  is assumed to exist as an added weight in a face  $\sigma$  incident on  $s$ , that is  $\zeta(\sigma) \leftarrow \zeta(\sigma) + h(s)$ .

First, for every face  $\sigma$ , we find the initialization dual path  $\alpha$  to the face at infinity. We build these paths such that the reduced graph of these paths is acyclic. We can build these paths as the shortest hop-count paths by flooding from the face at infinity. Alternatively, these can be computed by ordering faces (dual nodes) from west-to-east. In either case, this creates an aggregation tree  $\mathcal{T}$  rooted at the face at infinity. Now, starting at the leaves of  $\mathcal{T}$ , we compute an aggregate at each interior node by summing its value with those of its children in the the aggregation tree. Let us denote this function on the dual nodes as  $\mu$ .

Now, for every node  $\bar{\sigma} \in \mathcal{T}$ , consider the outgoing edge  $\bar{e}$  and its dual  $e$  in the original graph  $P$ . We set  $\zeta(e) = \mu(\bar{\sigma})$ .

Note that this initialization can be executed as a single aggregation sweep on the tree  $\mathcal{T}$ . Therefore, it can be computed at a cost of  $O(n)$ .

Note that for the function  $h$ , this can give slightly erroneous results, since we perturbed the function to assign it to a face. However, this is easily rectified. Observe that for a loop not passing through  $s$ , the contribution of  $h(s)$  is estimated correctly.

We only need to adjust carefully for nodes lying on the given loop. This we do by means of another differential form, calculated on the fly.

Given a loop  $L$ , we compute  $\zeta(L)$  in a single walk around  $L$ . Let us say  $e$  is the first edge traveled along  $L$ , and say  $\sigma_1$  and  $\sigma_2$  are the faces adjoining  $e$ . Now, we choose points  $p_1 \in \sigma_1$  and  $p_2 \in \sigma_2$  respectively. We maintain two other one-forms  $\eta_1$  and  $\eta_2$ , these are the *winding numbers* around  $p_1$  and  $p_2$  respectively. That is, for any edge  $(u, v)$  on  $L$ , we add the clockwise angle  $\angle up_iv$  to  $\eta_i$ . Suppose without loss of generality that  $p_1$  is on the exterior and  $p_2$  is on the interior of the region bounded by  $L$ , then we have  $\eta_1(L) = 0$ , and  $\eta_2(L)$  will sum to  $2\pi$  or  $-2\pi$  depending on orientation of  $L$ . This tells us if the interior of the bounded region lies to the right or left of the oriented cycle  $L$ . With this information, we know in which cases the  $h(s)$  needs to be added or subtracted from the  $\zeta(L)$  computed. Note that the method allows the user to query the interior of a region as well as a region closed as a point set.

Further, this automatically detects orientation of  $L$  so that the user does not always need to supply a clockwise loop. If  $\eta_2(L) < 0$  then the given  $L$  is oriented counter clockwise, and following our convention we should take  $-\zeta(L)$  as the result.

**Changing values.** Unlike the case of mobile targets, if an arbitrary function  $h$  changes with time, local updates may not suffice. In particular, the local update scheme works only when the function has certain local conservation properties, such as when a change of  $\delta$  in a face always causes a change  $-\delta$  in an adjacent face.

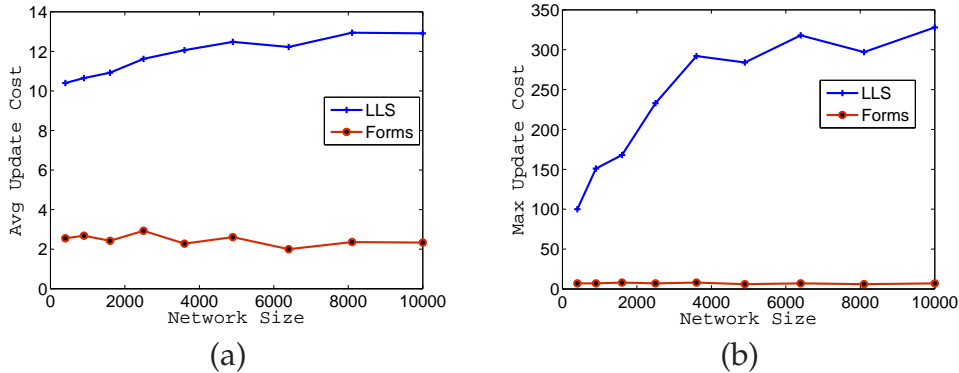
Instead we simply re-initialize the form at regular intervals or on sufficient changes. With an initialization of cost  $O(n)$ , we create a network-wide one-form with which we can find the aggregate in any region of the network.

### 10.3.8 Completely mobile networks

Consider a network where all nodes are mobile. That is, beyond the targets, the sensors themselves are mobile. Our method naturally extends to such scenarios. As a sensor moves, it may cross an edge of the planar graph. Suppose that  $s$  crosses an edge  $e$  to enter a face  $\tau$ . Then we update the network simply by first discarding all edges incident on  $s$ , then by inserting  $s$  into  $\tau$  as in Figure 10.7. Many existing planarization algorithms work for mobile networks [57]. We can use such methods to maintain the graph. In all cases, the removal of an edge will not incur a cost, the insertion of an edge will be made according to the idea in Figure 10.7.

Care needs to be taken in cases where we are considering forms to monitor values defined on nodes. For example, when a mobile network tracks its own nodes to be able to answer aggregate counts and weighted sums inside regions. Suppose in such a case  $s$  crosses an edge  $e \in \partial\tau$  to enter  $\tau$ . Then along with the usual insertion, the value  $h(s)$  must be assigned to one of the new faces, for example by  $\zeta(e) := \zeta(e) + h(s)$ .





**Figure 10.8.** cost per move of a target. (a) Average update cost per move. (b) Max update cost for any move.

### 10.3.9 Contour tree computation

The contour tree described in the previous chapter can also be computed using differential forms. The fundamental idea is that if a contour component encloses a set of maxima, then it also encloses the corresponding merge saddles. The same holds for the region bounded by any two contours.

To find the contour tree therefore, we need to count the number of maxima and minima inside a contour component, then search contour levels inside it for saddles. Also observe that given a contour component, we can as easily find the number of elements outside it. What is significant for contour tree computation is that a contour separates the plane into 2 components, and locally at the contour, one component represents the increasing direction of signal, one represents the decreasing direction of signal. In the following, we will refer to the increasing direction as the *inside*.

In the pre-processing, we create two different differential forms - for the maxima and the minima. Then the following procedure is executed.

From an arbitrary node  $p$ , build a monotone ascending path  $\mu$ , up to a maximum  $m$ . Next, count the number of maxima inside  $C(p)$ . If there is only one, then nothing is to be done. If there are more than one, then we do the following. Pick a node  $q$  in the middle of  $\mu$ . Count the number of maxima inside  $C(q)$ . If this number is the same as the number inside  $C(p)$ , that means there are no saddles in the regions bounded by these two contours. Otherwise the region contains saddle(s) and we repeat the procedure selecting a point in between  $p$  and  $q$  on the path.

Once we hit a saddle contour  $C(s)$  by this method, it will demarcate two inside components. One contains  $q$  and part of  $\mu$  and can be searched continuing the procedure as before. The other inside component can be searched by creating an ascending path from  $s$ . These searches run independently. Further, if the region bounded by  $C(p)$  and  $C(s)$  contains saddles, that can be searched independently as well. Note that while searching a region for merge saddles, we can simultaneously search it for split saddles.

At the same time, a search can be started from  $p$  with a descending path to construct the contour tree outside  $C(p)$ .



## 10.4 Simulations

We conducted extensive simulation tests to see how the theoretical guarantees of our algorithm translate to a network graph and compare with LLS [2] in performance, particularly in terms of communication costs. This section describes the findings. The simulations were done with networks that are quasi unit disk graphs<sup>2</sup> of inner radius  $1/\sqrt{2}$ . This choice of parameters allows local planarization algorithms [50, 130] to be used. The underlying sensor networks have nodes in a perturbed grid distribution, where the node is placed uniformly randomly in the grid box assigned to it. We consider networks without any significant coverage holes. In all cases, the average degree was about 10, and the size of the network was varied between 400 nodes and 10,000 nodes to test the scaling properties.

To evaluate the update costs, we introduce moving targets to the network domain. At each step, a target selects a random direction and moves up to a unit distance in that direction. After the move, the initial and final position are compared and updates are made.

**LLS scheme.** This is a locality aware location service for mobile networks. The principle here is to use location servers at different levels. At each level  $i = 0, 1, 2, 3, \dots$  the network region is tiled by squares of side  $2^i$ . The squares are aligned so that a square at level  $i$  is precisely covered by exactly 4 squares of level  $i - 1$ . In each square at each level, one node is designated to be the location server for that square, and keeps track of more precise locations of nodes in the square.

Location updates are performed in a certain lazy manner. Suppose mobile node  $p$  was in a square  $S_i$  at level  $i$ , and moves to a neighboring square at that level. The scheme does not update the location of  $p$  to the respective location servers. Instead, it waits until  $p$  has left this surrounding neighborhood of  $S_i$  before it actually performs an update. Thus, around  $S_i$  there is a ring of 8 squares moving where does not cause an update. As a compensation, LLS keeps its location information at the location servers of these nodes in addition to  $S_i$ . The idea here is to delay updates to avoid unnecessary communication. On average, if a node moves a distance  $d$ , then this scheme can be shown to have update costs of  $O(d \log d)$ . The cost is amortized. That is, the average cost is guaranteed to be low, but the update cost at a particular step can be arbitrarily high compared to the movement at that step.

The location search for a particular node starts at some other node in a network, and proceeds by searching nearby location servers at increasing levels. This goes on until some location server at the current or neighboring square for the current level claims to know the target location square at that level. Then the search proceeds in that square, successively searching lower levels. Of course, it is possible that due to the lazy update scheme, a server claiming to have the target is in fact in error. However in such a case, the target is guaranteed to be in one of the neighboring squares. It can be shown that this does not incur too high a cost. In fact, if the

---

<sup>2</sup>A quasi unit disk graph is one where nodes more than unit distance away do not have an edge, nodes less than a distance  $r$  away always have an edge, and for other distances, the presence of an edge is uncertain.

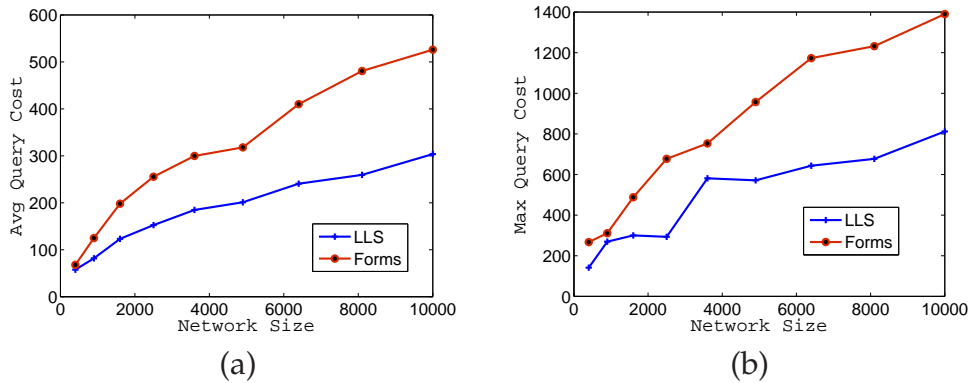


Figure 10.9. Search query costs. (a) Average cost per query. (b) Max cost for any query.

distance to the target is  $d$ , then the search finds the target at a cost of  $O(d)$ .

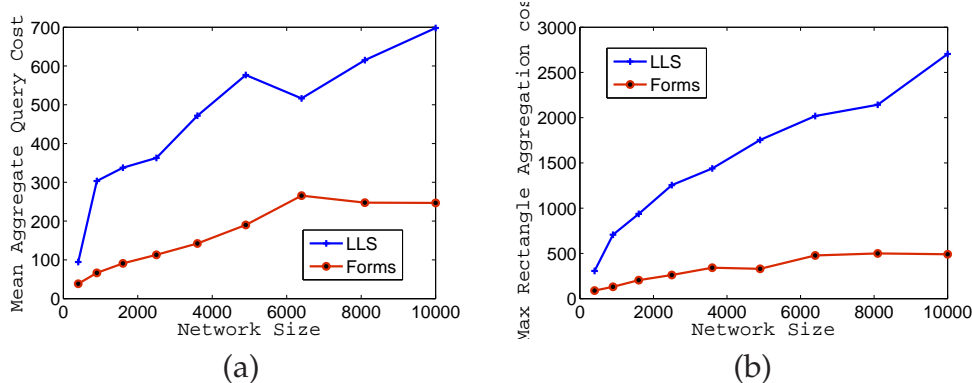
We compared costs with LLS in updates and query response. The following are the important observations:

- **Update costs.** Our algorithm adapts to node movements very efficiently. It has an average cost of about 2 messages per each small move (explained above) of the target, as compared to a cost of 10 to 12 messages for LLS. The maximum update cost for our scheme is about 7, while that for LLS is orders of magnitude higher — at 200 or 300 or more messages for a single small move. Most importantly, the costs of our scheme are independent of the network size, making it scalable to very large networks.
- **Search queries.** In answering queries where the one node searches for a specific target, our scheme performs slightly worse — consuming about 2 times the messages compared to *LLS*.
- **Aggregate range queries.** Given a geometric region such as a rectangle or ellipse, this query asks for the number of targets inside it. On this sort of queries, our scheme outperforms LLS by an order of magnitude.

### 10.4.1 Update costs

As a target moves, the tracking system has to update its data to be consistent with the current target position. LLS does this by suitably sending updates to its location servers, while our scheme changes the weights on the edges crossed by the target.

The results are shown in Figure 10.8(a). Our scheme is extremely efficient, since a small move does not cross too many edges, and the mean cost is about 2 per move. LLS is designed so that on certain moves, it does not require any updates. However, when the target has undergone sufficient displacement, it has to update several nearby lower level location servers - this incurs a reasonable cost. Later on, after further displacement, a move may require higher level servers further away to be updated, increasing the cost for that move, as well as the mean cost. The distance of the farthest server that may be tracking a target is proportional to the



**Figure 10.10.** Aggregation query costs for random rectangle regions. (a) Average Costs. (b) Max costs

network diameter. After a proportional displacement this server will need to be updated as well. Thus, the update costs of LLS depend on the network size, though the amortized cost of LLS is still quite manageable, at about 10 to 12 messages per move.

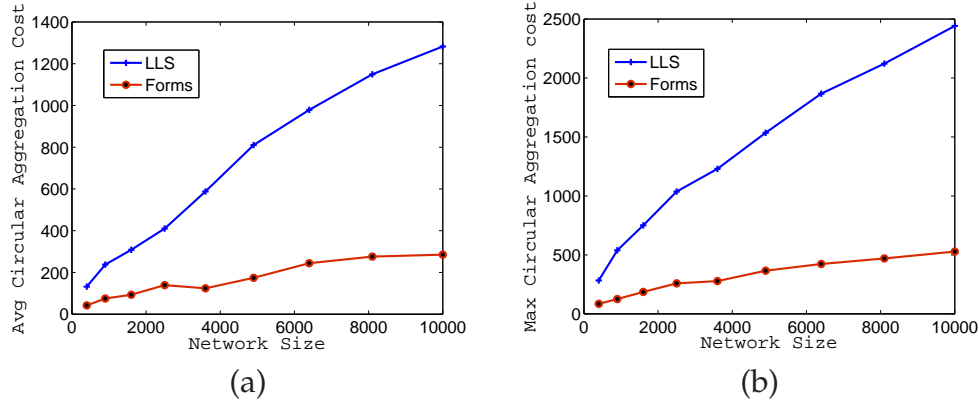
The worst case behavior of LLS is poor. This is because the strategy of avoiding updates until necessary means that the updates build up and on certain moves servers and neighboring servers at several levels of hierarchy need to be updated. Thus the update cost of a single move can go into several hundred messages (shown in Figure 10.8(a)). Our scheme, on the other hand, never has to update more than 8 edges.

Note that the costs in our scheme are taken to be proportional to the number of edge updates needed. In certain scenarios, where the target sensing does not require any communication, and when there is agreement among nodes on monitoring different parts of edges, it is possible to perform the updates at zero cost.

## 10.4.2 Search costs.

Location service schemes are designed to answer queries that ask for the location of a specific mobile target, or to deliver a message to the target. Our scheme of tracking forms on the other hand was designed with aggregate queries pertaining to groups of targets in mind. Nevertheless, we find that it is a good instrument for search of specific targets, and has performance comparable to the location service scheme. We can maintain a tracking form  $\zeta_i$  for each target  $T_i$  and then use that to search for it starting from the query node. The scheme is described in section 10.3.3.

In this experiment, we chose random query nodes, and random mobile targets. We execute a search for the target starting at the query node. The two schemes use analogous methods of searching exponentially growing regions for presence of the target, and in the suitable region searching exponentially smaller subregions until reaching the target. The asymptotic costs are the same for the two schemes. The simulation results in Figure 10.9 show that with tracking forms it costs about twice that of LLS to search.



**Figure 10.11.** Aggregation query costs for random circular regions. Costs for random ellipses show similar characteristics. (a) Average Costs. (b) Max costs

### 10.4.3 Aggregate range queries.

Given a region  $\mathcal{R}$ , say a rectangle or an ellipse, we wish to find the number of targets inside the region. With tracking forms, this is easy to do by summing the form in walk around the boundary. The details of the methods are in section 10.3.2. With a location sever scheme, the process is a little more complicated.

LLS maintains a quad-tree hierarchy, and recursively tracks nodes inside the quads at different levels. To find the aggregate, we need to look at quads of different levels that intersect with  $\mathcal{R}$ . In particular, if a quad  $Q$  intersects the boundary  $\partial\mathcal{R}$ , that means sub-quads of  $Q$  need to be analyzed further, to see which targets inside  $Q$  are actually inside  $\mathcal{R}$ . Therefore, the method boils down to finding quads at all levels that contain targets and intersect  $\partial\mathcal{R}$ . This turns out to be reasonably costly.

Figure 10.10 shows the costs when  $\mathcal{R}$  is a random rectangle inside the network region. Figure 10.11 shows the corresponding costs when  $\mathcal{R}$  is a random circle. Clearly, location server based schemes incurs a substantial cost in this type of query. Note that for target searching LLS using a different quadtree hierarchy for each target. This would be impractically expensive in this sort of query, where the presence of each target in  $\mathcal{R}$  will then have to be checked individually, driving the costs very high. We therefore used a common hierarchy where a location server can provide information about all targets in its quad region.

Even with this modification, the costs of our scheme are much lower, in principle only proportional to the size of the boundary of  $\mathcal{R}$ .

**Discussion.** In a network with mobile entities, it can be expected that a targets move often. Our scheme handles the movements very efficiently and locally. There is never any need to send updates to a distant point. This is also significant from power management point of view. If a target of interest is present in a part of the network, nearby nodes can be expected to be awake and actively monitoring it. If all movements are handled locally, then relatively distant nodes can sleep or go to low power mode to save energy without fear of interruptions.

## 10.5 Conclusions

In this chapter we presented the use of differential one-form in the application of target tracking and range queries. The method is simple, has low maintenance cost under target movement, is extremely flexible and robust to network changes and node mobility. The performance of our method is orders of magnitude better than previous location services schemes for tracking mobile targets. We expect that more applications can be found by using the differential one-form for a diverse set of queries of aggregated data, which remains as our future work.

# Chapter 11

## Conclusion Chapter

*...Our intuition is our most powerful tool. That is quite clear if you try to explain a piece of mathematics to a student or a colleague. You have a long, difficult argument and finally the student understands. What does the student say? The student says, "I see!" Seeing is synonymous with understanding...*

*Michael Atiyah  
Mathematics in the 20<sup>th</sup> Century*

The goal in each of the chapters were to construct distributed algorithms. These methods are more general, and effective on a wide range of platforms, as described in the introduction. This also makes the methods more robust. No particular node is substantially more significant than any other. The failure of a few nodes do not cause the setup to collapse. The overall workload is also more or less evenly distributed.

The challenge in a distributed method is to compute structures with global properties. This we achieved in the cases described here. In the questions of routing, this made it possible to route without explicitly finding routes for every other node in the network. In the questions of information processing, this made it possible to answer at a low cost, questions about aggregates, contours and individual data items anywhere in the network.

In general it is difficult to compute such global properties. Global properties requires global information, which is difficult to obtain and maintain without incurring large communication costs. That the nodes have small storage and computation abilities makes it impractical to store and compute large quantities of data frequently. The distributed computation therefore works best when nodes operate only on information from the local neighborhood. This provides two advantages. First, communication within the local neighborhood is efficient, therefore acquiring the local information is economical. Second, the local neighborhood is bounded in size, therefore the total information from this neighborhood can be stored, processed, and results communicated easily at a small cost. The ideal distributed algorithm therefore is one that extracts global properties from processing of local informations.

The local to global requirement is the reason that geometric methods work in finding distributed algorithms. Euler characteristic, Morse theory, Ricci flow, Gauss

Bonnet theorem, Stokes theorem are all examples of local-global relations that we used in constructing different distributed methods for networks. Spatial distribution is also in a sense a local-global relation. The distribution can be applied locally from each-node, oblivious to other nodes and without any need for coordination, and yet generates a structure with global properties.

The correlation with geometry makes it possible to use our visual intuition when designing distributed algorithms for sensors. As with any design and analysis process, the importance of pictures, diagrams and the power of the formidable visual center of the human brain should not be underestimated. That images and geometric constructs can be used to reason about distributed information makes it an attractive model of study.

This dissertation has a limited scope, and cannot cover all aspects of sensor network information processing. One important omission is perhaps the issue of communication failures and noises. In general, sensor data can be expected to be noisy. How to adapt techniques such as the contour tree method to such noises is not entirely clear. In general, adapting to noisy lossy data in a graceful manner is an interesting research challenge. Communication links, particularly wireless ones may be unreliable and transient. For a dense network, our essential assumption that to each node there is a small persistent neighborhood of low cost communication will hold, because it is likely that even when a link deteriorates, its end points will be connected through a short path within the neighborhood. But this may not be the case in a sparse network. In such a case, failure of a few links can entirely change the character of a network. How to adapt to such scenarios and take full advantage of live links is an interesting question. It will require a suitable abstraction that represents network topology and geometry compactly and distributedly, and can be updated suitably to reflect changes.



# Bibliography

- [1] Greenorbs project. <http://greenorbs.org/>.
- [2] I. Abraham, D. Dolev, and D. Malkhi. LLS: a locality aware location service for mobile ad hoc networks. In *DIALM-POMC '04: Proceedings of the 2004 joint workshop on Foundations of mobile computing*, pages 75–84, 2004.
- [3] I. Abraham, C. Gavoille, A. V. Goldberg, and D. Malkhi. Routing in networks with low doubling dimension. In *Proc. of the 26th International Conference on Distributed Computing Systems (ICDCS)*, July 2006.
- [4] I. Abraham and D. Malkhi. Name independent routing for growth bounded networks. In *SPAA '05: Proceedings of the seventeenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 49–55, 2005.
- [5] P. K. Agarwal. Range searching. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 31, pages 575–598. CRC Press LLC, Boca Raton, FL, 1997.
- [6] T. Alliance. Tinyos 2.1 adding threads and memory protection to tinyos. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 413–414, New York, NY, USA, 2008. ACM.
- [7] P. Angelini, F. Frati, and L. Grilli. An algorithm to construct greedy drawings of triangulations. In *Proc. of the 16th International Symposium on Graph Drawing*, pages 26–37, 2008.
- [8] J. Aslam, Z. Butler, F. Constantin, V. Crespi, G. Cybenko, and D. Rus. Tracking a moving object with a binary sensor network. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 150–161, 2003.
- [9] B. Awerbuch and D. Peleg. Concurrent online tracking of mobile users. In *SIGCOMM '91: Proceedings of the conference on Communications architecture & protocols*, pages 221–233, 1991.
- [10] X. Bai, S. Kuma, D. Xua, Z. Yun, and T. H. La. Deploying wireless sensors to achieve both coverage and connectivity. In *MobiHoc '06: Proceedings of the seventh ACM international symposium on Mobile ad hoc networking and computing*, pages 131–142, 2006.

- [11] B. A. Bash, J. W. Byers, and J. Considine. Approximately uniform random sampling in sensor networks. In *DMSN '04: Proceedings of the 1st international workshop on Data management for sensor networks*, pages 32–39, 2004.
- [12] P. Biswas and Y. Ye. Semidefinite programming for ad hoc wireless sensor network localization. In *Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks*, pages 46–54, 2004.
- [13] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. *Wireless Networks*, 7(6):609–616, 2001.
- [14] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Randomized gossip algorithms. *IEEE Transactions on Information Theory, Special issue of IEEE Transactions on Information Theory and IEEE/ACM Transactions on Networking*, 52(6):2508–2530, June 2006.
- [15] J. Bruck, J. Gao, and A. Jiang. Localization and routing in sensor networks by local angle information. In *Proc. of the Sixth ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'05)*, pages 181–192, May 2005.
- [16] J. Bruck, J. Gao, and A. Jiang. MAP: Medial axis based geometric routing in sensor networks. *Wireless Networks*, 13(6):835–853, 2007.
- [17] M. Caesar, M. Castro, E. B. Nightingale, G. O'Shea, and A. Rowstron. Virtual ring routing: network routing inspired by DHTs. In *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 351–362, 2006.
- [18] H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 918–926, 2000.
- [19] H. Carr, J. Snoeyink, and M. van de Panne. Simplifying flexible isosurfaces using local geometric measures. In *VIS '04: Proceedings of the conference on Visualization '04*, pages 497–504, 2004.
- [20] H. T.-H. Chan, A. Gupta, B. M. Maggs, and S. Zhou. On hierarchical routing in doubling metrics. In *SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 762–771, 2005.
- [21] M. B. Chen, C. Gotsman, and C. Wormser. Distributed computation of virtual coordinates. In *SCG '07: Proceedings of the twenty-third annual symposium on Computational geometry*, pages 210–219, 2007.
- [22] Y.-J. Chiang, T. Lenz, X. Lu, and G. Rote. Simple and optimal output-sensitive construction of contour trees using monotone paths. *Comput. Geom. Theory Appl.*, 30(2):165–195, 2005.

- [23] B. Chow. The ricci flow on the 2-sphere. *J. Differential Geom.*, 33(2):325–334, 1991.
- [24] B. Chow and F. Luo. Combinatorial ricci flows on surfaces. *Journal Differential Geometry*, 63(1):97–129, 2003.
- [25] M. Chu, H. Haussecker, and F. Zhao. Scalable information-driven sensor querying and routing for ad hoc heterogeneous sensor networks. *Int'l J. High Performance Computing Applications*, 16(3):90–110, 2002.
- [26] E. Cohen. Size-estimation framework with applications to transitive closure and reachability. *J. Comput. Syst. Sci.*, 55(3):441–453, 1997.
- [27] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate aggregation techniques for sensor databases. In *ICDE '04: Proceedings of the 20th International Conference on Data Engineering*, pages 449–460, 2004.
- [28] L. J. Cowen. Compact routing with minimum stretch. In *SODA '99: Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 255–260, 1999.
- [29] H. S. M. Coxeter. *Introduction to Geometry*. John Wiley & Sons, New York, 2nd edition, 1969.
- [30] M. de Berg and M. van Kreveld. Trekking in the alps without freezing or getting tired. *Algorithmica*, 18:306–323, 1997.
- [31] T. K. Delillo, A. R. Elcrat, and J. A. Pfaltzgraff. Schwarz-christoffel mapping of multiply connected domains. *Journal d'Analyse Mathématique*, 94(1):17–47, 2004.
- [32] R. Dhandapani. Greedy drawings of triangulations. In *SODA '08: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, 2008.
- [33] A. G. Dimakis, A. D. Sarwate, and M. J. Wainwright. Geographic gossip: efficient aggregation for sensor networks. In *IPSN '06: Proceedings of the fifth international conference on Information processing in sensor networks*, pages 69–76, 2006.
- [34] D. Mumford, C. Series, and D. Wright. *Indra's Pearls*. Cambridge University Press, 2002.
- [35] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *LCN '04: Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, pages 455–462, Washington, DC, USA, 2004. IEEE Computer Society.
- [36] H. Edelsbrunner, J. Harer, A. Mascarenhas, and V. Pascucci. Time-varying reeb graphs for continuous space-time data. In *SCG '04: Proceedings of the twentieth annual symposium on Computational geometry*, pages 366–372, 2004.

- [37] T. Eilam, C. Gavoille, and D. Peleg. Compact routing schemes with low stretch factor. *J. Algorithms*, 46(2):97–114, 2003.
- [38] D. Eppstein and M. T. Goodrich. Succinct greedy graph drawing in the hyperbolic plane. In *Proc. of the 16th International Symposium on Graph Drawing*, pages 14–25, 2008.
- [39] T. Eren, D. Goldenberg, W. Whitley, Y. Yang, S. Morse, B. Anderson, and P. Belhumeur. Rigidity, computation, and randomization of network localization. In *Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'04)*, volume 4, pages 2673–2684, March 2004.
- [40] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next century challenges: Scalable coordination in sensor networks. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, pages 263–270, August 1999.
- [41] Z. A. Eu, W. K. G. Seah, and H.-P. Tan. A study of mac schemes for wireless sensor networks powered by ambient energy harvesting. In *WICON '08: Proceedings of the 4th Annual International Conference on Wireless Internet*, pages 1–9, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).
- [42] Q. Fang, J. Gao, and L. Guibas. Locating and bypassing routing holes in sensor networks. In *Mobile Networks and Applications*, volume 11, pages 187–200, 2006.
- [43] Q. Fang, J. Gao, L. Guibas, V. de Silva, and L. Zhang. GLIDER: Gradient landmark-based distributed routing for sensor networks. In *Proc. of the 24th Conference of the IEEE Communication Society (INFOCOM)*, volume 1, pages 339–350, March 2005.
- [44] J. Faruque and A. Helmy. RUGGED: Routing on fingerprint gradients in sensor networks. In *IEEE Int'l Conf. on Pervasive Services (ICPS)*, pages 179–188, July 2004.
- [45] J. Faruque, K. Psounis, and A. Helmy. Analysis of gradient-based routing protocols in sensor networks. In *IEEE/ACM Int'l Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 258–275, June 2005.
- [46] R. Flury and R. Wattenhofer. MLS: an efficient location service for mobile ad hoc networks. In *MobiHoc '06: Proceedings of the seventh ACM international symposium on Mobile ad hoc networking and computing*, pages 226–237, 2006.
- [47] R. Fonesca, S. Ratnasamy, J. Zhao, C. T. Ee, D. Culler, S. Shenker, and I. Stoica. Beacon vector routing: Scalable point-to-point routing in wireless sensor networks. In *Proc. of the 2nd Symposium on Networked Systems Design and Implementation (NSDI)*, pages 329–342, May 2005.

- [48] W. Fulton. *Algebraic Topology: A First Course*. Springer, 1997.
- [49] S. Funke and N. Milosavljević. Guaranteed-delivery geographic routing under uncertain node locations. In *Proceedings of the 26th Conference of the IEEE Communications Society (INFOCOM'07)*, pages 1244–1252, May 2007.
- [50] S. Funke and N. Milosavljević. Network sketching or: “how much geometry hides in connectivity? - part II”. In *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 958–967, 2007.
- [51] S. Gandhi, J. Hershberger, and S. Suri. Approximate isocontours and spatial summaries for sensor networks. In *IPSN'07: Proceedings of the 6th international conference on Information processing in sensor networks*, pages 400–409, 2007.
- [52] D. Ganesan, D. Estrin, and J. Heidemann. DIMENSIONS: Why do we need a new data handling architecture for sensor networks. In *Proc. ACM SIGCOMM Workshop on Hot Topics in Networks*, pages 143–148, 2002.
- [53] D. Ganesan, B. Greenstein, D. Perelyubskiy, D. Estrin, and J. Heidemann. An evaluation of multi-resolution storage for sensor networks. In *Proceedings of the first international conference on Embedded networked sensor systems*, pages 89–102, 2003.
- [54] D. Ganesan, B. Greenstein, D. Perelyubskiy, D. Estrin, and J. Heidemann. Multi-resolution storage and search in sensor networks. *ACM Transactions on Storage*, 1(3):277–315, August 2005.
- [55] J. Gao, L. Guibas, J. Hershberger, and N. Milosavljevic. Sparse data aggregation in sensor networks. In *Proc. of the 6th International Symposium on Information Processing in Sensor Networks (IPSN'07)*, pages 430–439, April 2007.
- [56] J. Gao, L. Guibas, J. Hershberger, and L. Zhang. Fractionally cascaded information in a sensor network. In *Proc. of the 3rd International Symposium on Information Processing in Sensor Networks (IPSN'04)*, pages 311–319, April 2004.
- [57] J. Gao, L. J. Guibas, J. Hershberger, L. Zhang, and A. Zhu. Geometric spanners for routing in mobile networks. *IEEE Journal on Selected Areas in Communications Special issue on Wireless Ad Hoc Networks*, 23(1):174–185, 2005.
- [58] J. Gao and L. Zhang. Tradeoffs between stretch factor and load balancing ratio in routing on growth restricted graphs. In *Proc. of the 23rd ACM Symposium on Principles of Distributed Computing (PODC'04)*, pages 189–196, July 2004.
- [59] M. T. Goodrich and D. Strash. Succinct greedy geometric routing in  $r^2$ . Technical report on arXiv:0812.3893, 2008.
- [60] R. Govindan. Data-centric routing and storage in sensor networks. pages 185–205, 2004.



- [61] Y.-J. K. R. Govindan, B. Karp, and S. Shenker. Lazy cross-link removal for geographic routing. In *SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 112–124, 2006.
- [62] B. Greenstein, D. Estrin, R. Govindan, S. Ratnasamy, and S. Shenker. DIFS: A distributed index for features in sensor networks. In *Proceedings of First IEEE International Workshop on Sensor Network Protocols and Applications*, pages 163–173, Anchorage, Alaska, May 2003.
- [63] L. J. Guibas. Sensing, tracking and reasoning with relations. *IEEE Signal Processing Magazine*, 19(2):73–85, March 2002.
- [64] A. Gupta, R. Krauthgamer, and J. R. Lee. Bounded geometries, fractals, and low-distortion embeddings. In *FOCS '03: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, pages 534–543, 2003.
- [65] R. S. Hamilton. Three manifolds with positive ricci curvature. *Journal of Differential Geometry.*, 17:255–306, 1982.
- [66] A. Hatcher. *Algebraic Topology*. Cambridge University Press, 2001.
- [67] T. He, S. Krishnamurthy, L. Luo, T. Yan, L. Gu, R. Stoleru, G. Zhou, Q. Cao, P. Vicaire, J. A. Stankovic, T. F. Abdelzaher, J. Hui, and B. Krogh. Vigilnet: An integrated sensor network system for energy-efficient surveillance. *ACM Trans. Sen. Netw.*, 2(1):1–38, 2006.
- [68] Z.-X. He and O. Schramm. Fixed points, Koebe uniformization and circle packings. *Ann. of Math. (2)*, 137(2):369–406, 1993.
- [69] S. M. Hedetniemi, S. T. Hedetniemi, and A. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18(4):319–349, 1988.
- [70] J. M. Hellerstein, W. Hong, S. Madden, and K. Stanek. Beyond average: Toward sophisticated sensing with queries. In *Proc. Information Processing in Sensor Networks (IPSN)*, April 2003.
- [71] J. L. Hill and D. E. Culler. Mica: A wireless platform for deeply embedded networks. *IEEE Micro*, 22(6):12–24, 2002.
- [72] J. L. Hill, M. Horton, R. King, and L. Krishnamurthy. The platforms enabling wireless sensor networks. *Communications of the ACM*, 47(6):41–46, 2004.
- [73] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *ACM Conf. on Mobile Computing and Networking (MobiCom)*, pages 56–67, 2000.
- [74] M. Jin, J. Kim, F. Luo, and X. Gu. Discrete surface ricci flow. *IEEE Transaction on Visualization and computer Graphics*, 14(5):1030–1043, 2008.

- [75] D. B. Johnson and D. A. Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.
- [76] D. Karger and M. Ruhl. Find nearest neighbors in growth-restricted metrics. In *Proc. ACM Symposium on Theory of Computing*, pages 741–750, 2002.
- [77] B. Karp and H. Kung. GPSR: Greedy perimeter stateless routing for wireless networks. In *Proc. of the ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 243–254, 2000.
- [78] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *FOCS '03: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, page 482, Washington, DC, USA, 2003. IEEE Computer Society.
- [79] D. Kempe, J. Kleinberg, and A. Demers. Spatial gossip and resource location protocols. In *STOC '01: Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 163–172, 2001.
- [80] D. Kempe and F. McSherry. A decentralized algorithm for spectral analysis. In *STOC '04: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 561–568, New York, NY, USA, 2004. ACM Press.
- [81] W. Kim, K. Mechtov, J.-Y. Choi, and S. Ham. On target tracking with binary proximity sensors. In *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, page 40, 2005.
- [82] Y.-J. Kim, R. Govindan, B. Karp, and S. Shenker. Geographic routing made practical. In *Proceedings of the Second USENIX/ACM Symposium on Networked System Design and Implementation (NSDI 2005)*, May 2005.
- [83] Y.-J. Kim, R. Govindan, B. Karp, and S. Shenker. On the pitfalls of geographic face routing. In *DIALM-POMC '05: Proceedings of the 2005 joint workshop on Foundations of mobile computing*, pages 34–43, 2005.
- [84] C. Kinsey. *Topology of Surfaces*. Springer, 1993.
- [85] J. Kleinberg. Navigation in a small world. *Nature*, (406):845, 2000.
- [86] J. Kleinberg. The small-world phenomenon: an algorithm perspective. In *STOC '00: Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 163–170, 2000.
- [87] J. Kleinberg, A. Slivkins, and T. Wexler. Triangulation and embedding using small sets of beacons. In *Proc. 45th IEEE Symposium on Foundations of Computer Science*, pages 444–453, 2004.



- [88] R. Kleinberg. Geographic routing using hyperbolic space. In *Proceedings of the 26th Conference of the IEEE Communications Society (INFOCOM'07)*, pages 1902–1909, 2007.
- [89] G. Konjevod, A. W. Richa, and D. Xia. Optimal-stretch name-independent compact routing in doubling metrics. In *PODC '06: Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 198–207, 2006.
- [90] A. Kröller, S. P. Fekete, D. Pfisterer, and S. Fischer. Deterministic boundary recognition and topology extraction for large sensor networks. In *Proceedings of the 17th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'06)*, pages 1000–1009, 2006.
- [91] F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger. Geometric ad-hoc routing: Of theory and practice. In *Proc. 22<sup>nd</sup> ACM Int. Symposium on the Principles of Distributed Computing (PODC)*, pages 63–72, 2003.
- [92] F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger. Geometric ad-hoc routing: of theory and practice. In *PODC '03: Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 63–72, 2003.
- [93] S. S. Kulkarni and M. U. Arumugam. Tdma service for sensor networks. In *ICDCSW '04: Proceedings of the 24th International Conference on Distributed Computing Systems Workshops - W7: EC (ICDCSW'04)*, pages 604–609, Washington, DC, USA, 2004. IEEE Computer Society.
- [94] S. Kumar, T. H. Lai, and J. Balogh. On  $k$ -coverage in a mostly sleeping sensor network. In *MobiCom '04: Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 144–158, 2004.
- [95] B. Kusy, A. Ledeczi, and X. Koutsoukos. Tracking mobile nodes using RF doppler shifts. In *SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 29–42, New York, NY, USA, 2007. ACM.
- [96] T. Leighton and A. Moitra. Some results on greedy embeddings in metric spaces. In *Proc. of the 49th Annual Symposium on Foundations of Computer Science*, pages 337–346, October 2008.
- [97] P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: accurate and scalable simulation of entire tinys applications. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 126–137, 2003.
- [98] J. Li, J. Jannotti, D. Decouto, D. Karger, and R. Morris. A scalable location service for geographic ad-hoc routing. In *Proceedings of 6th ACM/IEEE International Conference on Mobile Computing and Networking*, pages 120–130, 2000.

- [99] X. Li, Y. J. Kim, R. Govindan, and W. Hong. Multi-dimensional range queries in sensor networks. In *Proceedings of the first international conference on Embedded networked sensor systems*, pages 63–75, 2003.
- [100] H. Lin, M. Lu, N. Milosavljević, J. Gao, and L. Guibas. Composable information gradients in wireless sensor networks. In *Proc. of the International Conference on Information Processing in Sensor Networks (IPSN'08)*, pages 121–132, April 2008.
- [101] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15:215–245, 1995.
- [102] N. Linial, L. Lovász, and A. Wigderson. Rubber bands, convex embeddings and graph connectivity. *Combinatorica*, 8(1):91–102, 1988.
- [103] J. Liu, F. Zhao, and D. Petrovic. Information-directed routing in ad hoc sensor networks. *IEEE Journal on Selected Areas in Communications*, 23(4):851–861, April 2005.
- [104] X. Liu, Q. Huang, and Y. Zhang. Combs, needles, haystacks: balancing push and pull for discovery in large-scale sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 122–133, 2004.
- [105] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: a tiny aggregation service for ad-hoc sensor networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):131–146, 2002.
- [106] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *SIGMOD*, June 2003.
- [107] Y. Mao, F. Wang, L. Qiu, S. S. Lam, and J. M. Smith. S4: Small state and small stretch routing protocol for large wireless sensor networks. In *Proceedings of the 4th USENIX Symposium on Networked System Design and Implementation (NSDI 2007)*, April 2007.
- [108] S. Meguerdichian, F. Koushanfar, M. Potkonjak, and M. B. Srivastava. Coverage problems in wireless ad-hoc sensor networks. In *Proc. of INFOCOM 2001*, volume 3, pages 1380–1387, 2001.
- [109] X. Meng, L. Li, T. Nandagopal, and S. Lu. Contour maps: Monitoring and diagnosis in sensor networks. *Computer Networks*.
- [110] S. Milgram. The small world problem. *Psychology Today*, (1), 1967.
- [111] J. W. Milnor. *Morse Theory*. Princeton University Press, Princeton NJ, 1963.
- [112] L. Mo, Y. He, Y. Liu, J. Zhao, S. Tang, X. Li, and G. Dai. Canopy closure estimates with greenorbs: Sustainable sensing in the forest. In *ACM SenSys 2009*, November 2009.

- [113] C. C. Moallemi and B. V. Roy. Consensus propagation. to appear.
- [114] S. Mortita. *Geometry of Differential Forms*. American Mathematical Society, 2001.
- [115] D. Mosk-Aoyama and D. Shah. Computing separable functions via gossip. In *PODC '06: Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 113–122, New York, NY, USA, 2006. ACM Press.
- [116] B. Nath and D. Niculescu. Routing on a curve. *SIGCOMM Comput. Commun. Rev.*, 33(1):155–160, 2003.
- [117] S. Nath, P. B. Gibbons, S. Seshan, and Z. R. Anderson. Synopsis diffusion for robust aggregation in sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 250–262, 2004.
- [118] J. Newsome and D. Song. GEM: graph embedding for routing and data-centric storage in sensor networks without geographic information. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 76–88, 2003.
- [119] A. Nguyen, N. Milosavljevic, Q. Fang, J. Gao, and L. J. Guibas. Landmark selection and greedy landmark-descent routing for sensor networks. In *Proceedings of the 26th Conference of the IEEE Communications Society (INFOCOM'07)*, pages 661–669, May 2007.
- [120] C. H. Papadimitriou and D. Ratajczak. On a conjecture related to geometric routing. *Theor. Comput. Sci.*, 344(1):3–14, 2005.
- [121] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM Monographs on Discrete Mathematics and Applications, 2000.
- [122] G. Perelman. The entropy formula for the ricci flow and its geometric applications. Technical Report arXiv.org, November 11 2002.
- [123] G. Perelman. Finite extinction time for the solutions to the ricci flow on certain three-manifolds. Technical Report arXiv.org, July 17 2003.
- [124] G. Perelman. Ricci flow with surgery on three-manifolds. Technical Report arXiv.org, March 10 2003.
- [125] C. E. Perkins and E. M. Royer. Ad hoc on-demand distance vector routing. In *Proc. of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, 1999.
- [126] P. Henrici. *Applied and Computational Complex Analysis*, volume 3. Wiley, New York, 1986.

- [127] M. Rabbat, J. Haupt, A. Singh, and R. Nowak. Decentralized compression and predistribution via randomized gossiping. In *IPSN '06: Proceedings of the fifth international conference on Information processing in sensor networks*, pages 51–59, New York, NY, USA, 2006. ACM Press.
- [128] A. Rao, C. Papadimitriou, S. Shenker, and I. Stoica. Geographic routing without location information. In *Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 96–108, 2003.
- [129] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker. GHT: A geographic hash table for data-centric storage in sensornets. In *Proc. 1st ACM Workshop on Wireless Sensor Networks and Applications*, pages 78–87, 2002.
- [130] R. Sarkar, X. Yin, J. Gao, F. Luo, and X. D. Gu. Greedy routing with guaranteed delivery using ricci flows. In *Proc. of the 8th International Symposium on Information Processing in Sensor Networks (IPSN'09)*, April 2009.
- [131] R. Sarkar, W. Zeng, J. Gao, and X. D. Gu. Covering space for in-network sensor data storage. In *Proc. of the 9th International Symposium on Information Processing in Sensor Networks (IPSN'10)*, April 2010.
- [132] R. Sarkar, X. Zhu, and J. Gao. Double rulings for information brokerage in sensor networks. In *Proc. of the ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 286–297, September 2006.
- [133] R. Sarkar, X. Zhu, and J. Gao. Hierarchical spatial gossip for multi-resolution representations in sensor networks. In *Proc. of the International Conference on Information Processing in Sensor Networks (IPSN'07)*, pages 420–429, April 2007.
- [134] R. Sarkar, X. Zhu, and J. Gao. Spatial distribution in routing table design for sensor networks. In *Proc. of the 28th Annual IEEE Conference on Computer Communications (INFOCOM'09), mini-conference*, April 2009.
- [135] R. Sarkar, X. Zhu, J. Gao, L. J. Guibas, and J. S. B. Mitchell. Iso-contour queries and gradient descent with guaranteed delivery in sensor networks. In *Proc. of the 27th Annual IEEE Conference on Computer Communications (INFOCOM'08)*, pages 1175–1183, May 2008.
- [136] A. Savvides, C.-C. Han, and M. B. Strivastava. Dynamic fine-grained localization in *ad-hoc* networks of sensors. In *Proc. 7th Annual International Conference on Mobile Computing and Networking (MobiCom 2001)*, pages 166–179, Rome, Italy, July 2001. ACM Press.
- [137] Y. Shang, W. Ruml, Y. Zhang, and M. P. J. Fromherz. Localization from mere connectivity. In *MobiHoc '03: Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, pages 201–212, 2003.

- [138] G. Sharma and R. Mazumdar. Hybrid sensor networks: a small world. In *MobiHoc '05: Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, pages 366–377, 2005.
- [139] S. Shenker, S. Ratnasamy, B. Karp, R. Govindan, and D. Estrin. Data-centric storage in sensornets. *SIGCOMM Comput. Commun. Rev.*, 33(1):137–142, 2003.
- [140] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri. Medians and beyond: New aggregation techniques for sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 239–249, 2004.
- [141] N. Shrivastava, R. M. U. Madhow, and S. Suri. Target tracking with binary proximity sensors: fundamental limits, minimal descriptions, and algorithms. In *SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 251–264, 2006.
- [142] J. Singh, U. Madhow, R. Kumar, S. Suri, and R. Cagley. Tracking multiple targets using binary proximity sensors. In *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, pages 529–538, New York, NY, USA, 2007. ACM Press.
- [143] P. Skraba, Q. Fang, A. Nguyen, and L. Guibas. Sweeps over wireless sensor networks. In *IPSN '06: Proceedings of the fifth international conference on Information processing in sensor networks*, pages 143–151, 2006.
- [144] K. Stephenson. *Introduction To Circle Packing*. Cambridge University Press, 2005.
- [145] I. Stojmenovic. A routing strategy and quorum based location update scheme for ad hoc wireless networks. Technical Report TR-99-09, SITE, University of Ottawa, September, 1999.
- [146] S. P. Tarasov and M. N. Vyalyi. Construction of contour trees in 3D in  $O(n \log n)$  steps. In *SCG '98: Proceedings of the fourteenth annual symposium on Computational geometry*, pages 68–75, 1998.
- [147] R. Thomas and X. Yu. 4-connected projective-planar graphs are hamiltonian. *J. Comb. Theory Ser. B*, 62(1):114–132, 1994.
- [148] M. Thorup and U. Zwick. Approximate distance oracles. In *Proc. ACM Symposium on Theory of Computing*, pages 183–192, 2001.
- [149] M. Thorup and U. Zwick. Compact routing schemes. In *SPAA '01: Proceedings of the thirteenth annual ACM symposium on Parallel algorithms and architectures*, pages 1–10, 2001.
- [150] W. P. Thurston. *Geometry and Topology of Three-Manifolds*. Princeton lecture notes, 1976.



- [151] J. Travers and S. Milgram. An experimental study of the small world problem. *Sociometry*, (32), 1969.
- [152] T. van Dam and K. Langendoen. An adaptive energy-efficient mac protocol for wireless sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 171–180, 2003.
- [153] M. van Kreveld, R. van Oostrum, C. Bajaj, V. Pascucci, and D. Schikore. Contour trees and small seed sets for isosurface traversal. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 212–220, 1997.
- [154] J. von Neumann. Various techniques used in connection with random digits. *U.S. National Bureau of Standards Applied Mathematics Series*, 12:36–38, 1951.
- [155] W. Wang and K. Ramchandran. Random distributed multiresolution representations with significance querying. In *IPSN '06: Proceedings of the fifth international conference on Information processing in sensor networks*, pages 102–108, New York, NY, USA, 2006. ACM Press.
- [156] L. Xiao and S. Boyd. Fast linear iterations for distributed averaging. *Systems and Control Letters*, 53:65–78, 2004.
- [157] L. Xiao, S. Boyd, and S. Lall. A scheme for robust distributed sensor fusion based on average consensus. In *Proceedings of the Fourth International Symposium on Information Processing in Sensor Networks*, pages 63–70, 2005.
- [158] L. Xiao, S. Boyd, and S. Lall. A space-time diffusion scheme for peer-to-peer least-squares estimation. In *Proceedings of the Fifth International Symposium on Information Processing in Sensor Networks*, pages 168–176, 2006.
- [159] F. Ye, H. Luo, J. Cheng, S. Lu, and L. Zhang. A two-tier data dissemination model for large-scale wireless sensor networks. In *MobiCom '02: Proceedings of the 8th annual international conference on Mobile computing and networking*, pages 148–159, 2002.
- [160] F. Ye, G. Zhong, S. Lu, and L. Zhang. GRAdient broadcast: A robust data delivery protocol for large scale sensor networks. *ACM Wireless Networks (WINET)*, 11(2), March 2005.
- [161] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient mac protocol for wireless sensor networks. In *INFOCOM 2002*, 2002.
- [162] F. Zhang, A. Jiang, and J. Chen. Robust planarization of unlocalized wireless sensor networks. In *Proc. of INFOCOM 2008*, pages 798–806, 2008.
- [163] F. Zhang, H. Li, A. A. Jiang, J. Chen, and P. Luo. Face tracing based geographic routing in nonplanar wireless networks. In *Proceedings of the 26th Conference of the IEEE Communications Society (INFOCOM'07)*, pages 2243–2251, May 2007.

- [164] H. Zhang and J. C. Hou. Maintaining sensing coverage and connectivity in large sensor networks. *Wireless Ad Hoc and Sensor Networks: An International Journal*, 1(1-2):89–123, January 2005.
- [165] F. Zhao, J. Shin, and J. Reich. Information-driven dynamic sensor collaboration. *IEEE Signal Processing Magazine*, 19(2):61–72, 2002.