EFFECTIVE ALGORITHMS AND PROTOCOLS FOR WIRELESS NETWORKING: A TOPOLOGICAL APPROACH

A Dissertation

by

FENGHUI ZHANG

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

August 2008

Major Subject: Computer Science

EFFECTIVE ALGORITHMS AND PROTOCOLS FOR WIRELESS NETWORKING: A TOPOLOGICAL APPROACH

A Dissertation

by

FENGHUI ZHANG

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Co-Chairs of Committee,	Jianer Chen
	Anxiao(Andrew) Jiang
Committee Members,	Donald K. Friesen
	Jennifer L. Welch
	Catherine H. Yan
Head of Department,	Valerie E. Taylor

August 2008

Major Subject: Computer Science

ABSTRACT

Effective Algorithms and Protocols for

Wireless Networking: A Topological Approach. (August 2008)

Fenghui Zhang, B.S., Fudan University

Co–Chairs of Advisory Committee: Dr. Jianer Chen
Dr. Anxiao Jiang

Much research has been done on wireless sensor networks. However, most protocols and algorithms for such networks are based on the ideal model Unit Disk Graph (UDG) model or do not assume any model. Furthermore, many results assume the knowledge of location information of the network. In practice, sensor networks often deviate from the UDG model significantly. It is not uncommon to observe stable long links that are more than five times longer than unstable short links in real wireless networks. A more general network model, the quasi unit-disk graph (quasi-UDG) model, captures much better the characteristics of wireless networks. However, the understanding of the properties of general quasi-UDGs has been very limited, which is impeding the design of key network protocols and algorithms.

In this dissertation we study the properties for general wireless sensor networks and develop new topological/geometrical techniques for wireless sensor networking. We assume neither the ideal UDG model nor the location information of the nodes. Instead we work on the more general quasi-UDG model and focus on figuring out the relationship between the geometrical properties and the topological properties of wireless sensor networks. Based on such relationships we develop algorithms that can compute useful substructures (planar subnetworks, boundaries, etc.). We also present

direct applications of the properties and substructures we constructed including routing, data storage, topology discovery, etc.

We prove that wireless networks based on quasi-UDG model exhibit nice properties like separabilities, existences of constant stretch backbones, etc. We develop efficient algorithms that can obtain relatively dense planar subnetworks for wireless sensor networks. We also present efficient routing protocols and balanced data storage scheme that supports ranged queries.

We present algorithmic results that can also be applied to other fields (e.g., information management). Based on divide and conquer and improved color coding technique, we develop algorithms for path, matching and packing problem that significantly improve previous best algorithms. We prove that it is unlikely for certain problems in operation science and information management to have any relatively effective algorithm or approximation algorithm for them.

To my wife Aihua and my son Anthony

ACKNOWLEDGMENTS

First of all I would like to thank my advisors, Dr. Jianer Chen and Dr. Anxiao (Andrew) Jiang. They are more than advisors to me. Dr. Chen has always been very supportive, even before he became my advisor. He has given me much inspiration, not only on my research, but also on my career planning and many other things in my life. He constantly encourages me to explore new territories and conquer difficulties. Dr. Jiang brought me into the area of sensor networking, which is the major topic of this dissertation. He taught me the difference between research in computer science and computer engineering and how to actually develop and apply algorithmic techniques in sensor networking research. The discussions with Dr. Chen and Dr. Jiang have been some of the most pleasant times during my PhD study. I am indebted to them now and in the future.

I am immensely grateful to Dr. Donald Friesen, Dr. Jennifer Welch, Dr. Sing-Hoi Sze and Dr. Catherine Yan. Their support and help were always present throughout the years. Dr. Friesen gave me valuable advice in all aspects of my Ph.D. life. Dr. Welch showed me how to be a researcher of excellence as well as a great teacher. Her guidance provided the inspiration of many results in this dissertation. Dr. Sze opened the door of bioinformatics for me. He not only taught me about research in bioinformatics, but also gave me priceless hints about academic writing. Dr. Yan refreshed my mathematical knowledge and taught me many elegant mathematical techniques and tools that are most useful in my research. It's my greatest fortune to have worked closely with them.

I would like to express my deep appreciation to all my colleagues, and in particular, to Dr. Iyad Kanj and Dr. Ge Xia. Cooperation with them has been very productive. And they also gave me very valuable advice on my career planning and

other things. I thank Xiuzheng Huang, Songjian Lu, Jie Meng, and Yang Liu, who were members of a wonderful theory group led by Dr. Chen. I also thank Hao Li, Yue Wang, and Vishal Kapoor, who were members of a excellent sensor networking group led by Dr. Jiang.

I would like to thank Dr. Yoonsuck Choe, Dr. Andreas Klapennecker, and Dr. Dimitri Logonouriv. They opened up a wide world for me so I had the chance to know about the latest research in artificial intelligence, quantum computing, peer-to-peer networks and many other fields in computer science and computer engineering.

I would like to take this opportunity to thank the faculty, staff, and students in the Department of Computer Science at Texas A&M University. Their support has made my study of computer science a pleasant experience and made this work possible.

Finally, my deepest gratitude goes to my beloved wife and my lovely son. I thank my wife for her love and supporting me during all these years. It is partly because of her that I chose Texas A&M University and met so many wonderful people here. Although my son is still very little, I thank him for doing well so I could focus on my work.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

LIST OF TABLES

TABLE Page

LIST OF FIGURES

FIGURE Page

FIGURE Page

FIGURE Page

FIGURE Page

CHAPTER I

INTRODUCTION

A wireless sensor network consists of autonomous units that use sensors to collect data from the environment and cooperate with each other to accomplish certain tasks. These sensor nodes are usually deployed in the sensor field which can be either the surface of an object or 3-D space. They monitor the physical conditions (temperature, motions, sounds, etc.) around them and collect these data. Although limited, these nodes have the ability to do relatively simple data processing. Among Radio frequency (RF), Optical communication (Laser) and Infrared, RF is the most popular wireless transmission media used in wireless sensor networks. The sensor nodes are usually based on battery power. In order to lower the cost of each sensor node and to make its size small, the computational power and communication power of a sensor node is limited in most cases.

Originally motivated by military applications like battlefield surveillance, wireless sensor networks are now widely used in other areas as well, including environment monitoring, traffic control, and healthcare applications. [82]

A typical sensor node consists of one or more sensors, a radio transceiver, a micro-controller and a battery as energy source. The size of a sensor node varies from a shoe box to the size of a pin. The cost for a single sensor node ranges from a few dollars to hundreds of dollars.

The computational power and communication power of the sensor nodes have been boosted recent years as the advancement in semiconductor industry and energy storage. The increasing of the power of wireless sensor networks has led to the

The journal model is *IEEE Transactions on Automatic Control.*

prevailing of applications based on such networks. The key feature of the wireless sensor networks is that it combines the functions of data collection, processing and communication into each node. A wide range of applications based on wireless sensor networks are proposed. Many of them are very promising and will change our future significantly. For example, the automobile sensing systems could reduce the number of automobile accidents by warning the drivers of dangers before hand or even stop the vehicles before tragedies.

On the other hand, because of their spatial property, the topology of the wireless sensor networks is closely correlated to their geometric structure. What are the relationships between these two (topological features and geometric features), how to utilize such properties to develop efficient protocols and reduce the dependencies on certain positioning devices have become very interesting research problems.

However, the research on wireless sensor networks turned out not to be an easy job. This is mainly because such networks are very different from the internet or a LAN. Wireless sensor nodes are usually based on battery power. This fact puts a limit on both the computational power and the communication power of the sensor nodes. The limit on the computational power usually includes limited storage and limited computing power. Protocols and algorithms for sensor networks have to be carefully designed to be applicable. On the other hand, the limitation of communication power gives wireless sensor network specific properties that are different from the internet and a LAN. First of all, the sensing system is often completely distributed since most centralize data processing will be very expensive. This is because data gathering and the result distributing are both global operations. Even if there is a server in the network, the frequency of such global operations has to be extremely low to avoid draining out of the power of each node quickly. Secondly, due to the non-uniformity of the transmission range of the nodes, the topology of the wireless sensor network can be very complex. Protocols and algorithms based on simple ideal models are often not applicable in practice. Furthermore, the links between two wireless nodes and the nodes themselves are not as reliable as that of computers on the internet or in a LAN. Hence network dynamics (node/link insertion and deletion) is a much more serious problem for wireless sensor networks.

One of the consequence of a network of complex topology is that many problems in the network are actually NP-complete. For example, finding a minimum dominating set is quite useful for network monitoring. It is easy to solve the problem if the network topology is simple enough (e.g., tree or ring structure). But it becomes very hard for general network topology. According to the complexity theory, we know that it is very unlikely that there will be efficient algorithms to solve these problems in general.

In this dissertation, I will introduce a topological approach to attack the difficulties in wireless sensor networking. We do not assume a simple model or the existence of expensive positioning device (such as GPS). We study the more general graph models for wireless sensor networks. Typical practical problems such as routing, data storage, topology discovery are first transformed in to problems in topological graph theory. We develop effective algorithms for these graph problems and apply these algorithms to derive protocols and schemes in wireless sensor networks.

A. Background

Throughout this dissertation we will always assume that the wireless sensor networks are deployed on a plane unless otherwise pointed out. An idealized model for wireless sensor networks is the well known Unit Disk Graph (UDG) model [21, 11]. In the UDG model, all the nodes are assumed to be exactly the same. A UDG sensor node has uniform transmission range, i.e., two nodes have a direct link in between if and only if their distance is no more than the unit distance r. A generalization of the UDG model is the quasi-Unit Disk Graph (quasi-UDG) model proposed in 2001 [5]. In the quasi-UDG model, we have two parameters R and r. Two nodes in the network have a direct link in between if their geographic distance is no more than r. If their distance is more than R, there will be NO direct link in between. In the case when the distance is between R and r, there may or may not be a direct link between the two nodes. It is straightforward that the UDG model is a special case of quasi-UDG when $R = r$. One can choose different scheme to specify the connectivity between two nodes if their distance is between R and r. One popular and easy way is to set a link between such two nodes u, v with given constant probability p. Another more realistic scheme is to set the probability p as a decreasing function of the distance between u and v.

Note that when r is set to 0 and R large enough, the quasi-UDG model can be used to model arbitrary network. The UDG model is much simpler than the quasi-UDG model. One fundamental property of the UDG model is the following *cross-link* property.

Lemma I.1 *If two edges $\{u, v\}$ and $\{x, y\}$ in a UDG cross each other, then at least one of u, v, x, y is the common neighbor of the other three.[38]*

Although the proof of the cross-link property of the UDG model is fairly simple, it turned out to be one of the most useful properties of the UDG model. A direct application of this property is the so called *Gabriel Planarization* that can be carried out in a completely distributed manner. Assuming that we know the coordinates of each node, two nodes u and v can "remove" (disable) the edge between them if there is another node w within the smallest disk that contains both u and v. After this, the remaining enabled edges induce a planar subgraph of the original network. We call a subgraph that contains all the nodes in the original graph a *spanner.* Since the planar subgraph obtained by Gabriel planarization contains every node in the original network, we call it a *planar spanner.*

Another efficient planarization of a UDG network is the *Relative Neighborhood Graph (RNG)* [96] proposed in 1980. The key idea is to destroy all triangles by removing the longest edge of each triangle. In [96] the authors showed that RNG is a sub graph of a Gabriel graph of the same input UDG network.

Let G be the original UDG network and H be a spanner of G. Let $d_G(u, v)$ be the distance between u, v in G and $d_H(u, v)$ be the distance between them in H (only using the edges in H). The *stretch factor* of H is then defined as

$$S_H = \max_{u,v \in G} \left\{ \frac{d_H(u, v)}{d_G(u, v)} \right\}.$$

The stretch factor of a spanner characterizes how good the spanner is in terms of preserving the connectivity of the original network. It is always a positive number at least 1. The smaller the stretch factor is, the better the spanner is.

If power efficiency is the major concern, one can define the distance between two nodes u and v to be the minimum power consumption that is necessary to send a message from u to v. For wireless nodes, the energy consumption P of transmitting a message to between two nodes of distance d usually satisfies $P = O(d^\beta)$ where β is a constant between 2 and 5. In this particular case, we call the stretch factor the *power stretch factor.* When this factor of a spanner is a constant, we say that the spanner is *power efficient.*

One can easily prove the following theorem.

Theorem I.2 *For a given UDG, the subgraph obtained by applying Gabriel planariza-*

tion is a planar power spanner of power stretch factor 1*, thus is power efficient.*

We can also take other measures of distance between nodes in wireless sensor networks and have different definitions of stretch factors. Another commonly used stretch factor is the *hop stretch factor* in which the distance between two nodes is simple the hop distance between them.

Routing is probably the most fundamental problem for any network. Different from other networks, developing efficient routing protocols is a great challenge for general wireless sensor networks.

Geographic routing was extensively studied to address this problem. It uses *greedy forwarding* in which a node always tries to rely the message to the neighbor that is closest to the destination. When there is no such neighbor, the message is then routed to get around the local maxima using local flooding.

In the case of UDG networks with each node knowing its own coordinates, it is much easier for a message to get around the local maxima. A celebrated idea called *perimeter routing* (or *face routing*) was proposed recently. The algorithm utilizes a planar spanner of the original network. Whenever a message reaches a local maxima, it switches to the face routing mode and gets around the local maxima. The key observation is that at least one of the faces containing the current node will contain a node that is closer to the destination. This routing protocol is ideal for UDG networks because of the existence of the Gabriel planar spanner.[38]

Later there was a similar routing scheme developed for a special class of quasi-UDG where $R/r \leq \sqrt{2}$ (also known as $\sqrt{2}$-quasi-UDG) [61] utilizing "virtual links". The key property of these quasi-UDG is that crosses can be locally detected based on the following property.

Lemma I.3 *If two edges $\{u, v\}$ and $\{x, y\}$ in a $\sqrt{2}$-quasi-UDG G cross each other,* *then at least one of the edges* $\{u, x\}, \{u, y\}, \{v, x\}, \{v, y\}$ *must be present in* G.

For networks based on general quasi-UDG models, planarization becomes difficult. Recently the Cross-Link Detection Protocol (CLDP) has been proposed as a nice attempt to tackle this problem [54]. In this protocol each node repeatedly probe its links to remove crossing links unless the removal of the cross links disconnect the network. Although CLDP does not guranttee a plane spanner, the authors showed that GPSR still works for the spanner.

There have been numerous geographic routing protocols based on perimeter routing, including GPSR [51], the work by Bose et al. [9], Compass routing, GOAFR [59], etc. There are also routing protocols that do not use node locations, but assign *virtual coordinates* to nodes for routing. Examples include GLIDER, MAP, GEM, etc. These protocols do not require the network to be a UDG.

In terms of network planarization, there are also some research done for unlocalized wireless sensor networks, i.e., the nodes' positions are not known. [35, 36, 37, 29, 97]

B. Parameterized computation

Many important problems that are encountered in real-world are NP-hard. According to complexity theory, it is very unlikely to have efficient algorithms to solve these problems. In many scenarios confronting such hardness is unavoidable. There have been a few approaches to deal with NP-hardness. Among them there are heuristic algorithms [69], approximation algorithms [4], and randomized algorithms [73].

However, in many cases the above approaches appear to be unsatisfactory. For example, many problems in the areas like database optimization, distributed computing and bioinformatics require exact solutions. Furthermore, the above approaches are not suitable for decision problems that ask “yes” or “no”. Recently a new approach called *parameterized computation* has been proposed to find exact solutions for NP-Complete problems. This new approach takes advantages of certain small parameters of the problems. The goal is to develop relatively efficient algorithms. The running time of these algorithms depend on the small parameters rather than the input size. Such algorithms are called *fixed-parameter tractable algorithm* or *FPT algorithm*. Study of such algorithms leads to a new line of research called *theory of fixed-parameter tractability* [26] that is concerned with designing practical parameterized algorithms for NP-hard problems. There have been many exciting results in this area. For example, the best known parameterized algorithm can decide if a graph of n vertices has a vertex cover of size at most k in time $O(1.274^k + kn)$ [19]. This algorithm is quite practical for parameter values up to $k = 400$ [15] and has been implemented to solve problems in bioinformatics.

Formally, a *parameterized problem* Q is a decision problem consisting of instances of the form (x, k), where the integer $k \geq 0$ is called the *parameter*. For example, the parameterized VERTEX COVER problem is to decide, given a pair (G, k) where G is a graph and k is a non-negative integer, whether G has a set of at most k vertices such that every edge of G is incident to at least one vertex in the set.

Many NP-hard parameterized problems become solvable in practice if the parameters are small. In the case of VERTEX COVER, for example, the parameter is the size of the solution set. In particular, the algorithm that runs in time $O(1.274^k + kn)$ suggests that the dominant factor in the time complexity, for this particular problem, is the size of the solution set, rather than the size of the input. Therefore, the VERTEX COVER problem is practically solvable if the size of the solution set is not too large, which is often the case in real applications.

Unfortunately, as one might have expected, not all problems have such nice properties. Many NP-hard parameterized problems remain difficult even when the values of the parameters are small. For instance, the INDEPENDENT SET problem is believed to be not solvable in time $f(k)n^{O(1)}$, for any function f.

To distinguish these two types of parameterized problems and characterize the hardness of such problems, Downey and Follows introduced the class of the *fixed-parameter tractable problems* denoted by FPT and the class of the *fixed-parameter intractable problems* which consists of various levels of *W-hierarchy* [26].

The class FPT contains problems that are solvable by parameterized algorithms in time $f(k)n^{O(1)}$, where k is given as a parameter, n is the size of the input, and f is an recursive function. The VERTEX COVER problem belongs to this class, along with many well known problems such as CUTWIDTH [30], TREEWIDTH [7], LONGEST PATH [1], and so on.

On the other hand, the W-hierarchy $\bigcup_{t\geq 0}$ W$[t]$ characterizes the inherent level of intractability of parameterized problems. The 0th level of the hierarchy is the class FPT. For any integer $i > 0$, the ith level is denoted by $W[i]$. A parameterized complexity preserving reduction (the *fpt-reduction*) is defined as follows. A parameterized problem Q is *fpt-reducible* to another parameterized problem Q' if there is an algorithm of running time $f(k)|x|^{O(1)}$ that on an instance (x, k) of Q produces an instance $(x', g(k))$ of Q', such that (x, k) is a yes-instance of Q if and only if $(x', g(k))$ is a yes-instance of Q', where the functions $f(k)$ and $g(k)$ depend only on k. A parameterized problem Q is $W[i]$*-hard* if every problem in $W[i]$ is fpt-reducible to Q, and is $W[i]$*-complete* if in addition Q is in $W[i]$. In particular, if any $W[i]$-hard problem is in FPT, then $W[i] =$ FPT, which, to the common belief, is very unlikely. The W[1]-hardness of a parameterized problem provides a strong evidence that the problem is not fixed-parameter tractable, or equivalently, cannot be solved in time $f(k)n^{O(1)}$ for any function f. It is proved that the W-hierarchy collapses to FPT only if CIRCUIT SATISFIABILITY (a very important problem in complexity theory) is solvable in subexponential time, which is widely believed to be unlikely. For a more detailed description and results on FPT and W-hierarchy, please refer to Downey and Fellows's book [26].

C. Our work

We present our research in wireless sensor networking and computational optimization. We study the fundamental properties of the general quasi-UDG model, namely separability and the existence of power efficient near planar spanner. We develop novel geographic routing algorithm based on face tracing that does not rely on the knowledge of node position information. We present a robust planarization algorithm that works for unlocalized wireless sensor networks. We also develop data storage scheme based on network sorting that can achieve data load balance and supports ranged queries in a natural way.

For computational optimization we present practical color coding technique and its applications in solving path, matching and packing problems. We develop fixed parameter tractable enumeration scheme for common techniques including color coding and bounded tree width. We show that a number of questions in the field of supply chain management are complete for the complexity classes $W[3]$ and $W[4]$. These problems are the first groups of natural problems complete for these two classes.

1. Wireless sensor networking

In the first part of this dissertation, we present our results in wireless sensor networking.

In Chapter II, we present the face tracing based geographic routing [99] protocol, which combines greedy forwarding with a mechanism called *face tracing*: when greedy forwarding fails, the message uses face tracing to route out of the dead-end region. The fundamental difference between face tracing and known perimeter routing algorithms is that with face tracing, the *faces* are not the faces of a planarized sub-network, but the faces of the network itself embedded in a high-genus topological surface. All the faces can be easily found, without any network embedding or planarization. In our extensive experiments these faces exhibited a prominent locality property and our routing protocol outperformed previous known protocols for general sensor networks.

In Chapter III, we present results on two important properties of the quasi-UDGs: separability and the existence of power efficient spanners. We show that quasi-UDGs are similar to planar graphs in the sense that given any quasi-UDG, we can construct a grid graph that is an abstraction of the given quasi-UDG and has a small balanced vertex separator (whose removal will partition the graph into two disjoint components of similar size). Furthermore, the grid graph has bounded node degree and bounded number of edge crossings. Based on the separability of the grid graph, we developed a compact routing protocol with stretch factor close to 2 utilizing a scheme similar to distance labelling. We also developed a distributed algorithm that constructs a spanner for any quasi-UDG with a constant power stretch factor and bounded node degrees and bounded average number of edge crossings.

In Chapter IV, we present a novel method that robustly planarizes (finds a planar subgraph in) sensor networks of realistic models: networks with non-uniform transmission ranges and unlocalized sensors. The method starts with a simple shortest path between two faraway nodes in the network, and progressively planarizes the whole network. The key to this algorithm is the so called ONE-SIDED TWO-LAYER PLANARIZATION problem. We first present an approximation algorithm for this NP-hard problem whose ratio can be arbitrarily close to 2. The algorithm is applicable to general networks, and achieves the best known approximation ratio. We then present an improved parameterized algorithm that solves the planarization problem exactly (namely, it finds the optimal solution). Our extensive simulations showed that the planar subgraphs maintain the distance between nodes with small stretches, allow detection of holes and boundaries with a much higher precision than existing methods, and are very robust to different quasi-UDG models.

In Chapter V, we present a sorting based data centric storage scheme. Data centric storage is a well known in-network storage scheme that adapts well to in-network processing. Previous approaches (GHT, GLS, etc.) either require the knowledge of location information or require a virtual embedding of the network. Our data centric storage scheme neither requires the location information nor relies on the planarization of the network. Our scheme achieves data load balance and supports ranged queries in a natural way. The idea is to construct a path as short as possible that traverses each node in the network at least once. Data objects in the network will then be sorted so that their order is consistent on the path. We developed distributed algorithms for sorting the data objects and practical algorithms to construct the path. We provide theoretical analysis for the complexity of the sorting algorithm as well as the demonstration of the storage performance through extensive simulations.

2. Computational optimization

In the second part of this dissertation, we describe our results in general computational optimization. We refined the color coding technique to make it practical and used it to improve the algorithms for a number of NP-hard problems including path, matching and packings. We developed general schemes such that the techniques of branch and bound, color coding, and bounded treewidth can be generalized to enumerate a given number of best solution for many problems.

In Chapter VI, we present our practical color coding technique and show how to apply it to get improved parameterized algorithms for path, matching and packing problems. We improve the size of an n, k color coding scheme to $O(6.4^k P(n))$ where $P(n)$ is a polynomial of n. Based on this result, we improve the algorithm for k-path problem to $O(12.8^k P(n))$.

In Chapter VII, we show the first groups of natural problems that are complete for the complexity classes $W[3]$ and $W[4]$ in the field of supply chain management. These results also imply the inapproximability of these problems.

D. Preliminaries

In the rest of this chapter, we give a concise introduction to most of the terms that will be used in the later chapters. Other terms will be introduced later in their proper setting. Most of the notations given here can be found in graph theory textbooks, such as [25].

A *graph* G is a pair (V, E), where V is a set of elements referred to as *vertices* of G and $E \subseteq V \times V$ is a binary relation on V. The elements of E are 2-element subsets of V, which are referred to as *edges* of G.

A graph G is called a *directed graph* or *digraph* if the elements of E are *ordered* pairs of vertices of G, otherwise it is called an *undirected graph.* Unless otherwise stated, the graph we consider in this dissertation are all undirected graphs.

The number of vertices of a graph G is its *order* (or *size*), written as $|G|$ (or $|V|$). The number of edges of a graph is written as $|E|$. A vertex v is *incident* with an edge e if $v \in e$. If $e = \{u, v\}$ is an edge of G, the vertices u and v are called *endpoints* (or *ends*) of e and they are considered to be *adjacent.*

Let v be a vertex of a graph G. The vertices that are adjacent to v is called its *neighbors.* The set of neighbors of v is denoted by $N_G(v)$ (or simply by $N(v)$ if the reference is clear). The set $N[v]$ denotes $N(v) \cup v$. More generally, for a set of vertices $U \subseteq V$, the neighbors in $V - U$ of vertices in U is called neighbors of U and denoted by $N(U)$. $N[U]$ denotes $N(U) \cup U$. The *degree* of v, denoted by $d_G(v)$ (or $d(v)$ if the reference is clear), is the number of edges that v is incident with, or equivalently, the number of vertices in $N(v)$. A vertex of degree 0 is *isolated.* If all vertices of G have the same degree k, then G is k*-regular*, or simply *regular.*

Let $G = (V, E)$ and $H = (V', E')$ be two graphs. If $V' \subseteq V$ and $E' \subseteq E$, then H is a *subgraph* of G. If in addition, H contains all edges in G that have both ends in V', H is an *induced subgraph* of G that is *induced* by V'. For a subgraph G' of G, denote by $G - G'$ the subgraph of G obtained by removing all vertices in G'.

A *path* in a graph G is a sequence of vertices $(v_0, v_1, \ldots, v_k)$ such that $(v_i, v_{i+1}) \in E$ for $i = 0, 1, \ldots, k-1$. A path is *simple* if all vertices in it are distinct. Unless otherwise stated, all the paths we consider in this dissertation are simple. A *cycle* in a graph G is a path $(v_0, v_1, \ldots, v_k)$ such that $v_0 = v_k$. A graph is *acyclic* is no cycle exists in the graph.

A non-empty graph $G = (V, E)$ is called *connected* if for any two vertices $u, v \in V$, there is a path $(v_0, v_1, \ldots, v_k)$ in G such that $v_0 = u$ and $v_k = v$. A maximal connected subgraph of G is called a *connected component* (or simple *component*) of G.

CHAPTER II

FACE TRACING BASED GEOGRAPHIC ROUTING IN NONPLANAR WIRELESS NETWORKS

Scalable and efficient routing is a main challenge in the deployment of large ad hoc wireless networks. An essential element of practical routing protocols is their accommodation of realistic network topologies. In this chapter, we study geographic routing in general large wireless networks. Geographic routing is a celebrated idea that uses the locations of nodes to effectively support routing. However, to guarantee delivery, recent geographic routing algorithms usually resort to *perimeter routing*, which requires the removal of communication links to get a planar sub-network on which perimeter routing is performed. Localized network planarization requires the wireless network to be a unit-disk graph (UDG) or its close approximation. For networks that significantly deviate from the UDG model, a common case in practice, substantially more expensive and non-localized network planarization methods have to be used. How to make such methods efficiently adaptable to network dynamics, and how to avoid the removal of an excessive number of links that leads to poor routing performance, are still open problems. To enable efficient geographic routing in general wireless networks, we present *face-tracing based routing*, a novel approach that routes the message in the *faces* of the network that are virtually embedded in a *topological surface*. Such faces are easily recognizable and constructible, and adaptively capture the important geometric features in wireless networks — in particular, holes, — thus leading to efficient routing. We show by both analysis and simulations that the face-tracing based routing is a highly scalable routing protocol that generates short routes, incurs low overhead, adapts quickly to network dynamics, and is robust to variations in network models.

A. Introduction

It is a challenging task to design practical routing schemes for large-scale *ad hoc* wireless networks (e.g., sensor networks). Limited energy and memory are often bottlenecks for such networks. And the complexity of connectivity and topology is key to the design of the routing protocols.

To support efficient and scalable routing, geographic routing has been extensively explored in recent years as a major technique. Geographic routing uses *greedy forwarding*: a relay node greedily forwards the message to a neighbor that is closer to the destination in Euclidean distance [9, 51, 59].[1] Such a step utilizes the close relation between a large-scale wireless network's topology and its deployment field, and greatly simplifies the design of the routing algorithm. Greedy forwarding, however, fails when the message reaches a *dead-end* node, a node that is closer to the destination than all its neighbors are. One method to solve this problem is to use local flooding (e.g., by expanding ring search) to find a node that is closer than the current dead-end node to the destination. This method turned out to be costly for networks that are relatively sparse or have holes.

In recently years, a celebrated idea called *perimeter routing* (or face routing) has been proposed and adopted in numerous routing algorithms [9, 51, 59]. The idea is to planarize the network by removing crossing edges. Then, when greedy forwarding fails in the original network, the message is routed from face to face in the planar sub-network toward the destination. That step, termed perimeter routing, is localized and nearly stateless. However, perimeter routing has its serious limitations. It relies on the planarization of the network. Localized network planarization requires the

[1] A source node can obtain a destination node's location based on its ID by using location service. And in some applications, such as data-centric storage, a message only needs to be sent to a location without knowing the destination node's ID.

wireless network to be a unit-disk graph (UDG) — defined as a network where two nodes can directly communicate if and only if their Euclidean distance is below a fixed value R — or its close approximation (e.g., a quasi-UDG where the communication range varies by a ratio of at most $\sqrt{2}$ [5]). In practice, however, such idealized connectivity models significantly deviate from many real wireless networks, due to reasons including antenna design, multi-path fading, etc. In addition, the errors in the node positions that the wireless nodes learn from the positioning system (e.g., Global Positioning System or localization methods) also moves the connectivity model away from the UDG model. It is not uncommon to observe stable long links that are five times or more longer than unstable short links in real wireless networks [40].

When a wireless network substantially deviates from the UDG model — a common case in practice, — it becomes provably infeasible to planarize it in a localized and efficient way. Also, planarizing such networks may force them to be disconnected. Recently, a nice attempt has been made to tackle this problem, where the Cross-Link Detection Protocol (CLDP) was proposed [54]. The idea of CLDP is to repeatedly probe the links of the network to remove crossing links (unless removing a link leads to problems such as network partition). Then in the network (nearly) planarized by CLDP, face routing algorithms, such as the well known Greedy Perimeter Stateless Routing (GPSR) algorithm [51], can be used. CLDP, however, does not resolve the *fundamental disadvantage* of network planarization: our experiments show that when the network deviates substantially from the UDG model, even if all the edges are short compared to the size of the network-deployment region, the action of planarizing the network requires the removal of a very large number of edges. That creates large distortion in routing distance in the perimeter routing phase, and can also easily lead the message in the opposite direction of the destination. Such a disadvantage appears hard to avoid for any routing algorithm based on direct planarization. Besides the high communication complexity of planarization and the distance distortion, how to make such methods adaptable to network dynamics (insertion and removal of links or nodes) is also a difficult open problem.

In this chapter, we present a novel routing approach for ad hoc wireless networks. We present *face-tracing based routing*, an efficient routing protocol that guarantees delivery for general wireless connectivity models. Similar to existing perimeter routing algorithms, the face-tracing based routing protocol combines greedy forwarding with a mechanism called *face tracing*: when greedy forwarding fails, the message uses face tracing to route out of the dead-end region. The fundamental difference between face tracing and perimeter routing is that with face tracing, the *faces* are not the faces of a planarized sub-network, but the faces of the network itself[2] embedded in a high-genus topological surface. Every edge is in one or two such faces. All the faces can be easily found, without any network embedding or planarization. These faces exhibit a prominent property: they automatically surround holes — regions where no node exist due to node sparsity or obstacles, around which dead-end nodes most likely appear — with high likelihood, and they tend to be localized in regions with no holes. Such a property is useful for routing a message out of dead-end regions, which is similar to the key reason for the success of perimeter routing in planar graphs. No edge removal is required for the correctness of the protocol, but to improve the performance, it does remove some edges in an efficient way. The number of edges removed, however, is much less than planarization, which makes face tracing much more efficient than perimeter routing due to its small distortion. We show that the face-tracing based routing protocol is highly efficient, scalable, adapts quickly to network dynamics, and is robust to variations in the network connectivity models.

[2] To be precise, in our protocol implementation, we consider the faces in a "cluster graph" derived from the network, which will be defined later.

There have been numerous geographic routing protocols based on perimeter routing, including GPSR [51], the work by Bose et al. [9], Compass routing, GOAFR [59], etc. There have also been routing protocols that do not use node locations, but assign *virtual coordinates* to nodes for routing. Examples include GLIDER, MAP, GEM, etc. The latter protocols do not require the network to be a UDG, similar to the face-tracing based routing protocol. Comparatively, the face-tracing based routing protocol uses node locations – which nodes obtain from position systems or localization methods — and does not require the embedding or the building of infrastructures to obtain virtual coordinates.

The rest of the chapter is organized as follows. In Section B, we introduce face tracing and study its properties. In Section C, we present the face-tracing based routing protocol. In Section D, we evaluate the protocol's performance through simulations. In Section E, we present concluding remarks.

B. Face tracing and its properties

In this section, we study face tracing and its properties in wireless sensor networks. Faces can be easily determined in a network, and they exhibit very nice locality properties. We will present the routing protocol based on face tracing in Section C.

1. Faces and face tracing

The concept of the *faces* of a network corresponds to an embedding of the network in a high-genus topological surface. Although our routing protocol *does not* embed the network in any way, understanding the relationship between face tracing and embedding is the key for proving the correctness of our protocol and its properties. In the following, we regard a network as a graph $G = (V, E)$ deployed in a plane, with V being the set of nodes and E the undirected edges.

A *topological surface* is an orientable 2-dimensional manifold in which each point has a neighborhood homomorphic to an open disk.[3] Informally speaking, a topological surface is the surface of a solid that contains no "infinitely thin joints". The simplest topological surfaces include spheres and toruses. (See Fig. 1 for examples.) On the other hand, the surface of two balls "glued" at a point does not make a topological surface.

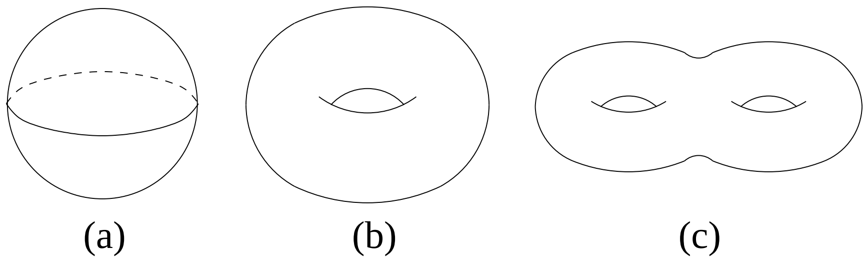


Fig. 1. Topological surfaces. (a) Sphere (genus=0). (b) Torus (genus=1). (c) Two–holed torus (genus=2).

Let G be a connected graph. An *embedding* of the graph G in a topological surface S is a "drawing" of G on S with no edge crossings. We will only consider "cellular embeddings" in which each face of the embedding is homeomorphic to an open disk.

To study graph embeddings, the concept of *graph rotation scheme* has to be introduced. Let v be a vertex in the graph G. A *rotation* at v is a cyclic labelling of the edges incident to v. That is, if v has p incident edges $[vu_0]$, $[vu_1]$, $\cdots$, $[vu_{p-1}]$, the rotation at v labels them by $\Pi(0)$, $\Pi(1)$, $\cdots$, $\Pi(p-1)$, where $\Pi(\cdot)$ is a permutation on $\{0, 1, \cdots, p-1\}$. We say that the edge labelled by $(i+1) \mod p$ *follows* the edge

[3] Formally, by "X is homeomorphic to Y", we mean there is a 1-to-1 mapping π from X to Y such that both π and its inverse are continuous.

labelled by $i \mod p$ or, equivalently, the edge labelled by $i \mod p$ *precedes* the edge labelled by $(i+1) \mod p$. A list of rotations, one for each vertex of G, is called a *rotation scheme* of the graph G. An example of a graph with a rotation scheme is shown in Fig. 2 (a), where the numbers beside edges are their labels.

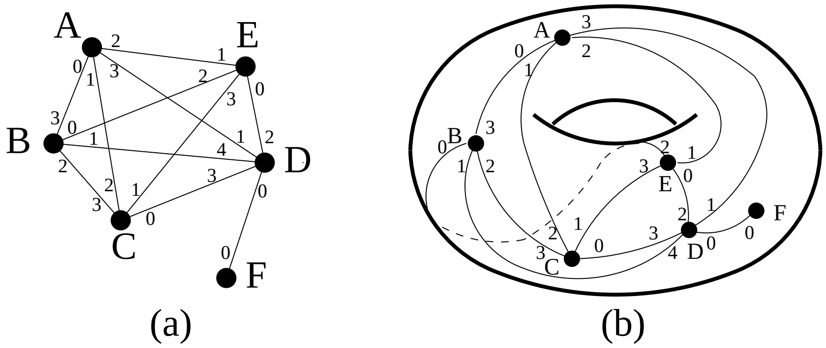


Fig. 2. Graph and its embedding. (a) A graph G with a rotation scheme. (b) Embedding of G in a topological surface.

An embedding of a graph G in a topological surface S naturally induces a rotation scheme for the graph G, as follows. For each vertex v of G, we take a sufficiently small neighborhood D of v on the surface S such that D is homeomorphic to a (planar) open disk. Then for the edges incident to v in D, we label them with $0, 1, 2, \cdots$ in the counterclockwise order, which defines a rotation scheme. (See Fig. 2 (b) for an example.) Conversely, by the classical Heffter-Edmonds Principle [46], *every rotation scheme of a graph G induces a unique embedding of G in a unique topological surface.* (See Fig. 2 for an example illustrating the correspondence between rotation scheme and embedding, where (a) is the graph G, and (b) is G's embedding in a topological surface.)

Therefore, as long as a rotation scheme of a graph G is given, we conceptually obtain an embedding of the graph G on some topological surface S. In our routing protocol, we always use the following rotation scheme: we label the edges incident to a node with $1, 2, 3, \cdots$ by the counterclockwise order of the edges *in the plane* where the wireless network is deployed. Note that the embedding corresponding to that particular rotation scheme is still highly non-trivial, because the network itself is usually not planar.

The edges of G partition the topology surface it is embedded in into *faces*. (See Fig. 2 (b).) By *face tracing*, we refer to the process of walking along the edges on the boundary of a face following the right-hand rule. For example, by walking along the edges from A to D to E to A to $D \cdots$ in Fig. 2, we are tracing a face. We can, in fact, do face tracing in the original graph G without finding out its embedding, as the following algorithm **FaceTrace** shows.

First we define a few notations. Each edge $e = [u, v]$ in a graph G has two directions: one is from u to v and the other is from v to u. We will call them "*edge-directions*" and denote them by $\langle u, v\rangle$ and $\langle v, u\rangle$, respectively. Let $\pi(G)$ be a rotation scheme of the graph G. To trace a face starting from an edge-direction $\langle u_0, v_0\rangle$, we apply the **FaceTrace** algorithm shown in Fig. 3.

```
Input: $G = (V_G, E_G)$: a graph with rotation $\pi(G)$ and edge direction $\langle u_0, v_0\rangle$
Output: traverse the face that $< u_0, v_0 >$ is in
$u \leftarrow u_0$, $v \leftarrow v_0$;
repeat
    output edge direction $\langle u, v\rangle$;
    let $[v, w]$ be the edge following the edge $[v, u]$ in the rotation at vertex $v$;
    $u = v$;    $v = w$;
until $(u = u_0)$ & $(v = v_0)$
```

Fig. 3. Algorithm **FaceTrace**: trace a face in a graph.

We give some remarks on the **FaceTrace** algorithm. The algorithm traces a sequence of edge-directions, following the orders in the rotations of the vertices appearing in the sequence, and stops when the first edge-direction is encountered again. It should be noted that *the first edge-direction must be encountered again and no edge-directions may appear more than once in the sequence.* To see this, we present here a proof by contradiction. Let $\langle u', v' \rangle$ be the first edge-direction that repeats in this sequence (such an edge-direction must exist because there are only finitely many edge-directions in the graph), and assume that $\langle u', v' \rangle$ is not $\langle u_0, v_0 \rangle$. By the algorithm, in order to trace the edge-direction $\langle u', v' \rangle$, we must first follow the edge-direction $\langle w', u' \rangle$, where $[w', u']$ is the edge preceding the edge $[u', v']$ in the rotation at vertex u'. Since $\langle u', v' \rangle$ is not the first edge-direction in the sequence, in order to trace $\langle u', v' \rangle$ twice, we must first trace $\langle w', u' \rangle$ twice. This contradicts the assumption that $\langle u', v' \rangle$ is the first edge-direction that repeats in the sequence. This contradiction proves that the first edge-direction $\langle u_0, v_0 \rangle$ must be the first repeated edge-direction in the sequence traced by the algorithm. In consequence, no edge-direction appears more than once in the sequence constructed by the algorithm **FaceTrace**.

Therefore, the sequence of edge-directions constructed by the algorithm **FaceTrace** forms a closed walk, which is the boundary of a face in the embedding $\pi(G)$ of the graph G.

The **FaceTrace** algorithm can start with any edge direction and trace the face that it is in. So clearly, each edge direction in a graph is contained in the tracing of exactly one face. An edge is involved in either one or two faces, because its two edge directions may or may not be in the tracing of the same face. If a vertex v is on the boundary of a face f — which we shall call "*the vertex v is in the face f*" in the rest of the chapter — a tracing of f must enter and leave v at least once each. So the number of faces that a vertex is in is upper bounded by its degree.

2. Face optimization for geographic routing

In our face-tracing based routing protocol, when greedy forwarding fails, we route the message along faces to get out of the dead-end region. Our extensive simulations show that to improve the routing performance, it is very beneficial to have small faces because of their good *locality* property: small faces tend to surround holes tightly, so they can guide messages to efficiently route around holes to escape the dead-end regions. In the following, we present three methods for reducing the sizes of faces, which also prove to be very effective in practice.

The first method splits a face into two smaller faces by removing an edge. Assume that a vertex v has p incident edges, which are labelled by $0, 1, \cdots, p-1$ in the rotation scheme. When we remove the edge labelled by j $(0 \leq j \leq p-1)$, the rotation at v changes in this way: now the edge labelled by $(j+1) \mod p$ follows the edge labelled by $(j-1) \mod p$, and the 'follow' relationship for the other edges remain unchanged. For a vertex v, we denote the neighboring vertices of v by $N(v)$. The first method is as follows:

- Let G be a connected graph with a rotation scheme. We remove an edge $[u, v]$ if it satisfies these two conditions: (1) there exists a face f that contains both the edge directions $\langle u, v\rangle$ and $\langle v, u\rangle$; (2) there exists another face g $(g \neq f)$ that contains a vertex in $N(u) - v$ and a vertex in $N(v) - u$ (those two vertices can be the same).

An example of the above method is shown in Fig. 4 (a), (b). The edge $[u, v]$ in Fig. 4 satisfies the two conditions: the corresponding face f is $u \rightarrow v \rightarrow B \rightarrow A \rightarrow v \rightarrow u \rightarrow A \rightarrow B \rightarrow u \rightarrow v \rightarrow \cdots$, and the face g is $u \rightarrow B \rightarrow v \rightarrow A \rightarrow u \rightarrow B \cdots$. So $[u, v]$ can be removed. After the removal, the graph is shown in Fig. 4 (b), where the face f has been split into two smaller faces, g remains intact, and the graph is

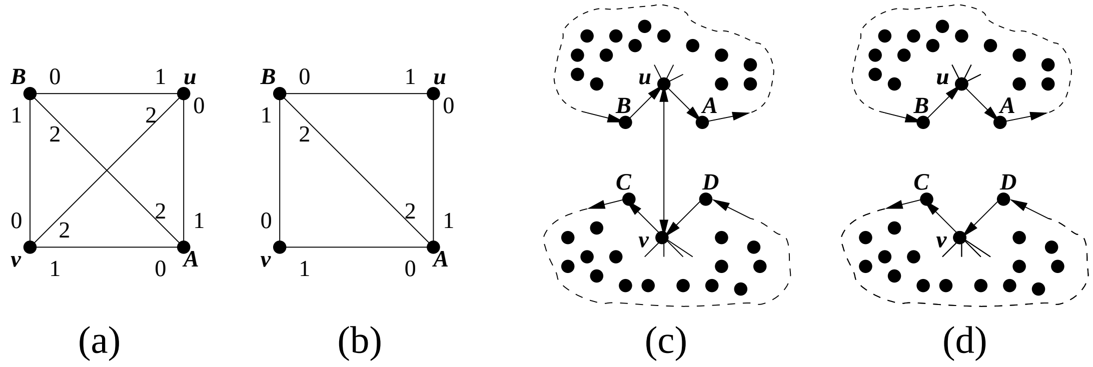


Fig. 4. Split a face by removing an edge. (a) A graph before removing an edge $[u, v]$. (b) After removing edge $[u, v]$. (c) A graph embedded in a topological surface, before removing edge $[u, v]$. (d) After removing edge $[u, v]$.

still connected. More generally, we have:

Theorem II.1 *After removing an edge $[u, v]$ using the above method, (1) the face f is split into two smaller faces; (2) the faces in G other than f all remain unchanged after the edge removal; (3) G remains connected after the edge removal.*

PROOF. Let's say that face g goes through $a \in N(u) - \{v\}$ and $b \in N(v) - \{u\}$. Face g is a cyclic walk, so g contains a walk from a to b. So naturally we get a walk $u \to a \to \cdots \to b \to v$, and clearly that walk does not contain the edge $[u, v]$. So $[u, v]$ is not a cut edge, removing which will not disconnect the network.

Face f contains both $\langle u, v\rangle$ and $\langle v, u\rangle$. Since f is a cyclic walk, the embedding of f in the topological surface is as illustrated in Fig. 4 (c) without loss of generality. (A vertex can appear multiple times in the shown face. The direction of the walk using right-hand rule is as shown by arrows. The face is *outside* what appears to be the two closed regions; but since the topological surface has 'bridges', the nodes in those two seemingly closed regions can be connected through the 'bridges'.) Before removing edge $[u, v]$, face f is $\{u \rightarrow A \rightarrow \cdots B \rightarrow u \rightarrow v \rightarrow C \rightarrow \cdots D \rightarrow v \rightarrow u\}$. After removing edge $[u, v]$, f is replaced by two smaller faces: $\{u \rightarrow A \rightarrow \cdots B \rightarrow u\}$ and $\{v \rightarrow C \rightarrow \cdots D \rightarrow v\}$. Removing edge $[u, v]$ does not affect other faces, because they do not contain the edge directions $\langle u, v\rangle$, $\langle v, u\rangle$, $\langle B, u\rangle$, $\langle u, A\rangle$, $\langle D, v\rangle$ or $\langle v, C\rangle$. □

The second method is simple: we remove the longest edge in every triangle in the graph. Its good performance for creating small faces is validated through experiments.

The third method is to work on a *cluster graph* $H = (V_H, E_H)$ instead of the original graph $G = (V, E)$. The cluster graph $H = (V_H, E_H)$ is defined as follows. Partition the vertex V into disjoint subsets S_1, S_2, $\cdots$, S_k such that for each S_i $(1 \leq i \leq k)$, there is a vertex $u_i \in S_i$ that is adjacent to all other vertices in S_i. V_H consists of vertices v_1, v_2, $\cdots$, v_k, such that (1) v_i has the same position as u_i in the plane; (2) there is an edge between v_i, v_j in H if and only if in G, there is an edge connected two vertices respectively in S_i and S_j. Such a graph H is called the *cluster graph of* G. Experiments show that the faces in H are much smaller than the faces in G for wireless networks. Our routing protocol actually routes messages *conceptually* along the faces in H instead of G.

3. Analysis on the locality property of faces

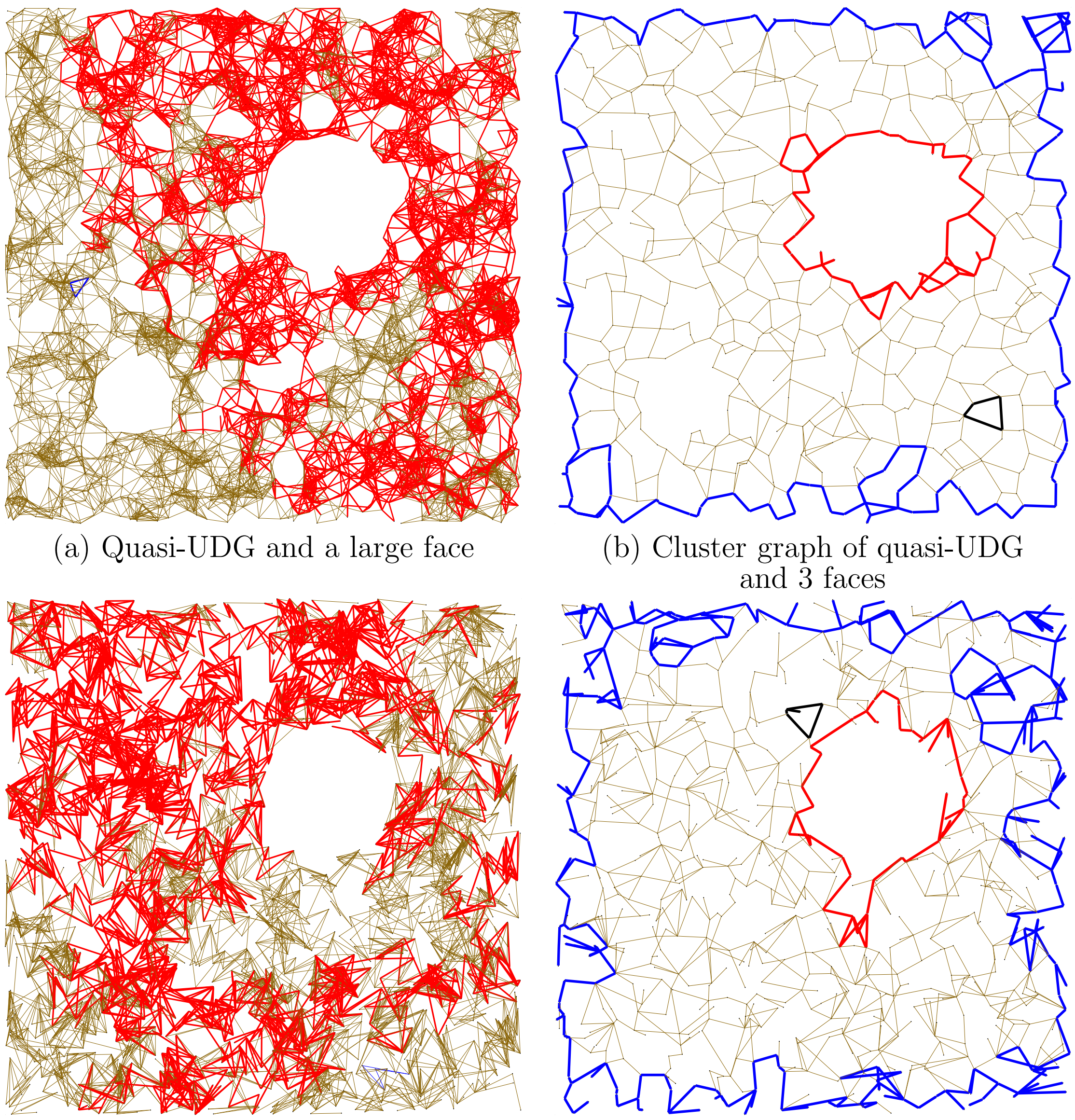
Our extensive simulations show that wireless networks strongly tend to have faces surrounding holes. This feature is especially nice for cluster graphs, where the faces exhibit the following *locality* property: (1) there are nearly always faces *closely* surrounding holes; (2) in the areas where no hole (of a moderate or large size) exists, the faces tend to be small and very localized. Those properties experimentally hold for a very wide range of network models. Examples of faces in wireless networks and in their cluster graphs are shown in Fig. 5. There, (a) and (c) show networks of two different models: quasi-UDG model and directional antenna model, two popular models of wireless network whose details will be introduced in Section D. (b) and (d) show their corresponding cluster graphs, to which the methods for reducing face sizes introduced in the previous subsection are applied. The original wireless networks tend to have faces surrounding holes but not very localized. Examples of two very large faces of that type are shown in (a) and (c) with thick lines. For the cluster graphs in (b) and (d), three typical faces are shown: a face *closely* surrounding a hole, a face enclosing the outside boundary of the network, and a randomly selected face in regions with no holes, which is very small and localized. We will show more statistics on the faces in our simulations in Section D.

The locality property of the faces in the cluster graph is key to the performance of the face-tracing based routing protocol. It is a very intriguing question why such a property exists, as it concerns the complex relationship between the wireless network's geometry in the Euclidean plane and its embedding in a topological surface. In this subsection, we attempt to shed some light on its understanding by studying the robustness of the faces surrounding holes.

The generation of an ad hoc wireless network can be seen as the random generation of nodes and edges following some rules (e.g., an edge cannot be too long). Assume that we have a graph that contains a face surrounding a hole. We consider the following question concerning the robustness of the face: *if we add or remove edges from the graph, in which case will there no longer be a face surrounding the hole?*

First, let's define a hole in the following way: *A hole is a continuous region in the plane that does not contain any vertex or part of any edge.*

The definition of if and how a face surrounds a hole is more subtle. To present the definition, we use a concept called *Surrounding Index (SI).*



(a) Quasi-UDG and a large face

(b) Cluster graph of quasi-UDG and 3 faces

(c) DA graph and a large face

(d) Cluster graph of DA graph and 3 faces

Fig. 5. Examples of quasi-UDG and directional antenna (DA) networks, their cluster graphs, and typical examples of the faces (represented by dark edges) in them. 2000 nodes are deployed in a 20×20 plane. For the quasi-UDG, $R = 1$, $r = 0.1$, $p = 0.5$, average degree is 7.885. For the DA graph, $\theta = 120^o$, $R_{DA} = 2$, average degree is 7.290.

Let G be a graph in a plane, and let h be a hole. Let P denote a walk in G. Let c be a fixed point in the hole h, and let p be a point on the walk P. Consider the ray starting at c and goes through p. When p moves along the walk P with a small step, the ray sweeps the plane with a small angle. We give the angle a positive (negative) sign if the ray sweeps in the counterclockwise (clockwise) direction. The *surrounding index* of the walk P for the hole h, $SI(P, H)$, is defined to be the total angle that the ray sweeps with when the point p moves through the whole walk P exactly once. We see a face as a close walk (where each edge direction is visited only once); therefore, a face's surrounding index must be $2\pi i$, where i is an integer. Note that a face may circle around a hole multiple times, so i may be an integer whose absolute vale is greater than 1. If a face does not enclose a hole, then its surrounding index is 0. Now we define: *A face f* **surrounds** *a hole h if $SI(f, H) \neq 0$.*

If we partition a face f into a set of smaller walks P_1, P_2, $\cdots$, P_k, then clearly, $SI(f, h) = \sum_{i=1}^{k} SI(P_i, h)$.

First we consider adding an edge $[u, v]$ to graph G. (In the following, we always assume that the rotation scheme labels the edges incident to a vertex based on their counterclockwise order in the plane. The results below can be extended for general rotation schemes.) Let $[u, w_1]$, $[u, w_2]$ be the two edges that, respectively, *precedes* and *follows* edge $[u, v]$ in u's rotation. Let $[v, a_1]$, $[u, a_2]$ be the two edges that, respectively, *precedes* and *follows* edge $[v, u]$ in v's rotation. Let the face that contains the edge directions $\langle w_1, u\rangle$, $\langle u, w_2\rangle$ (resp., $\langle a_1, v\rangle$, $\langle v, a_2\rangle$) before the edge $[u, v]$ is added be called face f_1 (resp., face f_2). Then, based on the definitions of face tracing and surrounding index, it is simple to see that the following proposition holds. We skip its detailed proof due to the space limitation.

Proposition II.2 *Let h be a hole. (1) If f_1 and f_2 are the same face, then the* *addition of the new edge $[u, v]$ splits it into two different faces f_3 and f_4, where f_3 is "$\cdots w_1 \rightarrow u \rightarrow v \rightarrow a_2 \cdots$" and f_4 is "$\cdots a_1 \rightarrow v \rightarrow u \rightarrow w_2 \rightarrow \cdots$". $SI(f_1, h) = SI(f_3, h) + SI(f_4, h)$. (2) If f_1 and f_2 are two different faces, then the addition of the new edge $[u, v]$ merges them into one face f_3: "$\cdots w_1 \rightarrow u \rightarrow v \rightarrow a_2 \cdots a_1 \rightarrow v \rightarrow u \rightarrow w_2 \rightarrow \cdots$". $SI(f_1, h) + SI(f_2, h) = SI(f_3, h)$.*

Now we consider removing an edge $[u, v]$ from graph G. Before $[u, v]$ is removed, let f_1 be the face containing the edge directions $\langle w_1, u\rangle$, $\langle u, v\rangle$, $\langle v, a_2\rangle$, and let f_2 be the face containing the edge directions $\langle a_1, v\rangle$, $\langle v, u\rangle$, $\langle u, w_2\rangle$. Similarly we have:

Proposition II.3 *Let h be a hole. (1) If f_1 and f_2 are the same face, then the removal of the edge $[u, v]$ splits it into two different faces f_3 and f_4: f_3 is "$\cdots w_1 \rightarrow u \rightarrow w_2 \cdots$" and f_4 is "$\cdots a_1 \rightarrow v \rightarrow a_2 \cdots$". $SI(f_1, h) = SI(f_3, h) + SI(f_4, h)$. (2) If f_1 and f_2 are two different faces, then the removal of the edge $[u, v]$ merges them into one face f_3: "$\cdots w_1 \rightarrow u \rightarrow w_2 \rightarrow \cdots \rightarrow a_1 \rightarrow v \rightarrow a_2 \cdots$". $SI(f_1, h) + SI(f_2, h) = SI(f_3, h)$.*

By the above two propositions, when we split a face f_1 surrounding a hole into two faces f_3 and f_4, one of them must still be surrounding the hole. That is because when $SI(f_1, h) = SI(f_3, h) + SI(f_4, h)$ and $SI(f_1, h) \neq 0$, either $SI(f_3, h) \neq 0$ or $SI(f_4, h) \neq 0$. When we merge two faces f_1 and f_2 into one face f_3, if — say, — f_1 surrounds a hole h (so $SI(f_1, h) \neq 0$), then f_3 also surrounds h unless $SI(f_2, h) = -SI(f_1, h) \neq 0$. So to eliminate a face in a graph that surrounds a hole, the only way is to merge it with another face of the opposite surrounding index, where at least one edge need be added. By the collected statistics on faces shown in Section D, we see that in the cluster graphs, the faces usually closely surround holes and the outer boundary; and because of the 'right-hand rule' for face tracing, often the only type of face pairs of opposite non-zero surrounding indices are a face closely surrounding a hole and a face enclosing the outside network boundary. These restrictions make it less likely to eliminate faces surrounding holes in a graph by adding or removing a small number of edges, which provides some insight on the robustness of the face surrounding property.

C. Face-tracing based routing protocol

The face-tracing based routing consists of two modes: the *greedy forwarding* mode, and the *face tracing* mode. A message is first routed in the original network using greedy forwarding. If it reaches a dead-end node v, the message enters the *face tracing* mode and routes along the faces in the network's cluster graph, until it reaches a node b that is geographically closer to the destination than v is. Then, the message returns to the greedy forwarding mode. The message alternates between these two modes until it reaches the destination. The nice locality property of the faces make this process very efficient. In the following, we introduce the components of the routing protocol.

1. Preprocessing

The network is preprocessed before any routing starts. The procedure consists of three elements: *building the cluster graph of the network*, *let nodes recognize the faces (of the cluster graph) they are in*, and *reduce the sizes of the faces using the methods described in Section B*. The specific process is: **First**, the nodes distributively partition the node set into very small clusters, where each cluster consists a 'cluster head' node that is adjacent to all the other nodes in the cluster. Every node remembers the connectivity between nodes in its own cluster; then the nodes distributively build a cluster graph by remembering the edges from their own clusters to the adjacent clusters; **Second**, for each triangle in the cluster graph, the two endpoints of its longest edge mark it as "removed"; **Third**, each node in the cluster graph uses the **FaceTrace** algorithm to learn the faces they are in, by sending inquiry messages along each incident edge in the cluster graph. The faces are assigned IDs, and the nodes remember the IDs of the faces they are in. To reduce the number of inquiry messages, we let the nodes initiate such inquiry messages asynchronously: if a node receives an inquiry message from an incoming edge e, it no longer needs to initiate an inquiry message along the outgoing link that *follows* e. To improve routing performance, each node also remembers the positions of t randomly selected nodes in every path. (We find through experiments that $t = 5$ is sufficient.) We call t the **sampling rate**. **Fourth**, if there is a link $[u, v]$ in the cluster graph that can be removed by using the first method presented in Section B for reducing faces sizes, we remove $[u, v]$. By Theorem II.1, only a face containing u and v are affected (which is split into two faces). So only u and v send out two messages to trace the two new faces, and inform all the nodes in those two faces of that change. The above operations can all be implemented in a very *efficient*, *distributed* and *asynchronous* way.

2. Routing

The routing consists two modes: *greedy forwarding* and *face tracing*. When a message just enters the face tracing mode at node v (of the cluster graph), among the faces containing v, we heuristically choose the face that contains a sampled node whose Euclidean distance to the destination is the minimum. (Recall that v remembers the positions of t sampled nodes in the face.) The message is routed in that face using the **FaceTrace** algorithm, which is nearly **stateless**. If that face does not get the message any closer to the destination, then we route from face to face, and route inside each face that we come to. Note that going from face to face is the same as going from vertex to vertex in the dual graph of the embedded cluster graph in its corresponding topological surface. Therefore, if we traverse all the faces in this way, we can reach the whole graph, including the destination. In our implementation, the message remembers the faces it has traversed in the current round of face tracing, and uses the depth-first search (DFS) to go from face to face. So the delivery is guaranteed. The overhead in remembering the traversed faces' IDs is very small, due to the high routing efficiency. Note that each cluster of the network acts as one node in the cluster graph. Since a cluster is of diameter 2 or less, realizing the face tracing in the true network is very simple.

3. Network dynamics

In a wireless network, links and node may come and go. Our protocol adapts to such network dynamics efficiently. By Propositions II.2 and II.3 described in the previous section, when a link is added or removed, at most two faces are affected, so only two messages need be sent by the two endpoints of the link to learn the new faces. Adding or removing a node is the same as adding or removing its incident links. The only additional case to consider is that when nodes/links are added or removed, clusters can change, appear or disappear. As nodes in the same cluster remember connectivity information about the whole cluster, such changes are efficiently processed.

D. Simulation

We have implemented the face-tracing based routing protocol, and conducted extensive simulations for various network connectivity models and deployment environments. The protocol has shown very stable performance across the various environments and parameter configurations. In this section, we present simulations for a typical set up for ad hoc wireless networks, and consider two different wireless connectivity models: the *quasi unit disk graph* (quasi-UDG) model, and the *directional antenna* (DA) model.

The *quasi-UDG* model is a generalization of the UDG model for wireless networks. It has three parameters R, r and p. ($R \geq r$, $0 \leq p \leq 1$.) An edge exists (does not exist) between two vertices if their Euclidean distance is less than r (more than R); if the Euclidean distance is between r and R, the edge exists with probability p.

The model we adopt for *directional antennas* (DA) is a simplification of the real DAs. It has two parameters R_{DA} and θ. ($0 \leq \theta \leq 2\pi$.) A vertex u can directly send messages to a vertex v if and only if v falls inside a cone of angel θ rooted at v and is also within Euclidean distance R_{DA} from v. The orientation of that cone is uniformly randomly selected. There is an edge between two vertices if and only if they can both directly send messages to each other.

In the experiments, we uniformly randomly deploy n_0 wireless nodes in a 2-D space of size 20×20. To mimic nontrivial deployment environments, we randomly put two holes (areas where nodes cannot be placed) of radius 1.5 and 2.5 in the plane. (The network also has naturally formed voids due to the sparsity of nodes.) Corresponding to each fixed set of parameters, we randomly generate 30 networks. Then in each network, we randomly pick 10,000 source and destination pairs for routing.

The focus of these experiments is to verify the validity of the new geographic routing approach based on face tracing. We concentrate on the topological level of the routing, and study the routing performance, properties of faces, network preprocessing overhead, packet overhead and adaptivity to network dynamics. Many important factors at the MAC layer, such as link quality or packet acknowledgement, have not been addressed and will be studied in our future work. We compare the face-tracing based routing protocol with the current geographic routing approach that uses perimeter routing. In particular, we compare it with the combination of GPSR [51] and CLDP [54]. GPSR is a widely known geographic routing protocol, and CLDP is a novel network planarization protocol that supports GPSR. The performance of combining GPSR with CLDP has been studied in [54]. We also compare with a popular geographic routing approach that combines greedy forwarding with local flooding. When greedy forwarding fails, that approach uses local flooding (expanding ring search with doubling radius) to route out of the dead-end region. The experiment results show that the face-tracing based routing approach leads to excellent routing performance.

1. Statistics on faces

Typical examples of the quasi-UDG networks, directional antenna (DA) networks, their cluster graph, and typical faces in them are shown in Fig. 5. Details of the figures were introduced in Section B, so we skip them here. We comment that in nearly all the cluster graphs generated in the experiments, there are faces surrounding the holes and the outside boundary. In areas with no (substantially) holes, the faces are very small and localized. The type of faces most helpful for getting a message out of a dead-end region are those that *closely* surround relatively large holes. Let's define a face surrounding a hole to be *close* if the average Euclidean distance from the vertices in the face to the boundary of the hole is less than Δ. We set $\Delta = 4$ here. The statistics on such *close* faces are shown in Table I.

As shown in Table I, the probability of having faces *closely* surrounding the holes is high. In such cases, that probability becomes relatively lower; that is because the faces surrounding the holes become complex and contain vertices further away from the holes, and we do not count them as '*close*.' The average face size is very small,

Table I. Statistics on faces in the cluster graphs of quasi-UDG networks and directional antenna (DA) networks. n_0 (m_0) and n (m) are the number of vertices (edges) in the original networks and their cluster graphs, respectively. $R = 1$, $p = 0.5$, $R_{DA} = 2.5$. 'Hole #1' and 'hole #2' are the two randomly placed holes of radius 1.5 and 2.5, respectively. 'Boundary' is the outside boundary of the deployment region.

		Network Connectivity Model: Quasi UDG							
		Statistics on faces surrounding holes		Statistics on all faces			Statistics on Network		
		Ratio of networks containing faces **closely** surround holes & boundary	Average distance between face vertices and the holes they surround	Average face size	Standard deviation of face size	Average number of faces a node is in	m_0	n	m
$n_0 = 2000$ $R/r = 2$	hole #1	100%	0.740	5.219	6.032	4.768	9878.0	449.1	1484.1
	hole #2	100%	0.751						
	boundary	100%	0.686						
$n_0 = 2000$ $R/r = 10$	hole #1	100%	1.036	6.465	11.257	4.821	7921.8	563.8	2022.3
	hole #2	97%	1.065						
	boundary	100%	1.070						
$n_0 = 4000$ $R/r = 2$	hole #1	100%	0.853	7.405	9.442	6.331	39600.3	573.8	2905.3
	hole #2	100%	0.664						
	boundary	100%	0.768						
$n_0 = 4000$ $R/r = 10$	hole #1	87%	3.851	10.705	29.011	7.519	31663.8	754.3	4637.2
	hole #2	60%	3.514						
	boundary	20%	1.485						
		Network Connectivity Model: Directional Antenna							
$n_0 = 2000$ $\theta = 90^o$	hole #1	57%	1.861	10.205	27.047	3.891	5451.6	838.1	2361.4
	hole #2	39%	1.323						
	boundary	12%	0.662						
$n_0 = 2000$ $\theta = 150^o$	hole #1	86%	1.682	12.114	25.731	5.968	12598.7	609.7	2811.7
	hole #2	62%	1.542						
	boundary	11%	0.673						
$n_0 = 4000$ $\theta = 90^o$	hole #1	77%	1.752	17.603	39.947	6.427	20818.7	1394.3	6698.2
	hole #2	60%	1.593						
	boundary	6.8%	0.675						
$n_0 = 4000$ $\theta = 150^o$	hole #1	100%	1.666	21.053	37.285	8.690	49956.0	937.9	6440.4
	hole #2	89%	1.671						
	boundary	11%	0.746						

and on average a vertex is contained only in a small number of faces. That is because of the strong locality of the faces.

2. Network preprocessing overhead

Both the face-tracing based routing protocol and CDLP require preprocessing when the network is initialized. Our protocol does clustering and requires nodes to recognize faces. CLDP probes the network to remove crossing links. The total number of messages sent is taken as the preprocessing overhead. When the same message is transmitted over k hops, we count it as k messages. The results are as shown in Table II. We see that the face-tracing based protocol improves the overhead significantly, by a factor of 10^3 to 10^4.

Table II. Network preprocessing overhead: number of messages sent, $R = 1$, $p = 0.5$, $R_{DA} = 2.5$.

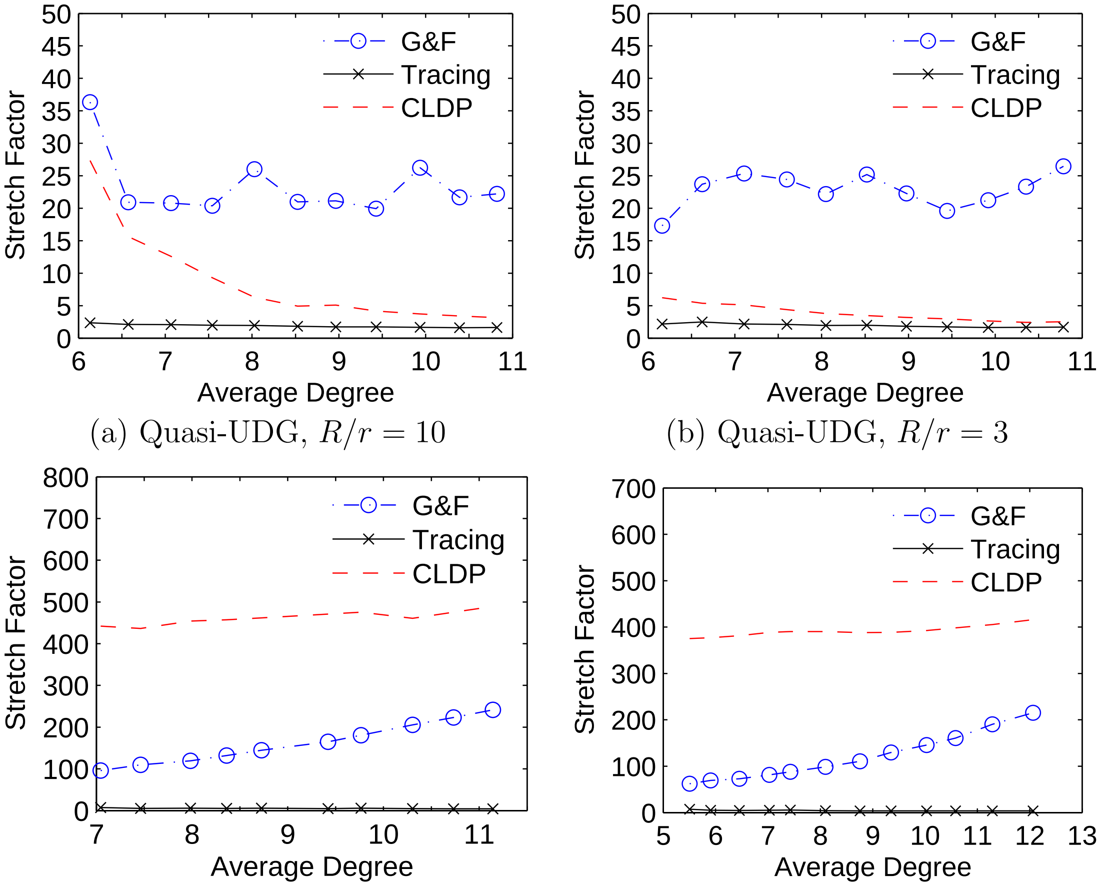
	Quasi-UDG model			
	$n_0 = 2000$	$n_0 = 2000$	$n_0 = 4000$	$n_0 = 4000$
	$R/r = 2$	$R/r = 10$	$R/r = 2$	$R/r = 10$
Tracing	1.40×10^4	1.37×10^4	4.15×10^4	3.93×10^4
CLDP	1.67×10^7	1.63×10^7	1.77×10^8	1.79×10^8
	Directional antenna model			
	$n_0 = 2000$	$n_0 = 2000$	$n_0 = 4000$	$n_0 = 4000$
	$\theta = 90^o$	$\theta = 150^o$	$\theta = 90^o$	$\theta = 150^o$
Tracing	1.73×10^4	2.06×10^4	4.24×10^4	6.17×10^4
CLDP	1.94×10^7	5.37×10^7	1.83×10^8	4.67×10^8

3. Quality of routing paths

Given a source and a destination, if greedy forwarding alone succeeds, the face-tracing based routing (short as *Tracing here*), CLDP+GPSR (short as *CLDP* here), and greedy forwarding plus local flooding (short as *G&F* here) produce exactly the same routing path. In our experiments, greedy forwarding succeeds in around 30% to 70%

cases for quasi-UDGs, while for directional antenna (DA) graphs this percentage is less than 1%. We compare the routing performance only for the cases where greedy forwarding alone does not succeed. Define *stretch factor* as the average ratio of the hop distance in a routing path (generated by one of the three routing protocols) to the minimum hop distance between the source and the destination. For a good understanding, we measure the routing performance considering the changes in vertex degree, vertex density, network size, and face sampling rate.

a. Stretch factor vs. node degree



(a) Quasi-UDG, $R/r = 10$

(b) Quasi-UDG, $R/r = 3$

(c) DA, $\theta = 90^o$

(d) DA, $\theta = 120^o$

Fig. 6. Stretch factor vs. average vertex degree in original networks. $R = 1$.

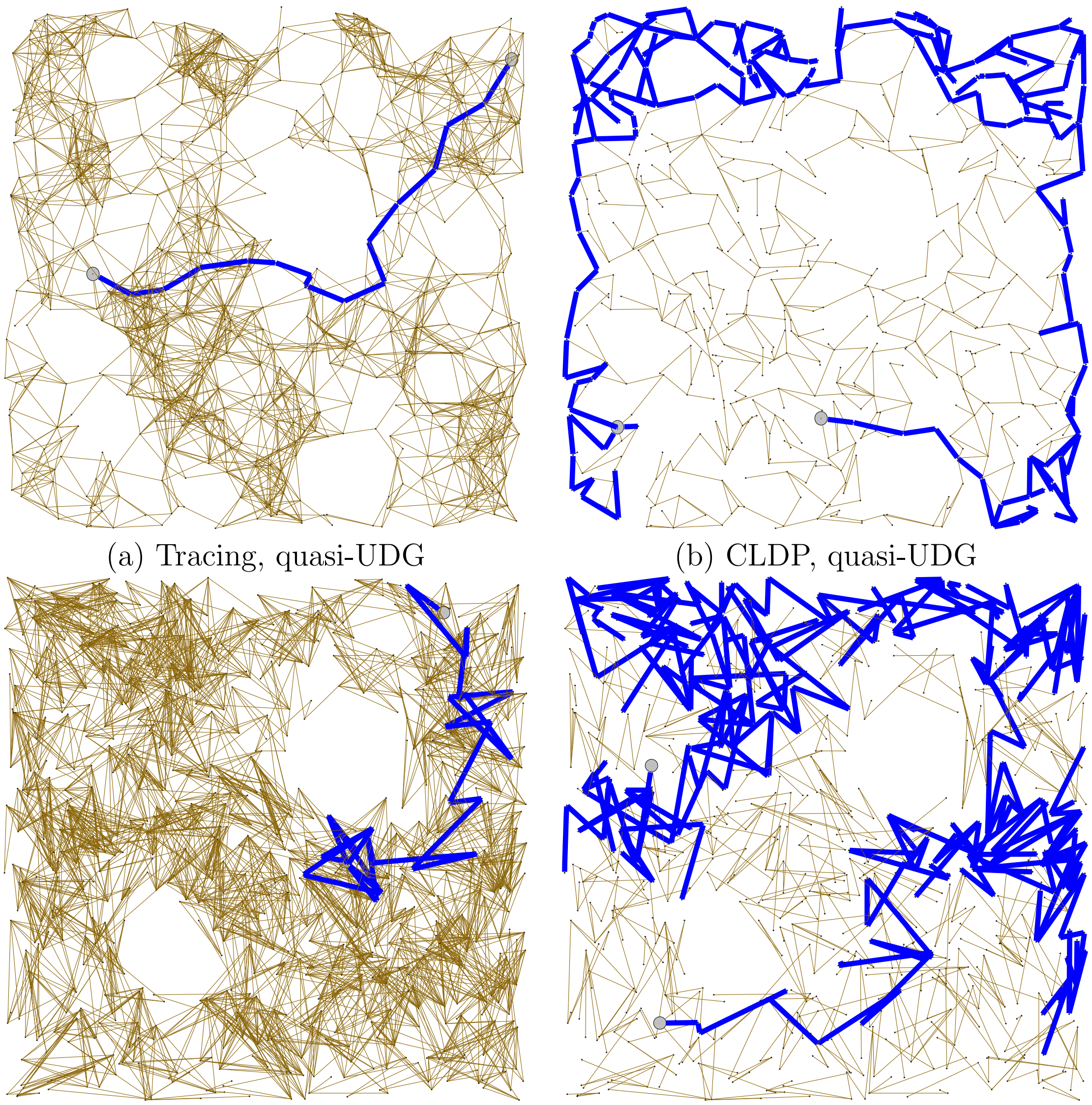
For quasi-UDGs (DA graphs), we adjust the value of connectivity probability p (the radius R_{DA}) to change the average vertex degree. The average routing stretch factors are shown in Fig. 6. The face-tracing based routing protocol performs much better than the other two. Its stretch factor is in the ranges of $[1.59, 2.37]$, $[1.63, 2.47]$, $[4.48, 6.87]$, $[3.57, 7.20]$ for Fig. 6 (a), (b), (c) and (d). It exhibits a particular stable performance, with a stretch factor that is several, tens or even hundreds of times better than the other two routing protocols.

Examples of the routing paths are shown in Fig. 7. The paths are represented by thick lines. The face-tracing based routing protocol outperforms greedy forwarding plus local flooding because the faces guides messages around holes efficiently. (See Fig. 7 (a) for an example, where the left end of the path is the source.) It also outperforms CLDP+GPSR because it removes much fewer edges than CLDP does. (Note that both use all the edges in the greedy forwarding mode.) CLDP usually removes about twice the number of edges than our protocol. Fig. 7 (b) and (d) show how sparse the network can be when links are removed by CLDP for perimeter routing.

b. Stretch factor vs. vertex density, network size, and face sampling rate

We increase the number of vertices, and measure the routing stretch factors. The results are shown in Fig. 8 (a), (b). (We skip the results for directional antenna graphs due to limited space.) The stretch factor for our protocol is in the ranges $[1.38, 2.35]$ and $[1.39, 3.41]$ for Fig. 8 (a) and (b). Again, it stably outperforms the other two protocols.

We increase the network size by increasing the number of vertices while keeping the vertex density and average degree constant. The results are shown in Fig. 8(c), which indicate that all three routing protocols are scalable in the network size for



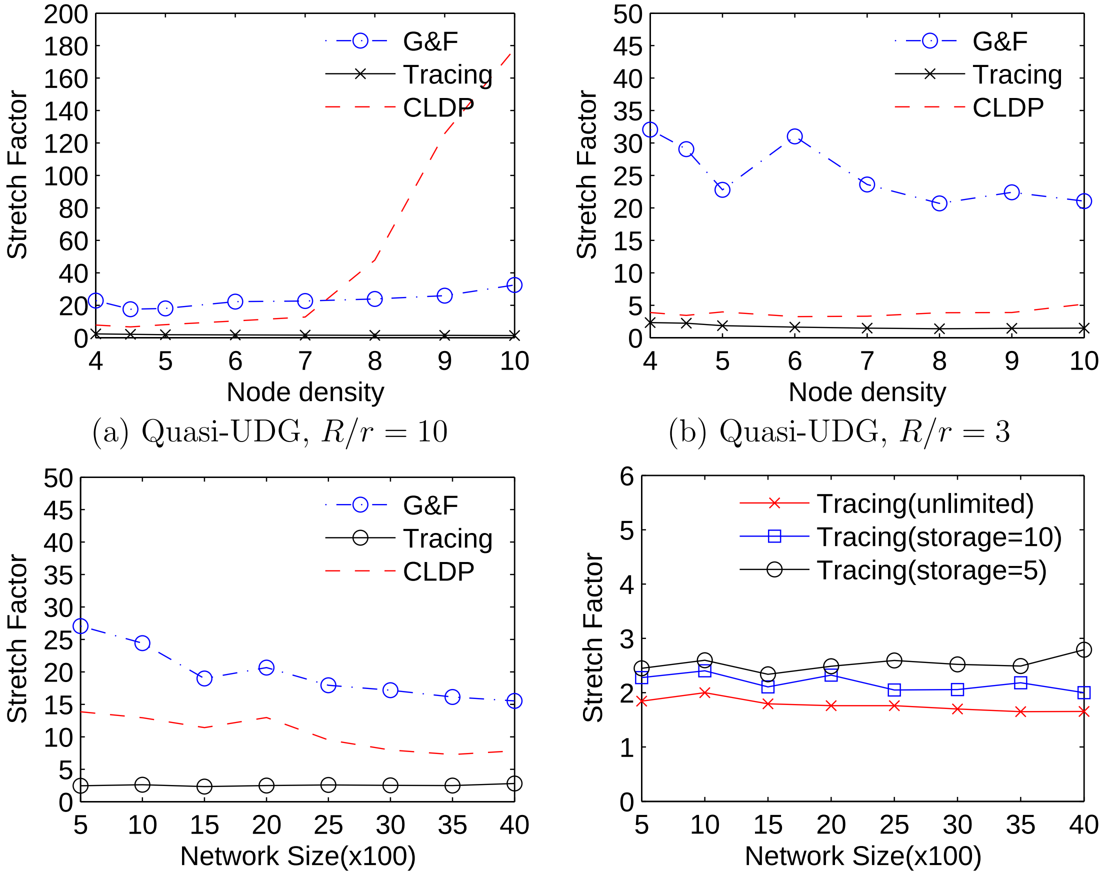
(a) Tracing, quasi-UDG

(b) CLDP, quasi-UDG

(c) Tracing, DA

(d) CLDP, DA

Fig. 7. Examples of routing paths (represented by dark edges). The underlying networks in (a) and (c) are the original networks. The underlying networks in (b) and (d) are showing only the edges not removed by CLDP.



(a) Quasi-UDG, $R/r = 10$

(b) Quasi-UDG, $R/r = 3$

(c) Quasi-UDG, $R/r = 10$

(d) Quasi-UDG, $R/r = 10$

Fig. 8. (a) and (b): Stretch factor vs. node density (number of nodes per unit area). (c) Stretch factor vs. network size (the total number of nodes) (d) Stretch factor vs. face sampling rate.

stretch factors. We also change the face sampling rate (the number of sample vertices a node remembers about a face) from 5 to 10 to infinity. As shown in Fig. 8 (d), the performance of our protocol does not change much. It means that setting the sampling rate to be 5 is already sufficient. In all our other experiments, we use the sampling rate 5.

4. Packet overhead

With the face-tracing based routing, when a message enters the face tracing mode, it needs to remember the faces it has traversed in this round. We call such a storage overhead in the message the *packet overhead*. The average packet overhead is shown in Table III. The unit there is the the ID of a face, which is roughly $\log_2 n_o$ bits. We see that on average the message traverses only a few faces, which is very efficient.

Table III. Average packet overhead of the face-tracing based algorithm. The average value is taken over the different routing paths. $R = 1$, $p = 0.5$, $R_{DA} = 2.5$.

Quasi-UDG				Directional antenna			
$n_0 =$ 2000, R/r $= 2$	$n_0 =$ 2000, R/r $= 10$	$n_0 =$ 4000, R/r $= 2$	$n_0 =$ 4000, R/r $= 10$	$n_0 =$ 2000, $\theta =$ 90^o	$n_0 =$ 2000, $\theta =$ 150^o	$n_0 =$ 4000, $\theta =$ 90^o	$n_0 =$ 4000, $\theta =$ 150^o
1.24	1.69	1.04	2.21	2.82	2.98	4.52	4.20

5. Adaptivity to network dynamics

When there are dynamics in wireless networks — the deletion/insertion of links or nodes — the face-tracing based routing protocol needs to recognize the new faces, while CLDP needs to re-probe the network to remove crossing links and restore some other links. Our protocol has the advantage that its adaptation to the network dynamics is *on demand*: it can efficiently recognize new faces with localized operation only when links/nodes are deleted/inserted. CLDP chooses the strategy of periodic probing of the whole network. We measure the number of messages that our protocol needs to send to recognize new faces and adapt to the changed network. When the same message is sent over k hops, we count it as k messages. The results are shown in Table. IV. We see that the overhead is very low.

Table IV. Average number of messages sent for the deletion/insertion of a link/node. When a message is sent over k hops, it is counted as k messages.

Quasi-UDG				Directional antenna			
$n_0 =$ 2000, R/r = 2	$n_0 =$ 2000, R/r = 10	$n_0 =$ 4000, R/r = 2	$n_0 =$ 4000, R/r = 10	$n_0 =$ 2000, $\theta =$ 90^o	$n_0 =$ 2000, $\theta =$ 150^o	$n_0 =$ 4000, $\theta =$ 90^o	$n_0 =$ 4000, $\theta =$ 150^o
Deletion of a link							
17.4	23.7	24.2	21.8	37.5	25.8	49.7	32.2
Insertion of a link							
12.7	9.7	24.0	18.2	1.9	9.1	4.1	19.1
Deletion of a node							
138.5	205.7	140.6	177.9	491.4	269.6	557.4	298.1
Insertion of a node							
222.7	249.2	263.9	269.2	462.3	284.7	470.0	312.7

CHAPTER III

SEPARABILITY AND TOPOLOGY CONTROL OF QUASI UNIT DISK GRAPHS

A deep understanding of the structural properties of wireless networks is critical for evaluating the performance of network protocols and improving their designs. Many protocols for wireless networks — routing, topology control, information storage/retrieval and numerous other applications — have been based on the idealized unit-disk graph (UDG) network model. The significant deviation of the UDG model from many real wireless networks is substantially limiting the applicability of such protocols. A more general network model, the quasi unit-disk graph (quasi-UDG) model, captures much better the characteristics of wireless networks. However, the understanding of the properties of general quasi-UDGs has been very limited, which is impeding the designs of key network protocols and algorithms.

In this chapter, we present results on two important properties of quasi-UDGs: separability and the existence of power efficient spanners. Network separability is a fundamental property leading to efficient network algorithms and fast parallel computation. We prove that every quasi-UDG has a corresponding grid graph with small balanced separators that captures its connectivity properties. We also study the problem of constructing an energy-efficient backbone for a quasi-UDG. We present a distributed localized algorithm that, given a quasi-UDG, constructs a *nearly planar* backbone with a constant stretch factor and a bounded degree. We demonstrate the excellent performance of these auxiliary graphs through simulations and show their applications in efficient routing.

A. Introduction

The connectivity structures of wireless networks exhibit strong correlations with the physical environment due to the signal transmission model of wireless nodes. A deep understanding of the structural properties of wireless networks is critical for evaluating the performance of network protocols and improving their designs. So far, many protocols have been based on the idealized unit-disk graph (UDG) network model, where two wireless nodes can directly communicate if and only if their physical distance is within a fixed parameter R. Examples of these protocols include routing [9, 51], topology control [2], distributed information storage/retrieval [29] and a great variety of other applications. In practice, however, the UDG model significantly deviates from many real wireless networks, due to reasons including multi-path fading [40, 93], antenna design issues, inaccurate node position estimation, etc. It is not uncommon to observe stable links that are five times longer than unstable short links [93]. The significant deviation of the UDG model from the real/practical models is substantially limiting the applicability of protocols based on UDGs. To combat this problem, a more general network model, the quasi unit-disk graph (quasi-UDG) model, has been recently proposed to capture the nonuniform characteristics of (most) wireless networks. Formally, this model is defined as follows.

Definition III.1 *A quasi-UDG with parameters r and R (r and R are positive numbers) over a set of points in the plane is defined as follows. The points in the set are the vertices of the graph. For any two points u and v in the set with Euclidean distance $|uv|$: if $|uv| \leq r$ then uv is an edge in the graph; if $|uv| > R$ then uv is not an edge in the graph; and if $r < |uv| \leq R$ then uv may or may not be an edge in the graph.*

In sharp contrast to the UDG model whose properties have been well understood ([2, 51]), the understanding of the properties of general quasi-UDGs has been very limited. Among the limited knowledge about quasi-UDG, a notable result is the "link-crossing" property discovered for quasi-UDGs where $R \leq \sqrt{2} \cdot r$ [5]. The serious lack of understanding of the properties of general quasi-UDGs is impeding the design of key network protocols and algorithms for this model.

In this chapter, we present results on two important properties of quasi-UDGs: separability and the existence of power efficient spanners. Network separability is a fundamental property that leads to efficient network algorithms and fast parallel computation [65]. A (vertex) *separator* of a graph G is a set of vertices whose removal splits the graph into two non-adjacent parts of similar sizes. We call a graph G *well separable* if any subgraph of G has relatively small separators. A well separable graph has strong localized properties. As a result, the performance of protocols for routing, information retrieval, network monitoring, etc., can be significantly improved for such graphs. We first construct a grid graph that is an abstraction of the given quasi-UDG G and show that the grid graph is well separable. The separator we obtain for the grid graph is of size $O(\sqrt{N})$ and can split the graph into two parts of size roughly $N/2$, where N is the number of nodes of the grid graph. In addition, both the degree of the grid nodes and the number of edges crossing any given edge, are upper bounded by constants. Among many applications of the separators, we present, as an example, a compact routing protocol based on the grid graph construction and the distance labeling technique. We prove that the routing table size of each node in our protocol is bounded by $O(\sqrt{N} \log N)$, which is much better than the tight bound proved for general graphs and close to the lower bound of $\Omega(\sqrt{N})$ for bounded-degree graphs in [42]. The ratio of the routing path length to the shortest path length is upper bounded by $2 + \epsilon$, where ϵ is a small constant. More extensions of the results are also included.

In the second part of the chapter we study the existence and the construction of energy efficient backbones for quasi-UDGs. A backbone is a spanning subgraph of the wireless network used for efficient communication. By using only those edges in the backbone for communication, signal interference, routing table size and power usage can be substantially reduced. A major requirement for the backbone construction is preserving the shortest path distances between vertices as much as possible. For a backbone B of a graph $G = (V, E)$, the *stretch factor* of B is defined as $s(B) = \max\{f_B(u, v)/f_G(u, v)|u, v \in V\}$, where $f_B(u, v)$ and $f_G(u, v)$ are the lengths of the shortest paths between vertices u and v in B and G, respectively. The stretch factor reflects the quality of the backbone. There have been results showing that for UDGs, bounded degree and planar spanners can be constructed when the distance function $f(u, v)$ is defined as the minimum power needed to send a message from u to v [50][95]. In this chapter, we present a distributed algorithm that constructs a backbone B for any quasi-UDG G with a constant power stretch factor. The node degree of the backbone B is upper bounded by a constant. In addition, although it is in general impossible to construct planar backbones with constant stretch factors for quasi-UDG, we show that B is *nearly planar*. More specifically, we show that B has a constant upper bound on the average number of edges crossing any given edge. The latter property is useful for geographic routing algorithms based on cross link detection [54].

We evaluate the performance of the separators, the routing protocol, and the backbone construction, through extensive simulations. Their performance is much better compared to the theoretical analysis of the worst cases. This shows that, although the quasi-UDG model is quite different from the UDG model, efficient algorithms can still be developed by exploiting the locality of the model.

The rest of the chapter is organized as follows. In Section B, we present the grid graph construction and prove its separability properties. In Section C, we present the backbone construction. In Section D, we present the compact routing protocol based on the grid graph and the distance labeling technique, as well as the simulation results. We conclude the chapter in Section E.

B. Overview

A quasi-UDG is called a *Unit Disk Graph*(UDG) if $R = r$. The UDG has been the major graph model for studying wireless sensor networks. Recognizing and embedding a UDG when the location information is not available, have been proved to be NP-hard [11]. To approximate an embedding within ratio $\sqrt{3/2}$ is also NP-hard [58]. Even when the angles between adjacent edges or the distances between nearby nodes are known the embedding of the UDG remains NP-hard, it is also NP-hard to find an α-approximate embedding where $\alpha < \sqrt{2}$ [3, 12]. However a planar spanner can be found in polynomial time [12]. If the distances between every pair of nodes are given, the UDG can then be embedded efficiently [6, 92].

But if the locations of the nodes in the network are known, many hard problems become easier. For example, the well known APX-hard MAXIMUM INDEPENDENT SET problem has a polynomial time approximation scheme [77]. Many other hard problems are also relatively easier for UDG [21].

The UDG has the nice cross link property(if the edge $\{u, v\}$ crosses the edge $\{p, q\}$ in a UDG, at least one of u, v, p, q must be the common neighbor of the other three) what enables efficient distributive planarization of the network. When the lengths of each edge is given, one can construct a Gabriel graph [38] and a relative neighborhood graph [96] that are planar and the Gabriel graph is power efficient. And when the angles between adjacent edges are known, a planar subgraph containing a restricted Delaunay graph (the residual graph after removing all edges longer than R in a Delaunay graph) can be found [97] and has constant stretch factor.

Based on the planarization of UDG, quite a few *perimeter routing* (face routing) protocols have been proposed [9, 51, 59].

In contrast to UDG, the quasi-UDG does not have such a nice property to exploit. However researches have been done for quasi-UDG model because it is more practical than the ideal UDG model. For quasi-UDG with $R/r \leq \sqrt{2}$, the network can still be planarized with the help of virtual edges [5]. Thus the protocols based on the planar subgraph of the network can still be applied.

Separability is another very import property in graph theory. Given a graph of n nodes, a separator is usually a set of vertices (edges) whose removal will disconnect the original graph into two parts of similar size (each of size $O(n)$). In this chapter, if not specified, by separator we mean vertex separator only. We say that a separator is small if the number of vertices in the separator is $o(n)$ or even a constant. A graph G is said to be *well separable* if all subgraphs of G has small separators. Many NP-hard graph problems, e.g., VERTEX COVER, MAXIMUM INDEPENDENT SET, can be solved in polynomial time if the graph is well separable and the size of the separator concerned is constant [8]. When the separators we have are not constant but $o(n)$, the same set of NP-hard problems can usually be solved much more effectively and have much better approximation results [8]. For distributed networks, shortest path routing can be realized with small routing tables when the graph has small separators, as in the case of planar graphs or graphs with bounded tree width [42].

In networking field, small separators are small cuts through the network. Thus it is important to study separability in many scenarios, e.g., network flow and reliability study [10], routing [33] and distributed short path algorithm [32].

Since separability is closely related to graph partitioning, it naturally fit in the scope of parallel computing [79].

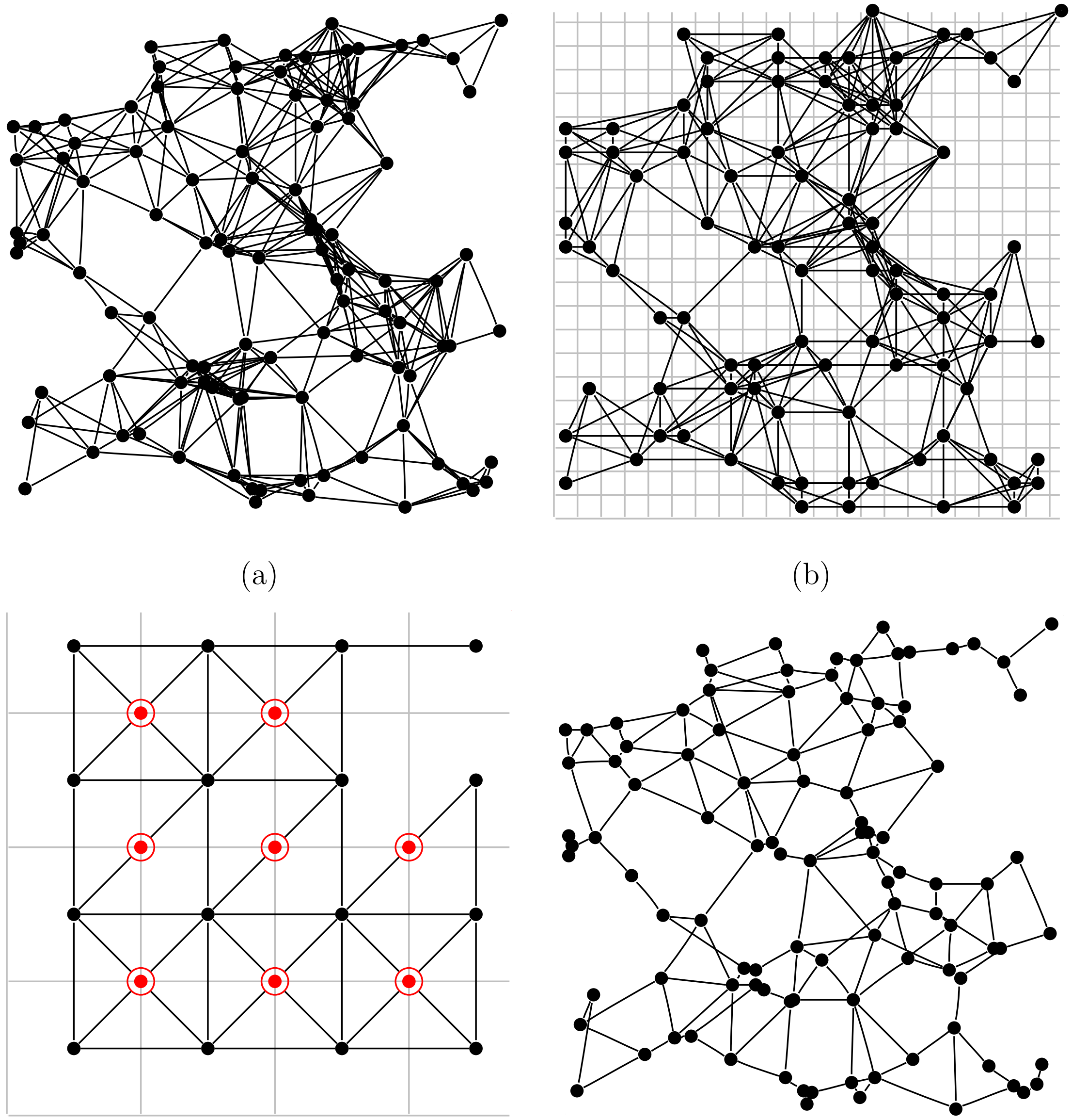
Planar graphs are known to have $O(\sqrt{n})$-size separators [65]. General graphs usually do not have such property. For quasi-UDG, however, we can extract a subgraph or an abstract graph that have small separators and preserve the overall topological features of the network.

C. Grid graph of quasi-UDGs

In this section, we present a distributed algorithm for constructing a grid graph for any quasi-UDG, and prove that the grid graph is well separable. The grid graph, whose node density and edge density are both upper bounded by constants, is an abstraction of the quasi-UDG. A quasi-UDG may have highly variable node and edge densities, which prevent it from having small separators. The grid graph is a "sparsified" version of the quasi-UDG, which retains the distance information for vertices and represents well the deployment region of the quasi-UDG. As a result, the connectivity results for the grid graph can be easily mapped to results for the quasi-UDG. An example of a quasi-UDG and its corresponding grid graph is shown in parts (a) and (b) of Fig. 9, respectively.

1. Construction of the grid graph H

To obtain the grid graph H for a quasi-UDG G, we first impose a grid on the plane. The size of the grid is chosen to be small enough so that all nodes of G within each cell are fully connected. The key operation is then to contract all the nodes within the same cell into a vertex of H, i.e., we view each non-empty cell as a vertex and two vertices of H have an edge if there are two adjacent nodes of G in each cell. The


(a)

(b)

(c)

(d)

Fig. 9. (a) A quasi-UDG G with 100 vertices and $R/r = 0.5$; (b) The grid graph corresponding to G; (c) The auxiliary graph used to find the top level separator of G; (d) The backbone of G.

construction algorithm is given in Fig. 10.

```
Input: $G = (V_G, E_G)$: a quasi-UDG with parameters $R$ and $r$
Output: $H = (V_H, E_H)$: the grid graph for $G$
Impose a grid of cell size $\frac{r}{\sqrt{2}} \times \frac{r}{\sqrt{2}}$ on the plane;
For each cell that has at least one vertex of $G$, $H$ has a corresponding vertex
    positioned at the center of the cell;
There is an edge between two vertices of $H$ if and only if there is at least one
    edge connecting two vertices of $G$ that are, respectively, in the two corresponding
    cells.
```

Fig. 10. Algorithm **GridGraph**: Constructing the grid graph for given quasi-UDG.

All the vertices of G in the same grid cell are adjacent. The algorithm **GridGraph** can be easily implemented in a distributed manner. The following theorem proves the constant upper bounds for the node density, edge density, and the number of edges crossing any given edge in the grid graph H.

Theorem III.2 *The algorithm* **GridGraph** *constructs a grid graph H for a given quasi-UDG G such that: (1) inside any disk of radius y, there are at most $O(y^2/r^2)$ vertices of H; (2) the degree of each vertex in H is upper bounded by $O(R^2/r^2)$; and (3) the number of edges crossing any given edge in H is upper bounded by $O(R^4/r^4)$.*

Proof. From the algorithm we know that the Euclidean distance between any two vertices of H is at least $r/\sqrt{2}$. Hence, if we place an open disk of radius $r/(2\sqrt{2})$ centered at every vertex, then no two disks will intersect. Therefore given any disk of radius y, the number of such open disks intersecting it is upper bounded by $O(y^2/r^2)$, and so is the number of vertices of H inside the disk.

Consider a vertex U of H and denote by $v(U)$ the set of nodes of G inside the cell represented by U. The number of vertices of H within distance $R + r$ from U is bounded by $O(R^2/r^2)$. No node of G in $v(U)$ can be adjacent to a node in $v(V)$ if the distance between V and U is more than $R + r$. Hence, the degree of U is upper bounded by $O(R^2/r^2)$.

Similarly, for an edge $\{U, V\}$ of H, the number of grid vertices within distance $R + r$ to any point on the line segment connecting U and V is also upper bounded by $O(R^2/r^2)$. Therefore, the total number of edges crossing $\{U, V\}$ is upper bounded by $O(R^4/r^4)$. □

If two vertices of H are h hops away from each other, then two vertices of G in the two corresponding cells are at most $2h + 1$ hops away from each other. Note that the above method for constructing grid graphs, and the above results, can be easily extended to higher dimensional spaces.

2. Separability of the grid graph H

Network separability is a fundamental property that leads to efficient network algorithms (in particular, those algorithms based on the divide and conquer paradigms),—fast parallel computation, and improvements in the study of computational complexity [65]. Many applications in wireless ad hoc networks (routing, information retrieval, etc.), as well as quite a number of hard theoretical problems, have more efficient solutions if the underlying graph is well separable. For example, shortest path routing can be realized with small routing tables when the graph has small separators, as in the case of planar graphs or graphs with bounded tree width [42]. Also, NP-hard problems such as vertex cover and independent set, are solvable in polynomial time if the input graph and all its subgraphs have small separators.

In this subsection, we study the separability of the grid graph obtained above. We begin with a formal definition of the separability of graphs.

Definition III.3 *Given a graph G of n vertices, a b-**separator** of G is a set of ver-*

tices whose removal splits G into two non-adjacent subgraphs, each of which contains at most bn vertices. We call a graph G $(f(n'), b)$**-separable** *if every subgraph of G has a b-separator of at most* $O(f(n'))$ *vertices, where* $f(n')$ *is a function of the number of vertices* n' *in that subgraph.*

In order to compute a small separator for the grid graph H, we use the help of a planar auxiliary graph T. Similarly to the construction of the grid graph, we first impose a larger grid on the plane. Each non-empty cell (i.e., there is at least one vertex of H in that cell) of the grid is then mapped to a vertex of the auxiliary graph T placed at the center of the cell. Two vertices of T are connected by an edge if there are two adjacent vertices of H in each of the two corresponding cells. The size of the cells is large enough so that each vertex of T can only have edges connecting to its adjacent cells on the plane. Therefore each edge of T is either horizontal, vertical, diagonal or anti-diagonal. Edge crossings can only involve a diagonal edge and an anti-diagonal edge. Each of these diagonal edges and anti-diagonal edges can cross at most one other edge. Then, we planarize it by adding a virtual vertex at each edge crossing(placed at the crossing point), thus eliminating all edge crossings. (Note that we consider all the edges to be straight line segments.) The detailed construction of the auxiliary graph T is presented in Fig. 11. All the virtual vertices in T are denoted by *red vertices* and the others — which represent cells — are denoted by *black vertices.* Each red vertex has weight zero, while each black vertex has weight equals to the number of vertices of H in the corresponding cell.

Fig. 9(c) shows an example of the auxiliary graph. The longest edge in the auxiliary graph has length $R + \sqrt{2}r/2$, and red vertices are either of degree 2 or 4. Since the cell we apply in this algorithm is large enough (of side length $R + r/\sqrt{2}$) and all black vertices are placed at the centers of their corresponding cells, any black

```
Input: $H = (V_H, E_H)$: a grid graph with parameters $R$ and $r$
Output: $T = (V_T, E_T)$: the auxiliary graph for $H$
Impose a grid of cell-size $(R + \frac{r}{\sqrt{2}}) \times (R + \frac{r}{\sqrt{2}})$ on the plane;
For each cell that has at least one vertex of $H$, $T$ has a corresponding black vertex $v$ whose position is set at the center of the cell; we assign to $v$ a weight equals to the number of vertices of $H$ in that cell;
Add an edge between two black vertices $u$ and $v$ of $T$ if and only if there is at least one edge connecting two vertices of $H$ that are, respectively, in the two corresponding cells;
For each pair of crossing edges $\{u, v\}$, $\{w, x\}$, add a red vertex at the intersection of the two edges and replace those two original edges with four new edges that connect the red vertex, respectively, to the four black vertices $u$, $v$, $w$, and $x$; let the weight of the red vertex be 0;
For each diagonal edge between two black vertices, add a red vertex of weight 0 at the middle of the edge and replace that original diagonal edge with two new edges that connect the red vertex, respectively, to those two black vertices.
```

Fig. 11. **AuxiliaryGraph**: create an auxiliary graph for constructing separators.

vertex may only connect to the eight black vertices around it before the red vertices were added. Therefore, around each black vertex, there can be at most four red vertices; and no two red vertices are adjacent to each other. Formally, we have the following lemma.

Lemma III.4 *Let $N_{T,b}$ be the number of black vertices in the auxiliary graph T. Then T is a planar graph of at most $2N_{T,b}$ vertices and no two red vertices are adjacent.*

Lipton and Tarjan proved in their celebrated Separation Theorem [65] that for any vertex-weighted planar graph of n vertices, there exists a set of $O(\sqrt{n})$ vertices that separates the graph into two non-adjacent subgraphs, each of which weighs at most 2/3 of the total weight of the graph. The separator algorithm presented in [65], however, is relatively complex. For the planar auxiliary graph T, which has a constrained structure, we present a simpler and practically more efficient algorithm for finding such a small separator. Based on that, the algorithm also finds a small separator for the grid graph H.

To find a separator for T, the idea of the algorithm is to peel off the outer face of T repeatedly to find a thin cut. To accomplish that, we build a BFS tree rooted on the outer face of the graph. When a vertex u is discovered in the process, we mark all undiscovered vertices $F(u)$ that share faces with u so that they will be put into no later than the next level of the BFS tree. This can be done by adding edges from u to all vertices in $F(u)$. After we have this BFS tree, one of the so called *fundamental cycles* (a cycle formed by exact one non-tree edge and some tree edges) will contain the separator we want, i.e., it separates T in the most balanced way. The details of the algorithm are presented in Fig. 12.

We now prove that the algorithm **Separator** constructs small balanced separators for H and T. We start with a lemma.

Lemma III.5 *Let $\hat{T}$ be any subgraph of the auxiliary graph T. If its outer face has k vertices, then the number of inner vertices (the vertices not on the outer face) is at most $\lfloor k^2/(2\pi) \rfloor$.*

Proof. The outer face of the planar graph T' is a closed curve (or closed curves, if $\hat{T}$ is disconnected) on the plane. Let $x = R + r/\sqrt{2}$ be the side-length of the cells in the construction of the auxiliary graph T. For each inner vertex of $\hat{T}$, we place a $\sqrt{2}x/2 \times \sqrt{2}x/2$ square centered at it, then rotate the square by 45 degrees. It is simple to see that now these (diamond shaped) squares centered at the inner vertices do not overlap each other. The area of each square is $x^2/2$.

First consider the case when the outer face is connected, i.e. $\hat{T}$ is connected. The outer face of $\hat{T}$ consists of several (at least one) simple cycles. Suppose there are i such simple cycles of size $k_1, k_2, \ldots, k_i$ in the outer face. Note that the value $\sum_{j=1}^{i} k_j$ can be greater than k, the number of vertices in the outer face, because in

```
**Input:** $H$: a grid graph with parameters $R$ and $r$
**Output:** $S_H$: a separator for $H$.
    $S_T$: a separator for $T$. ($T$ is the auxiliary graph of $H$.)
1: Let $T$ be the auxiliary graph of $H$. Let $T'$ be a copy of $T$.
2: Build a breadth-first search (BFS) tree for a dynamically changing graph $T'$ ($T'$ changes because new edges are added to it during the BFS procedure) in the following way: (1) pick a vertex $v$ on the outer face of $T'$ to be the root and start BFS; (2) during the BFS process, when a vertex $u$ is discovered (put it into the BFS tree), for every face containing $u$, add edges from $u$ to as many other vertices in the face as possible so long as $T'$ remains a simple planar graph; if after adding those edges, there are still faces containing $u$ that are not triangulated, add edges to triangulate them arbitrarily. During BFS, the undiscovered neighbors of a vertex are visited in the clockwise order (starting with the parent of the vertex in the BFS tree as the reference point);
3: Check every fundamental cycle in the BFS tree. Let $S_T$ be a fundamental cycle that separates $T'$ (therefore also $T$) in the most balanced way, i.e. the difference between the summation of the weights of vertices in the two separated subgraphs $A_1, B_1$ is minimized.
4: Consider the graph $T$. Let $S'_T$ be a copy of $S_T$. For each red vertex $u$ in $S'_T$ with the set of neighboring vertices $N(u)$, we distinguish two cases: **Case (1)** All vertices in $N(u)$ belong to $A_1$(respectively, $B_1$) except those in $S'_T$. Then, we move $u$ from $S'_T$ to $A_1$(respectively, $B_1$); **Case (2)** Both $A_1$ and $B_1$ contain vertices of $N(u)$. Then, we put all vertices in $N(u)$ into $S'_T$ and move $u$ from $S'_T$ to $A_1$.
5: Let $S_H$ be the set of vertices of $H$ in those cells corresponding to the black vertices of $T$ in $S'_T$. Let $A_2$, $B_2$ be the two sets of vertices of $H$ in those cells corresponding to the black vertices of $T$ in $A_1$ and $B_1$. Clearly, $S_H$ separates $H$ into $A_2$ and $B_2$.
```

Fig. 12. Algorithm **Separator**: constructor seperators for given grid graph.

the summation a vertex can be counted more than once. The simple cycles form the outer face of a planar graph, so the number of times vertices are over-counted is exactly $i-1$. Thus $\sum_{j=1}^{i} k_j = k+i-1$.

First we have $k^2 = \left[\left(\sum_{j=1}^{i} k_j\right) - i + 1\right]^2 = \sum_{j=1}^{i} k_j^2 + \sum_{j=1}^{i}\left(k_j \sum_{l\neq j}^{i} k_l\right) - \sum_{j=1}^{i} 2(i-1)k_j + (i-1)^2 \geq \sum_{j=1}^{i} k_j^2 + \sum_{j=1}^{i}\left[k_j\left(\sum_{l\neq j}^{i} k_l - 2(i-1)\right)\right] \geq \sum_{j=1}^{i} k_j^2$. The last inequality holds because $k_j \geq 2$ and $\sum_{l\neq j} k_l$ contains exactly $i-1$ terms. The equality holds when $i=1$.

Each simple cycle of k_j vertices has k_j edges, thus the perimeter of the cycle is at most $k_j x$. Therefore the area of the region inside the cycle k_j is at most $\lfloor k_j^2 x^2/(4\pi)\rfloor$ and the total area of the regions inside the outer face is bounded by $\sum_{j=1}^{i}\lfloor k_j^2 x^2/(4\pi)\rfloor \leq \lfloor k^2x^2/(4\pi)\rfloor$.

Now if there are several disconnected cycles in the outer face, each connected part, say of k' vertices, surrounds a region of area no more than $\lfloor k'^2 x^2/(4\pi)\rfloor$, since $\sum k'^2 \leq (\sum k')^2 = k^2$, the total area of the regions surrounded by the outer face is also bounded by $\lfloor k^2x^2/(4\pi)\rfloor$. Thus, in all cases, the total number of inner vertices is bounded by $\lfloor k^2x^2/(4\pi)\rfloor / x^2/2 = \lfloor k^2/(2\pi)\rfloor$. □

Define the *depth* of a tree to be the maximum number of edges in a path from the root to a leaf. We have:

Lemma III.6 *Let N_T be the number of vertices in the auxiliary graph T. The BFS tree constructed in Step 2 of the algorithm* **Separator** *is of depth at most $\sqrt{N_T}$.*

Proof. Let d be the depth of the BFS tree. Because of the triangulation operation enforced on the graph T' during the BFS process, for $i = 1, 2, \cdots, d-1$, the vertices at level i (if $i = 1$, include the root as well) of the BFS tree actually contain all the vertices on the outer face of the subgraph induced by the vertices at levels $i, i+1, \cdots, d$. So it suffices to show that if we "peel off" one outer face from T' at each step, T' becomes an empty graph after $t \leq \sqrt{N_T}$ steps.

Let n_x be the number of vertices remaining in the graph T' after x steps. (By convention, define $n_0 = N_T$.) By Lemma III.5, we know that in the x-th step we have "peeled off" at least $\lceil \sqrt{2\pi N_x} \rceil$ vertices. So $n_{t-1} \geq 1$, $n_i \geq n_{i+1} + \lceil \sqrt{2\pi n_{i+1}} \rceil$ for $i = t-2, t-3, \cdots, 0$. Now let us prove that $n_{t-j} \geq j^2$ by induction: when $j = 1$, we have $n_{t-1} \geq 1$ and when $j = 2$, we have $n_{t-2} \geq 4$; suppose our claim is true for $2 \leq j \leq i$; consider the case $j = i+1$, where $n_{t-(i+1)} \geq n_{t-i} + \lceil \sqrt{2\pi n_{t-i}} \rceil \geq i^2 + \lceil \sqrt{2\pi} \rceil i \geq i^2 + 2i + 1 = (i+1)^2$.

We have $N_T = n_0 = n_{t-t} \geq t^2$. So $t \leq \sqrt{N_T}$. □

By Lemma 2 in [65], if a vertex-weighted planar graph has a spanning tree of depth h, then there exists a fundamental cycle of size at most $2h+1$ that separates the graph into two non-adjacent subgraphs each of which weighs no more than 2/3 of the total weight of the graph. As the BFS tree obtained in Step 2 of Algorithm **Separator** is of depth at most $\sqrt{N_T}$, we have the following theorem immediately.

Theorem III.7 *Let N_T be the number of vertices in the auxiliary graph T, and let N_H be the number of vertices in H. Then, the total weight of the vertices of T is N_H, and the set S_T obtained in Algorithm* **Separator** *contains at most $2\sqrt{N_T}+1$ vertices and separates T into two non-adjacent subgraphs each of which weighs no more than $2N_H/3$.*

We now prove that the algorithm **Separator** also finds a small balanced separator for the grid graph H.

Theorem III.8 *Let N_H be the number of vertices in the grid graph H. Then, the algorithm* **Separator** *constructs a separator S_H of size $O(\sqrt{N_H})$ that separates H into two non-adjacent subgraphs each of which has no more than $2N_H/3$ vertices.*

Moreover, the grid graph H is $(\sqrt{n'}, 2/3)$-separable when the weights of all the vertices of H are set to be 1.

Proof. Let N' be the number of black nodes in T. Clearly $N' \leq N_H$; and it is straightforward that each cell corresponding to a black vertex of T contains at most $\lceil 2(R + \sqrt{2}r/2)^2/r^2 \rceil$ vertices of H. Hence we have $N' = \Theta(N_H)$. From lemma III.4 we know that the number of red vertices is no more than N', and the total weight of vertices in T is N_H. Hence the separator S_T for T contains no more than $2\sqrt{2N'} + 1$ vertices whose weights sum up to $O(\sqrt{N_H})$, and separates T into two parts each of which weighs no more than $2N_H/3$.

Now we show that after Step 4 of Algorithm **Separator**, S'_T is still a separator for T of size $O(\sqrt{N'})$, and A_1 and B_1 are still of weight no more than $2N_H/3$. Consider any red vertex $u \in S'_T$ in Step 4, in the case where all of u's neighbors are either in S_T or A_1 (respectively, B_1), $S'_T \setminus \{u\}$ can separate T into $A_1 \cup \{u\}$ and B_1 (respectively, A_1 and $B_1 \cup \{u\}$). Note that u has weight 0, so moving u from S'_T to A_1 (or, B_1) does not change their weights. In the complimentary case, the algorithm moves all u's neighbors into S'_T and moves u into A_1; clearly S'_T still separates A_1 and B_1. And by doing that, we decrease the weights of both A_1 and B_1. The size of S'_T increases by at most 3 for each red vertex.

Hence after Step 4, we have replaced all red vertices in S'_T by black ones, increasing the size of S'_T by at most three times, not increasing the weights of A_1 and B_1. Most importantly, S'_T still separates A_1 and B_1. Therefore S'_T is still of size $O(\sqrt{N'}) = O(\sqrt{N_H})$, and the weights of A_1 and B_1 are no more than $2N_H/3$. Each cell corresponding to a black vertex of T contains a bounded number of vertices of H, so S_H is of size $O(\sqrt{N_H})$. Also, the number of vertices in A_2 (resp., B_2) equals the weight of A_1 (resp., B_1) (at most $2N_H/3$).

By the construction of the auxiliary graph T, if no two black vertices are joined by an edge or two edges with a red vertex in the middle, there is no edge connecting vertices of H in those two corresponding cells. A_1 and B_1 are not adjacent in T, and S'_T has no red vertex. So A_2 and B_2 obtained in Step 5 are not adjacent in H, and S_H separates A_2 and B_2 in H.

It is simple to see that any subgraph of H can be used as the input of Algorithm **Separator**, and the above arguments still hold. Hence H is $(\sqrt{n'}, 2/3)$-separable. □

For some applications, a perfectly balanced separator is desirable. By using the same technique described in [65], we can construct a separator of size $O(\sqrt{N_H})$ that separates H into two parts each of which has no more than $N_H/2$ vertices. The idea is to separate the larger part of the outcome of the algorithm recursively. Hence we have:

Corollary III.9 *Let N_H be the number of vertices in the grid graph H. H is $(\sqrt{n'}, 0.5)$-separable.*

For the grid graph, we can develop a shortest-path routing scheme based on its separators, using the idea of distance labeling [42]. We can then transform it into a compact routing scheme for the underlying quasi-UDG G with a small stretch factor. We leave the details of the routing algorithm and the extended results to Section D.

3. Separability for degree/edge crossing bounded graphs

Actually for any degree bounded graph G of n vertices deployed on a plane with $O(1)$ number of edge crossings at any edge, the technique described previously can be used to find balanced separator for G. The algorithm is outlined in Fig. 13

The following theorem proves that the above algorithm finds a constant-balanced separator for graph G of size $O(\sqrt{n})$.

```
Input: $G = (V_G, E_G)$: a graph with constant edge crossings and bounded degree
Output: A separator for $G$
1: add a virtual vertex at each edge crossing and we have a planar auxiliary graph
   $T$;
2: Use Lipton-Tarjan's algorithm to find a balanced separator for $T$;
3: If a virtual vertex $u$ is in the separator, replace it by all four endpoints(original
   nodes) of the two edges crossing at $u$ and remove $u$ from $T$, let the residual graph
   be $T'$;
4: return the separator found.
```

Fig. 13. Algorithm **Separator-Gen**: construct separators for graphs of constant degree and edge crossings.

Theorem III.10 *The algorithm* **Separator-Gen** *constructs a* $(\sqrt{n}, O(1))$*-separator for the input graph* G *with edge crossing bounded by a constant* C *and node degree bounded by* D*.*

Proof. It is easy to see that the auxiliary graph T has at most $Dn/2$ edges and $(CD/4+1)n$ vertices. Hence the separator found in step 2 by the Lipton-Tarjan's algorithm has size $O(\sqrt{n})$ and it separates the graph T in to two equally parts, each of size N and $n/2 - O(\sqrt{n}) \leq N \leq (CD/4+1)n/2$. The third step adjusts the separator so that every node is real. And the separator obtained actually separates T' and still has size $O(\sqrt{n})$. The graph T' also has $O(n)$ nodes and the separator returned separates T' into two parts T_1, T_2 (each of size $(N \pm O(\sqrt{n}))$. More importantly, every virtual vertices in T_1 and T_2 is surrounded by real(original) nodes. Since each original node can create at most DC many virtual vertices, each of T_1 and T_2 must contain at least $[(N - O(\sqrt{n})]/(DC+1) > n/[2(DC+1)]$ many real nodes. Apparently this separator also separates G into two parts, each of size at least $n/[2(DC+1)]$ (at most $n[2(DC+1)-1]/[2(DC+1)] = O(n)$). Hence this size-$O(\sqrt{n})$ separator of G is a $O(1)$-separator. □

Apply the same technique as described in [65] we can then obtain a perfectly balanced separator for graphs with bounded degree and edge crossings.

Corollary III.11 *A graph G of n nodes with node degree and edge crossings bounded by $O(1)$ is $(\sqrt{n}, 0.5)$-separable.*

D. Backbone with constant stretch factor

We denote by a *backbone* of a given graph any spanning subgraph. Examples of backbones are spanning trees. Backbones, particularly those with small stretch factors and node degrees, have very important applications in wireless communication because they can help reduce signal interferences and simplify algorithms (such as routing algorithms).

In this section, we present a distributed localized construction of a backbone for a quasi-UDG with constant stretch factor, constant node degree, and a small number of edge crossings. We will show in Section D that these backbones can help reduce the routing table size in our routing scheme.

1. Backbone construction

Energy is a major limitation in wireless networks. Accordingly, the stretch factor of a backbone is often defined based on energy consumption. We start with the formal definition.

Definition III.12 *Denote by $|uv|_G$ the Euclidean distance between any two nodes u and v. Let $(u = u_1, u_2, \cdots, u_k = v)$ be a path from u to v in the graph G. The* **communication cost** *between u and v following the above given path is defined as:*

$$c_G(u, v) = \textstyle\sum_{i=1}^{k-1} \alpha |u_i u_{i+1}|_G^{\beta},$$

where β is the power exponent with $2 \leq \beta \leq 5$, and α is a scaling factor linear in the number of bits sent. If there is no path from u to v then $c_G(u, v)$ is defined to be $+\infty$.

Definition III.13 *Given a graph $G = (V, E)$ and a backbone B of G, the* **stretch factor** *of B is defined as:*

$$\max_{u,v \in V} \left\{ c_{B,min}(u, v)/[c_{G,min}(u, v)] \right\},$$

where $c_{B,min}(u, v)$ and $c_{G,min}(u, v)$ denote the minimum *communication cost (over all paths) between u, v in the graph B and G, respectively.*

The stretch factor defined above is also called the *power stretch factor*. We say that a backbone is *energy efficient* if its power stretch factor is bounded by a constant.

We next present a distributed localized algorithm that, given a quasi-UDG G, constructs a backbone B whose maximum degree is $O(R^2/r^2)$, the average number of edges crossing a given edge is $O(R^4/r^4)$, and the power stretch factor is bounded by $3 + \epsilon$, for any constant $\epsilon > 0$.

We classify the edges in the quasi-UDG G into two types: **short edges** whose length is not greater than r, and **long edges** whose length is strictly larger than r.

Let E_{short} the set of short edges. Then the graph induced by E_{short} is a unit disk graph of unit length r. Denote this graph by U_{short}, and note that U_{short} may not be connected.

The results in [50] describe a distributed localized algorithm that, given a unit disk graph and a positive integer $k \geq 9$ as a parameter, constructs a planar power spanner of the graph with degree at most $k + 5$ and stretch factor $1 + (2\sin(\pi/k))^\beta$, where β is the power exponent. Since each component in U_{short} is a unit disk graph, we can apply the algorithm in [50] to each component to construct a planar power spanner of the component of degree bounded by $k + 5$ and stretch factor $1 + (2\sin(\pi/k))^\beta$.

Let B be the set of edges in the spanners of all these components.

Now impose a grid of cell-size $r/\sqrt{2} \times r/\sqrt{2}$ on the plane. Note that any two points in the same cell are connected in G, and that any long edge must connect points in two different cells. For each pair of cells, add to B the shortest edge between those two cells (i.e., the shortest edge having one endpoint in one of the two cells and the other endpoint in the other cell). Observe that determining the shortest edge between two cells can be done in a localized fashion since the points in a cell form a clique. This completes the construction of B. Note that after adding the long edges to B, G' may no longer be planar. However, as we will show below, the average number of edges crossing a given edge will be bounded by a constant. The algorithm is summarized in Fig. 14.

```
Input: $G$: a quasi-UDG with parameters $R$ and $r$; an integer parameter $k \geq 8$
Output: $B$: a backbone of $G$
let $U_{short}$ be the unit disk graph induced by the set of edges in $G$ of length at
    most $r$;
let $B$ be the set of edges computed by applying the algorithm in [50] to each
    component of $U_{short}$ to compute a spanner of the component;
impose a grid of cell-size $\frac{r}{\sqrt{2}} \times \frac{r}{\sqrt{2}}$ on the plane;
for every two grid-cells do
    add to $B$ a shortest edge between the two cells (in case such
    an edge exists);
return $B$;
```

Fig. 14. Algorithm **Backbone**: construct a backbone for a given quasi-UDG

Theorem III.14 *For any integer parameter $k \geq 9$, the algorithm* **QuasiUDG-Backbone** *constructs a backbone of the given quasi-UDG G whose maximum degree is $O(R^2/r^2)$, average number of edges crossing a given edge is $O(R^4/r^4)$, and power stretch factor is $3 + 2^{\beta+1}\sin^{\beta}(\pi/k)$ (which, for any $\epsilon > 0$, is bounded by $3 + \epsilon$ for large enough k). Moreover, the algorithm is localized and exchanges $O(m)$ messages, where*

m is the number of edges in G.

Proof. Call two grid-cells *adjacent* if there is an edge between a node in the first cell and another node in the second cell, and note that the number of cells that are adjacent to a given cell is $O(R^2/r^2)$.

Since the algorithm in [50] constructs a backbone for each component in U_{short}, B contains a spanning subgraph for each component in U_{short}. Since all the vertices in a given grid-cell form a clique, and hence belong to the same component in U_{short}, all the vertices in a given cell remain connected in B. From step 4 of the algorithm, every two adjacent cells that were adjacent in G, remain adjacent in B by virtue of adding the shortest edge between the two cells to B. It follows from the above, and from the connectivity of G, that B is a spanner of G.

Since each node in the spanner of U_{short} constructed by the algorithm in [50] has degree bounded by $k + 5$, each node in B has at most $k + 5$ short edges incident on it in B. Since for any node in B the number of cells adjacent to the cell containing the node is $O(R^2/r^2)$, any node in B can have at most $O(R^2/r^2)$ long edges incident on it in B. It follows that every node has $O(R^2/r^2)$ incident edges in B.

To prove the bound on the number of edge-crossings, note that since the algorithm in [50] constructs planar spanners of the components of U_{short}, no two short edges in B cross. Therefore, the number of edge-crossings is the number of crossings between short edges and long edges, plus the number of crossings between long edges. We bound each of these numbers next.

Let e_{short} be a short edge in B. Then e_{short} must join two nodes in the same grid-cell C. Any long edge in B that crosses e_{short} must join two nodes, each located in an adjacent cell to C. Since cell C has $O(R^2/r^2)$ adjacent cells, C has $O(R^4/r^4)$ pairs of adjacent cells. Since exactly one edge between any two adjacent cells is kept in B, the total number of long edges crossing e_{short} is $O(R^4/r^4)$. Therefore, the total number of edge-crossings involving short edges is $O(m \cdot R^4/r^4)$.

To bound the number of edge-crossings between two long edges, let e_{long} be a long edge. Suppose that the endpoints of e_{long} reside in the two cells C and C'. Any long edge crossing e_{long} must join a node in a cell that is adjacent to either C or C' to another node in a cell adjacent to C or C'. Since there are $O(R^4/r^4)$ pairs of cells that are adjacent to either C or C', and since exactly one edge between any two adjacent cells is kept in B, the total number of long edges crossing e_{long} is $O(R^4/r^4)$. Therefore, the total number of edge-crossings involving long edges is $O(m \cdot R^4/r^4)$.

It follows that the total number of edge-crossings is $O(m \cdot R^4/r^4)$, and hence the average number of edges crossing a given edge is $O(R^4/r^4)$.

To prove the bound on the stretch factor, it is enough to show that every edge in G is stretched by no more that $3 + 2^{\beta+1}\sin^{\beta}(\pi/k)$ in B. Let $e = (u, v)$ be an edge in G. If $e \in B$ then the statement follows directly. Suppose now that $e \notin B$. If e is a short edge, then since B contains a power spanner of U_{short} with stretch factor $1 + 2^{\beta}\sin^{\beta}(\pi/k)$, the result follows. If e is a long edge, let C_u and C_v be the two cells containing u and v, respectively. Let $e_{min} = (u', v')$ be the shortest edge between C_u and C_v that was included in B, where $u' \in C_u$ and $v' \in C_v$. Since B contains a power spanner of C_u of stretch factor $1 + 2^{\beta}\sin^{\beta}(\pi/k)$, there is a path $P_{uu'}$ in B from u to u' of cost at most $1 + 2^{\beta}\sin^{\beta}(\pi/k)|uu'|^{\beta}$. Similarly, there is a path $P_{v'v}$ in B from v' to v of cost at most $1 + 2^{\beta}\sin^{\beta}(\pi/k)|vv'|^{\beta}$. It follows that the path from u to v in B consisting of the concatenation of $P_{uu'}$, e_{min}, and $P_{v'v}$, has cost bounded by $1 + 2^{\beta}\sin^{\beta}(\pi/k)|uu'|^{\beta} + 1 + 2^{\beta}\sin^{\beta}(\pi/k)|vv'|^{beta} + |u'v'|^{\beta}$. Since (u, u') and (v, v') are both short edges, and hence, are shorter than the long edge $e = (u, v)$, and since e_{min} is not longer than e, it follows that there is a path in B of cost at most $3 + 2^{\beta+1}\sin^{\beta}(\pi/k)|uv|$. Noting that $2^{\beta+1}\sin^{\beta}(\pi/k)$ can be made arbitrarily small by choosing a sufficiently large parameter k, the stretch factor can be bounded by $3 + \epsilon$ for any $\epsilon > 0$.

Finally, to analyze the number of messages exchanged by the algorithm, note first that algorithm in [50] exchanges $O(m)$ messages. To compute the shortest edge between two adjacent cells, each node in the cell computes the shortest edge incident on it and whose other point is in the other cell. Then vertices in one cell elect the shortest among all these edges. Since all the vertices in one cell form a clique, and since the number of adjacent cells to a given cell is $O(R^2/r^2)$, the total number of messages exchanged for computing the shortest edges between adjacent cells is $O(m)$. Moreover, all the computation can be done locally: every node only communicates with its neighbors. It follows that the total number of messages exchanged by the algorithm is $O(m)$. □

The following theorem is then straightforward by Corollary III.11.

Theorem III.15 *The backbone constructed by the algorithm* **QuasiUDG-Backbone** *is* $(\sqrt{n}, 0.5)$*-separable where* n *is the number of nodes in the network.*

E. Applications and performance evaluation

In this section, we first present out routing algorithm based on the separators, then prove the bound for the path stretch factor of our routing protocol. As the second part of the section, we show the simulation results of the backbone constructions and the routing performance of our routing algorithms to verify the theoretical bounds we prove.

1. A routing scheme based on the separators

As one of the applications of the small separators of the grid graphs, we present a routing scheme for quasi-UDG based on the grid graph and analyze its performance. Our routing scheme is suitable for systems in which the size of the messages itself is relatively large. We will give the simulation results later in this section.

Our routing scheme is based on the distance labelling scheme described in [42]. The basic idea of distance labelling is to give each vertex a label such that the distance between two vertices can be computed using only their labels. A straightforward labelling scheme is to store in each node a full table of the distances to all the other vertices. The goal of the distance labelling scheme in [42] is to find the labels of minimum length. The separability of the underlying graph is a key factor of how good a distance labelling scheme is available for the network. In [42] the authors proved that for a graph which has a separator of size k, there is a distance labelling scheme of label size $O(k \log n + \log^2 n)$, and the distance between two nodes can be computed in time $O(\log n)$, where n is the number of nodes in the network.

Although a quasi-UDG G may not possess a small separator, we have proved that the grid graph H with n vertices constructed for G does have a balanced separator of size $O(\sqrt{n})$. Conceptually, our routing protocol utilizes two-level routing: virtually, the message is sent in the grid graph from the cell containing the source to the cell containing the destination, via the shortest path in the grid graph; in reality, the routing is implemented in the underlying quasi-UDG to route from cell to cell. (Note that in each cell, the quasi-UDG vertices are fully connected, so routing from one cell to the next takes at most two hops.) The basic idea to achieve shortest path routing in the grid graph is to split H into two non-adjacent parts using the small separator. Each vertex of H remembers the distance to all separator vertices. Thus, two vertices in the two parts (or the separator) can compute their shortest path distance using that information, because their shortest path must go through a separator vertex. We recursively apply the same process to partition each part into small parts, to enable any two vertices to compute their shortest path distance using their stored information (their labels). We stop partitioning a part when its size is below a certain constant. (We call such a part a *basic block*.) Since we use balanced separators, the process ends after $O(\log n)$ levels of partitioning.

For a vertex W of H, let $v(W)$ be the set of quasi-UDG vertices of G that reside in the cell corresponding to W. The following list contains the information that each vertex $u \in v(W)$ in G stores in our protocol.

- The minimum distances (in H) to all the separator vertices that are on the boundaries of all the partitions W is in;
- The neighboring quasi-UDG vertex through which it can get to other cells adjacent to W in H;
- A shortest path routing table for the vertices of H in the basic block where u resides.

The routing protocol assumes that the source knows the label of the destination. This piece of information can be obtained from location service. Since location service is not directly related to our topic, we skip the details here.

If the destination is not in the same cell as the source, the message will follow a shortest path in H from the source cell to the destination cell. By utilizing the second part of the list (label), a vertex can send a message to any of its neighboring cell in two hops. Within a basic block, the third part of the routing table points out the shortest path between cells directly. Our routing protocol compares favorably with shortest path routing algorithms and compact routing algorithms for general networks for its significantly smaller routing table size and maintained constant stretch factor.

Now we are ready to prove the following theorem.

Theorem III.16 *For any quasi-UDG G of N_G vertices, let $h(u, v)$ be the minimum hop distance between vertices u and v. The above routing protocol guarantees the routing path from u to v to have at most $2h(u, v) + 1$ hops, for any two vertices u and v. The size of the routing table at each node and the message overhead are both $O(\sqrt{N_G} \log N_G)$.*

Proof. In the routing protocol described above, the first part of a node's routing table is of size $O(\sqrt{N} \log N)$. The second and third parts of the routing table both consist of a constant number of entries because the number of neighboring cells and the number of cells in each basic block are both constants. The size of the routing table is then $O(\sqrt{N} \log N)$. Inside each message we need only to carry the label of the destination vertex, so the overhead in the message size is also bounded by $O(\sqrt{N} \log N)$.

Given a path p from u to v, let $d(p)$ denote its number of hops, and let $c(p)$ denote the number of times the path p travels from one cell to another. Let p_{opt} be the shortest path from u to v, and let p' be the routing path of our protocol. Clearly, $c(p_{opt}) \leq d(p_{opt})$, and $c(p') \leq c(p_{opt})$ because our protocol uses shortest path routing in the grid graph. The path p' travels from one cell to the next in at most two hops, so $d(p') \leq 2c(p') + 1$. It follows that $d(p') \leq 2d(p_{opt}) + 1$. □

Sometimes we are more concerned about the energy consumption than the hop distance if the wireless nodes are able to adjust their communication range to save power. Let the communication cost be as defined in Section C. In reality, it is infeasible for a node to reduce its communication range to an infinitely small number. There is always a constant range δ below which the wireless node cannot reduce its communication range. With this assumption, we prove the following theorem.

Theorem III.17 *Let the communication cost be as defined in Section C, and assume that the minimum communication range is δ. (Therefore, the communication cost of an edge of Euclidean length d is $\alpha \cdot (\max\{d, \delta\})^{\beta}$.) Then, the communication cost of a routing path from u to v generated by our routing protocol is upper bounded by a constant times the minimum communication cost over all the paths from u to v.*

PROOF. Let p_{opt} be the optimal path from u to v with the minimum communication cost C_{opt}, and let p' be the routing path of our algorithm with cost C'. If u, v are in the same cell of the grid graph H, then $C_{opt} \geq \alpha\delta^{\beta}$, and $C' \leq \alpha r^{\beta}$ since vertices in the same cell form a clique. So $C' \leq (r^{\beta}/\delta^{\beta})\alpha\delta^{\beta} \leq (r^{\beta}/\delta^{\beta})C_{opt} = C_{opt} \cdot O(1)$.

Now assume that u, v are in different cells of H. Let l_{opt} and l' denote, respectively, the number of hops in p_{opt} and p'. By Theorem III.16, $l' \leq 2l_{opt} + 1$. So $C' \leq l'\alpha R^{\beta} \leq (2l_{opt}+1)\alpha R^{\beta} \leq 2l_{opt} + 1/l_{opt} \cdot R^{\beta}/\delta^{\beta} \cdot l_{opt}\alpha\delta^{\beta} \leq 3 \cdot R^{\beta}/\delta^{\beta} \cdot C_{opt} = C_{opt} \cdot O(1)$.

□

2. Simulations

We conducted extensive simulations to evaluate the performance of our backbone construction algorithm and routing protocol. The performance has been stable and consistent. In the following experiment, we randomly deploy N quasi-UDG nodes in a 2-D space of size 1500×1500. We increase the number of nodes, N, in the system from 1000, 1500 to 2000 to verify the effects of density change on the performance. We also increase the value R/r from $1, 1.5, 2, 3$ to 10 to see the performance of our algorithms for different wireless connectivity models. To mimic nontrivial network topologies, we randomly generate a big hole of radius randomly picked in the range $[R, 2R]$ and five small random holes of radius picked in the range $[0, R]$. The centers of the holes are uniformly randomly chosen in the plane. If the distance between two nodes is in the range $(r, R]$, we put a direct link between them probabilistically. For each configuration, we run the simulation 30 times and take the average of the performance metrics.

We would like to point out that the performances of our routing algorithm and backbone construction method are relatively independent of the size of the network. Our theoretical bounds and the simulation results both show that the quality of the backbone constructed and the stretch of the routing paths are closely related to the ratio of r to R.

a. Backbone construction

In the backbone construction simulations, we measure the power stretch factor, maximum degree, the average degree and the average number of edge-crossings on an edge in the backbone constructed and compare them to the original graph. For node pairs whose distances are between r and R, we adjust the probability of their being connected to ensure an expected average degree of 10 in order to compare the results between different densities and values of R/r. The results shown in Table V and Fig. 15 are for backbones constructed by only performing the first step and the last step in Algorithm **Backbone**. We eliminated the results for the case when $R = r$ since there the backbones are known to be planar with power stretch factor being 1 because of the Gabriel operation in the first step of the algorithm. Our results showed that for all configurations the backbones have very small power stretch factors, much smaller maximum degree and in most cases, we can bring the average degree to below 6 (which is the upper bound of the average degree for planar graphs). Even when

$R/r = 10$, the average degree of our backbones is no more than 8. As for the number of crossings, our algorithm reduced it by at least 60% in all cases.

Table V. Power stretch factor for the backbones($\beta = 2$).

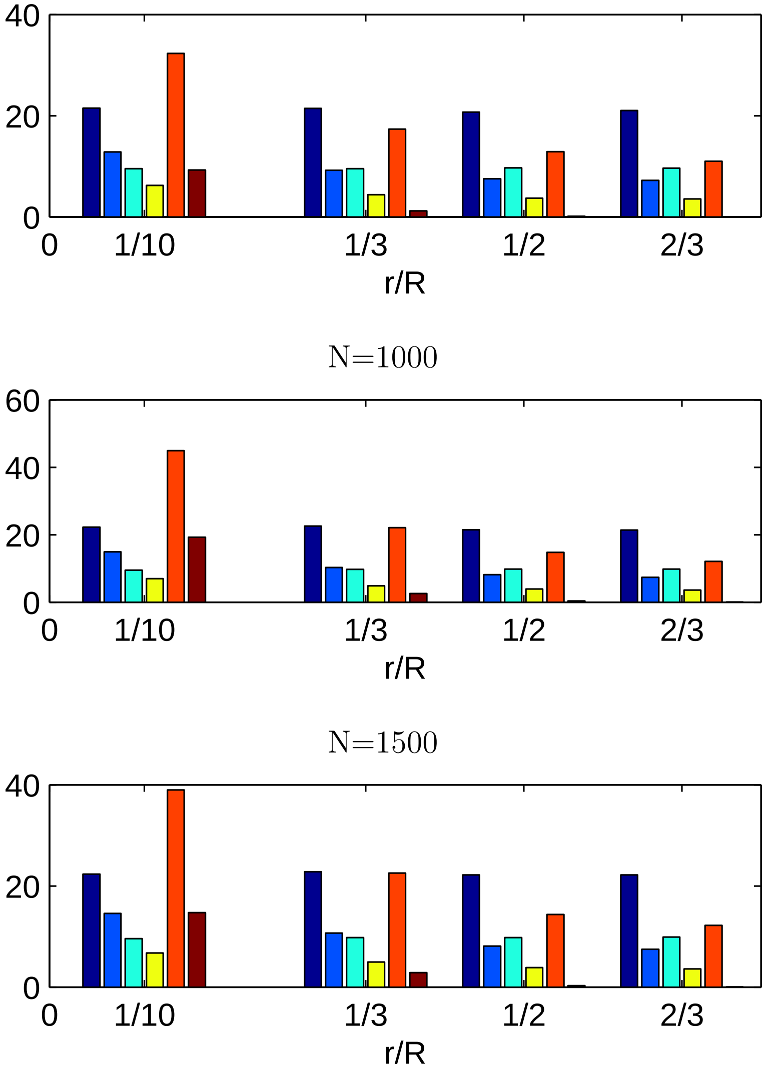
	Stretch Factor			
N	R/r=10	R/r=3	R/r=2	R/r=1.5
1000	1.048	1.141	1.190	1.184
1500	1.044	1.155	1.198	1.129
2000	1.046	1.176	1.239	1.204

From the three bar graphs in Fig. 15, the reduction in the metrics is quite uniform. It implies that the performance of our algorithm is stable for different sizes of the network.

b. Routing performance

We apply our routing protocol not only to the original quasi-UDGs but also to the backbones we obtain. To study the performance, we measure the maximum label size, average label size and the stretch factor of routing path that is defined as the distance in the actual routing path over the shortest path between the source and the destination. The length of the path for routing in the original graphs is defined as the hop distance between two nodes, while in the backbones, we use the communication cost with $\beta = 2$ as the length of the path. In both cases we randomly pick 1000 source-destination pairs in the graph, simulate the routing process and compare the length of the path with the shortest. Due to the page limit we only present the results on the quasi-UDG with expected degree 10 and remark that the results are consistent for graphs with other edge densities.

Table VI shows the average values of the maximum label size and the average label size (with a node ID as a unit) over the experiments for two cases. We observe that the label sizes with the algorithm applied to the backbones are smaller than those



N=1000

N=1500

N=2000

Fig. 15. The maximum degree, the average degree, and the average number of edges crossing an edge for quasi-UDGs and their backbones. The 6 bars in each group are, from left to right (1) maximum degree in quasi-UDG; (2) maximum degree in backbone; (3) average degree in quasi-UDG; (4) average degree in backbone; (5) average number of crossings per edge in quasi-UDG; (6) average number of crossings per edge in backbone. Note that in some groups, the last bar is not shown, because the average number of crossings per edge in backbones equals 0 there.

to the original graphs. This is mainly because the backbones are sparser than the original quasi-UDGs, hence the grid graphs we get are also sparser and have smaller separators. We will see later that this advantage comes at a cost of slightly larger stretch factors.

Table VI. Label sizes of routing scheme based on separators.

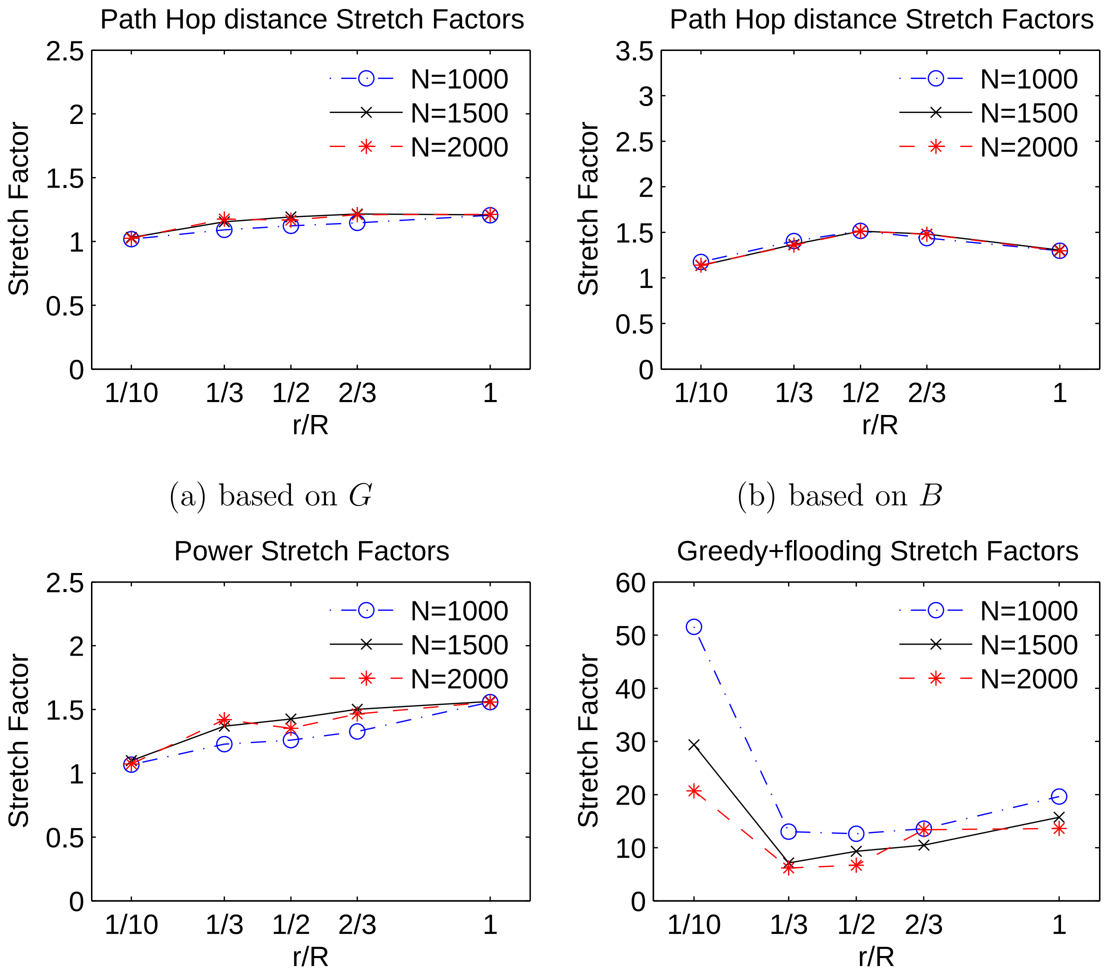
		On original		On backbone	
N	R/r	Max Size	Avg Size	Max Size	Avg Size
1000	10	220.667	155.911	184.800	131.614
1000	3	139.733	106.553	130.733	89.561
1000	2	129.933	91.318	97.434	68.825
1000	1.5	100.367	72.960	90.634	60.562
1000	1	75.834	55.876	72.434	51.231
1500	10	325.900	233.056	287.567	205.997
1500	3	218.933	152.448	166.067	121.646
1500	2	165.767	115.528	143.400	90.668
1500	1.5	133.900	91.548	122.967	78.739
1500	1	102.167	72.627	90.800	63.852
2000	10	320.033	243.665	292.733	219.245
2000	3	232.100	172.638	219.200	151.097
2000	2	196.500	142.079	151.400	102.410
2000	1.5	271.500	224.919	124.433	86.575
2000	1	115.867	84.759	108.933	74.227

Fig. 16(a) shows the average hop distance stretch factors of the routing path for the routing algorithm applied to the original graphs directly. In all cases we have the path stretch factors no larger than 1.3.

Fig. 16(c) shows the power stretch factors and Fig. 16(b) shows the hop distance stretch factors of the routing paths when the algorithm is applied to the backbones. The hop stretch factors shown in Fig. 16(b) are moderately larger than the ones shown in Fig. 16(a). It is the price we paid for the reduction in the size of the routing tables.

It looks interesting from the figures that when R/r is large (10), the algorithm generally performs better than the other cases. This is because to maintain the same average node degree of the graphs we have to decrease the value of r. In that case a grid graph actually describes the original graph more accurately and with more details. Hence the sizes of the labels are larger(see Table VI), but the paths we discovered are closer to the shortest ones.

We have also implemented the well known greedy-forwarding plus local-flooding routing algorithm, and performed the same number of experiments on the same set of graphs. The average stretch factors are shown in Figure 16(d). Our results indicate that compared to that algorithm, the routing protocol based on separators has a much better stretch factor because of its robustness to the existence of holes.



(a) based on G

(b) based on B

(c) based on B

(d) Greedy+flooding

Fig. 16. Stretch factors for routing algorithms. G is the original graph, and B is the backbone.

CHAPTER IV

ROBUST PLANARIZATION OF UNLOCALIZED WIRELESS SENSOR NETWORKS

Wireless sensor networks need very efficient network protocols due to the sensors' limited communication and computation capabilities. Network planarization – finding a planar subgraph of the network that contains all the nodes – has been a very important technique for many network protocols. It first became the foundation of various well known routing protocols, including GPSR, GOAFR and several other protocols. Since then, it has also been used in numerous other applications, including data-centric storage, network localization, topology discovery, etc. However, an important problem remains: network planarization itself is very difficult. So far, efficient planarization algorithms exist only for very restrictive models: the network must be a unit-disk graph, and accurate measurements related to the node locations (e.g., node positions or angles between adjacent links) need to be known. For more practical network models, where the transmission ranges are usually not uniform and sensors cannot obtain their accurate location information via expensive localization devices, no efficient planarization algorithm is available.

In this chapter, we present a novel method that robustly planarizes sensor networks of a realistic model: networks with non-uniform transmission ranges and unlocalized sensors (that is, static sensors whose locations are unknown). Our method starts with a simple shortest path between two nodes, and progressively planarizes the whole network. It achieves both efficiency and a good planarization result. We present two planarization algorithms for different settings. Our results not only solve the planarization problem, but also outperform some known results in the graph drawing research field. We demonstrate the practical performance of our method –

as well as its application in topology discovery, – through extensive simulations.

A. Introduction

Wireless sensor networks usually need very efficient network protocols due to the limited communication and computation capabilities of small sensors. Therefore, it is especially important to exploit the special topological properties of sensornets for the many network functions. A common observation is that the topology of a wireless sensornet usually has a strong correlation with the geometry of the sensor field. That observation has been used in numerous notable applications, including geographic routing [54], etc. In these applications, network planarization has become a very important technique, because a well planarized network not only exhibits the geometric properties of the sensor field, but can also be efficiently utilized in network protocols.

The objective of network planarization is to get a connected planar subgraph of the network that contains all the nodes of the network. To well reflect the geometry of the sensor field, the planar subgraph should contain many links, so that the hop distance (or communication distance) between nodes does not change a lot after planarization. Network planarization first became the foundation of several well known geographic routing protocols, including GPSR [54], Compass Routing [57], GOAFR [60], etc. Such protocols use the faces in the planar subgraph to perform perimeter routing, which guarantees packet delivery. Since then, network planarization has also become a fundamental tool in numerous other applications, including data-centric storage [81], network localization [72] and topology discovery [97] [36] [37]. Here the data-centric storage schemes use the planar graph to help determine on which set of nodes to store each datum, as well as for routing; the network localization schemes can use the properties of planar graphs to facilitate localization; and the topology discovery schemes can use the faces of the planar graph to recognize and locate boundaries and holes in the sensor field. Discovering boundaries and holes in a sensor field is useful for understanding the collected sensor data (because the meaning of sensor data often depends on the type of physical environment where they are collected), for understanding the sensing environment (e.g., building floor plan, transportation network, lakes) and detecting events (e.g., fire in a forest), and for load-balanced routing.

Although network planarization has been proven to be an excellent technique for sensor network protocols, an important problem remains: network planarization itself is difficult. So far, efficient planarization algorithms exist only for network of very restrictive models: unit-disk graphs with accurately known measurements related to the nodes' physical locations. The measurements are the nodes' positions, the angles between all adjacent links, or the lengths of all links. A unit-disk graph (UDG) is a graph where two nodes have a link between them if and only if their physical distance is at most one. So it corresponds to a network where the transmission ranges are the same for all nodes and in all directions. For unit-disk graphs with the accurate location measurements, network planarization can be performed efficiently and distributively by using the techniques based on Gabriel-graph, Relative-Neighborhood graph or Delaunay graph [12]. When the network knows the accurate node positions and is very similar to the unit-disk graph, – specifically, when it is the so called $\sqrt{2}$-quasi unit disk graph, – although no network planarization algorithm is known, the problem can be circumvented to some extent by using "virtual links" [5]. However, the virtual links may need to be realized by long paths in the network, which makes the approach not so useful for many applications. For more general networks, no efficient planarization method is known. Practical sensor networks often deviate significantly from the unit-disk graph model. The transmission ranges of sensors usually vary substantially in different directions due to reasons including multi-path fading, antenna design, etc., and it is common to observe a variation ratio up to five or more [40]. Also, it is often hard for sensors to obtain accurate location measurements via expensive localization devices (e.g., GPS) or localization algorithms [12]. No efficient network planarization algorithm is currently available for such practical wireless sensor networks.

In this chapter, we present a novel method that robustly planarizes sensor networks of realistic models: networks with non-uniform transmission ranges and unlocalized sensors (that is, sensors whose location information is unknown). The method starts with a simple shortest path between two faraway nodes in the network, and progressively planarizes the whole network.

The key for our approach is to solve the so called BIPARTITE PLANARIZATION problem. It has been proved to be NP-hard [28]. We present two planarization algorithms for different settings. We first present a $(2 + 3\epsilon)$-approximation algorithm for this problem. The algorithm is applicable to general networks, and achieves the best known approximation ratio. It outperforms the known results in the graph drawing research field [27].

We then present a fixed parameter tractable (FPT) algorithm that solves the problem exactly (namely, it finds the optimal solution). The algorithm uses the key observation that when a certain parameter is small, the problem can be solved in polynomial time. We show the usefulness of the algorithm to practical networks by simulations.

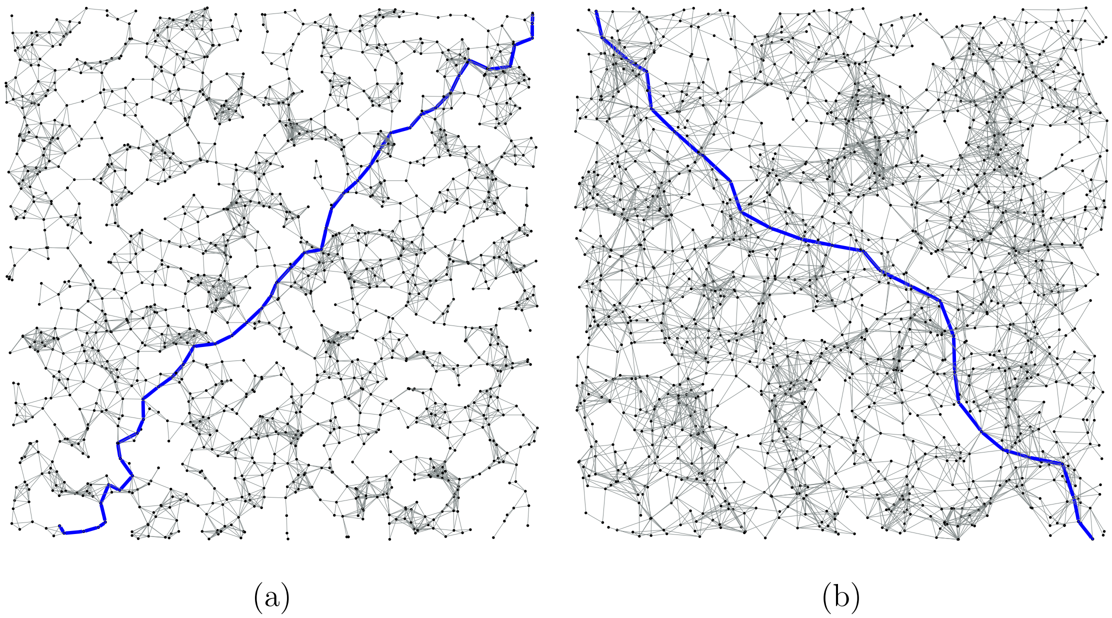
Since no information on node locations is known, the planar subgraph output by our method is not embedded. It is already sufficient for some applications, such as topology discovery (boundary recognition) [36]. If embedding is needed, the planar graph can be embedded efficiently as a plane graph by using existing planar embedding algorithms in graph drawing [23, 75] or in [36] [37]. The embedded graph can then be used for many applications, including geographic routing [51]. We demonstrate the performance of our planarization method and its application to topology discovery by extensive simulations. We show that the planar subgraphs maintain the distance between nodes with small stretches, detect holes and boundaries with a much higher precision than existing methods, and are robust to the network models.

B. Overview of the planarization scheme

In this section, we present an overview of the planarization scheme. It consists of five steps, described as follows.

1. Finding a shortest path between wwo faraway nodes

The first step is to find a shortest path between two faraway nodes. The two faraway nodes can be found with the following common approach: first, randomly choose a node a, use one flooding to build a shortest path tree rooted at a, and find the node b that is the furthest (in hops) from a in the network; then, use a similar method to find the node c that is the furthest (in hops) from b. b and c are the two faraway nodes we need, and the unique path between b and c in the shortest path tree rooted at b is the shortest path between b and c. The advantage of a shortest path between two faraway nodes is that such a path usually does not twist, regardless of the uniformity of the transmission ranges. Thus it has a good planarity property. We illustrate the property in Fig. 17. The two networks have drastically different features in the uniformity of transmission ranges, and the path is similar to a straight line in both cases. (When there are holes in the sensor field, the path may not be straight but still spreads out well.) This observation is validated by extensive simulations.



(a) (b)

Fig. 17. The shortest path between two faraway nodes, in two sensor networks with drastically different transmission ranges. The average degree is 7 in both cases. (a) A sensornet with uniform transmission ranges. (b) A sensornet where the transmission ranges in different directions vary by up to 10 times.

In the following, we will call the shortest path between b and c the *base path.*

2. Building a shortest path tree

The second step is to build a shortest path tree. First, we find the node r_1 that is the furthest away (in hops) from the base path. This can be easily done by viewing the nodes in the base path as a super node, and build a shortest path tree rooted at this super node. Then, we build a shortest path tree rooted at node r_1. See Fig. 18 (a) for an illustration.

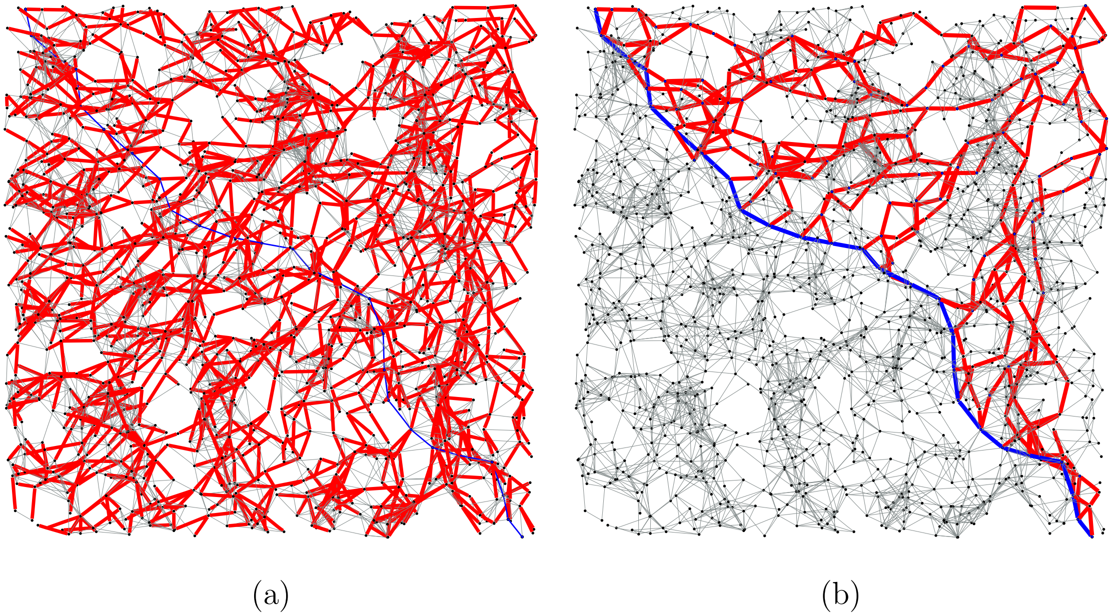


Fig. 18. Red edges indicate (a) The shortest path tree rooted at node r_1. Here r_1 is near the up right corner of the network. (b) The nodes in the layers, and the edges between layers.

3. Planarizing the network layer by layer

In this third step, we planarize part of the network layer by layer. The nodes in the base path are in Layer 1. Recursively, in the shortest path tree rooted at r_1, if a node is the parent of a node in Layer i and is not included in any of the first i layers, then it is in Layer $i+1$. A node that is not the ancestor (in the tree) of any node in the base path is not included in any layer. Let us say that the maximum layer found this way is Layer M. See Fig. 18 (b) for an illustration.

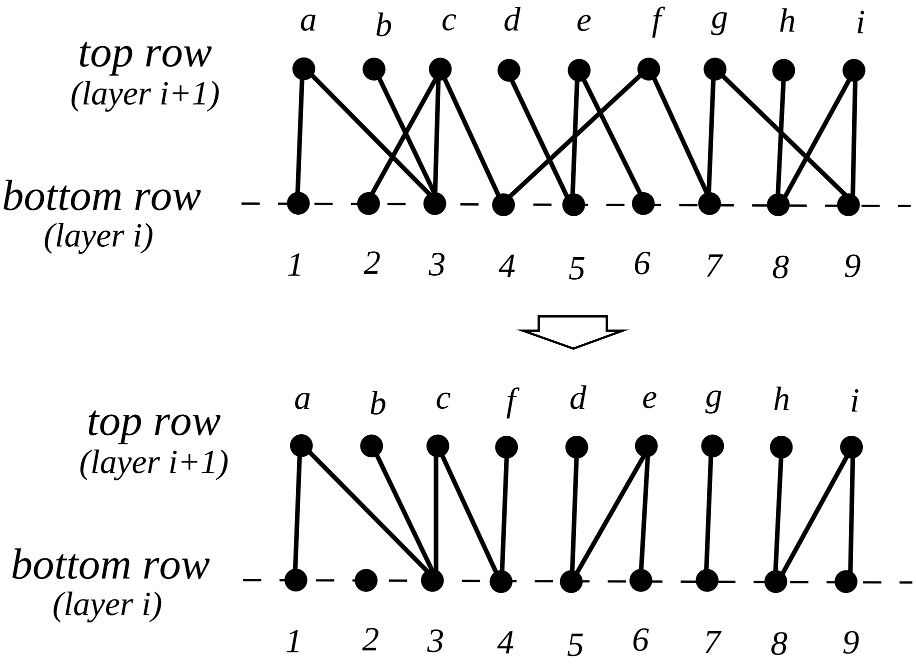


Fig. 19. Planarize two adjacent layers. The upper graph is before planarization, and the lower graph is after planarization. (1) In the upper graph, e, f are *cover nodes* of the virtual edge $\{5, 6\}$ and make its *cover number* be 2. (2) If c is the only cover node for virtual edge $\{3, 4\}$ in an optimal solution, neither of the walls $\{c, 3\}$ and $\{c, 4\}$ is removed by that solution.

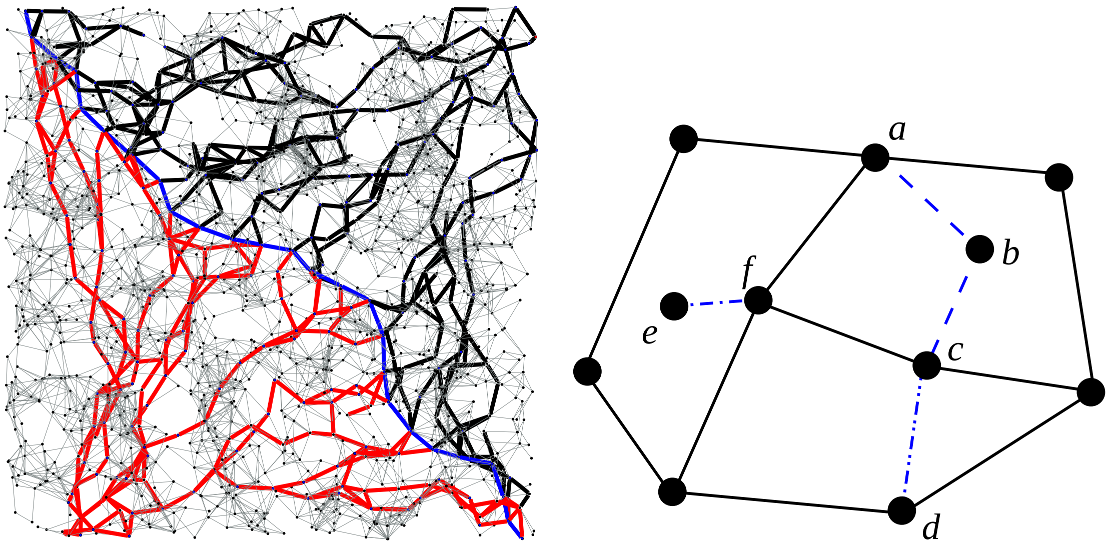
We progressively build a planar graph that includes the nodes in the M layers. First, we process Layer 1 and Layer 2. Let $G = (V_1 \cup V_2, E)$ denote the bipartite graph where the nodes in Layer 1 are in one row and the nodes in Layer 2 are in the other row. The edges in G are all those network links between Layer 1 and Layer 2. See Fig. 19 for an illustration. In the bipartite graph G, the nodes in the bottom row – which are the nodes in Layer 1 – are placed following their order in the base path. The nodes in the top row – which are the nodes in Layer 2 – are not ordered yet.

We then use a planarization algorithm to remove some edges from G, and order the nodes in the top row, so that the remaining edges do not cross each other. (All the edges are straight.) Thus we obtain a planar subgraph between Layer 1 and Layer 2. Then, we process Layer 2 and Layer 3 in the same way, then Layer 3 and Layer 4, $\cdots$, and finally Layer $M-1$ and Layer M. Note that when we are processing Layer i and $i+1$, the nodes in Layer i (the bottom row) have been ordered. So the same algorithm can be used $M-1$ times. All the edges we keep in these $M-1$ steps form a planar graph.

The general idea is that the base path and the shortest path tree rooted at r_1 both act as good references for planarization. By processing the nodes layer by layer, we *comb through* the network and obtain a planar subgraph. The planarization algorithm for planarizing the edges between two adjacent layers is the *key operation* in our method. We present the details of the algorithm in the following sections.

4. Building a second tree and planarizing the network

The planar graph built so far is a skeleton of the network covering part (often about half) of the sensor field. In this step, we build a second shortest path tree and planarize more of the network. This second tree is rooted at the node r_2 that is the furthest (in hops) from the node r_1. The planarization process is exactly the same as the process in the previous step (namely, the third step), except that here node r_2 replaces node r_1, and we do not include in the layers here those nodes that have been included in Layer 2 through Layer $M-1$ in the previous step. See Fig. 20 (a) for an illustration. The planar graphs built in this step and the previous step together form a large planar subgraph that covers most of the sensor field.



(a) (b)

Fig. 20. (a) After finishing the first tree, we build the second and planarize it similarly. Edges of the second tree are red; (b) The path $a \rightarrow b \rightarrow c$, the edge $c - d$, and the edge $e - f$ can be added into the planar graph.

5. Refining the planar graph

The planar graph built so far is a skeleton of the network, which usually covers the whole sensor field. Those nodes outside it are usually within a few hops from it. To include all nodes into the planar graph and to include more edges, three simple steps are performed. First, if a node in the planar graph – which we shall call G_{planar} – finds that it has many 1- or 2-hop neighbors outside G_{planar}, it uses a 4-hop localized flooding to add one or more paths to G_{planar}, as long as the new path connects nodes in the same face and therefore preserves the planarity of G_{planar}. Note that the previous planarization steps already tell us what the faces in the planar graph are, so this operation is easily doable. Second, if there are still nodes outside G_{planar}, they connect themselves to G_{planar} via small trees, which is a simple operation. The trees preserve the planarity of the graph. At this moment, the planar graph contains all the nodes. Then, to add more edges to G_{planar}, the nodes add their incident edges to it if the new edge connects two nodes in the same face (and therefore preserves the planarity). Now we get the final planar subgraph of the network. See Fig. 20 (b) for an illustration on how the refining is done.

C. An approximation planarization algorithm

In the previous section, the planarization scheme is presented. The *key* operation is how to planarize two adjacent layers. That is also the only part of the scheme whose detail has not been specified. Naturally, our objective is to remove as few edges as possible during planarization, because a dense planar graph is desirable. In this section, we present an approximation algorithm to this NP-hard problem.

We formally define the problem as follows:

> BIPARTITE PLANARIZATION problem (BiPP): In a bipartite graph $G = (V_1 \cup V_2, E)$, the nodes in the bottom row V_1 are already linearly ordered, and the nodes in the top row V_2 are not (See Fig. 19 for an example). How to remove some edges from G, and linearly order the nodes in the top row V_2, so that no two edges cross each other? The objective is to minimize the number of removed edges.

In the graph drawing research field, this problem is also called the ONE SIDED TWO-LAYER PLANARIZATION problem. It is known to be NP hard, even when the nodes in V_2 all have degrees at most two and the nodes in V_1 all have degrees at most one [28]. The best existing solution that runs in polynomial time is a 3-approximation algorithm [27]. In this section, we present a new algorithm whose approximation ratio can be arbitrarily close to 2, thus improving the best known result.

Let us define a few terms. For any two nodes v_1 and v_2 in the bottom row V_1, if they are adjacent (in the sense of the given ordering of the nodes in V_1), then we imagine there is a *virtual edge* between v_1 and v_2. We say that a node $u \in V_2$ *covers* a virtual edge e if u has neighbors in the graph G that are on both the left side and the right side of e in the bottom row. Such a node u is called a *cover node* of e. The number of nodes in V_2 covering e is called the *cover number* of e. All the edges incident to a cover node of e are called the *walls* of e. Note that every cover node of e is incident to at least two walls of e.

Our approximation algorithm is based on the following observations.

Observation IV.1 *In any solution to the* BIPARTITE PLANARIZATION *problem, the cover number of any virtual edge e is at most* 1*. Therefore, if the cover number of e is y in the graph G, any solution must remove at least $y-1$ walls of e.*

Therefore, when the cover number of a virtual edge is large, if we remove two walls around e for each of the cover nodes, we would have removed no more than twice the number of edges removed by any solution plus one. This technique is used in our algorithm.

On the other hand, if the cover numbers of all the virtual edges are relatively small, we can solve the problem efficiently with the divide and conquer technique. The following observation is the basis for the divide and conquer technique.

Observation IV.2 *If the cover number of a virtual edge e is y in the graph G, then in any solution, at most one of e's cover nodes can keep all its corresponding walls around e. For all the other cover nodes of e, each of them must remove all the walls on at least one side of e. Therefore, if we enumerate all the possible ways to solve the conflicts at e in any solution, there are at most $y2^{y-1} + 2^y$ cases to consider.*

(Specifically, for each cover node, we first consider if its walls should be removed; if yes, we consider which side of the walls to remove. So there are $y2^{y-1} + 2^y$ cases.)

If a virtual edge e has no cover node in a solution, the nodes in the bottom row can be separated into two parts: the left of e and the right of e. Then every node in the top row is adjacent to nodes in only one of the two parts. So in that case, the problem can be solved for the two subgraphs separately. If in an optimal solution, e has one cover node, then the following observation tells us that we can still divide the problem into two parts.

Observation IV.3 *Suppose that in an optimal solution, a virtual edge e has exactly one cover node u. If the wall w_l (respectively, w_r) incident to u is the closest edge to e from the left(respectively, right) side, then w_1, w_2 must have not been removed by that optimal solution (as shown in Fig. 19). In that case (assuming that we have guessed this case to be true), when we search for the optimal solution, we can mark the two walls closest to e from each side (namely, w_1 and w_2) to be irremovable (namely, we do not remove them in the algorithm).*

The approximation algorithm is shown in Fig. 21.

```
**Input:** $G = (V_1 \cup V_2, E), \epsilon$
**Output:** $G_{planar}$: A solution to the BiPP.
**repeat**
    $e \leftarrow$ a virtual edge in $V_1$ with cover number $> 1/\epsilon$
    for each cover node of $e$, remove a wall incident to that node from each side of $e$
**until** no virtual edge in $V_1$ has cover number larger than $1/\epsilon$
call the procedure **Exact-BiPP**($G$) and return the result.
```

Fig. 21. Algorithm **APX-BiPP**: approximate the BiPP problem.

```
Input: $G = (V_1 \cup V_2, E)$
Output: $G_{planar}$: A solution to the BiPP.
 1: Let $n$ be the number of nodes in $V_1$. If $n < 5$, solve the problem in the brute force
    way, and return the solution.
 2: $e \leftarrow$ the $\lfloor n/2 \rfloor$-th virtual edge in $V_1$
 3: $y \leftarrow$ the cover number of $e$
 4: for all $y2^{y-1} + 2^y$ cases at $e$ (see Observation IV.2) do
 5:    skip this iteration if in this case, an irremovable edge is to be removed
 6:    the problem is now split into two disjoint subproblems of roughly the same size,
       recursively call Exact-BiPP on them.
 7:    record the solution of this case if it removes the least number of edges among
       all cases considered so far,
 8: end for
 9: remove edges according to the best solution got above
10: return $G$
```

Fig. 22. Algorithm **Exact-BiPP**: solve the BiPP problem exactly.

Let us first look at the subroutine **Exact-BiPP** in Fig. 22, in any case the edge e will be a separator for the problem in the sense that no two nodes on both side of e share a neighbor in V_2 except v, the one we decided to keep "over" e. In that case $\{v, u_1\}$ and $\{v, u_2\}$ are both kept in the final solution and marked irremovable, the two subproblems formed in line 6 of **Exact-BiPP** will not affect each other. In the case e has no cover node in the solution, it is clear that we have split the problem into two independent subproblems. Since we enumerate all possible cases at e in the recursive stage, we will get the optimal solution.

Assuming that the maximum cover number for virtual edges in V_1 is c and $T(n)$ is the running time of the algorithm where $n = |V_1|$. We have the recurrence relationship $T(n) < 2 \times c2^c T(n/2)$. This recursion has a solution $T(n) < (2c2^c)^{\log n} = n^{c+\log c+O(1)}$. Hence we have the following lemma immediately.

Lemma IV.4 *The algorithm* **Exact-BiPP** *finds the optimal solution for the* BIPARTITE PLANARIZATION *problem and runs in time* $n^{c+\log c+O(1)}$ *where* c *is the maximum* *cover number for all virtual edges in V_1 and n is the number of nodes in V_1.*

In the following theorem, we prove the algorithm **APX-BiPP**'s approximation ratio, $2+3\epsilon$, and its polynomial time complexity. The parameter ϵ can be a arbitrarily small positive number, so we assume that $\epsilon \leq 1/3$.

Theorem IV.5 *The planarization algorithm* **APX-BiPP** *is a* $(2+3\epsilon)$*-approximation for the* BIPARTITE PLANARIZATION *problem. It runs in time* $O(|V_1|^{1/\epsilon+\log\lceil 1/\epsilon\rceil+O(1)} + |E||V_1|)$.

PROOF. First it is simple to see that the algorithm produces a solution for the problem.

Let us look at the approximation ratio. In the recursive stage (subroutine **Exact-BiPP**) of the algorithm, by Lemma IV.4 we get the exact solution.

While in the preprocessing stage (lines 1 through 4 in **APX-BiPP**), there are $a(e) > 1/\epsilon$ nodes in V_2 covering e, and at most one of them can be the cover node of e in any solution. We removed $2a(e)$ edges over e. That is at most $a(e)+1$ more than the number of edges removed by an optimal solution S.

Suppose that in the preprocessing stage (lines 1 through 4 in **APX-BiPP**), we have run the loop m times. Let M be the total number of edges removed, we have $M > 2m/\epsilon$. Let the residual graph be B'. Among these M edges at most $M/2+m$ are unnecessary, in other words, S (the optimal solution) would have to remove at least $M/2-m$ edges. Therefore the number of edges removed by S, denoted by $R(B)$, is then at least $M/2-m+R(B'')$ where B'' is the residual graph after we removed these $M/2-m$ edges according to S. Since in the approximation stage we have removed all walls at those virtual edges, B' is a subgraph of B''. Hence we have $R(B') \leq R(B'')$

and $R(B) \geq M/2 - m + R(B')$. While in our approximated solution the number of edges removed, denoted by $A(B)$, is exactly $M + R(B')$. Hence we have

$$\begin{aligned}\frac{A(B)}{R(B)} &\leq \frac{M + R(B')}{M/2 - m + R(B')}2 + \frac{4m - 2R(B')}{M - 2m + 2R(B')} \\ &\leq 2 + \frac{4m}{2m/\epsilon - 2m} \qquad (4.1)\end{aligned}$$

Now we show that the algorithm runs in polynomial time. The preprocessing stage takes time $O(|E||V_1|)$. In the stage when we call **Exact-BiPP**, since the cover numbers of all virtual edges in V_1 are bounded by $1/\epsilon$, by Lemma IV.4 the running time is upper bounded by $n^{1/\epsilon + \log[1/\epsilon] + O(1)}$. Therefore the total running time is then $O(|V_1|^{\frac{1}{\epsilon} + \log\frac{1}{\epsilon} + O(1)} + |E||V_1|)$. □

D. Fixed parameter tractable algorithm for optimal planarization

In our extensive simulations, we observed that while planarizing the subgraph induced by two adjacent layers, the number of edges that need to be removed is usually much smaller than the number of nodes in those two layers, for a network of moderate density (average degree roughly between 6 and 12). If the network is very dense, we can use a simple preprocessing operation to reduce the edge density to the moderate level. (Details of the operation will be discussed later.) Therefore a question remains: can we use that observation to practically improve the planarization algorithm?

In recent years, a new approach called parameterized computation has been proposed to solve NP hard problems by exploiting small parameters [26]. Let k be a parameter in a parameterized problem. We say that the problem is *fixed parameter tractable* (FPT) if it can be solved optimally in time $O(f(k)n^{O(1)})$, where n is the input size and $f(k)$ is a function of k. When k is bounded, not only is the time complexity polynomial, but it usually also grows much slower than $O(n^k)$ when n increases. Quite a few NP-hard problems have been proved to be in FPT with effective algorithms. For example, the VERTEX COVER problem with parameter k as the cover size can be solved in time $O(1.286^k + n^3)$ [19].

In [27], the authors showed that the BIPARTITE PLANARIZATION problem (also called the ONE SIDED TWO-LAYER PLANARIZATION problem) is fixed parameter tractable when the parameter k is an upper bound for the number of edges to be removed. They developed an FPT algorithm for the problem running in time $3^k n^{O(1)}$. In this section, we present an improved FPT algorithm running in time $(2+\gamma)^k n^{O(1)}$, where γ can be an arbitrarily small positive number.

Formally, the parameterized version of the BIPARTITE PLANARIZATION problem can be stated as follows:

> Given an instance of the BIPARTITE PLANARIZATION problem and a parameter k, either find a solution to the instance that removes at most k edges, or report that no such solution exists.

Let us first define a few terms. Let $u \in V_1$ be the leftmost neighbor of a node $z \in V_2$, and $v \in V_1$ be the rightmost. We call the pair of edges $\{u, z\}, \{z, v\}$ an *arc*, and denote it by $z(u, v)$. We say that the arc $z(u, v)$ *conflicts* with an edge $\{w_1, w_2\}$ if it is necessary to remove $\{w_1, w_2\}$ in order to keep both edges $\{z, u\}, \{z, v\}$ in a solution. To help simplify the following discussion, for each pair of arcs $z(u, v), z'(u, v)$ – i.e., they share the same leftmost and rightmost neighbors u, v in V_1 – we also consider $\{z, u\}$ a conflict edge of $z'(u, v)$. Similarly $\{z', u\}$ is also a conflict edge of $z(u, v)$.

The *conflict number* of an arc $z(u, v)$ is the number of edges that conflict with the arc. (See Figure 23 (a) for an illustration of the terms.)

The conflict number of a given arc $z(u, v)$ is simply the summation of the following two numbers: (1) the number of edges incident to nodes (exclusively) between u, v

in V_1 but not z; (2) the number of arcs that of the forms $z'(u,v)$ (where $z' \neq z$), i.e. they share the same leftmost (rightmost) node in V_1. The set of edges that are conflicting with a given arc can also be decided easily by definition.

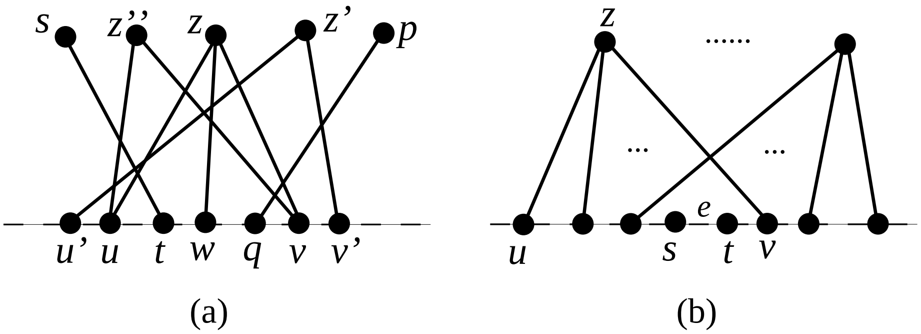


Fig. 23. (a) $z(u,v)$ is an arc; the edges $\{p,q\}, \{s,t\}, \{z'',u\}$ conflict with $z(u,v)$; the conflict number of $z(u,v)$ is then 3; $z'(u',v')$ strictly covers $z(u,v)$ and has a larger conflict number 7. (b) cover number and conflict number at virtual edge $e(s,t)$: there are at least $f-1$ edges incident to nodes between u and s but not z.

The following lemma shows the relationship between cover numbers and conflict numbers.

Lemma IV.6 *If there is a virtual edge e in V_1 with cover number f, there must exist an arc that has conflict number at least $f-1$. In other words, if no arc has conflict number larger than $f-1$, no virtual edge in V_1 will have cover number larger than f.*

Proof. Suppose the left-most walls of e are incident to u in V_1. Let $z(u,v)$ be the arc such that v in V_1 is the furthest away to the right of u. Hence $\{z,u\}$ is a leftmost wall of e and v is the right-most neighbor of the cover nodes adjacent to u. Each of the $f-1$ cover nodes (other than z) of e either is adjacent to u and has no neighbor exclusively to the right of v, or has its left-most wall exclusively to the right of u. By definition, each of these $f-1$ cover nodes (other than z) of e will contribute at least 1 to $z(u, v)$'s conflict number. Either their rightmost walls (if they are adjacent to u but not v) or their leftmost walls will be conflict edges of $z(u, v)$. The conflict number of $z(u, v)$ is, therefore, at least $f - 1$. □

We present the FPT algorithm as Fig. 24. The constant β is any predefined positive number that is no smaller than 1. The algorithm returns a solution if there exists a solution that removes at most k edges, and returns **false** otherwise. (With a little abuse of notations, when the algorithm returns **true**, it returns the solution as well.) Note that when a solution is found, the conflict number of every arc becomes 0.

```
Input: $G(V_1 \cup V_2, E), k, \beta \geq 1$
Output: $G_{planar}$: A solution removing no more than $k$ edges.
$z(u, v) \leftarrow$ the arc with the maximum conflict number $c(z)$.
if $c(z) = 0$ then return true
if $k < c(z)$ then return false
if $c(z) \leq \beta$ then
    planarize $G$ using the procedure Exact-BiPP. (That is, run Exact-BiPP($G$).)
    if Exact-BiPP($G$) removes at most $k$ edges, return true; else return false
end if
$E(z) \leftarrow$ the set of edges conflicting with $z(u, v)$
if FPT-BiPP($G(V_1 \cup V_2, E \setminus E(z)), k - c(z)$) return true;
if FPT-BiPP($G(V_1 \cup V_2, E \setminus \{\{z, u\}\}), k - 1$) return true;
if FPT-BiPP($G(V_1 \cup V_2, E \setminus \{\{z, v\}\}), k - 1$) return true;
return false
```

Fig. 24. Algorithm **FPT-BiPP**: solve the BiPP problem in FPT time.

The following theorem proves the correctness and complexity of the algorithm.

Theorem IV.7 *The algorithm* **FPT-BiPP** *will either find a solution for the* BIPARTITE PLANARIZATION *problem by removing at most k edges, or correctly report that there is no such solution. The running time of the algorithm is upper bounded by $(2 + 1/\beta)^k n^{\beta + \log \beta + O(1)}$.*

Proof. During the calling of the routine **Exact-BiPP** in line 5 (that is, when $c(z) \leq \beta$), since the conflict numbers of the nodes in V_1 are upper bounded by β, by Lemma IV.6, the cover number of the virtual edges in V_1 will be bounded by $\beta + 1$. Again by Lemma IV.4, the running time of **Exact-BiPP** in this algorithm will be bounded by $T_1 = n^{\beta+\log\beta+O(1)}$.

As for the recursive calls in lines 9, 10 and 11, we branch into three cases where k decreases by $1, 1$ and $c(z)$, respectively. Since we enter this stage only if $c(z) > \beta$, if we use $T(k)$ to denote the running time of the algorithm, the following recurrence relationship holds for $T(k)$:

$$T(k) \leq 2T(k-1) + T(k-\beta).$$

We use induction to show that $T_1(2+1/\beta)^k$ is an upper bound for $T(k)$. In the case when $k < c(z)$, we return false right away. If $c(z) \leq k \leq \beta$, we already have $T(k) \leq T_1$. Hence when $k \leq \beta$, we always have $T(k) \leq T_1(2+1/\beta)^k$.

Suppose for $\beta \leq k < t$ we have $T(k) \leq T_1(2+1/\beta)^k$. Using basic calculus, it is easy to verify that when $\beta \geq 1$, $2(2+1/\beta)^{\beta-1} + 1 \leq (2+1/\beta)^\beta$. Thus we have

$$\begin{aligned}
T(t) &\leq 2T(t-1) + T(t-\beta) \\
&\leq 2T_1(2+1/\beta)^{t-1} + T_1(2+1/\beta)^{t-\beta} \\
&\leq T_1(2+1/\beta)^{t-\beta}\left[2(2+1/\beta)^{\beta-1} + 1\right] \\
&\leq T_1(2+1/\beta)^t
\end{aligned}$$

Therefore we have proved that the running time of the algorithm is bounded by $(2+1/\beta)^k n^{\beta+\log\beta+O(1)}$. □

The exponential part of the running time of the algorithm **FPT-BiPP** can be arbitrarily close to 2^k by choosing a large enough value for β.

The difference between the approximation approach and a parameterized approach is that the latter gives the optimal solution. Although the problem is NP-hard, the FPT algorithm solves the problem in polynomial time when the parameter is small, i.e., the exponential part of the running time is independent of the number of the nodes in the input graph. As we will show later in simulations, the number of edges to be removed is actually very small (less than 15 in most cases for networks of up to 2500 nodes) for networks with practical models.

E. Implementation and simulations

We conducted extensive simulations to test the performance of the planarization method. The performance has been very stable for different network models, sensor deployment methods, network sizes and sensor densities. In the following, we present some typical simulation results. A planarized network has numerous applications, including topology discovery, localization, geographic routing, etc. (For geographic routing, embedding is needed and such embedding algorithms are available [23, 36, 37, 75].) We illustrate the performance of our result by showing its application to topology discovery. We show that our planarized network can locate holes and outer boundaries of the sensor fields accurately, significantly improving the known results on topology discovery.

Let us explain the distributed implementation of our planarization method. The method utilizes a few shortest path trees, and a practically limited amount of localized flooding to refine the final result. (The total cost of the localized flooding is about the same as flooding the network once or twice.) Both building shortest path trees and localized flooding are very mature techniques in networking. Both algorithms presented in the chapter planarize the network layer by layer, which naturally corresponds to a distributed implementation. While planarizing two adjacent layers, we take the simple approach of letting one node in the two layers act as a proxy and run the algorithm in a centralized way. (The nodes in the two layers can easily send their information to the proxy node via the corresponding shortest path tree.) We comment that our algorithms take the divide and conquer approach, so it is simple to decentralized its implementation; but we decide not to use it for networks of less than 2500 nodes because of the already very low communication and computation overhead.

The faces in the planarized network are always very clear throughout the planarization process. That is because nodes are planarized layer by layer, and the planarization algorithm sorts the edges incident to each node by ordering the nodes in the two adjacent layers. That makes the refinement step and the topology discovery application of using faces to recognize holes/boundaries very simple.

Our method works well for networks of moderate or sparse edge densities. For dense networks (average degree 15 or more), the following simple preprocessing approach can effectively reduce the edge density and edge crossing: for every maximal clique in the network, we remove some edges from it so that the remaining edges form a star. Note that the average degree of a planar graph is always less than 6. So such an edge-reduction preprocessing step goes along well with planarization.

We have presented two planarization algorithms: an approximation algorithm and an FPT algorithm. They have very similar performance in practice. Due to the space limitation, we present the simulation results for the FPT algorithm only. (The results for the approximation algorithm are very close.) In most simulations, the optimal planarization solution removes less than 15 edges for all the layers. So by setting $k = 15$ or a little above, the FPT algorithm finds the optimal solution and maintain a low time complexity at the same time.

1. Network planarization

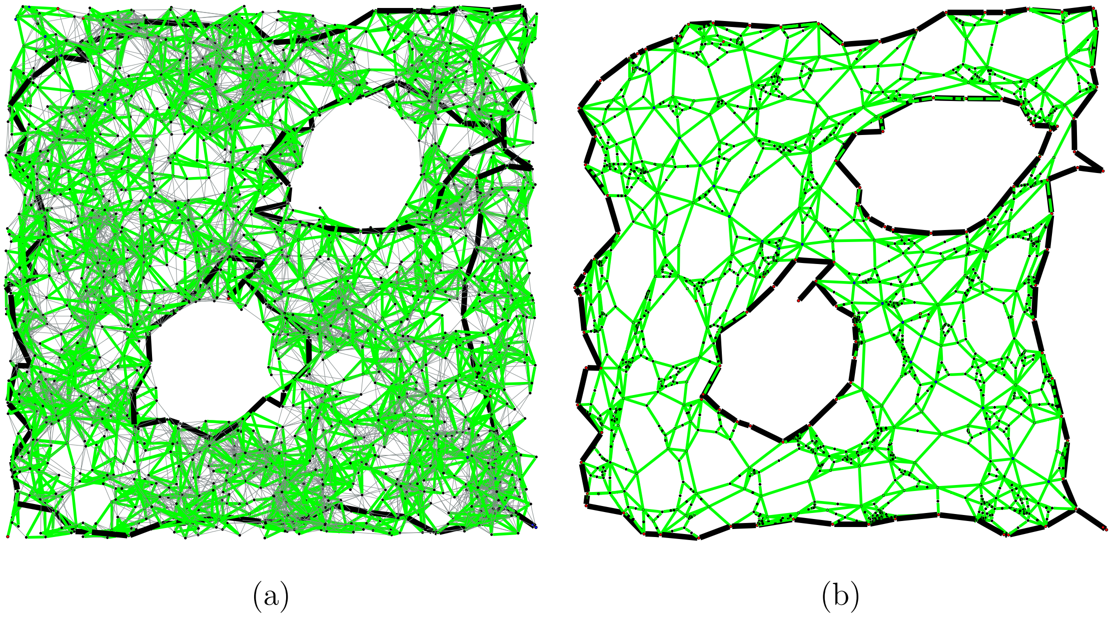


Fig. 25. Wireless sensor network and its planarization. (a) The dark (green and black) edges are the planarized network. The light (grey) edges are the remaining edges in the original network. The original network is a quasi-UDG with $N = 2000$ nodes and average degree 12, where transmission ranges vary by as much as $R/r = 5$. (b) A plane embedding of the planarized network.

We randomly deploy N sensors uniformly randomly in a 15000×15000 square area. The network follows the quasi unit disk graph (quasi-UDG) model: two nodes do not have a link if their distance is greater than R, have a link if their distance is less than r, and have a link with probability 1/3 if their distance is between r and R. (Here $r \leq R$.) Let $\alpha = R/r$. When N and α are given, we adjust r and R to obtain the desired average node degree. To make the sensor field non-trivial, we also randomly place holes in the network. Our reported results are for two holes of radius about $2R$. For each parameter configuration, 1000 networks are generated and measured.

A typical planarization result is shown in Fig. 25. To quantitatively analyze the performance of planarization, we measure the stretch (*str.*) of hop distance and its standard deviation σ. Let u, v be two nodes, whose hop distance is $h(u, v)$ before planarization and is $h'(u, v)$ after planarization. The *multiplicative stretch* is defined as $h'(u, v)/h(u, v)$, and the *additive stretch* is $h'(u, v) - h(u, v)$. To better characterize the stretch of different node pairs, we measure the multiplicative stretch for nodes whose hop distance is greater than 1/4 of the network diameter, and measure the additive stretch for the others. The results are shown in Table VII where d is the average degree of the input networks, d_p is the average degree of the planar spanning subgraph found and D is the average diameter of the input networks.

Table VII. Planarization results for $N = 1500$.

	d=7			d=9			d=11		
Additive stretch for nodes within $Diameter/4$ hops									
α	D	*str.*	σ	D	*str.*	σ	D	*str.*	σ
1	48.3	4.03	9.80	40.1	3.56	8.04	36.1	3.36	7.27
2	36.2	3.45	8.11	30.6	3.28	7.19	27.4	3.24	6.66
5	28.9	3.23	6.76	24.0	3.05	6.25	21.5	2.98	6.00
10	27.0	3.11	6.82	23.7	3.01	6.13	21.4	2.96	5.83
Multiplicative stretch for nodes more than $Diameter/4$ hops apart									
α	d_p	*str.*	σ	d_p	*str.*	σ	d_p	*str.*	σ
1	3.5	1.27	0.57	3.8	1.28	0.56	3.8	1.30	0.57
2	3.7	1.31	0.54	3.8	1.35	0.56	3.9	1.38	0.58
5	3.7	1.36	0.55	3.8	1.40	0.55	3.8	1.43	0.56
10	3.7	1.36	0.54	3.8	1.40	0.55	3.8	1.43	0.57

We see that for networks of very different connectivity models and densities, the stretch is constantly small. This shows the good performance of the planarized networks.

2. Topology discovery

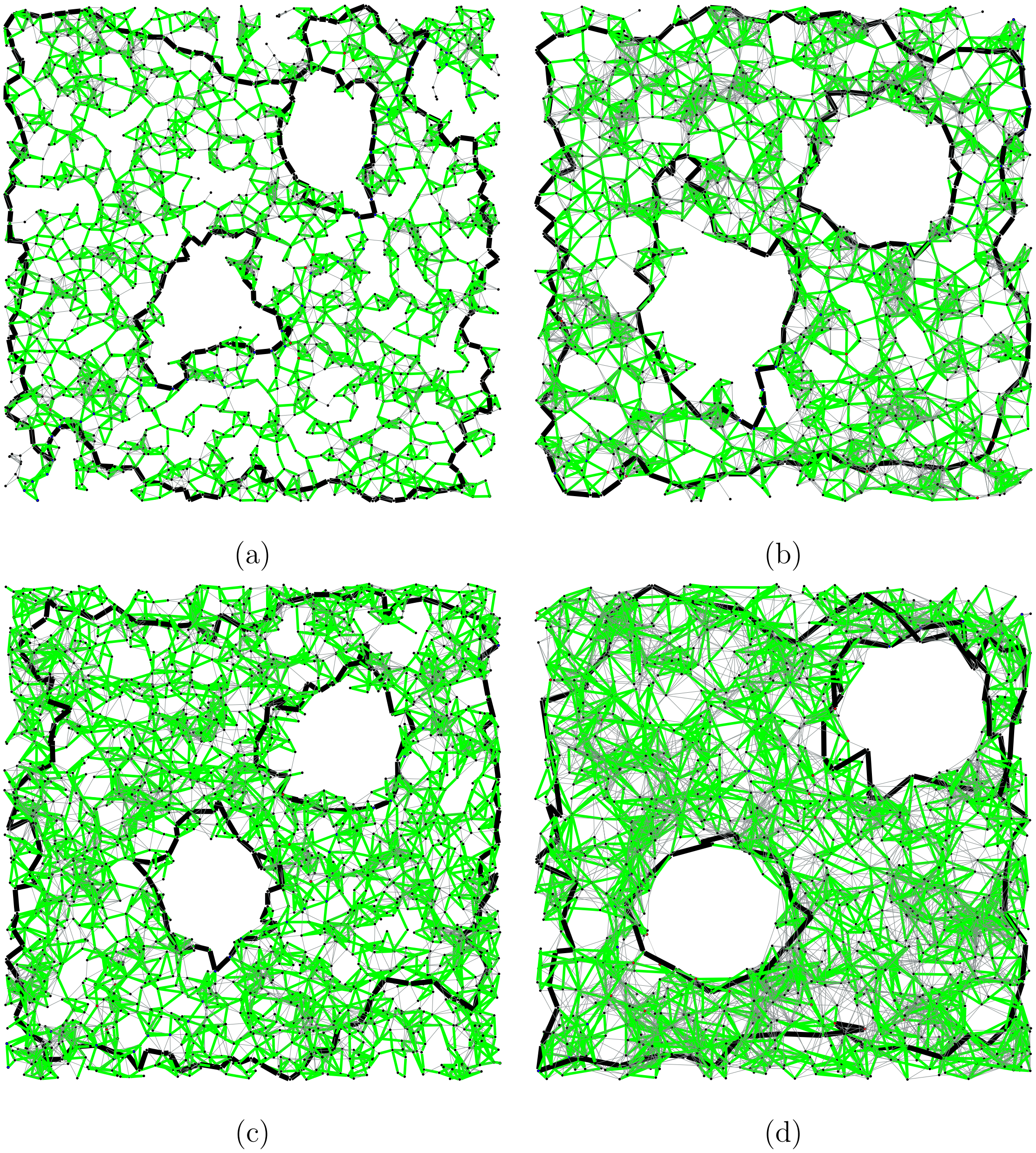
One important application of network planarization is topology discovery [97] [36] [37]. In topology discovery, we use the large faces in a planarized network to find holes and the outer boundary of the sensor field. The importance of such prominent features in sensor fields has been presented earlier in the Introduction. Mainly, they help us learn about the sensor field, detect obstacles and abnormal events, interpret the sensed data, and balance routing load.

In our simulations, we place two circular holes of radius $2R$ in the sensor field, where the sensors cannot be deployed. We comment that the actual holes are very irregular and look far from circles, because of the random deployment of nodes and the sparsity of edges. By checking large faces (faces containing 20 edges or more), we are able to identify the faces close to the boundaries of holes and the outer boundary in more than 95% cases for all configurations.

Four typical results are shown in Fig. 26. To quantitatively analyze the performance of topology discovery, we compare the Euclidean length of the actual hole or outer boundary, l, to the Euclidean length of the face, l', surrounding it. Both l and l' are computed off-line using real coordinates of the nodes. (Note that the actual boundary consists of a set of fractional segments of edges since edges can cross.) The results are shown in Table VIII where σ is the standard deviation of l'/l and m is the average of the total number of edges removed to planarize both shortest path trees for each network.

Table VIII. Hole detection results for $N = 1500$.

	d=7			d=9			d=11		
α	l'/l	σ	m	l'/l	σ	m	$l'/l.$	σ	m
1	1.02	0.06	4.8	1.15	0.31	9.5	1.20	0.05	14.6
2	1.13	0.05	6.4	1.22	0.04	10.1	1.29	0.19	13.2
5	1.37	0.26	8.9	1.43	0.46	11.2	1.50	0.16	11.2
10	1.41	0.82	8.5	1.48	0.49	9.9	1.42	0.46	12.2


(a)

(b)

(c)

(d)

Fig. 26. Typical hole detection results: green edges are those in the planarized network; black cycles are detected boundaries. (a) UDG network with $N = 2000$ nodes and average degree 8.14. (b) UDG network with $N = 1500$ nodes and average degree 12.72. (c) quasi-UDG network with $N = 2500$ nodes and average degree 8.05, where the transmission ranges vary by as much as $R/r = 10$ times. (d) quasi-UDG network with $N = 1500$ nodes and average degree 12.05, where the transmission ranges vary by as much as $R/r = 5$ times.

We see that the length ratio is usually *very* close to 1. This shows the excellent performance of the results. Our results compare favorably with the known results on topology discovery. In [36, 37, 97], the authors presented very nice algorithms for detecting holes and outer boundaries. Their algorithms work for large holes (radius are 5 times the communication range of the sensors or more) or very dense networks (e.g., average degree larger than 15). In comparison, our method detects holes whose radius are twice the communication range or more, and works for both low and high densities. So the improvement in performance is actually remarkable.

CHAPTER V

SORTING BASED DATA CENTRIC STORAGE

Data-centric storage is a very important concept for sensor networks, where data of the same type are aggregated and stored in the same set of nodes. It is essential for many sensornet applications because it supports efficient in-network query and processing. Multiple approaches have been proposed so far. Their main technique is the hashing technique, where a hashing function is used to map data with the same key value to the same geometric location, and sensors closest to the location are made to store the data. Such solutions are elegant and efficient for implementation. However, two difficulties still remain: load balancing and the support for range queries. When the data of some key values are more abundant than data of other key values, or when sensors are not uniformly placed in the geometric space, some sensors can store substantially more data than other sensors. Since hashing functions map data with similar key values to independent locations, to query a range of data, multiple query messages need to be sent, even if the data of some key value in the range do not exist. In addition to the above two difficulties, obtaining the locations of sensors is also a non-trivial task.

In this chapter, we propose a new data-centric storage method based on sorting. The idea is to sort the data in the network based on their key values, so that queries – including range queries – can be easily answered. The sorting method balances the storage load well, and we present a sorting algorithm that is both decentralized and very efficient. It requires no location information, and is robust to different network models. We present both rigorous theoretical analysis and extensive simulations for analyzing its performance. They show that the sorting-based method has excellent performance for both communication and storage.

A. Introduction

Wireless sensor networks are widely deployed nowadays to collect data from different types of environments. In many systems, sensors also collaborate to aggregate data and answer queries. For sensor networks, where and how to store data is an important issue.

There have been two basic approaches for data storage: the *sink-based storage*, and the *in-network storage*. In the sink-based storage approach, all the data collected by sensors are transmitted to a powerful server, which is connected to the sensor network through a gateway. This approach is simple and easy to configure. But it also have some shortcomings. First, such a connection to a server may not be available, especially for sensors deployed in remote areas, emergency areas, combat fields, or robotic/mobile systems. A server can also be expensive. Second, transmitting all the data to the sink makes the sensors near the sink lose their energy quickly.

The in-network storage approach, where data are stored in the sensors themselves, removes the dependency on servers, and balances power consumption better. It also supports in-network processing well, which is important for real-time applications. Like a database, the data in a sensor network are labelled by their key value. To answer queries efficiently, the data with the same key value should be aggregated and stored in the same place, and that place should be easily accessible by any sensor querying the data. That is the basic principle of *data-centric storage*, which has been a well accepted concept in sensor networks.

There have been a number of data-centric storage schemes proposed in recent years. Their main technique is the *geographic hash table*, where the location to stored the data with the same key value is determined by a hash function. The hash function takes the key as input and outputs a geographic location, and the data are stored in the sensor (or sensors) closest to that location. Any sensor querying the data can use the same hash function to find their location. To better support the routing process – both for transmitting the data from the sensors collecting them to the sensor storing them, and for a sensor querying and retrieving the data – the geographic routing algorithm GPSR is used.

Although the hash-based method is an elegant solution for data-centric storage, two difficulties still remain: *storage load balance* and *the support for range queries.* When the data with some key values are more abundant than the data with other key values, the sensors corresponding to the first group of data need to store more data than the second group. Also, since the hash function builds a simple mapping between keys and locations, when there is a hole in the sensor network, the sensors around the hole often need to store much more data than others. Both are common scenarios in sensor networks, which are usually deployed in complex environments. And in fact, we will show that even the random deployment of sensors alone can lead to substantial load balancing problems. On the other side, since the hash function maps similarly key values independently to different locations, range query becomes expensive. Range query is a very common and useful query in sensor network, where a sensor queries the data whose key values are in a range. With the hash-based method, for every key value in the range, a separate query message need to be sent, even if the data of some key values do not exist. Therefore, the communication cost can be high. In addition to the above two difficulties, obtaining the locations of nodes is also a non-trivial task for sensor networks.

In this chapter, we propose a new data-centric storage method: *sorting-based storage*. The idea is to sort the data in the network based on their key values. A primary path in the network is used to provide a linear order of the data, and all the edges are used to make both the sorting process and the query process efficient. The sorting process naturally balances the storage load for sensor well, regardless of the distribution of data or the shape of the network. And since adjacent data are stored sequentially in nearby sensors, range queries can be easily answered. We present a sorting algorithm that is both decentralized and efficient. It requires no location information, and is robust to different network models. We present both rigorous theoretical analysis and extensive simulations for analyzing its performance. They show that the sorting-based method has excellent performance for both communication and storage.

The rest of the chapter is organized as follows. In Section B, we present an overview of related work. In Section C, we introduce the basic concepts of the sorting-based storage scheme. In Section D, we present the sorting algorithm, and provide a general performance evaluation. In Section E, we analyze the sorting algorithm for two-dimensional networks. We describe the implementation of the storage scheme in Section D, and use simulation to evaluate its performance in Section G. In Section H, we present the concluding remarks.

B. Related work

The concept of data-centric storage was proposed in [81], where a storage scheme named GHT (Geographic Hashing Table) was presented. GHT maps the key value of data to a geographic location using a hash function, and stores the data in the nodes closest to the geographic location. GHT uses the geographic routing algorithm GPSR for data forwarding and query. Since the locations of sensors are often hard to obtain and GPSR is not always a feasible routing algorithm, a new routing algorithm named GEM was proposed in [76]. GEM embeds a tree with additional short-cut edges in the network, which the authors called the ringed tree. The ringed tree defines a virtual polar coordinates for the nodes. GEM uses the ringed tree for routing with guaranteed delivery. It can also be used for data-centric storage, where data are mapped to the virtual coordinates using a hash function.

A number of schemes have been proposed to store the indexes of data, instead of the data themselves, in an organized way so that they can be easily looked up. In [24], a two-tier storage architecture called TSAR was proposed. In TSAR, the high level proxy nodes keep track of the indexes of data using a virtual structure called the interval skip graph, which is a distributed search tree for indexing intervals. TSAR supports spatio-temporal and value queries. In [64], a storage architecture called DIM was proposed to store indexes in a distributed way. DIM maps key values to physical zones, which are organized similar to k-d trees and are independent of the network topology itself. The key values are assigned to zones in a way that makes similar keys more likely to be stored near each other. DIM supports multi-dimensional queries, and it uses the GPSR geographic routing protocol.

The idea of replicating data along a curve or in a set of nodes has been explored in multiple publications. In [83], a scheme called double ruling was studied, where the data from each sensor is replicated along a curve called the replication curve. To query the data from a sensor, a query message is transmitted along a curve called the retrieval curve. If the two curves are guaranteed to meet, the retrieval of data is guaranteed. In [29], data are replicated on a curve called finger trees. In the above schemes, for the replication curve and the retrieval curve to meet, a planar network is desirable. In addition to the above schemes, some location service schemes can also be applied to replicate data and support data query. In [63], a scheme called GLS (Geographic Location Service) was proposed, where the location information from each sensor is replicated in a set of sensors. To query the information, a query message is routed using ring-structured routing tables until it meets a sensor storing the replicated information. In the above schemes data are replicated to many nodes, so that finding the data becomes easier.

Multi-resolution storage of data was studied in [39], where a scheme called DIMENSIONS was used to store data with multiple resolutions. It supports multi-resolution data access and spatio-temporal pattern mining. The idea of using multiple powerful nodes to provide data service was studied in [89]. The authors proposed algorithms that aim at minimizing the energy cost needed to collect the aggregated data.

Distributed in-place sorting has been a classic topic in the field of parallel computing [62]. In particular, synchronized parallel algorithms have been studied extensively to sort data that are stored in different topologies. For a linear array of size n, the *odd-even transposition sort* algorithm takes $O(n)$ rounds to sort n numbers. For an $n \times n$ mesh, the *Shearsort* algorithm takes $O(n \log n)$ rounds to sort n^2 numbers, and an asymptotically tight $O(n)$-round algorithm is also available [62]. Here each round usually involves only a constant number of operations for each processor (node) in the network.

C. Basic concepts and terms

In this section, we introduce some basic concepts of the sorting-based data-centric storage scheme, and some terms that will be used in the rest of the chapter.

Consider a network G with n nodes and m data objects. Every data object has a key, whose value is an integer. The network contains a path P that goes through every node at least once. The length of the path is N (i.e., N nodes, where $N \geq n$). (Ideally, the path would go through every node exactly once. However, such a Hamiltonian path may not exist, or may not be easy to construct.) Sorting data means that the data are stored in the nodes so that the order of their key values is consistent with the order of the corresponding nodes in the path. In other words, if we walk through the path P, we sequentially visit the m data objects whose key values keep monotonically increasing (or decreasing).

If two data objects have the same key value, then their relative order in the path can be arbitrarily decided on. Note that a node may appear in the path more than once. The following is a simple example of sorting.

Example 1 *The network G contains four nodes: a_1, a_2, a_3 and a_4. An example of the path P is $P = a_1, a_2, a_1, a_3, a_4$. There are six data objects in the network, whose key values are 2, 4, 6, 6, 7 and 8, respectively. Then the following is an example of sorting: a_1 stores 2 and 6, a_2 stores 4, a_3 stores 6, a_4 stores 7 and 8.*

To well balance the storage load, the nodes should store approximately the same number of data objects. If we construct such a path in the network and sort the data based on the path, then data query, including the range query, will be simple, because a message can follow the key values to the queried data. The routing need not strictly follow the path. In fact, most of the time, the routing can use the edges outside the path that serve as *shortcuts*.

We will present an asynchronous sorting algorithm for an arbitrary network. The following definition defines two terms used for evaluating the complexity of an asynchronous algorithm.

Definition V.1 *Given a distributed network G with n nodes, during an asynchronous computational process in the network each node takes actions. We call it a round when every node has taken exactly one action or given up the chance to take an action. We call each action a step.*

Note that the round we defined here is logical, i.e., *different rounds could be overlapping in the time frame due to the asynchrony of the process.* By definition, there can only be at most n steps taken in each round. The work of the process can then be measured by the total number of steps taken and be estimated by the total number of rounds in the process.

We will use the term *load factor* to measure the load balance of a data storage scheme, as follows.

Definition V.2 *Given a storage scheme where the average load is d data objects per node, the maximum load is d_{max} data objects in a node and the minimum load is d_{min} data objects in a node, the load factor of the scheme is then* $\max\{d_{max}/d, d/d_{min}\}$.

The sorting algorithm we present does not depend on how the path P is constructed. In particular, if the nodes know their locations, it is simple to construct a path with good performance. However, obtaining locations is a highly non-trivial task for sensors. In this chapter, we present a method to construct P for non-localized sensor networks, where node location information is unknown. The construction uses some concepts related to planar graphs, which are defined below.

Definition V.3 *Given a planar graph G_p embedded in the plane, a face F is* adjacent *to another face F' if F and F' share an edge or a vertex. Similarly we say that a face F is* adjacent *to a path P if F and P share an edge or a vertex. The* distance *from a face F to a path P is the minimum number of faces we need to walk through in order to walk from F to a face adjacent to P (by walking through adjacent faces, not counting F itself). The* distance *between an edge e and a path P is the smallest value among the distances from the faces adjacent to e to the path P. The distance between two paths P, P' is the smallest value among the distances from the edges in P to the path P'.*

An example of the above terms is shown in Fig. 27.

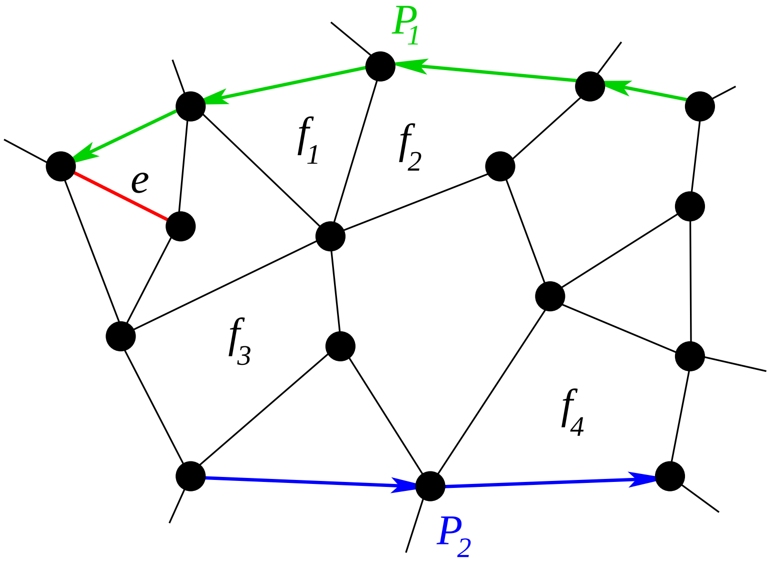


Fig. 27. Adjacency and distance between edges, paths and faces: the faces f_1, f_2, f_3 are adjacent to each other (i.e., distance 1 from each other); f_1 and f_2 are adjacent to the path P_1 (indicated by the green edges at the top); f_3 and f_4 are adjacent to the path P_2 (indicated by the blue edges at the bottom); the distance from f_2 to P_2 is 1; the distance from the edge e to P_1 is 0, and the distance from e to P_2 is 1; the distance between P_1 and P_2 is 1.

D. Network sorting and balanced data storage

In this section, we present a distributed algorithm to solve the NETWORK SORTING problem. That is, given a network G of n nodes, a path P of length N, and m data objects that are stored in the network, the nodes sort the data with the additional constraint that in the end, the numbers of data objects stored in the nodes are as balanced as possible.

We first describe an algorithm for a simplified version of the problem, where the path P is a Hamiltonian path (i.e., each node appears in P exactly once) and every node initially stores one data object (namely, $n = N = m$). Later we will show how to generalize the algorithm to solve the sorting problem for general cases.

During the sorting process, every node remembers the keys of its own data and its neighbors' data. The basic step in the sorting algorithm is a *local sorting operation*:

- A node sorts its own data and its neighbors' data based on their positions in the path. After it sorts the data, each of those nodes (whose data are sorted) informs its neighbors of the key of its new data.

For example, suppose that a node a has four neighbors b, c, d, e, and the keys of their data are 2, 4, 1, 5, 6, respectively. Suppose their order in the path is (b, a, c, e, d). Then after a sorts the data, the nodes a, b, c, d, e store data with the keys 2, 1, 4, 6, 5, respectively.

The key of the algorithm is to carry out the local sorting operations asynchronously and efficiently. To do that, we need to use a scheduling scheme to avoid multiple nodes accessing the same data simultaneously (namely, conflicts), and avoid deadlocks. Our algorithm utilizes a modified scheduling scheme from the d-scheduling algorithm described in [67]. Our algorithm assigns IDs 1, 2, $\cdots$, n to the nodes according to their order in the path P. Every node u is assigned a *priority* (r_u, ID), where r_u is a integral variable whose initial value is 0, and ID is the ID of u. The value of the priority is determined by the concatenated value of r_u and ID, which is $r_u \cdot n + ID$. For example, if the priority of node u is $(2, 16)$, then the value of u's priority is $2n + 16$. In general, in the sorting algorithm, the smaller a node's priority is, the higher a priority the node has for carrying out a local sorting operation.

The general process of sorting with scheduling is as follows:

- If a node u finds that its priority is the smallest among the priorities of the nodes within two hops, and that the data within one hop need sorting, node

u sorts the data within one hop. After this local sorting operation, node u increases its variable r_u by one. Every node in the network sorts data in this way.

We can see that when a node u is sorting data, its neighbors can neither sort data nor have their data sorted by any node other than u. For the nodes two hops away from u, although they cannot sort data, they can have their data sorted by other nodes.

The formal presentation of the sorting algorithm is presented in Fig. 28. It describes the protocol that a node u follows. Every node follows the same protocol, and the sorting ends when no node needs to sort more data. The algorithm shows the detailed messages sent in the sorting process.

When the graph underlying the network is a simple path, the above algorithm becomes the distributed bubble sort algorithm.

Since during the sorting process, each node u's priority is decided by (r_u, ID), the priorities of all the nodes are in total order at any moment. The dependent graph obtained from the network G by putting a directed edge from a node to each neighbor with a higher priority is always a DAG (directed acyclic graph). Therefore there will be no deadlock and the execution of the algorithm will not be halted. Here we call the actual local sorting operation performed by a node in the network a *step*.

Note that by the scheduling scheme, each node will have a chance to perform a step at least after all its two hop neighbors have performed a step.

We show an example of the sorting process in Table IX, which is an instance of the network sorting problem shown in Fig. 29. Initially, the data objects in the network are in the reversed order of the nodes. (The order of the nodes in the path P is $a_1, a_2, \cdots, a_8$. The numbers in the nodes are the keys of data.) Each row in

```
Input: A node $u$ in a network $G$
Output: Data objects in the network are sorted
Initially: $u$ sends its ID and the key of the data it currently has to all its neighbors; $r_u \leftarrow 0$.
if the order of the data objects in $u$'s neighborhood is not consistent to the order of the IDs then
    Send REQUEST to every neighbor
else
    Send SKIP to all neighbors
end if
Wait until it hears from all neighbors
if $u$ gets REQUESTs from neighbors  then
    Put all requests in a priority queue (the smaller the value of the priority is, the higher the priority)
    Send CONFIRM to the node with the smallest priority (including itself) that is requesting
    Send PENDING to all others
end if
if $u$ gets CONFIRM from all neighbors then
    send LOCK to its neighbors
    Perform sorting operation, i.e., sort the data objects in the neighborhood
    increase its $r_u$ by 1
    send UNLOCK to its neighbors
end if
if $u$ gets PENDING from a neighbor then
    $u$ waits for further message from the neighbor
end if
if $u$ gets LOCK from a neighbor then
    $u$ sends REJECT to its requesters
end if
if $u$ gets REJECT from a neighbor then
    $u$ send CANCELREQUEST to its neighbors
    $u$ send CONFIRM to the next requested node in queue
end if
if $u$ gets CANCELREQUEST from a neighbor then
    $u$ send CONFIRM to the next requested node in queue
end if
if $u$ gets UNLOCK from a neighbor then
    if $u$'s data has been exchanged then
        Notify its neighbors about the new key
    end if
    Send REQUEST to every neighbor if necessary
end if
```

Fig. 28. Algorithm **NetworkSorting**: running by each node in the network to solve the network sorting problem.

the table presents the result after the nodes in the first column perform local sorting operations (i.e, steps) simultaneously. It takes only two rounds to sort the data objects in the example.

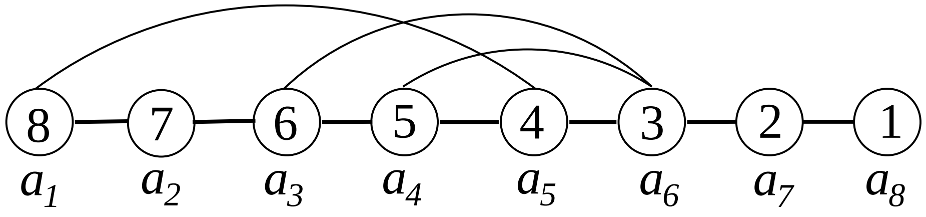


Fig. 29. A sample network sorting instance

Table IX. The sorting process of the sample network in Fig. 29.

Nodes	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8
Initially	8	7	6	5	4	3	2	1
Round 1								
n_1, n_8	4	7	6	5	8	3	1	2
n_3	4	3	5	6	8	7	1	2
n_2, n_7	3	4	5	6	8	1	2	7
n_4	3	4	1	5	6	8	2	7
n_6	3	4	1	2	5	6	8	7
n_5	2	4	1	3	5	6	8	7
Round 2								
n_2, n_8	1	2	4	3	5	6	7	8
n_3	1	2	3	4	5	6	7	8

The total number of rounds is a measure of the maximum work performed by each node. In the following theorem we show that it takes at most $O(N)$ rounds to sort all the data objects in the network. We note that it is a general bound. For two dimensional networks, the actual performance can be much better, which we will show later. The theorem below not only gives a upper bound on the number of rounds needed to solve the simplified NETWORK SORTING problem, but also proves the correctness of the algorithm.

Theorem V.4 *The sorting algorithm described above solves the* NETWORK SORTING *problem for $N = n = m$ after at most N rounds.*

PROOF. W.l.o.g., suppose the keys of the data objects are $1, 2, \ldots, N$ and the nodes are $a_1, a_2, \ldots, a_N$ as ordered on P. For simplicity we identify each data object with its key. Assume after $k \leq N$ rounds, the data i is on the node $a_{f_{i,k}}$. If suffices to show that for any k, i, if $f_{i,k} > i$, $f_{i,k} \leq N - k + i - 1$; otherwise, if $f_{i,k} < i$, $f_{i,k} \geq i + k - N + 1$. That is, the data i is at most $N - k - 1$ hops away from its destination on P.

We use induction on i and k. For the data 1, it is trivial since each round it makes progress towards a_1 unless it is already there. Now suppose for any $s < i$, it is true for the data s. Let us look at the data $i + 1$. At $k = 0$ rounds, the conclusion is true because any two nodes in P have distance at most $N - 1$.

Suppose after $k - 1$ rounds, it is still true. Assume that $f_{i+1,k-1} > i + 1$; by the induction hypothesis, we know that $f_{i+1,k-1} \leq N - k + i + 1$. In the k-th round, if data $i+1$ has made progress towards its destination, then $f_{i+1,k} \leq f_{i+1,k-1} - 1 \leq N - k + i = N - k + (i + 1) - 1$. On the other hand, if data $i + 1$ failed to make progress in the k-th round, it must have been passed by a smaller data object i' (here $i' \leq i + 1$). Again by the induction hypothesis, we have $f_{i+1,k} = f_{i',k-1} = N - (k - 1) + i' - 1 = N - k + i' \leq N - k + i$.

The case when $f_{i+1,k-1} < i + 1$ can be proved by symmetry.

When $f_{i+1,k-1} = i + 1$, if data $i + 1$ did not move, the statement is trivially true. Otherwise it has either been exchanged with a smaller data i' or a larger data i''. We will either have $f_{i+1,k} = f_{i',k-1} = N - (k - 1) + i' - 1 = N - k + i' \leq N - k + i$, or have $f_{i+1,k} = f_{i',k-1} = k - 1 + i' - N + 1 = k + i' - N \geq (i + 1) + k - N - 1$. This completes the proof. □

Since by definition, each round consists of at most N steps, the following corollary is straightforward.

Corollary V.5 *The sorting algorithm described above solves the* NETWORK SORTING *problem when $N = n = m$ in at most $O(N^2)$ steps.*

Now let us consider the case where P is not a Hamiltonian path (i.e., $N > n$) and we have $m = N$ data objects (one for each occurrence of a node in the path P). Each node may have multiple IDs because it may appear multiple times in the path P. Since we can always view a node with multiple occurrences on P as multiple copies of the same node, we can map the data to the IDs so that they are consistent in the path. That is, in the end, the data object with the smallest key should be matched to the smallest ID, the second smallest key should go with the second smallest ID, ..., and so on. The above NetworkSorting algorithm can be applied to sort data, which will again end in $O(N)$ rounds and $O(N^2)$ steps. The only unsolved issue is that data may not be well balanced, whose solution we will discuss in the following most general case.

In the general case, there is no constraint for n, N and m. We call this problem the GENERALIZED NETWORK SORTING problem. To solve this problem, we need to modify the algorithm. First of all, a node at the end of the path learns the values of n, N and m (which is easy), and sends a message along the path to let every ID know how many data objects it should have in the end. (Every node should have $\lceil m/n \rceil$ or $\lfloor m/n \rfloor$ data objects in total in the end.) Then, during the sorting process, every node u keeps track of the number of data objects stored by each neighboring ID (a copy of the neighbor). When a node u performs the local sorting operation, u collects all the keys in its neighborhood, sorts them, and re-distribute the data objects so that the neighbors and itself store the data as evenly as possible. Note that here the IDs may be attached with different numbers of data objects to ensure the nodes hold similar numbers of data objects. The scheduling part of the algorithm remains unchanged. The following theorem proves that the modified NetworkSorting algorithm solves the general network sorting problem in $O(Nm)$ rounds and $O(N^2m)$ steps.

Theorem V.6 *The sorting algorithm described above solves the* generalized network sorting *problem in $O(Nm)$ rounds and $O(N^2m)$ steps.*

Proof. As we have showed before, there is no deadlock during the execution of the algorithm. In that case the smallest data will reach its destination in at most $N-1$ rounds. After that the second smallest data will be in place in at most $N-2$ rounds. And so on ... Since there are m data objects in the network, the conclusion of the theorem follows. □

E. Sorting performance in mesh networks

The upper bounds presented so far for the sorting performance are for general networks. In this section, we analyze the sorting performance for one- and two-dimensional arrays. We show that the performance of the NetworkSorting algorithm meets the lower bound when the network is a one-dimensional array. When the network is a two-dimensional array, the algorithm achieves the same performance as the well-known *synchronized* shear-sort algorithm. The performance for arrays is much better than the previous upper bounds.

The analysis here provides evidence that the actual sorting performance in sensor networks can be much better than the upper bounds presented earlier, because sensor networks often resemble two-dimensional mesh networks. We will further demonstrate the good sorting performance in sensor networks by extensive simulations later.

In the following analysis, we assume the previously discussed simplified model, where the path P is a Hamiltonian path, and the number of data objects equals the number of nodes. That is, $n = N = m$.

When the network is an $n_1 \times n_2$ array, then it is easy to get the following simple lower bounds.

Fact V.7 *For any distributed sorting algorithm sorting data in an $n \times 1$ array, in the worst case, $\Omega(n)$ rounds are needed. Furthermore, $\Omega(n^2)$ data exchanges are necessary to sort data in this network.*

Fact V.8 *For any distributed sorting algorithm sorting data in a $\sqrt{n} \times \sqrt{n}$ array, in the worst case, $\Omega(\sqrt{n})$ rounds are needed. Furthermore, $\Omega(n\sqrt{n})$ data exchanges are necessary to sort data in this network.*

1. Sorting in linear array

When the underlying network is a linear array of n nodes, our algorithm has the same performance as the odd-even transposition sort algorithm described in [62] and meets the lower bound performance, by Theorem V.4 and Fact V.7. We have the follow theorem directly.

Theorem V.9 *The* NetworkSorting *algorithm solves the network sorting problem on a linear array of size n in $O(n)$ rounds.*

2. Sorting in $\sqrt{n} \times \sqrt{n}$ grid

Now suppose our network is a $\sqrt{n} \times \sqrt{n}$ grid and the path P is a snake-like path [62]. An example of such path is shown in Fig. 30, where the path is indicated by the arrows. Making a path be snake-like helps improve the sorting performance, which we have adopted in our implementation of the storage scheme.

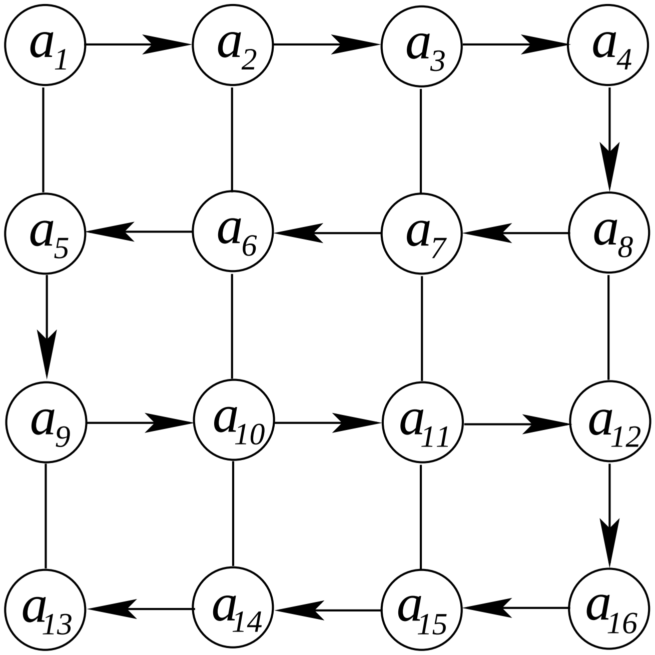


Fig. 30. A snake-like path marked by arrows in a 4×4 mesh.

We show that our sorting algorithm will sort data in $O(\sqrt{n} \log n)$ rounds. Since our algorithm is oblivious, we will use the following 0-1 sorting lemma [62] in our proof.

Lemma V.10 The 0-1 Sorting Lemma. *If an oblivious comparison-exchanged algorithm sorts all input sets consisting solely of 0s and 1s, then it sorts all input sets with arbitrary values. [62]*

In the proof of the following lemma, we assume that the data in the network are either 0 or 1.

Lemma V.11 *Suppose the data to be sorted are only 0s and 1s. Suppose in a $n_1 \times n_2$ grid ($n_1 \leq n_2$), all the data objects in the first i ($1 \leq i < n_1$) rows are sorted according to the snake-like path P, and so are all the data objects in the last $n_1 - i$ rows. The* NetworkSorting *algorithm will sort all the data objects in the grid in $2n_2$ rounds.*

Proof. We use induction on n_1. When $n_1 = 2$, the only choice for i is 1. Suppose we have more 1s than 0s in the network. In the first round, all 1s in the first row will be put into the second row and the second row will contain no 0. After n_2 rounds, the algorithm will sort the data in the first row (hence all data in the network). Similarly when there are more 0s than 1s, in the first round all 0s in the second row will be pushed up to the first row. And the algorithm will sort the second row in n_2 rounds.

Now suppose for $n_1 = k$, our conclusion holds. Let us look at the case when $n_1 = k + 1$. Since the first i rows are sorted, if the first row contains only 0s, this row will not be touched by the algorithm. Hence the sorting only involves the $i - 1 + n_1 - i = k$ rows. By the induction hypothesis, the proof is done. Similarly, if the n_1-th row contains only 1s, the algorithm can also sort the network in $2n_2$ rounds.

If on the contrary, the first row contains 1s and the last row contains 0s, then we know that all the rows from 2 to i are filled with 1s and all the rows from $i + 1$ to $n_1 - 1$ are filled with 0s. In this case, the following sorting is essentially inside each column, which is similar to sorting in a linear array. So the algorithm will sort the network in $n_1 + n_2 \leq 2n_2$ rounds. This completes the proof. □

Now we are ready to prove the following lemma.

Lemma V.12 *The* NetworkSorting *algorithm solves the network sorting problem on a $\sqrt{n} \times \sqrt{n}$ grid with data being 0s and 1s in $O(\sqrt{n} \log n)$ rounds.*

Proof. We first consider a constrained version of the algorithm. For the first $2\sqrt{n}$ rounds, we disallow data exchanges between different rows; for the next $2n$ rounds, we allow data exchanges only between row $2i-1$ and $2i$ where $1 \leq i \leq \sqrt{n}/2$; and so on .... That is, in general, at the k-th $2\sqrt{n}$ rounds, we split the grid into chunks each of which has 2^{k-1} rows, and disallow data exchanges between rows that belong to different chunks. By Lemma V.11, after each k-th $2\sqrt{n}$ rounds, we have the subnetwork within each chunk sorted according to the snake-like path P. Hence after $O(\sqrt{n}\log\sqrt{n})$ rounds, the whole network will be sorted.

Since the NetworkSorting algorithm does not have the above constraints, its data exchange is more efficient than the above constrained version, as explained in the following. For both the original algorithm and the constrained version, note that we only have 0s and 1s in the network. Consider the row that a data object is in. The data object will never move "backwards". that is, once a 0 has reached row i, it will never be pushed down to a row j with $j > i$; on the other hand, once a 1 has reached row i, it will never be pushed up to a row j with $j < i$. Therefore for a given instance and each row i in the mesh, the number of 0s pushed up from rows with id greater than i to rows with id smaller than or equal to i is fixed in either algorithm. Since the NetworkSorting algorithm does not have the constraints in the constrained version, the 0's are pushed up at least as fast as the above constrained version of the algorithm. Hence the number of rounds needed for the NetworkSorting algorithm is also $O(\sqrt{n}\log n)$. □

By the 0-1 sorting lemma, we get have the following theorem.

Theorem V.13 *The* NetworkSorting *algorithm solves the* network sorting *problem on a* $\sqrt{n} \times \sqrt{n}$ *grid in* $O(\sqrt{n}\log n)$ *rounds and* $O(n\log n)$ *steps.*

The above upper bound for the NetworkSorting algorithm matches the performance achieved by the Shear Sort algorithm [62], which requires global synchronization. Since the NetworkSorting algorithm is asynchronous, it is more appropriate for sensor networks.

F. Implementation for unlocalized sensor networks

In this section, we present our sorting-based balanced data storage scheme. Recall that the network sorting problem requires a path P that traverses all nodes in the network. For better performance, we prefer the path to be snake-like. If the sensors' locations are known, constructing such a path is simple. However, obtaining location information is often difficult. In this chapter, we consider unlocalized sensor networks, where no information is known, and present a solution.

The storage scheme stores data in a sorted and load-balanced way. The former property ensures that queries, including range queries, can be efficiently answered. In addition to presenting the construction of the path P, we also discuss other aspects of the data-centric storage scheme.

1. Constructing the path P

Although the NetworkSorting algorithm has no specific requirement on the path P, the actual complexity of the sorting algorithm relies on its shape and length. A short snake-like path is desirable. As we will show later with our simulation results, a shorter path incurs much lower communication costs. We know that determining if a graph has a Hamiltonian path is NP hard [41]. So clearly, finding the shortest path that traverses all nodes in a given graph is also NP hard. It is therefore very unlikely that the optimal solution can be found efficiently. On the other side, the path's length is at least n, and the traversing of any spanning tree of the network will give us a path of length $2n$. This gives us a simple ratio-2 approximation for the path.

In this subsection, we present a practical algorithm that constructs the path P, which is typically much shorter than $2n$ for unlocalized wireless sensor networks. The performance of our algorithm will be verified later with extensive simulations.

The algorithm for finding such a path consists of three major steps: (1) planarize the network; (2) construct a snake-like backbone path in the planarized network; (3) extend the path to include all nodes in the original network.

There have been several papers on how to obtain a planarized network efficiently for unlocalized wireless sensor networks [35, 97, 98]. We skip the details here because it is not the focus of this chapter. In the following we will discuss the second and the third steps.

a. Construct the backbone path

Assume that the network is already planarized and we have a topological embedding of the planar network, which we denote by G_{planar}. We also assume that the true network G is deployed in a plane. G_{planar} is a subgraph of G. To simplify the discussion, we assume that the outer face of the planar graph G_{planar} is a simple cycle. (Otherwise the graph is 1-connected and we can deal with each part of the tree-like structure easily.)

Let f be the outer face of the network G_{planar}. We split f into four path segments P_1, P_2, P_3, P_4 of roughly equal lengths. Next we stretch the outer face into a square such that P_1, P_2, P_3, P_4 are on the north, east, south and west side of the square, respectively. In the following discussion, the outer face is not to be considered in super paths or the measurement of distances.

Let us first look at a simple algorithm that will find the medial axis of two paths on the outer face. The medial axis is defined as a path that is at roughly the same distance from those two paths. We assume that the distance between the two input paths is at least one, i.e., no two edges on P_1 shares any face with edges on P_3.

```
Input: $G_{planar}, P_1, P_3$
Output: $P'$: a path formed by nodes whose distances to $P_1, P_3$ are roughly equal.
$G' \leftarrow$ the subgraph induced by faces and edges (excluding the outer face and its
    edges) whose distances to $P_1$ and $P_3$ are equal.
Let $u, v$ be two nodes in $G'$ that are on the east and west boundaries of $G_{planar}$,
    respectively.
return a path $P'$ from $u$ to $v$ in $G'$
```

Fig. 31. Algorithm **Medial-Axis**: find the medial axis between two disjoint paths in a planar graph.

Now we prove that the **Medial-Axis** algorithm shown in Fig. 31 returns a simple path if P_1, P_3 have distance at least one.

Theorem V.14 *Given the planar graph G_{planar} and the paths P_1, P_3 on the outer face, the* **Medial-Axis** *algorithm in 31 returns a simple path P' such that the distances from the edges in P' to P_1 and P_3 differ by at most* 1*, and P' separates P_1 and P_3 in G. Furthermore, P' does not share edges with P_1 or P_3.*

Proof. If two faces are adjacent, their distances to a given path differ by at most one. Hence for each edge in G', its distances to P_1 and P_3 differ by at most one. It suffices to show that G' is a connected graph.

Define the *super path* between two faces f_s, f_t in the planar graph G_{planar} as the path in the dual graph of G_{planar}, i.e., the super path is a sequence of faces in G such that every two adjacent faces in the sequence share at least one edge and the sequence starts at f_s and ends at f_t.

Consider any two edges e_1 in P_1 and e_2 in P_3. Any super path between any face adjacent to e_1 and any face adjacent to e_2 must come across a face or an edge in G'. If the super path visits a face that has equal distance to P_1 and P_3, we are done. Otherwise there must be an edge e in G_{planar} shared by two faces on the super path such that the distances from e to P_1 and P_3 are equal. Let the distances from e_1 to P_1, P_3 be 0 and Y, respectively. Let the distance from e_2 to P_1, P_3 be X and 0, respectively. Here $X > 1, Y > 1$. These distances are also distances from the first and the last faces on the super path to P_1, P_3. We know that on the super path, the distance from two adjacent faces to a path can change by at most 1. Consider the distance pairs as coordinates on a plane. If we connect the points defined by each pair of adjacent faces on the super path, we obtain a curve from $(0, Y)$ to $(X, 0)$. The continuity of the plane implies that this curve must cross the line $x = y$. If the crossing point is also on the super path, we have a face in G'. Otherwise there are two adjacent faces F_1, F_2 on the super path such that $d(F_1, P_1) = d(F_2, P_2)$ and $d(F_1, P_2) = d(F_2, P_1)$. (Here $d(\cdot)$ is the distance.) Hence the edge shared by F_1, F_2 is in G'.

Therefore G' separates P_1 and P_3, not only in G_{planar} but also in the dual graph. In other words, if we block faces and edges in G' from G_{planar}, there will be no super path between faces adjacent to P_1 and faces adjacent to P_3. Suppose G' is disconnected, since both G_{planar} and G' are planar and G_{planar} is connected, there would exist a super path from some edge on P_1 to some edge on P_3 that do not cross G', which is a contradiction.

Now that G' is a connected graph and separates P_1 and P_3, it must contain nodes on both the east and the west boundaries of G_{planar}. Therefore the algorithm returns a path P' that separates P_1 and P_3.

Suppose P' shares an edge e with P_1. Then the distance from e to P_1 is 0. Hence the distance from e to P_3 is 0, which is impossible since we assume that P_1 is at least distance 1 away from P_3. □

The path P' separates the network G_{planar} into two subgraphs. For each subgraph, we apply the same algorithm to find its medial axis path. We recursively partition the network and find medial axis paths. In the end, we get a set of "horizontal paths" in the network G_{planar} that do not cross each other. (A technical detail: during the recursive process, if the northern and southern boundaries of a new subgraph are adjacent, the subgraph can be disconnected by removing a face. In this case, we can either choose not to run the algorithm on it, or choose to find a medial axis path for each of the two separated components, and then connect the two paths into a single path by going through the northern boundary.)

The algorithm for constructing all the horizontal paths in the network is presented in Fig. 32.

```
Input: $G_{planar}, P_1, P_3$
Output: Horizontal paths containing all the nodes in $G_{planar}$.
if $P_1 = P_3$, return
$P' \leftarrow$ Medial-Axis($G_{planar}, P_1, P_3$)
Ouput $P'$
$G_1 \leftarrow$ the subgraph between $P_1$ and $P'$
Horizontal-paths($G_1, P_1, P'$)
$G_2 \leftarrow$ the subgraph between $P'$ and $P_3$
Horizontal-paths($G_2, P', P_3$)
```

Fig. 32. Algorithm **Horizontal-paths**: construct a set of horizontal paths.

b. Construct the path traversing all nodes

In the previous step we have obtained a set of horizontal paths that do not cross each other. They form a total order from top to bottom (that is, from north to south). A straightforward way to get the snake-like path is to take the first path from left to right, connect it to the right end of the second path (which is simple to do), and walk through the second path from right to left, then turn on the third path in a similar way, and so on.

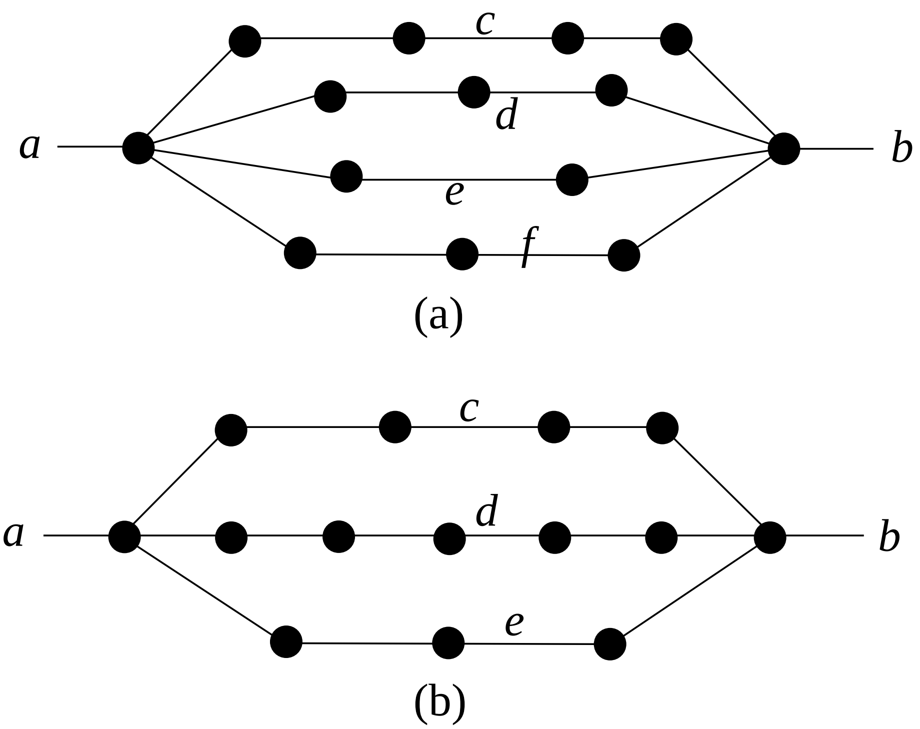
The horizontal paths could have some overlapping edges. To make the path short, we try to avoid repeating the common segment of the horizontal paths. Since every two adjacent horizontal paths must have adjacent end nodes, we only need to distinguish two cases here.

If there are an even number of paths sharing both ends, as shown in Fig. 33 (a) (in which a, b, c, d, e, f are sub paths in the horizontal paths), assume e is the shortest one among c, d, e, f, the we traverse these paths as $a - c - d - e - f - e - b$. (That is, use e twice.) If there are an odd number of paths sharing both ends, as shown in Fig. 33 (b), we traverse these paths in the order $a - c - d - e - b$. The approach can be used recursively to minimize the length of the path.

The path constructed so far may not include all the nodes of the original network G. To include all the nodes into the path P, we can use the following simple heuristic. If a node has two neighbors that are consecutive in the path, it inserts itself into the middle of the neighbors. After that, if there are still a few nodes not in the path, use a tree to attach to the path, and use the traverse of the tree as part of the path. This approach makes P include all the nodes well and very quickly.

2. Sorting data in the network

We assume that each node knows the total number m of data objects stored in the network, the number n of nodes in the network, and the length N of the path P. (These numbers can be easily learned with a simple information collection operation, especially with the help of the path P.) Then every node should store m/n data



(a)

(b)

Fig. 33. Overlapping horizontal paths.

objects. A node knows how many times it appears in the path, so it can decide how many data objects to assign to each of its copy in the path. The sorting process is described as before. It generates a sorted and balanced storage result.

3. Data access and query

With the sorted data and the path P, querying data is simple. When a node u wants to query data objects with the key value k, it sends a query message that contains the value k and its own ID. Every relay node chooses the neighbor whose data objects have a key closest to k as the next hop. The process continues until the query message reaches the destination node. The routing is guaranteed to succeed because the edges in P can always be used for routing if necessary. In practice, most of the time the edges not in P are *shortcuts* and make routing much more efficient. To send the data back to u, a similar routing protocol is used, except that node u's ID is used for routing instead of the key k. And the routing is also guaranteed to succeed.

The range query is answered with the same method, because data with adjacent key values are stored next to each other in the path P.

4. Data dynamics

When a data object of key k is inserted into the network, the initiator of the insertion will send the object as if it were a query message for k. Once the message reaches its destination (where it should be stored), the new data object will be stored there. Certainly, the load balance is affected over time. We discuss its solution in the following.

The total number of the data objects in the network will be calculated and broadcasted periodically. With the existence of the path P, this task is simple. Once a node discovers that its load is too heavy or too light, it first tries to solve the imbalance locally by exchanging data with its neighbors on the path P. If local operations fail to bring back load balance, it triggers a re-balance process. This process is similar to the sorting process, only cost less.

In practice we make the threshold of balanced ratio be 2 and 1/2. That is, if a node's load is more than twice the average load or less than half the average load, it will try to work with its neighbors to bring the load factor to below 2, or trigger the re-balance process. Therefore at any moment the load factor of our scheme is no worse than 2.

The more data objects there are in the network, the more data insertions will be needed to trigger the re-balance process, given that the distribution of the new data objects is not too skewed. In practice the overhead brought by the re-balance process is quite light if it is averaged over each data insertion.

Data deletion is dealt in a very similar way as insertions. We skip the details.

G. Performance evaluation

We conduct extensive simulations to test the performance of our sorting based data centric storage scheme. Our results show that the performance is very stable for different network models and different degree of data loads. We compare our results with GHT. Our schemes outperforms GHT in terms of communication costs for storage, query and, more importantly, data load balance. In this section we present some typical simulation results.

We randomly deploy $n = 1500$ nodes in the sensor field. The network model we explore contains UDG and quasi-UDG. We also test the storage schemes with holes present in the sensor field. For quasi-UDG(R, r), when two nodes' distance is between r and r, we set the probability them having a direct link in between to be 0.3. The data objects are generated by random nodes with keys in the range $[0, n]$. The total number of data objects we examined ranged from $100n$ to $1000n$. In each experiment, 1000 random queries are initiated by random nodes.

Three storage scheme were compared: GHT, sorting based on snake like path (sorting(snake) for short), and sorting based on spanning tree path (sorting(tree) for short). Sorting(snake) is the scheme we discussed throughout this chapter. Sorting(tree) scheme differs from sorting(snake) in the sense that instead of going through the preprocessing stage to construct a path that traverses all nodes, it simply traverse the graph using DFS method and use the trajectory as the path. Sorting(tree) has very simple preprocessing stage. However, the simulation results show that its overall performance is worse than sorting(snake) approach.

Since GHT requires nontrivial pre-process, especially when GPSR is to be used for routing. This is true even when the location information of the nodes is known. In the following discussion, we will assume GHT has the position of each node and we ignore the cost for GHT to find which node is the closest to given location in the sensor field.

The qualities of the paths obtained by sorting(snake) and sorting(tree) are compared in table X. The lengths of the paths are the product of the average frequency of the nodes and the number of nodes, n. From the table we see that the tree based path is substantially longer than that of the snake path. As we have shown previously in the analysis of our sorting algorithm, the sorting and query performance are closely related to the length of the path we have. We will show later that the performance of the storage scheme based on the tree path is worse than the one based on the snake path. In figure 34 we show four typical networks in our simulations. In the figures the red (thick) paths are the intermediate snake-like paths during the construction of the path that traverses all nodes. Blue (thicker) paths are typical query paths in the

Table X. Path qualities: maximum /average freq of nodes appearing.

	UDG		UDG(hole)		2-qUDG	
	max	avg	max	avg	max	avg
Snake path	9.31	1.51	9.36	1.50	9.62	1.49
Tree path	4.01	1.98	4.04	1.98	4.58	1.98

networks after the storage process is done.

During the sorting process, we assume the total number of data objects in the network is known by every node (This information can be collected and broad casted easily by traversing our path no more than twice). Although the sorting algorithm described in previous sections is for one data per node, it can be easily generalized to deal with multi data object. In the sorting process, each node that is performing a sorting operation will distribute the data objects in its neighborhood equally to the nodes within the neighborhood. If every node in the neighborhood is holding roughly the same amount of data (with differences $\delta \leq 20$, no redistribution will happen. Instead, data objects will be exchanged to ensure the order property. Table XI contains the load distribution of total $500N$ data objects in the network. In that case the average load of the nodes is all 500. Fig. 35 shows the typical data load distribution of GHT and sorting(snake) based scheme on a UDG with average degree roughly 7. We could make the load distribution for the sorting based schemes more balanced by make δ smaller. But that in turn will increase the communication cost for storage and data rebalance.

The comparison of the communication costs among the three schemes we have is presented in Table XII and Table XIII. For GHT, the costs per data object for storage, query, data dynamics are all the same since they are basically the length of the routing path between two nodes. In GHT, routing is done by combining greedy forwarding and local flooding. A message is always tried to be relied to a node that

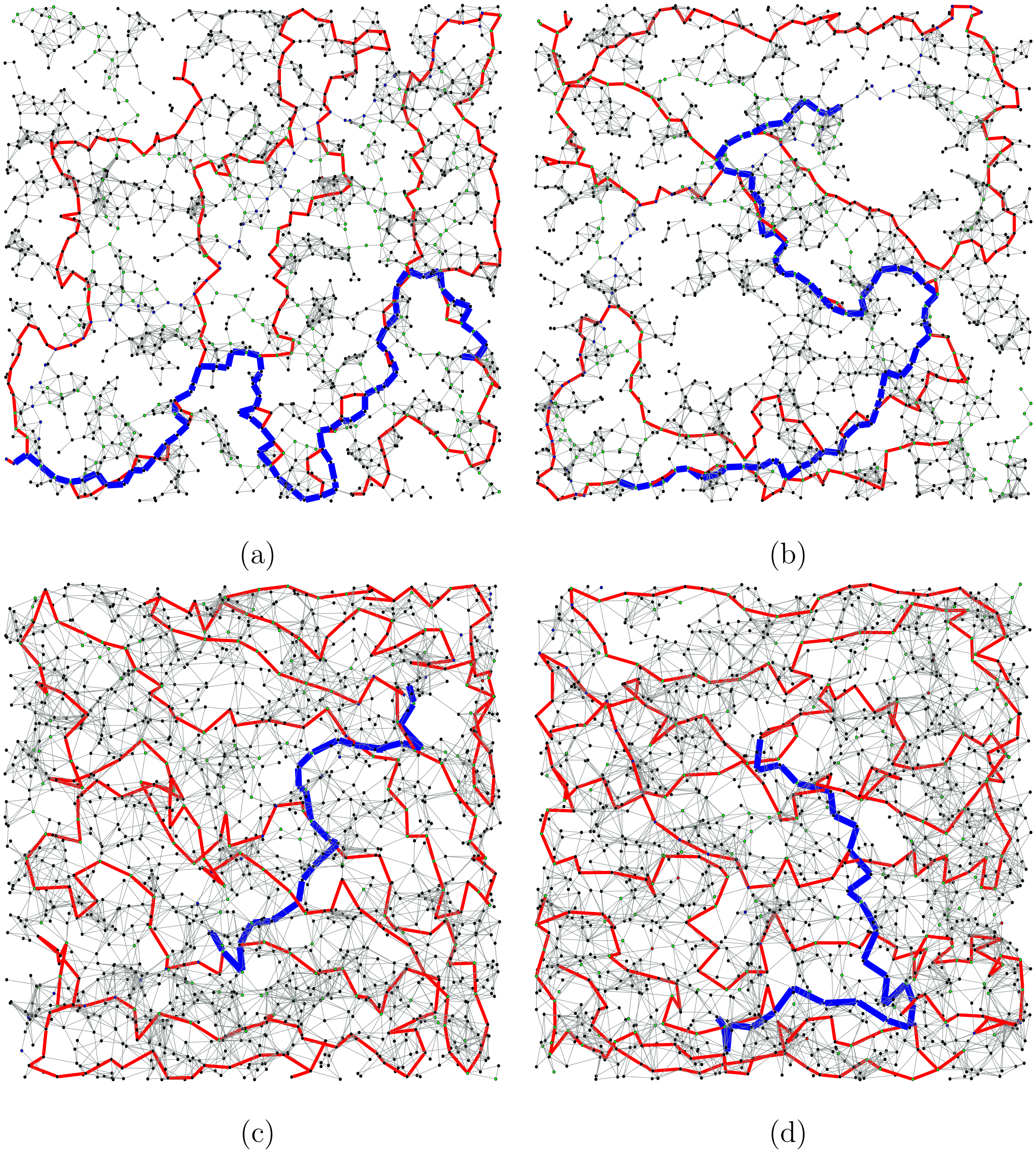


Fig. 34. Sample networks: the red paths are the intermediate snake-like path we are work on; the blue(thicker) paths are typical query paths in each network: (a) UDG with average degree 6.8; (b) UDG with average degree 7.1 and holes; (c) quasi-UDG with average degree 7.3 and $R/r = 5$; (d) quasi-UDG with average degree 7.4 and $R/r = 10$.

Table XI. Data load: max load/std. deviation. Note that the averages are all the same: 500.

	UDG		UDG(hole)		2-qUDG	
	max	σ	max	σ	max	σ
GHT	4271.18	577.56	4737.01	592.03	3740.59	571.05
Snake	564.42	6.18	561.50	6.10	560.17	5.54
Tree	633.05	6.16	627.97	6.12	613.54	5.54

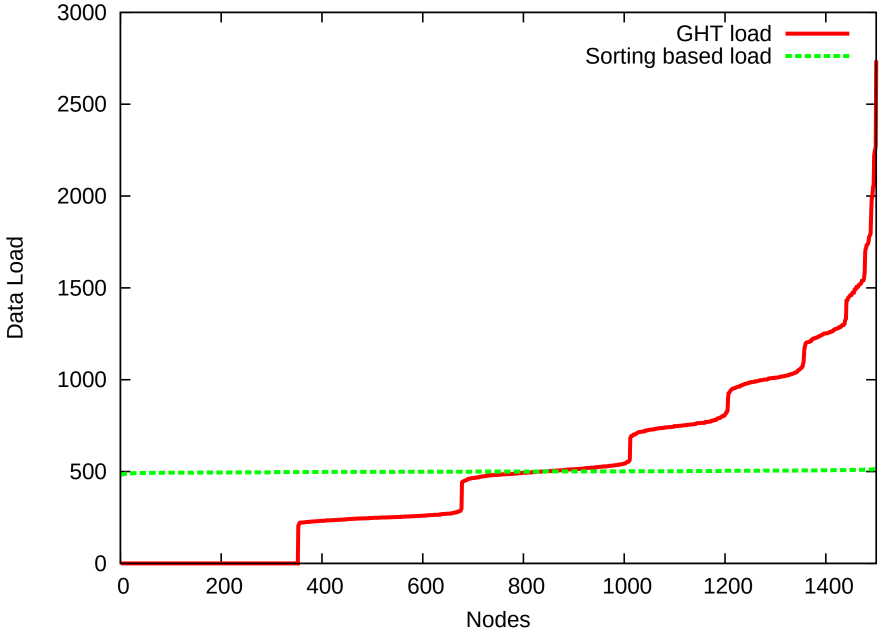


Fig. 35. Typical data load distributions of GHT scheme and sorting based scheme on the same UDG network with average degree 7.

Table XII. Communication cost for storage and data dynamics: average number of messages per data object per operation.

Scheme	operation	average load	UDG	UDG (hole)	2-qUDG
GHT	*	*	291.14	372.47	59.45
Sorting (snake)	*storage*	100	34.39	33.23	27.46
		500	182.37	175.97	142.77
	insertion	100	62.24	62.62	57.35
		500	67.62	66.14	62.20
Sorting (tree)	*storage*	100	42.00	40.55	34.67
		500	217.66	207.67	177.69
	insertion	100	95.66	97.46	98.32
		500	93.79	93.18	93.75

is closer to the destination. When a local maxima is encountered, the message enters the local flooding mode to overcome it. While doing local flooding, a node doubles the radius of flooding if the last attempt fails until a closer node is found.

We show the performance of our sorting based schemes for the cases when the average number of data objects is 100 and 500 on each node. During the sorting process we ignore the control messages being exchanged between adjacent nodes because they are all very small in size comparing to the data objects and routing messages. From the table, we can see that the query performance is very stable for both scheme. In all cases, the query cost of our scheme is better than that of GHT. To measure the cost for data dynamics, we insert data into the network. When a node find that its load is more than two times the average or less than half the average, it triggers a rebalance operation to force the network to redistribute the load. We measure the cost for data insertion as the cost for storing the data on proper nodes plus rebalancing the network. The simulations show that average cost for data insertions is independent with the total amount of data we have in the network.

To characterize the storage process more vividly, we sample 1000 data object

Table XIII. Communication cost for query: average number of messages per data object per operation.

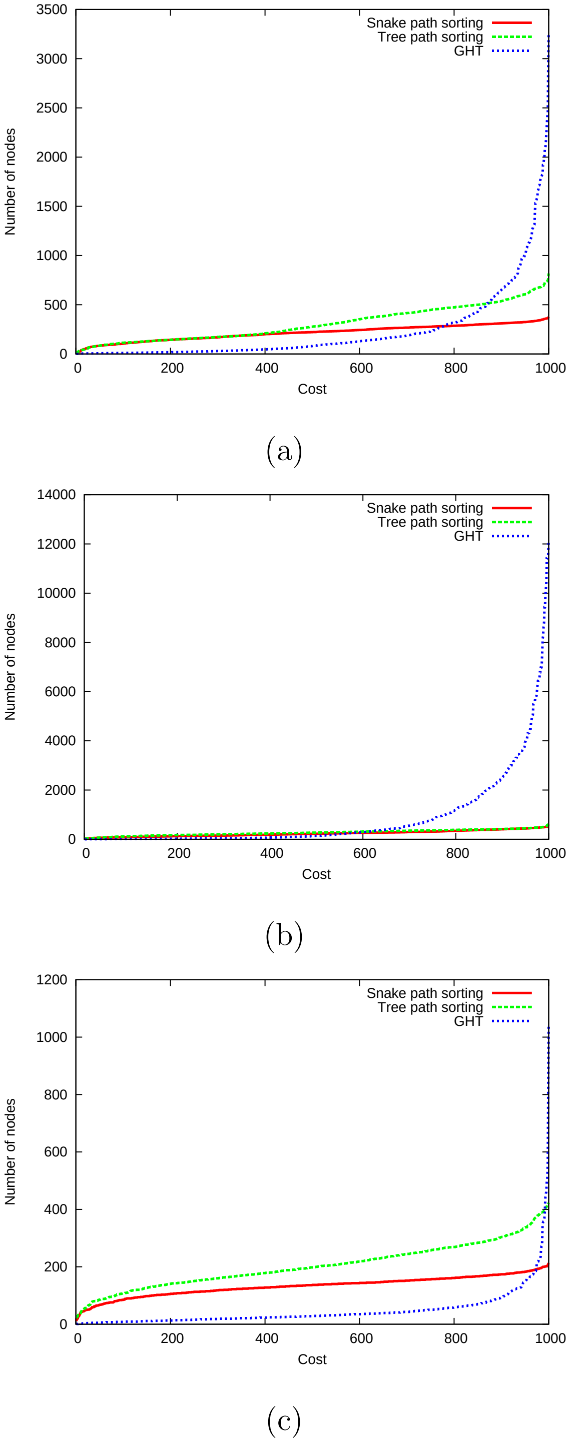
Scheme	average load	UDG	UDG (hole)	2-qUDG
GHT	*	291.14	372.47	59.45
Sorting	100	45.22	45.41	35.37
(snake)	500	45.33	45.54	35.03
Sorting	100	53.40	54.63	43.27
(tree)	500	53.56	53.22	42.50

randomly and measure the distances they travelled in the storage process for each storage scheme. In Fig. 36 we show the distribution of such distances for the three network model we present. The distance of the data objects travelled is more stable in the sorting based schemes than that of GHT.

H. Conclusion

We present sorting based data centric storage schemes for wireless sensor networks. Our protocol does not require location information of the nodes and achieves load balance in term of the amount of data each node is storing. Our schemes support ranged query in a natural way. The communication costs for both storage and query outperform the GHT scheme even with the knowledge of the location of each node. The average maintenance cost per data insertion is also plausible.

Our future work includes (1) further improve the performance of our storage scheme; (2) generalize our method to multi-dimensional data. We observed that during the data query process, the messages tend to follow the base path when the shortcuts in the network are more than 3 hops long. One possible way to improve the query performance is to store simple shortcut information on each node so that messages will take the shortcuts more.



(a)

(b)

(c)

Fig. 36. The distribution of the distance of 1000 sample data objects travelled during the storage process. (a) UDG with average degree 6; (b) UDG with average degree 6 and holes; (c) quasi-UDG with average degree 6 and $R/r = 2$.

CHAPTER VI

IMPROVED ALGORITHMS FOR PATH, MATCHING, AND PACKING PROBLEMS

We develop new and improved randomized and deterministic algorithmic techniques for PATH, MATCHING, and PACKING problems. We introduce a new divide-and-conquer technique. It leads to significant improvements to the random algorithms for these problems. For example, our randomized algorithm for the k-PATH problem runs in time $O(4^k k^{3.42} m)$ and space $O(nk \log k + m)$, improving the previous best randomized algorithm of running time $O(5.44^k km)$ and space $O(2^k kn + m)$. For deterministic algorithms, we present an improved k-color coding scheme. We develop an improved upper bound $O(6.4^k n)$ on the number of k-colorings in a k-color coding scheme. This leads directly to a deterministic algorithm of time $O(12.8^k nm)$ for the k-PATH problem, improving the previous best deterministic algorithm for the problem that runs in time $O(c^k nm)$, where $c > 8000$. Our techniques also lead to similar or more significant improvements on randomized and deterministic algorithms for MATCHING and PACKING problems, such as 3-D MATCHING, 3-SET PACKING, and TRIANGLE PACKING.

A. Introduction

This chapter studies new and improved algorithmic techniques for exact and parameterized algorithms for the NP-hard problems PATH, MATCHING, and PACKING. This research direction has recently drawn considerable attention [1, 17, 31, 49, 52, 56, 68, 71, 80, 88].

The k-PATH problem (given a graph G and an integer k, either construct a simple path of k vertices in G or report that no such path exists) is closely related to a number of well-known NP-hard problems, such as the LONGEST PATH problem, the HAMILTONIAN PATH problem, and the TRAVELING SALESMAN problem. Earlier algorithms [7, 71] for the k-PATH problem have running time bounded by $O(2^k k! n^{O(1)})$. Papadimitriou and Yannakakis [78] studied a restricted version of the problem, the $(\log n)$-PATH problem, and conjectured that it can be solved in polynomial time. This conjecture was confirmed by Alon, Yuster, and Zwick [1], who presented for the k-PATH problem randomized and deterministic algorithms of running time $O(2^{O(k)} n^{O(1)})$. This also provides currently the best polynomial time approximation algorithm of ratio $O(n/\log n)$ for the LONGEST PATH problem. Very recently, the k-PATH problem has found applications in bioinformatics for detecting signaling pathways in protein interaction networks [88] and for biological subnetwork matchings [52].

The exact and parameterized MATCHING and PACKING problems were first studied in [26]. In particular, deterministic algorithms of running time $O(2^{O(k)}(3k)! n \log^4 n)$ were developed for the 3-D MATCHING problem (given a set S of triples and an integer k, either find a subset of k disjoint triples in S or report that no such subset exists) and the 3-SET PACKING problem (given a collection C of 3-sets and an integer k, either find a sub-collection of k disjoint 3-sets in C or report that no such sub-collection exists). The upper bounds for the complexity of these problems were subsequently improved to $O((5.7k)^k n)$ based on the *greedy localization* techniques [17, 49]. Koutis [56] developed randomized algorithms of time $O(10.88^{3k} n^{O(1)})$ and space $O(2^{3k} + m)$, and deterministic algorithms of time $O(2^{O(k)} n^{O(1)})$ for these problems. This upper bound was further improved to $O((12.7D)^{3k} n^{O(1)})$ (where $D \geq 10.4$) by Fellows et al. [31]. Algorithms for packing a small subgraph in a given graph, such as TRIANGLE PACKING (given a graph G and an integer k, either find a set of k vertex-disjoint triangles or report that no such set exists), have also been studied [31, 68, 80].

Currently, the best randomized and deterministic algorithms for the k-PATH problem [1] and the MATCHING and SET PACKING problems [31, 56] are all based on a technique called *color coding* developed by Alon, Yuster, and Zwick [1]. Take the k-PATH problem as an example. We say that a simple path in a graph G is *properly colored* under a coloring of the vertices in G if no two vertices on the path are colored with the same color. The algorithms proposed in [1] proceed as follows. Suppose that there is a path P of k vertices in G, starting from a vertex v_0. To find the path P, first we color the vertices of the graph G using k colors so that the path P is properly colored. Then we use a (deterministic) dynamic programming algorithm, which for each vertex u records every possible color set C such that there is a properly colored simple path from v_0 to u that uses exactly the colors in the set C. Since there are at most 2^k different color sets, the dynamic programming algorithm runs in time $O(2^k km)$ and space $O(2^k kn + m)$.

Therefore, the critical step is how to construct a coloring for the graph G so that the path P is properly colored. Alon, Yuster, and Zwick [1] proposed two approaches to this problem. The first is a randomized algorithm of running time $O(e^k n)$ that produces $O(e^k)$ colorings for the graph G in which with high probability at least one coloring properly colors the path P. The second is a deterministic algorithm based on the hashing schemes studied by Fredman, Komlos, and Szemeredi [34] and Schmidt and Siegel [87], which constructs a set of $O(c^k n^{O(1)})$ colorings for the graph G in which at least one colors the path P properly, where $c \gg 1$ is a constant. This, plus the above dynamic programming algorithm, gives for the k-PATH problem a randomized algorithm of running time $O((2e)^k n^{O(1)}) = O(5.44^k n^{O(1)})$ and space $O(2^k kn + m)$ and a deterministic algorithm of running time $O((2c)^k n^{O(1)})$. The currently best randomized and deterministic algorithms for MATCHING and SET PACKING [56, 31] follow the same principle: first color the elements so that no two elements in the subset of interest are colored with the same color, then apply a deterministic algorithm (e.g., dynamic programming) on the set of colored elements to search for the subset.

This method is of great theoretical importance. In particular, it confirms Papadimitriou and Yannakakis's conjecture that the $(\log n)$-PATH problem can be solved in polynomial time. On the other hand, both the time complexity and the space complexity are quite high. It can be verified that for the deterministic algorithm of time $O((2c)^k kn^{O(1)})$ for the k-PATH problem described in [1], the constant c is at least 4000 (a more detailed analysis on the algorithm will be given in later discussions). In consequence, the deterministic algorithm proposed in [1] has running time of the form $O(d^k n^{O(1)})$, where $d > 8000$. Obviously, such an algorithm will quickly become impractical even for very small values of k. Moreover, the space complexity of all the randomized algorithms described above for PATH, MATCHING, and PACKING is exponential in k, which is also remarkable.

In this chapter, we study new techniques to develop improved randomized and deterministic algorithms for the PATH, MATCHING, and PACKING problems. Our first result is a randomized divide-and-conquer method. Roughly speaking, suppose that we are looking for a subset S_k of k elements in a universal set S. We first randomly partition the set S into two parts, then recursively look for a subset of $k/2$ elements in each part. This simple method leads directly to improved randomized algorithms. For the k-PATH problem, this new method gives a randomized algorithm of time $O(4^k k^{3.42} m)$ and space $O(nk \log k + m)$, improving the previous best randomized algorithm for the problem of time $O(5.44^k km)$ and space $O(2^k kn + m)$ [1]. For the 3-D MATCHING and 3-SET PACKING problems, the method gives randomized algorithms of time $O(2.52^{3k} n)$ and space $(nk \log k + m)$, improving the previous best randomized algorithms for the problems of time $O(10.88^{3k} n^{O(1)})$ and space $O(2^{3k} + m)$ [56].

To develop improved deterministic algorithms for the PATH, MATCHING, and

PACKING problems, we study the complexity of *k-color coding schemes*. A *k-coloring* of a set S is a mapping from S to $\{0, \dots, k-1\}$. A collection $\mathcal{C}$ of k-colorings of S is a *k-color coding scheme* for S if for every subset S_k of k elements in S there is a k-coloring F in $\mathcal{C}$ such that no two elements in S_k are colored with the same color under F. Denote by $\tau(n, k)$ the minimum size of a k-color coding scheme for a set of n elements. We study upper bounds for the function $\tau(n, k)$. To improve the upper bound for $\tau(n, k)$, we propose a new four-level hashing procedure that uses the techniques of kernelization, collision minimization, and recursive construction. The kernelization technique used in the first level of the procedure is based on the hashing function studied in [34]. The next two levels in the procedure use a hashing function that is nearly optimal in terms of collision minimization. The last level of the procedure uses a non-trivial recursive formula for $\tau(n, k)$ and performs careful constructions of k-color coding schemes for small values of k. These techniques induce a new k-color coding scheme of size $O(6.4^k n)$. This is much better than the previous upper bound for $\tau(n, k)$, which is larger than 4000^k [1].

The improved upper bound on $\tau(n, k)$ directly induces significant improvements on deterministic algorithms for the PATH, MATCHING, and PACKING problems. A comparison of the previous best algorithms and our improved algorithms is given in Table XIV.

We make a few remarks on our results before proceeding to technical discussions. Many NP-hard problems are concerned with searching for a subset S_k of k elements in a given set S such that the subset S_k satisfies certain properties. The k-color coding schemes seem to provide a convenient "pre-classification" of the elements in S so that all elements in the subset S_k are colored with distinct colors, which will significantly narrow down the search space for the problem. From this point of view, the study of k-color coding schemes seems to be of general interest in solving NP-hard problems.

Table XIV. Comparison of deterministic algorithms (each entry t denotes a running time of $O(tn^{O(1)})$).

Reference	k-path	3-D match	3-set pack	triangle pack
[1]	$> 8000^k$			
[56]		$> 32000^{3k}$	$> 32000^{3k}$	
[31]*		$(12.7D)^{3k}$	$(12.7D)^{3k}$	$(12.7D)^{3k}$
Ours	12.8^k	12.8^{3k}	12.8^{3k}	12.8^{3k}
Ours+[66]		2.77^{3k}	4.61^{3k}	4.61^{3k}

* Fellows et al. [31] developed a $13k$-color coding scheme of size 12.7^{3k}, and left the remaining procedures for the algorithms unspecified. Here we use $O(D^{3k})$ to denote the complexity of searching for a subset of $3k$ symbols in a set colored with $13k$ colors. A straightforward implementation of this process will have $D \geq 10.4$.

We also remark on how significant our improvement of the upper bound for $\tau(n,k)$ is. All the functions 4^k, 5.44^k, 6.4^k, and 4000^k are of the form $2^{O(k)}$. However, these functions are not linearly related. In fact, for algorithms whose running time is of the form $O(c^k n^{O(1)})$, where c^k is the dominating term (as is the case for many NP-hard problems), a small reduction on the constant c will result in a significant improvement on the running time. In particular, 4000^k is larger than the fourth power of 6.4^k.

B. Randomized divide-and-conquer

In this section, we describe a new randomized divide-and-conquer method, which will induce improved randomized algorithms for a number of PATH, MATCHING, and PACKING problems. To make our discussion more specific, we will describe the method in detail based on the k-PATH problem. We then explain briefly how the method is applied to MATCHING and PACKING problems. Throughout this chapter, we will denote by $e = 2.718\cdots$ the base of the natural logarithm.

The randomized algorithm **find-paths** for k-PATH is given in Fig. 37. A simple path in a graph G is a (u, k)*-path* if it contains exactly k vertices and if one end of the path is u. In particular, for any vertex u, a $(u, 1)$-path consists of a single vertex u. When the vertex u is irrelevant, a (u, k)-path will be simply called a k*-path.* Our algorithm **find-paths**(G', P', k) on the subgraph G' and a set P' of k'-paths (where no vertex on the paths in P' is in G') returns a set P of paths, each is a concatenation of a k'-path in P' and a k-path in G' (if no such paths exist, the algorithm returns an empty set). In particular, the algorithm **find-paths**$(G', \emptyset, k)$ returns a set of k-paths in the graph G'.

Lemma VI.1 *For integer* $k > 1$, $\lceil \log k \rceil = \lceil \log(\lceil k/2 \rceil) \rceil + 1$.

Theorem VI.2 *On a graph* $G = (V, E)$ *with* n *vertices and* m *edges and an integer* $k \geq 1$, *if the graph* G *contains a* (u, k)*-path for a vertex* u, *then with probability larger than* $1 - 1/e > 0.632$, *the set* P *returned by the algorithm* **find-paths**$(G, \emptyset, k)$ *contains a* (u, k)*-path. The algorithm* **find-paths**$(G, \emptyset, k)$ *runs in time* $O(4^k k^{3.42} m)$ *and in space* $O(nk \log k + m)$.

Proof. To prove the first part, we prove the following claims using induction on k:

1. If $P' = \emptyset$ and G' has a (u, k)-path, then with probability larger than $1 - 1/e$, the set P returned by the algorithm **find-paths**(G', P', k) includes a (u, k)-path.
2. If $P' \neq \emptyset$ and G' has a (u, k)-path whose other end is connected to an end vertex of a path in P', then with probability larger than $1 - 1/e$, the set P returned by the algorithm **find-paths**(G', P', k) contains a $(u, k' + k)$-path.

The claims are obviously true for $k = 1$. Let $k > 1$. First consider the case when

```
Input: $G'$: a subgraph of $G$, $P'$ a set of $k'$-paths in $G$ that contains no vertex in $G'$,
    an integer $k \geq 1$;
Output: a set $P$ of paths, each a concatenation of a $k'$-path in $P'$ and a $k$-path in
    $G'$
$P \leftarrow \emptyset$
if k=1 then
    if $P' \neq \emptyset$ then
        return all 1-paths in $G'$
    else
        for each $(u, k')$-path $p$ in $P'$ and each vertex $w$ in $G'$ do
            if $(u, w)$ is an edge in $G$ then
                concatenate $p$ and $w$ to make a $(w, k' + 1)$-path $p'$
                add $p'$ into $P$ if no $(w, k' + 1)$-path is in $P$
            end if
        end for
    end if
    return $P$
end if
for $2.51 \cdot 2^k$ times do
    randomly partition the vertices of $G$ into $V_L$ and $V_R$
    let $G_L$ and $G_R$ be the subgraphs induced by $V_L$ and $V_R$, respectively
    $P_L \leftarrow$find-paths($G_L, P', k - \lceil k/w \rceil$)
    if $P_L \neq \emptyset$ then
        $P_R \leftarrow$find-paths($G_R, P_L, k - \lceil k/w \rceil$)
        for each $(u, k' + k)$-path $p$ in $P_R$ do
            add $p$ to $P$ if no $(u, k' + k)$-path is in $P$
        end for
    end if
end for
return $P$
```

Fig. 37. Algorithm **find-paths**: extend a set of k'-paths to longer length.

$P' = \emptyset$. Suppose that

$$[u_1, u_2, \ldots, u_{k_1}, u_{k_1+1}, \ldots, u_k]$$

is a (u_k, k)-path in G', where $k_1 = \lceil k/2 \rceil$. Then with probability $1/2^k$, step 3.1 of the algorithm puts vertices $u_1, u_2, \ldots, u_{k_1}$ into V_L, and vertices $u_{k_1+1}, \ldots, u_k$ into V_R. If this is the case, then the graph G_L contains the (u_{k_1}, k_1)-path $[u_1, \ldots, u_{k_1}]$, and the graph G_R contains the $(u_k, k-k_1)$-path $[u_{k_1+1}, \ldots, u_k]$. By the inductive hypothesis, with probability larger than $1-1/e$, P_L obtained from step 3.3 includes a (u_{k_1}, k_1)-path. The $(u_k, k-k_1)$-path $[u_{k_1+1}, \ldots, u_k]$ in G_R has its other end u_{k_1+1} connected to the (u_{k_1}, k_1)-path in P_L. Therefore with probability larger than $1-1/e$, P_R obtained in step 3.5 contains a path of length $k_1 + (k - k_1) = k$ that ends with u_k, i.e., a (u_k, k)-path. Therefore in each loop of step 3, the probability ρ that a (u_k, k)-path is added to the set P is larger than

$$\frac{(1-1/e)^2}{2^k} > \frac{0.6322}{2^k} > \frac{1}{2.51 \cdot 2^k}.$$

When $P' \neq \emptyset$, we follow the same argument as before except that we require that the (u_k, k)-path in G' its other end connected to the end of a k'-path in P'. So P_L contains a $(u_{k_1}, k'+k_1)$-path p that is a concatenation of a k'-path in P' and a k_1-path in G_L, and P_R contains a $(u_k, k'+k)$-path that is a concatenation of a $(k'+k_1)$-path in P_L and a $(k-k_1)$-path in G_R.

Since step 3 of the algorithm loops $2.51 \cdot 2^k$ times, the overall probability that the algorithm returns a set of paths that contains a (u_k, k)-path (when the set P' is empty) or a $(u_k, k'+k)$-path (when the set P' is not empty) is

$$1 - (1-\rho)^{2.51 \cdot 2^k} > 1 - \left(1 - \frac{1}{2.51 \cdot 2^k}\right)^{2.51 \cdot 2^k} > 1 - \frac{1}{e}.$$

This proves the first part of the theorem.

To analyze the time complexity, let $T(k)$ be the running time of the algorithm **find-paths**(G', P', k). Without loss of generality, we can assume that $m \geq n$. From the algorithm, we get the following recurrence relation:

$$T(k) = 2.51 \cdot 2^k (cm + T(\lceil k/2 \rceil) + T(k - \lceil k/2 \rceil)),$$

where $c > 0$ is a constant. We claim that for all $k > 0$,

$$T(k) \leq c \cdot (10.7)^{\lceil \log k \rceil} 2^{2k} m, \tag{6.1}$$

and we prove it by induction on k. Obviously $T(1) \leq cm$ if c is sufficiently large, so inequality (6.1) holds for $k = 1$. Let $k > 1$, then

$$\begin{aligned}
T(k) &= 2.51 \cdot 2^k \left(cm + T(\lceil k/2 \rceil) + T(k - \lceil k/2 \rceil)\right) \\
&\leq 2.51 \cdot 2^k \left(cm + 2c \cdot (10.7)^{\lceil \log \lceil k/2 \rceil \rceil} m 2^{2\lceil k/2 \rceil}\right) \\
&\leq 2.51 \cdot 2^k \left(cm + \frac{2c}{10.7} \cdot (10.7)^{\lceil \log \lceil k/2 \rceil \rceil + 1} m 2^{k+1}\right) \\
&= c \cdot (10.7)^{\lceil \log k \rceil} 2^{2k} m \cdot 2.51 \left(\frac{1}{10.7^{\lceil \log k \rceil} 2^k} + \frac{4}{10.7}\right) \\
&\leq c \cdot (10.7)^{\lceil \log k \rceil} 2^{2k} m \cdot 2.51 \left(\frac{1}{10.7 \cdot 4} + \frac{4}{10.7}\right) \\
&< c \cdot (10.7)^{\lceil \log k \rceil} 2^{2k} m.
\end{aligned}$$

Here in the second step of the above derivation, we have used $k - \lceil k/2 \rceil \leq \lceil k/2 \rceil$. In the third step, we have used $2\lceil k/2 \rceil \leq k + 1$, and in the fourth step we have used Lemma VI.1. Thus the running time $T(k)$ of the algorithm **find-paths**$(G, \emptyset, k)$ is $O((10.7)^{\lceil \log k \rceil} 2^{2k} m) = O(4^k k^{3.42} m)$.

In terms of the space complexity, each recursive call to the algorithm **find-paths** uses $O(nk)$ space (mainly for the sets P_L, P_R, and P). Since on a graph G and an integer k, the recursive depth of the algorithm is $O(\log k)$, we conclude that the space complexity of the algorithm **find-paths**$(G, \emptyset, k)$ is $O(nk \log k + m)$. □

Using Theorem VI.2, we can develop $O(4^k k^{3.42} m)$ time randomized algorithms of arbitrarily small error bound for the k-PATH problem. For example, to achieve an error bound of 0.0001, we can run the algorithm in Theorem VI.2 t times, where t satisfies $(1/e)^t \leq 0.0001$ (e.g., $t = 10$).

We compare our algorithm in Theorem VI.2 with previously known algorithms for the k-PATH problem. To our knowledge, there are two kinds of randomized algorithms for the k-PATH problem. The first kind is based on random permutations of vertices followed by searching in a directed acyclic graph (see [1, 52] for details). The algorithm runs in time $O(mk!)$ and space $O(m)$. The second kind, proposed by Alon, Yuster, and Zwick [1], is based on random coloring of vertices followed by dynamic programming to search for a k-path in the colored graph. The algorithm runs in time $O((2e)^k km) = O(5.44^k km)$ and space $O(2^k kn + m)$ (the space is mainly for the dynamic programming phase). Compared to these algorithms, our algorithm has a significantly improved running time and uses polynomial space. In fact, if we only need to know if the graph has a k-path or not, a slight modification of our algorithm can reduce the space complexity to $O(n \log k + m)$.

Remark 1. Recently, Kneis et al. [55] have independently developed a similar randomized algorithm for the problem, whose complexity is slightly worse than ours.

Remark 2. It seems that we have to be more careful when we analyze an exponential time algorithm based on the divide-and-conquer method. Certain common techniques from traditional algorithm analysis do not seem to be directly applicable. For example, we cannot simply assume that the parameter k is a power of 2 since the extension from this special case to the case for general k does not seem to give the same complexity bound. In fact, when k is a power of 2, it is quite trivial to verify (by induction) that $T(k) \leq O(4^k k^{2.52} m)$. However, it seems not easy to extend this bound to the case for general k.

The above randomized divide-and-conquer method can also be used to develop improved algorithms for MATCHING and PACKING problems. Consider the 3-D MATCHING problem (given a set S of triples and an integer k, either find a subset S_k of k disjoint triples or report that no such subset exists). In the case when such a subset S_k exists, let A_k be the set of $3k$ symbols in the triples in S_k. With probability $\binom{k}{k/2}/2^{3k} = O(1/(2^{2k}\sqrt{k}))$, we can partition the symbols in A_k into two subsets A'_k and A''_k such that A'_k contains $3k/2$ symbols in the $k/2$ triples in S_k and A''_k contains $3k/2$ symbols in the other $k/2$ triples in S_k. The set S of triples can be partitioned into two subsets S' and S'' in terms of A'_k and A''_k, and a subset of $k/2$ triples is searched recursively in each of the sets S' and S''. An analysis similar to that in Theorem VI.2 shows that this algorithm runs in time $O(2.52^{3k}n)$ and space $O(nk\log k + m)$ and finds the subset of k triples with high probability. It is straightforward to modify this approach to obtain an algorithm of the same time and space complexity for the 3-SET PACKING problem (given a collection of 3-sets and an integer k, either find a sub-collection of k disjoint 3-sets or report that no such sub-collection exists).

Theorem VI.3 *The* 3-D MATCHING *and* 3-SET PACKING *problems can be solved by randomized algorithms in time $O(2.52^{3k}n)$ and space $O(nk\log k + m)$.*

Note that the previous best randomized algorithms for the problems [56] take time $O(10.88^{3k}n^{O(1)})$ and space $O(2^{3k} + m)$, where the space is exponential in k.

C. A new color coding scheme

The k-color coding scheme is a general used technique. The concept and techniques were first proposed by Alon, Yuster, and Zwick [1], who also showed how to use the techniques to solve a number of important NP-hard problems more effectively.

The complexity of those algorithms depends directly on the size of the k-color coding scheme they used. Any improvement on the size of the scheme will improve algorithms based on color coding (e.g., [1, 31, 56]). In this section, we develop a new k-color coding scheme of size significantly smaller then previous results.

We start from some definitions. Let S be a set and let W be a subset of S. A function f on S is *injective from* W if for any two different elements x and y in W, we have $f(x) \neq f(y)$. For each integer m, denote by Z_m the integer set $\{0, 1, \ldots, m-1\}$. In particular, if m is a prime number, then Z_m is a field under the addition and multiplication modular m.

Definition VI.4 *A k-coloring of a set S is a function from S to Z_k. A collection $\mathcal{F}$ of k-colorings of S is a k-color coding scheme for S if for every subset W of k elements in S, there is a k-coloring f_W in $\mathcal{F}$ that is injective from W. The size of the k-color coding scheme $\mathcal{F}$, noted by $|\mathcal{F}|$, is the number of k-colorings in $\mathcal{F}$.*

1. A special collection of color coding schemes

Let us first look at a few simple lemmas on color coding schemes.

Lemma VI.5 *For any positive integers n and k, $n \geq k$, there is a k-color coding scheme for Z_n of size bounded by $\binom{n}{k}$.*

Proof. For each subset W of k elements in Z_n, construct a k-coloring F_W for Z_n that assigns each element in W a distinct color and colors all other elements in Z_n arbitrarily. The k-coloring F_W is obviously injective from W. The collection $\{F_W \mid W \subseteq Z_n,\ |W| = k\}$ of $\binom{n}{k}$ k-colorings is a k-color coding scheme for the set Z_n. □

Lemma VI.6 *For $n \geq 2$, there exist* (1) *a* 1*-color coding scheme for Z_n of size* 1*;* (2) *an n-color coding scheme for Z_n of size* 1*; and* (3) *a* 2*-color coding scheme for Z_n of size $\leq \lceil \log n \rceil$.*

Proof. Statement (1) and statement (2) are obvious. We prove statement (3) by induction on n. It can be easily verified for the cases of $n \leq 4$ the statement holds. Inductively, suppose that for $n \leq 2^g$, $g \geq 2$, statement (3) is true. Now consider the case $2^g < n \leq 2^{g+1}$.

Partition the n elements in Z_n into two subsets Z' and Z'' such that $|Z'| = 2^g$ and $|Z''| = n - 2^g \leq 2^g$. By the inductive hypothesis, there exist a 2-color coding scheme $\mathcal{F}' = \{F'_1, \ldots, F'_{g'}\}$ of size g' for Z' and a 2-color coding scheme $\mathcal{F}'' = \{F''_1, \ldots, F''_{g''}\}$ of size g'' for Z'', where $g' \leq \lceil \log |Z'| \rceil = g$ and $g'' \leq \lceil \log |Z''| \rceil \leq g$. Without loss of generality, we assume $g' \geq g''$.

Construct g' 2-colorings for Z_n:

$$(F'_1, F''_1), (F'_2, F''_2), \ldots, (F'_{g''}, F''_{g''}), (F'_{g''+1}, F''_{g''}), \ldots, (F'_{g'}, F''_{g''}), \tag{6.2}$$

where the coloring (F'_i, F''_j) colors the elements in Z' using the 2-coloring F'_i, and the elements in Z'' using the 2-coloring F''_j (both F'_i and F''_j use the color set $\{0, 1\}$). We also construct a new 2-coloring F for Z_n that assigns 0 to all elements in Z' and 1 to all elements in Z''.

Let W be any subset of two elements in Z_n. If the two elements in W are both in Z' or both in Z'', then since $\mathcal{F}'$ and $\mathcal{F}''$ are 2-color coding schemes for Z' and Z'', respectively, one of the 2-colorings in (6.2) will assign the two elements in W with different colors. On the other hand, if one element of W is in Z' and the other element of W is in Z'', then the 2-coloring F colors the two elements of W with different colors. In conclusion, the 2-coloring F plus the g' 2-colorings in (6.2) makes a 2-color coding scheme of size $g' + 1 \leq g + 1 = \lceil \log n \rceil$ for the set Z_n. The induction goes through and the lemma is proved. □

For general k, we have the recurrence relation shown in the following lemma.

Lemma VI.7 *Let $n = n_1 + \cdots + n_r$, where all $n_j \geq 1$ are integers. Let $\tau(n', k')$ be an upper bound for the size of a k'-color coding scheme for the set $Z_{n'}$, where $n' < n$ and $k' \leq k$. Then there is a k-color coding scheme for the set Z_n whose size is bounded by*

$$\sum_{0 \leq k_1 \leq n_1, \ldots, 0 \leq k_r \leq n_r}^{k_1 + \cdots + k_r = k} \left(\frac{\tau(\#[k_j \leq 1], \#[k_j = 1])}{\binom{\#[k_j \leq 1]}{\#[k_j = 1]}} \prod_{k_j \geq 2} \tau(n_j, k_j) \right)$$

where $\#[k_j \leq 1]$ and $\#[k_j = 1]$ are the numbers of k_j's in the list $[k_1, \ldots, k_r]$ such that $k_j \leq 1$ and $k_j = 1$, respectively.

Proof. Arbitrarily partition the set Z_n into r disjoint subsets Y_1, …, Y_r, where $|Y_j| = n_j$ for all j. Let L be the collection of all lists $[k_1, \ldots, k_r]$ of r integers satisfying $k_1 + \cdots + k_r = h$ and $0 \leq k_j \leq n_j$ for all j. We say that two lists $[k_1, \ldots, k_r]$ and $[k'_1, \ldots, k'_r]$ in L are *conjugate* if for every j either $k_j \geq 2$ or $k'_j \geq 2$ will imply $k_j = k'_j$. It is clear that this conjugation is an equivalence relation and partitions the lists in L into equivalence classes. A conjugation equivalence class will be called a $(k_1, \ldots, k_r)$-class for any list $[k_1, \ldots, k_r]$ in the class. Each $(k_1, \ldots, k_r)$-class contains exactly $\binom{\#[k_j \leq 1]}{\#[k_j = 1]}$ lists in L: this is because when the values and positions for all $k_j \geq 2$ are fixed in $[k_1, \ldots, k_r]$, there are exactly $\binom{\#[k_j \leq 1]}{\#[k_j = 1]}$ ways to determine the $\#[k_j = 1]$ "1"s in the remaining $\#[k_j \leq 1]$ positions in $[k_1, \ldots, k_r]$.

Fix a $(k_1, \ldots, k_r)$-class. For each j such that $k_j \geq 2$, let $\mathcal{F}_{n_j, k_j}$ be a k_j-color coding scheme of size bounded by $\tau(n_j, k_j)$ for the set Z_{n_j}. Moreover, let $\mathcal{F}$ be a $(\#[k_j = 1])$-color coding scheme of size bounded by $\tau(\#[k_j \leq 1], \#[k_j = 1])$ for the set $Z_{\#[k_j \leq 1]}$. Let the color sets used by all these schemes be disjoint. We construct a set of at most

$$\tau(\#[k_j \leq 1], \#[k_j = 1]) \prod_{k_j \geq 2} \tau(n_j, k_j)$$

k-colorings for the set Z_n: each of these k-colorings consists of a k_j-coloring from the scheme $\mathcal{F}_{n_j,k_j}$ for the set Y_j for each $k_j \geq 2$, plus a $(\#[k_j = 1])$-coloring from the scheme $\mathcal{F}$ that treats each set Y_j with $k_j \leq 1$ as a single element and assigns all elements in Y_j with the same color.

We apply the above process to each $(k_1, \ldots, k_r)$-class, which gives a collection of

$$\sum_{0 \leq k_1 \leq n_1, \ldots, 0 \leq k_r \leq n_r}^{k_1 + \cdots + k_r = k} \left(\frac{\tau(\#[k_j \leq 1], \#[k_j = 1])}{\binom{\#[k_j \leq 1]}{\#[k_j = 1]}} \prod_{k_j \geq 2} \tau(n_j, k_j) \right)$$

k-colorings for the set Z_n (note that each $(k_1, \ldots, k_r)$-class contains exactly $\binom{\#[k_j \leq 1]}{\#[k_j = 1]}$ lists in the collection L). To complete the proof of the lemma, it remains to show that this collection makes a k-color coding scheme for the set Z_n.

Let W be an arbitrary subset of k elements in Z_n. Suppose that for each j, W has exactly k_j elements in the set Y_j. Note that $[k_1, \ldots, k_r]$ is a list in the collection L. For each $k_j \geq 2$, since $\mathcal{F}_{n_j,k_j}$ is a k_j-color coding scheme for Y_j, one k_j-coloring F_j in $\mathcal{F}_{n_j,k_j}$ must be injective from the k_j elements of W that are in Y_j. On the other hand, since $\mathcal{F}$ is a $(\#[k_j = 1])$-color coding scheme for the set $Z_{\#[k_j \leq 1]}$, one $(\#[k_j = 1])$-coloring F in $\mathcal{F}$ assigns each of the $\#[k_j = 1]$ sets Y_j with $k_j = 1$ a distinct color. Therefore, the combination of these k_j-colorings F_j and the $(\#[k_j = 1])$-coloring F, which is one of the k-colorings constructed above, makes a k-coloring for the set Z_n that is injective from the subset W. This completes the proof of the lemma. □

By Lemma VI.6 and Lemma VI.7, for small values of n and k, we can construct a k-color coding scheme easily based on Lemma VI.7. We use a computer program to construct color coding schemes for special pairs (n, k) for small values of $n = k(k-1)$ and k. The sizes of the color coding scheme we constructed are given in Table XV (the last column in the table will be used for later discussion).

Table XV. Upper bound on the size of a k-color coding scheme for Z_n

k	$n = k(k-1)$	upper bound $\tau(n,k)$	$B_k = (\tau(n,k))^{4/n}$
2	2	1	1
3	6	3	2.0801
4	12	12	2.2895
5	20	82	2.4142
6	30	434	2.2474
7	42	2,937	2.1394
8	56	16,960	2.0050
9	72	115,251	1.9108
10	90	655,756	1.8136
11	110	4,731,907	1.7488
12	132	33,489,268	1.6906
13	156	260,723,566	1.6437
14	182	1,426,381,707	1.5893
15	210	13,008,846,025	1.5584
16	240	58,465,192,360	1.5117
17	272	676,712,910,839	1.4928
18	306	6,079,615,220,515	1.4693

2. A k-coloring algorithm with a parameter set

Assuming that we have two integers n, k where $n \geq k$, we introduce a k-coloring algorithm of the set Z_n in this subsection and prove that the algorithm actully constructs a k-color coding scheme later.

Let p_0 and p be prime numbers satisfying $n \leq p_0 < 2n$ and $k^2 < p < 2k^2$ (such prime numbers exist by Bertrand's Conjecture [48]). The prime numbers p_0 and p can be obtained in time $O(n\sqrt{n})$ and $O(k^2\sqrt{k^2}) = O(k^3)$, respectively, using a trivial primality testing algorithm.

We present a k-coloring algorithm for the set Z_n. The algorithm is associated with a set of parameters satisfying the following properties:

C0. an integer a_0, where $0 \leq a_0 \leq p_0 - 1$;

C1. a pair of integers (a, b), where $0 < a \leq p-1$, $0 \leq b \leq p-1$;

C2. an ordered list $C = [c_0, c_1, \ldots, c_{k'}]$ of non-negative integers, where $k' = k/4 - 1$, $\sum_{j=0}^{k'} c_j = k$, and $\sum_{j=0}^{k'} c_j(c_j - 1) \leq 4k$. Let $C_{>1}$ be the sublist of C by removing all $c_j \leq 1$;

C3. an ordered list $L = [(a_1, b_1), (a_2, b_2), \ldots, (a_r, b_r)]$ of pairs of integers, where $0 < a_i \leq p-1$, $0 \leq b_i \leq p-1$, and $r \leq \log |C_{>1}|$;

C4. a mapping from the elements in the list $C_{>1}$ to the elements in the list L such that at least half of the c_j's in $C_{>1}$ are mapped to (a_1, b_1), at least half of the c_j's that are not mapped to (a_1, b_1) are mapped to (a_2, b_2), at least half of the c_j's that are not mapped to (a_1, b_1) and (a_2, b_2) are mapped to (a_3, b_3), and so on.

C5. an ordered list of colorings $[F_{c_0}, F_{c_1}, \ldots, F_{c_{k'}}]$, where for each $c_j \geq 2$, F_{c_j} is a c_j-coloring from the c_j-color coding scheme $\mathcal{F}_{c_j}$ for $Z_{c_j(c_j-1)}$ given in Lemma VI.8 (for $c_j \leq 1$, F_{c_j} is irrelevant).

We also define two functions as follows. For an integer m, let p_m be the smallest prime number such that $m \leq p_m < 2m$. For two given integers s and a, where $1 < s < m$ and $0 \leq a \leq p_m - 1$, we define a function $\psi_{a,s}$ from Z_m to Z_s by

$$\psi_{a,s}(x) = (ax \bmod p_m) \bmod s, \tag{6.3}$$

and for three given integers s, a, b, where $1 < s < m$, $0 < a \leq p_m - 1$, and $0 \leq b \leq p_m - 1$, we define a function $\phi_{a,b,s}$ from the set Z_m to the set Z_s by

$$\phi_{a,b,s}(x) = ((ax + b) \bmod p_m) \bmod s. \tag{6.4}$$

Our k-coloring algorithm on the set Z_n is given in Fig. 38.

```
Input: $n, k$ and parameters as specified in C1–C5;
Output: a $k$-coloring of the set $Z_n$;
 1: for $j = 0$ to $k' = k/4 - 1$ do
 2:   $U_j = \{x \mid x \in Z_n,\ \phi_{a,b,k/4}(x) = j\}$;
 3: end for
 4: for each $U_j$ such that $c_j > 1$ do
 5:   suppose that $c_j$ is mapped to $(a_i, b_i)$
 6:   for $t = 0$ to $c_j(c_j - 1) - 1$ do
 7:     $U_{j,t} = \{x \mid x \in U_j,\ \phi_{a_i,b_i,c_j(c_j-1)}(x) = t\}$;
 8:     create $c_j$ new colors $\tau_{j,0}, \tau_{j,1}, \ldots, \tau_{j,c_j-1}$;
 9:     assign all elements in $U_{j,t}$ with color $\tau_{j,s}$ if the $c_j$-coloring $F_{c_j}$ for $Z_{c_j(c_j-1)}$
        assigns color $s$ to the element $t$
10:   end for
11: end for
12: for each $c_j = 1$ do
13:   create a new color $\tau_j$ and assign all elements in $U_j$ the color $\tau_j$
14: end for
15: assign all elements in $\bigcup_{c_j=0} U_j$ arbitrarily using the colors created in steps 2–3
```

Fig. 38. Algorithm **Coloring**: construct a color coding scheme.

We make some remarks on the algorithm **Coloring**:

(1) the function ψ_{a_0,k^2} in step 0 is from Z_n to Z_{k^2}, which implies that $\bar{x} \in Z_{k^2}$;

(2) the function $\phi_{a,b,k/4}$ in step 1 is from Z_{k^2} to $Z_{k/4}$, which implies that $\phi_{a,b,k/4}(\bar{x}) \in Z_{k/4}$;

(3) for each j such that $c_j > 1$, the function $\phi_{a_i,b_i,c_j(c_j-1)}$ in step 2 is from Z_{k^2} to $Z_{c_j(c_j-1)}$,

which implies that $\phi_{a_i,b_i,c_j(c_j-1)}(\bar{x}) \in Z_{c_j(c_j-1)}$;

(4) by steps 2-3, for each $c_j \geq 1$, we create c_j new colors. By **C2**, $\sum_{j=0}^{k/4-1} c_j = k$. Therefore,

the total number of colors used by the algorithm **Coloring** is exactly k. In consequence,

the algorithm produces a k-coloring for the set Z_n.

For each given set of parameters satisfying the conditions **C0-C5**, the algorithm

Coloring produces a k-coloring for the set Z_n. A collection $\mathcal{F}$ of k-colorings of Z_n can be obtained by running over all possible sets of parameters that satisfy the conditions. We first consider the size of this collection $\mathcal{F}$.

3. The size of the collection $\mathcal{F}$

To get an upper bound for the size of $\mathcal{F}$, we will need the following lemma.

Lemma VI.8 *Let $[c_0, c_1, \ldots, c_r]$ be a list of non-negative integers such that $\sum_{j=0}^{r} c_j = k$ and $\sum_{j=0}^{r} c_j(c_j - 1) \leq 4k$. Then there is a collection $\{\mathcal{F}_{c_0}, \mathcal{F}_{c_1}, \ldots, \mathcal{F}_{c_r}\}$ of color coding schemes, where $\mathcal{F}_{c_j}$ is a c_j-color coding scheme for the set $Z_{c_j(c_j-1)}$, such that $\prod_{c_j \geq 2} |\mathcal{F}_{c_j}| \leq 2.4142^k$.*

Proof. For $2 \leq c_j \leq 18$, we use the c_j-color coding scheme $\mathcal{F}_{c_j}$ for the set $Z_{c_j(c_j-1)}$ given in Table XV, whose size is bounded by $\tau(c_j(c_j - 1), c_j)$ in the third column of the table. For $c_j > 18$, we simply use the trivial c_j-color coding scheme $\mathcal{F}_{c_j}$ for the set $Z_{c_j(c_j-1)}$ given in Lemma VI.5, whose size is bounded by $\tau(c_j(c_j - 1), c_j) = \binom{c_j(c_j-1)}{c_j}$.

Let $B_{c_j} = (\tau(c_j(c_j - 1), c_j))^{4/c_j(c_j-1)}$. It is easy to verify from the table in Table XV that $B_{c_j} \leq 2.4142$ for $c_j \leq 18$ (see the fourth column in the table).

Now let $c_j > 18$. By our definition,

$$B_{c_j} = (\tau(c_j(c_j - 1), c_j))^{4/c_j(c_j-1)} = \binom{c_j(c_j - 1)}{c_j}^{4/c_j(c_j-1)}.$$

Consider

$$\begin{aligned}
f(x) &= \binom{x(x-1)}{x}^{\frac{1}{x(x-1)}} \\
&= \left[\frac{[x(x-1)][x(x-1)-1]\cdots[x(x-1)-x+1]}{x!}\right]^{\frac{1}{x(x-1)}} \\
&\leq \left[\frac{[x(x-1)-\frac{x-1}{2}]^x}{\sqrt{2\pi x}(x/e)^x}\right]^{\frac{1}{x(x-1)}} \\
&< \left[\frac{(\frac{2x(x-1)-(x-1)}{2})^x}{(x/e)^x}\right]^{\frac{1}{x(x-1)}} \\
&= \left(\frac{e(2x-1)(x-1)}{2x}\right)^{\frac{1}{x-1}},
\end{aligned}$$

where in the first inequality, we have used the inequalities $ab \leq [(a+b)/2]^2$ and $x! \geq \sqrt{2\pi x}(x/e)^x$).

Let $g(x) = [e(2x-1)(x-1)/(2x)]^{1/(x-1)}$. It can be verified that when $x \geq 7$, $g(x)$ is strictly decreasing. In particular, for $c_j \geq 19$, we have

$$B_{c_j} = (f(c_j))^4 < (g(c_j))^4 \leq (g(19))^4 = 2.3599.$$

Thus, $B_{c_j} \leq 2.4142$ for all c_j. Combining this with $\sum_{j=0}^r c_j(c_j-1) \leq 4k$, we obtain

$$\begin{aligned}
&\prod_{c_j\geq 2} |\mathcal{F}_{c_j}| \leq \prod_{c_j\geq 2} \tau(c_j(c_j-1), c_j) = \prod_{c_j\geq 2} B_{c_j}^{c_j(c_j-1)/4} \leq \prod_{c_j\geq 2} 2.4142^{c_j(c_j-1)/4} \\
= \;& 2.4142^{\sum_{c_j\geq 2} c_j(c_j-1)/4} = 2.4142^{\sum_{j=0}^r c_j(c_j-1)/4} \leq 2.4142^k
\end{aligned}$$

This completes the proof of the lemma. □

To derive an upper for the size of the collection $\mathcal{F}$ we constructed in the previous subsection, we discuss the number of possible combinations of the parameters satisfying the conditions **C0**-**C5**.

Condition **C0**: the parameter a_0 satisfies $0 \leq a_0 \leq p_0 - 1$, where $p_0 < 2n$. Therefore, there are at most $2n - 1 = O(n)$ possible values for the parameter a_0.

Condition **C1**: the parameters a and b satisfy $0 < a \leq p - 1$ and $0 \leq b \leq p - 1$, where $p < 2k^2$. Therefore, there are at most $O(k^4)$ pairs of integers (a, b) satisfying condition **C1**.

Condition **C2**: we represent each list $C = [c_0, \ldots, c_{k'}]$ satisfying condition **C2** using a single binary string B_C of length $5k/4 - 1$ in which there are exactly $k/4 - 1$ 0-bits. The $k/4 - 1$ 0-bits in B_C divide B_C into $k/4$ "segments" such that the j-th segment contains exactly c_j 1-bits (in particular, the segment between two consecutive 0's in B_C corresponds to a $c_j = 0$). It is easy to verify that any list C satisfying condition **C2** is uniquely represented by such a binary string B_C. Note that the number of binary strings of length $5k/4 - 1$ with exactly $k/4 - 1$ 0-bits is equal to $\binom{5k/4-1}{k/4-1} \leq 1.8692^k$. We conclude that the total number of different lists satisfying condition **C2** is bounded by 1.8692^k. Note that all these lists can be systematically enumerated based on the binary string representation described above.

Condition **C3**: since $r \leq \log(|C_{>1}|) \leq \log(k/4) = \log k - 2$, there are at most $\log k - 2$ pairs in each list L satisfying condition **C3**. By condition **C3**, there are $O(p^2) = O(k^4)$ possible pairs for each (a_i, b_i). Thus, the total number of lists L satisfying condition **C3** is $O(k^{4\log k - 8})$.

Condition **C4**: now we discuss the case when a list $C = [c_0, \ldots, c_{k'}]$ satisfying condition **C2** and a list $L = [(a_1, b_1), \ldots, (a_r, b_r)]$ satisfying condition **C3** are given, how many different mappings from $C_{>1}$ to L can be there that satisfy condition **C4**. Let $q = |C_{>1}| \leq k/4$. We use a binary string $A_{>1}$ to represent a mapping from $C_{>1}$ to L, as follows. The binary string $A_{>1}$ has q 0-bits, which divide $A_{>1}$ into q segments, each starting with a 0-bit. For each j, the j-th segment of form 01^{i-1} in $A_{>1}$ represents the mapping from the j-th element in $C_{>1}$ to the integer pair (a_i, b_i)

in L. By Condition **C4**, at least half of the segments in $A_{>1}$ have no 1-bit, at least half of the remaining segments in $A_{>1}$ have the form 01, and at least half of the remaining segments that are not of the form 0 or 01 in $A_{>1}$ have the form 011, and so on. Therefore, the length of the binary string $A_{>1}$ is bounded by

$$\frac{q}{2} + 2\frac{q}{2^2} + 3\frac{q}{2^3} + \cdots < 2q \leq \frac{k}{2}$$

In consequence, the number of different mappings from the list $C_{>1}$ to the list L satisfying condition **C4** is bounded by $2^{k/2} = 1.4143^k$.

Condition **C5**: in the list $[F_{c_0}, F_{c_1}, \ldots, F_{c_{k'}}]$ of colorings, for each $c_j \geq 2$, F_{c_j} is a c_j-coloring from the c_j-color coding scheme $\mathcal{F}_{c_j}$ for $Z_{c_j(c_j-1)}$ given in Lemma VI.8. Moreover, by Lemma VI.6, for $c_j = 1$, we have $|\mathcal{F}_{c_j}| = 1$. Therefore, the total number of different lists $[F_{c_0}, F_{c_1}, \ldots, F_{c'_k}]$ satisfying condition **C5** is equal to $\prod_{j=0}^{k'} |\mathcal{F}_{c_j}| = \prod_{c_j \geq 2} |\mathcal{F}_{c_j}|$, which, by Lemma VI.8, is bounded by 2.4142^k (note that the list $[c_0, c_1, \ldots, c_{k'}]$ satisfies condition **C2**).

Summarizing the above discussion, we have the following theorem.

Theorem VI.9 *Running the algorithm* **Coloring** *in Fig. 38 over all possible parameters satisfying the conditions* **C0-C5** *gives a collection $\mathcal{F}$ of $O(6.383^k k^{\log k - 4} n)$ k-colorings for the set Z_n. These k-colorings can be constructed in time $O(6.383^k k^{\log k - 4} n^2)$.*

Proof. By the above analysis, the total number of possible combinations of the parameters satisfying the conditions **C0-C5** is bounded by

$$O(n) \cdot O(k^4) \cdot 1.8692^k \cdot O(k^{4\log k - 8}) \cdot 1.4143^k \cdot 2.4142^k = O(6.383^k k^{\log k - 4} n^2)$$

From the above discussion, these k-colorings can be constructed systematically. Since each k-coloring of the set Z_n can be printed in time $O(n)$, the collection $\mathcal{F}$ can be constructed in time $O(6.383^k k^{\log k - 4} n^2)$. □

4. The collection $\mathcal{F}$ makes a k-color coding scheme for Z_n

We derive in this subsection that the collection $\mathcal{F}$ of k-colorings for the set Z_n in Theorem VI.9 is a k-color coding scheme for the set Z_n. We show that for any subset $W = \{w_1, \ldots, w_k\}$ of k elements in Z_n, there is a combination of parameters satisfying the conditions **C0-C5** on which the algorithm **Coloring** produces a k-coloring for the set Z_n that is injective from W.

First let us consider the function $\psi_{a,s}$ used in step 0 of the algorithm **Coloring**. The function $\psi_{a,s}$ has been studied in the construction of universal hashing functions [34, 87]. It was first suggested by Alon, Yuster, and Zwich for the implementation of k-color coding schemes [1]. One of the key properties of the function is summarized in the following theorem.

Theorem VI.10 ([34]) *For the subset W of k elements in Z_n, there is an integer a_0, $0 \leq a_0 \leq p_0 - 1$, such that the function ψ_{a_0,k^2} is injective from W.*

Therefore, there is an integer a_0 satisfying the condition **C0** such that if we let $\overline{w}_i = \psi_{a_0,k^2}(w_i)$ for all $1 \leq i \leq k$, then the set $\overline{W} = \{\overline{w}_1, \overline{w}_2, \ldots, \overline{w}_k\}$ is a subset of k elements in Z_{k^2}. For each $1 \leq i \leq k$, define a subset W_i of Z_n by $W_i = \{x \mid x \in Z_n \ \& \ \psi_{a_0,k^2}(x) = \overline{w}_i\}$. Let $\mathcal{W} = \{W_1, \ldots, W_k\}$ be the collection of these subsets.

Now consider the function $\phi_{a,b,s}$, which has been studied by Carter and Wegman [13]. We first study some basic properties of the function.

Consider the following two sets of ordered pairs of integers (recall that p is a prime number satisfying $k^2 < p < 2k^2$):

$$\begin{aligned} F_1(p) &= \{(a,b) \mid 0 < a \le p-1 \text{ and } 0 \le b \le p-1\} \\ F_2(p) &= \{(r,q) \mid 0 \le r, q \le p-1 \text{ and } r \ne q\} \end{aligned}$$

Fix two distinct integers x and y, $0 \le x, y \le p-1$, we construct a mapping as follows:

$$\pi : \quad (a,b) \longrightarrow ((ax+b) \bmod p, \quad (ay+b) \bmod p)$$

Lemma VI.11 *For any two integers x and y such that $0 \le x, y \le p-1$ and $x \ne y$, the mapping π is a one-to-one mapping from $F_1(p)$ to $F_2(p)$.*

Proof. Since p is a prime number, the set Z_p is a field in terms of the additions and multiplications modular p. For a pair (a,b) in $F_1(p)$, from $(ax+b) \bmod p = (ay+b) \bmod p$, we would get $x = y$ (recall that p is a prime and $a \ne 0$). Therefore, the mapping π maps each element in $F_1(p)$ to an element in $F_2(p)$. Moreover, for a pair (r,q) in $F_2(p)$, where $r \ne q$, the linear equations $(ax+b) \bmod p = r$ and $((ay+b) \bmod p = q$ have a unique solution (a,b), where $a, b \in Z_p$ and $a \ne 0$, i.e., $(a,b) \in F_1(p)$. The lemma now follows directly from the fact that both sets $F_1(p)$ and $F_2(p)$ have exactly $p(p-1)$ elements. □

Let $0 < a \le p-1$, $0 \le b \le p-1$, and $1 < s \le k^2$. For the subset $\overline{W}$ of k elements in Z_{k^2} and for each integer j, $0 \le j \le s-1$, denote by $B(a,b,s,\overline{W},j)$ the number of integers x in $\overline{W}$ such that $\phi_{a,b,s}(x) = j$. We have the following lemma.

Lemma VI.12 *Suppose $p \bmod s = h$. Then for the subset $\overline{W}$ of k elements in Z_{k^2}, we have*

$$\sum_{(a,b)\in F_1(p)} \sum_{j=0}^{s-1} \binom{B(a,b,s,\overline{W},j)}{2} = \frac{k(k-1)(p-h)(p-(s-h))}{2s} \tag{6.5}$$

Proof. Let $p = gs+h$, where g and h are integers. Then $Z_p = \{0, 1, \ldots, gs+h-1\}$. Let

$$A_0 = \sum_{(a,b)\in F_1(p)} \sum_{j=0}^{s-1} \binom{B(a,b,s,\overline{W},j)}{2} = \sum_{j=0}^{s-1} \sum_{(a,b)\in F_1(p)} \binom{B(a,b,s,\overline{W},j)}{2}$$

The value A_0 is equal to the number of different ways of picking an ordered pair (a,b) in $F_1(p)$, and two different elements x and y in $\overline{W}$ such that $\phi_{a,b,s}(x) = \phi_{a,b,s}(y)$, or equivalently

$$((ax+b) \bmod p) \bmod s = ((ay+b) \bmod p) \bmod s.$$

By Lemma VI.11, for two different elements x and y in $\overline{W}$, the mapping

$$\pi : \quad (a,b) \longrightarrow ((ax+b) \bmod p, \quad (ay+b) \bmod p)$$

is a one-to-one mapping from $F_1(p)$ to $F_2(p)$. Therefore, the value A_0 is also equal to the number of different ways of picking an ordered pair (r,q) in $F_2(p)$ and two different elements x and y in W such that

$$r \bmod s = q \bmod s.$$

The number of different ways to pick two different elements x and y in $\overline{W}$ is equal to $k(k-1)/2$. Therefore, the value A_0 is equal to $k(k-1)/2$ times the number of different ways of picking an ordered pair (r,q) in $F_2(p)$ such that $r \bmod s = q \bmod s$.

For each j, if $0 \le j \le h-1$, then there are $g+1$ elements q in Z_p such that $q \bmod s = j$; while if $h \le j \le s-1$, then there are g elements q in Z_p such that $q \bmod s = j$. Therefore, for each j, $0 \le j \le h-1$, there are $g(g+1)$ ordered pairs (r,q) in $F_2(p)$ such that $r \bmod s = q \bmod s = j$ while for each j, $h \le j \le s-1$, there are

$g(g-1)$ ordered pairs (r, q) in $F_2(p)$ such that $r \bmod s = q \bmod s = j$. In summary, there are totally $hg(g+1) + (s-h)g(g-1) = g(p-(s-h)) = (p-h)(p-(s-h))/s$ ordered pairs (r, q) in $F_2(p)$ such that $r \bmod s = q \bmod s$.

Therefore, we have proved

$$A_0 = \sum_{(a,b) \in F_1(p)} \sum_{j=0}^{s-1} \binom{B(a, b, s, \overline{W}, j)}{2} = \frac{k(k-1)(p-h)(p-(s-h))}{2s}$$

This completes the proof of the lemma. □

Corollary VI.13 *Let $1 < s \leq k^2$. For the subset $\overline{W}$ of k elements in the set Z_{k^2}, there is an ordered pair (a, b) in $F_1(p)$ such that*

$$\sum_{j=0}^{s-1} \binom{B(a, b, s, \overline{W}, j)}{2} < \frac{k(k-1)}{2s}$$

Proof. Since there are exact $p(p-1)$ ordered pairs in $F_1(p)$, from Lemma VI.12, there is at least one ordered pair (a, b) in $F_1(p)$ such that

$$\sum_{j=0}^{s-1} \binom{B(a, b, s, \overline{W}, j)}{2} \leq \frac{k(k-1)(p-h)(p-(s-h))}{2sp(p-1)}$$

Since $p = gs + h$ is a prime number, we have $1 \leq h \leq s-1$ and $1 \leq s-h \leq s-1$. Therefore, both $(p-h)$ and $(p-(s-h))$ are not larger than $p-1$. In particular, $(p-h)(p-(s-h))$ is strictly smaller than $p(p-1)$. Now the corollary follows. □

The result in Corollary VI.13 is a significant improvement over the bound given in [34], which is the bound used to implement the color coding scheme suggested by Alon, Yuster, and Zwick [1]. In particular, the result derived in [34] uses the hash function $\psi_{a,s}$ and gave an upper bound of k^2/s. We will see that the bound improvement from k^2/s to $k(k-1)/(2s)$ significantly improves the size of k-color coding schemes.

Lemma VI.14 *For the subset $\overline{W}$ of k elements in Z_{k^2}, there is a pair (a, b) in $F_1(p)$ such that*

$$\sum_{j=0}^{k/4-1} B(a, b, k/4, \overline{W}, j)(B(a, b, k/4, \overline{W}, j) - 1) < 4k$$

Proof. Let $s = k/4$. From Corollary VI.13,there is a pair (a, b) in $F_1(p)$, such that

$$\sum_{j=0}^{k/4-1} \binom{B(a, b, k/4, \overline{W}, j)}{2} < 2(k - 1),$$

which directly implies the lemma. □

Corollary VI.15 *For the subset $\overline{W}$ of k elements in Z_{k^2}, there is a pair (a, b) satisfying the condition* **C1**, *such that if we let $\overline{W}_j = \{\overline{w} \mid \overline{w} \in \overline{W}$ & $\phi_{a,b,k/4}(\overline{w}) = j\}$ and $c_j = |\overline{W}_j|$ for all $0 \leq j \leq k' = k/4 - 1$, then the list $C = [c_0, \ldots, c_{k'}]$ satisfies the condition* **C2**.

Since each element w_i in the set W corresponds uniquely to an element $\overline{w}_i$ in the set $\overline{W}$, according to Corollary VI.15, there is a pair (a, b) satisfying condition **C1**, such that each set U_j, $0 \leq j \leq k' = k/4 - 1$, constructed in step 1 of the algorithm **Coloring** contains exactly c_j elements in W, and the list $C = [c_0, \ldots, c_{k'}]$ satisfies the condition **C2**.

Lemma VI.16 *Let $\overline{\mathcal{W}}$ be a collection of some of the subsets $\overline{W}_j$ with $c_j > 1$, as given in Corollary VI.15. Then there is a pair (a, b) satisfying condition* **C1** *such that for at least one half of the subsets $\overline{W}_j$ in $\overline{\mathcal{W}}$, the function $\phi_{a,b,c_j(c_j-1)}$ is injective from $\overline{W}_j$ to $Z_{c_j(c_j-1)}$.*

Proof. Fix a subset $\overline{W}_j$ in $\overline{\mathcal{W}}$, where $c_j = |\overline{W}_j| > 1$. Applying Lemma VI.12 to

$\overline{W}_j$ and let $s = c_j(c_j - 1)$ (where we have let $p \bmod s = h > 0$), we get:

$$\begin{aligned}\sum_{(a,b)\in F_1(p)} \sum_{i=0}^{s-1} \binom{B(a,b,s,\overline{W}_j,i)}{2} &= \frac{c_j(c_j-1)(p-h)(p-(s-h))}{2s} \\ &< \frac{c_j(c_j-1)p(p-1)}{2s} = \frac{p(p-1)}{2}\end{aligned}$$

Since the set $F_1(p)$ totally has $p(p-1)$ pairs, the above relation claims that for at least one half of the pairs (a, b) in $F_1(p)$, the equality

$$\sum_{i=0}^{s-1} \binom{B(a,b,s,\overline{W}_j,i)}{2} = 0$$

holds, i.e., $B(a, b, s, \overline{W}_j, i) \leq 1$ for all i. Therefore, for at least one half of the pairs (a, b) in $F_1(p)$, the function $\phi_{a,b,c_j(c_j-1)}$ is injective from the subset $\overline{W}_j$. Applying this analysis to each subset $\overline{W}_j$ in $\overline{\mathcal{W}}$ and using simple counting argument, we derive that there is at least one pair (a, b) in $F_1(p)$ (i.e., a pair (a, b) satisfying condition **C1**) such that for at least one half of the subsets $\overline{W}_j$ in $\overline{\mathcal{W}}$, the function $\phi_{a,b,c_j(c_j-1)}$ is injective from $\overline{W}_j$. □

Corollary VI.17 *Let $\overline{\mathcal{W}}_{>1}$ be the collection of all subsets $\overline{W}_j$ in Corollary VI.15 with $c_j = |\overline{W}_j| > 1$. Then there is an ordered list $L = [(a_1, b_1), \ldots, (a_r, b_r)]$ satisfying the condition* **C3** *such that for at least one half of the subsets $\overline{W}_j$ in $\overline{\mathcal{W}}_{>1}$, the function $\phi_{a_1,b_1,c_j(c_j-1)}$ is injective from $\overline{W}_j$ to $Z_{c_j(c_j-1)}$, for at least one half of the remaining subsets $\overline{W}_j$ in $\overline{\mathcal{W}}_{>1}$, the function $\phi_{a_2,b_2,c_j(c_j-1)}$ is injective from $\overline{W}_j$ to $Z_{c_j(c_j-1)}$, and for at least one half of the remaining subsets $\overline{W}_j$ in $\overline{\mathcal{W}}_{>1}$, the function $\phi_{a_3,b_3,c_j(c_j-1)}$ is injective from $\overline{W}_j$ to $Z_{c_j(c_j-1)}$, and so on.*

Proof. Applying Lemma VI.16 to $\overline{\mathcal{W}}_{>1}$, we get a pair (a_1, b_1) satisfying condition **C1** that, for at least one half of the subsets $\overline{W}_j$ in $\overline{\mathcal{W}}_{>1}$, is injective from $\overline{W}_j$. Let $\overline{\mathcal{W}}'_{>1}$ be the remaining subsets in $\overline{\mathcal{W}}_{>1}$. Applying Lemma VI.16 to $\overline{\mathcal{W}}'_{>1}$, we get a pair

(a_2, b_2) satisfying condition **C1** that, for at least one half of the subsets $\overline{W}_j$ in $\overline{\mathcal{W}}'_{>1}$, is injective from $\overline{W}_j$, and so on. This process stops after at most $r \leq \log q$ steps, where q is the total number of subsets in $\overline{\mathcal{W}}_{>1}$. The list of pairs $L = [(a_1, b_1), \ldots, (a_r, b_r)]$ constructed this way satisfies the condition **C3**. □

From Corollary VI.17, we get immediately,

Corollary VI.18 *Let $\overline{W}_j$ be the subset, $0 \leq j \leq k'$, as given in Corollary VI.15, $c_j = |\overline{W}_j|$, and $C = [c_0, \ldots, c_{k'}]$. Then there is a list $L = [(a_1, b_1), \ldots, (a_r, b_r)]$ satisfying condition* **C3** *and a mapping from $C_{>1}$ to L satisfying condition* **C4***, such that for all j, if $c_j > 1$ is mapped to (a_i, b_i), then the function $\phi_{a_i, b_i, c_j(c_j-1)}$ is injective from $\overline{W}_j$.*

Therefore, for the pair (a', b') and the list $C' = [c'_0, \ldots, c'_{k'}]$ in Corollary VI.15, which satisfy conditions **C1** and **C2**, respectively, and for the list

$$L' = [(a'_1, b'_1), \ldots, (a'_r, b'_r)]$$

and the mapping from $C'_{>1}$ to L' in Corollary VI.18, which satisfy conditions **C3** and **C4**, respectively, each function $\phi_{a'_i, b'_i, c'_j(c'_j-1)}$ is injective from the subset W_j to $Z_{c'_j(c'_j-1)}$ for all j. For each j, let W'_j be the image of W_j under the function $\phi_{a'_i, b'_i, c'_j(c'_j-1)}$, then $W'_j \subseteq Z_{c'_j(c'_j-1)}$ and $|W'_j| = c'_j$. Now since $\mathcal{F}_{c'_j}$ is a c'_j-color coding scheme for the set $Z_{c'_j(c'_j-1)}$, one $F_{c'_j}$ of the c'_j-colorings in $\mathcal{F}_{c'_j}$ is injective from W'_j. According to the algorithm **Coloring**, when this c'_j-coloring $F_{c'_j}$ is used for the algorithm, the c'_j elements in W_j are colored with distinct colors. Running this for all j, we conclude that there is a list $[F_{c'_0}, F_{c'_1}, \ldots, F_{c'_{k'}}]$, where $F_{c'_j}$ is a c'_j-coloring in the c'_j-color coding scheme $\mathcal{F}_{c'_j}$, satisfying condition **C5** such that all elements in the subset W are colored with distinct colors.

Summarizing the above discussion, we have the following theorem.

Theorem VI.19 *For each subset W of k elements in Z_n, there is a combination of parameters satisfying conditions* **C1-C5** *on which the algorithm* **Coloring** *produces a k-coloring for Z_n that is injective from W.*

Combining Theorem VI.19 with Theorem VI.9, and recall that we have let $n = k^2$, we get

Theorem VI.20 *If k is divisible by 4, then $\tau(k^2, k) = O(6.383^k k^{\log k - 4})$, i.e., there is a k-color coding scheme of size $O(6.383^k k^{\log k - 4})$ for the set Z_{k^2}.*

5. Extension to general n and k

By Theorem VI.10, the following theorem for general value of n is straightforward.

Theorem VI.21 *For any integer n, and integer k divisible by 4,*

$$\tau(n, k) = O(6.383^k k^{\log k - 4} n),$$

i.e., there is a k-color coding scheme of size $O(6.383^k k^{\log k - 4} n)$ for the set Z_n .

Now let us consider the case when k is not divisible by 4. Suppose that $k = 4k' - h$, where $1 \leq h \leq 3$. We first construct a $(4k')$-color coding scheme $\mathcal{F}'$ of size

$$O(6.383^{4k'} (4k')^{\log(4k') - 4} n) = O(6.383^k k^{\log(k+h) - 4} n)$$

for the set Z_n. Now for each $(4k')$-coloring F in $\mathcal{F}'$, we construct $\binom{4k'}{h} = O(k^3)$ k-colorings for Z_n by selecting every subset of h colors in F and replacing them arbitrarily by the remaining $k = 4k' - h$ colors. This gives a collection $\mathcal{F}$ of

$$O(k^3 6.383^k k^{\log(k+h) - 4} n) = O(6.4^k n)$$

k-colorings for the set Z_n. To show that this is a k-color coding scheme for the set Z_n, let W be any subset of k elements in Z_n. Let W' be a subset of $4k'$ elements in Z_n obtained from W by adding arbitrarily h elements. Since $\mathcal{F}'$ is a $(4k')$-color coding scheme for Z_n, there is a $(4k')$-coloring F' in $\mathcal{F}'$ that is injective from W'. In particular, this $(4k')$-coloring F' is also injective from W. Now the k-coloring F in $\mathcal{F}$ obtained from F' by removing the other h colors is injective from W.

Hence we have proved the following theorem.

Theorem VI.22 *For any integers n and k, where $n \geq k$, $\tau(n, k) = O(6.4^k n)$, i.e., there is a k-color coding scheme of size $O(6.4^k n)$ for the set Z_n .*

D. Final remarks

We have developed new randomized and deterministic algorithms for PATH, MATCHING, and PACKING problems. Our randomized algorithms are the first group of randomized algorithms of running time $O(2^{O(k)} n^{O(1)})$ and polynomial space for these problems. Moreover, our algorithms also improve the running time of the best previous algorithms.

Our deterministic algorithms for PATH, MATCHING, and PACKING problems significantly improve the previous best algorithms. Our algorithms are based on a new k-color coding scheme of significantly improved size. A number of new techniques have been used in the development of the new k-color coding scheme, including an upper bound on the size of k-color coding schemes and a four-level hashing procedure.

The color coding technique seems to provide a new approach to exact algorithms for solving NP-hard problems, especially for those that are concerned with finding k proper elements in a set of n elements. Recent research [18] has shown that for certain NP-hard problems, such searching seems to have to take time $n^{\Omega(k)}$. Therefore, a preprocessing by k-coloring the n elements so that the k searched elements are colored distinctly seems to significantly narrow down the search space. For example, suppose that we have colored the vertices of a graph G with k colors such that the vertices of a k-clique in G are all colored with distinct colors. Then searching for the k-clique can be easily done in time $O((n/k)^k)$, which seems to be significantly faster than searching for the k-clique in an uncolored graph.

We would like to point out that it is possible to further improve the upper bound on the size of k-color coding schemes for the set Z_n.

CHAPTER VII

ON PRODUCT COVERING IN 3-TIER SUPPLY CHAIN MODELS: NATURAL COMPLETE PROBLEMS FOR $W[3]$ AND $W[4]$

The field of supply chain management has been growing at a rapid pace in recent years, both as a research area and as a practical discipline. In this chapter, we study the computational complexity of product covering problems in 3-tier supply chain models, and present natural complete problems for the classes $W[3]$ and $W[4]$ in parameterized complexity theory. This seems the first group of natural complete problems for higher levels in the parameterized intractability hierarchy (i.e., the W-hierarchy), and the first precise complexity characterizations of certain optimization problems in the research of supply chain management. Our results also derive strong computational lower bounds and inapproximability for these optimization problems.

A. Introduction

Parameterized complexity theory [26] is a recently proposed and promising approach to the central issue of how to cope with intractable problems – as is so frequently the case in the natural world of computing. An example is the NP-complete problem VERTEX COVER (determining whether a given graph has a vertex cover of size k), which now is solvable in time $O(1.285^k + kn)$ [19] and becomes quite practical for various applications. The other direction of the research is the study of *parameterized intractability*, based on a parameterized intractability hierarchy, the W-hierarchy $\bigcup_{t\geq 1} W[t]$. Under a parameterized reduction, the *fpt-reduction*, a large number of well-known computational problems have been proved to be complete for certain levels of the W-hierarchy [26]. For example, CLIQUE, INDEPENDENT SET, SET PACKING, V-C DIMENSION, and WEIGHTED 3-SAT are complete for the class $W[1]$, and DOMINATING

SET, HITTING SET, SET COVER, and WEIGHTED SAT are complete for the class $W[2]$. The completeness of a problem in a level of the W-hierarchy characterizes precisely the parameterized complexity of the problem.

However, no complete problem is known for any level $W[t]$ for $t > 2$, except the generic problems based on weighted satisfiability on bounded depth circuits and their variations [14, 26][1]. Therefore, it is interesting to know whether high levels of the W-hierarchy, which are defined in terms of formal mathematics, catch the complexity of certain natural computational problems.

In this chapter, we present natural complete problems for the classes $W[3]$ and $W[4]$, based on computational problems studied in the areas of supply chain management. The study of supply chain management has been growing at a rapid pace in recent years, as a research area and as a practical discipline (see recent survey papers [43, 70]). It has provided extremely rich contexts for the definition of new large-scale optimization problems. Efforts to improve supply chain management have gained the attention of academic researchers, along with the enthusiastic support of government and industry. Therefore, our completeness results in the W-hierarchy for computational problems in the study of supply chains will also contribute to the understanding of this new computation model. Moreover, based on the recent research on parameterized intractability and inapproximability [18], our results also imply directly inapproximability for these problems.

We give a quick review on the related background.

A *parameterized problem* consists of instances of the form (x, k), where x is the *problem description* and k is an integer called the *parameter*. A parameterized

[1] We note that a similar situation has occurred in the study of the popular polynomial time hierarchy, for which complete problems for the first level Σ_1^p =NP have been extensively studied while the research on natural complete problems for higher level Σ_t^p for $t > 1$ has just started recently [84, 85, 86].

problem Q is *fixed parameter tractable* if it can be solved by an algorithm of running time $O(f(k)n^{O(1)})$, where f is a function independent of $n = |x|$. Denote by FPT the class of all fixed parameter tractable problems.

A Π_t-*circuit* of n input variables $x_1, \ldots, x_n$ is a $(t+1)$-leveled circuit in which (1) the 0-th level is a single output gate that is an AND-gate; (2) each level-t gate is an input gate labeled by either x_i (a positive literal) or $\overline{x}_i$ (a negative literal), $1 \leq i \leq n$; (3) the outputs of a level-j gate can only be connected to the inputs of level-$(j-1)$ gates; and (4) AND-gates and OR-gates are organized into t alternating levels. A circuit is *monotone* (resp. *antimonotone*) if all its input gates are labeled by positive literals (resp., negative literals). A circuit represents naturally a boolean function. A truth assignment α to the variables of a circuit C *satisfies* C if α makes C output 1. The *weight* of an assignment α is the number of variables assigned value 1 by α.

The problem *weighted satisfiability on* Π_t-*circuits*, briefly WCS$[t]$, consists of instances of the form (C, k), where C is a Π_t-circuit that is satisfied by an assignment of weight k. The W-*hierarchy*, $\bigcup_{t \geq 1} W[t]$, in parameterized complexity theory is defined based on WCS$[t]$ via a new reduction, the *fpt-reduction*. We say that a parameterized problem Q is *fpt-reducible* to another parameterized problem Q' if there are two recursive functions f and g, and an algorithm A of running time bounded by $f(k)|x|^{O(1)}$, such that for an input (x, k), the algorithm A produces a pair (x', k'), where $k' \leq g(k)$, and (x, k) is a yes-instance of Q if and only if (x', k') is a yes-instance of Q'. It is easy to verify that the fpt-reducibility is transitive [26]. For an integer $t \geq 2$, a parameterized problem Q_1 is in the class $W[t]$ if Q_1 is fpt-reducible to the problem WCS$[t]$, a parameterized problem Q_2 is $W[t]$-*hard* if the problem WCS$[t]$ is fpt-reducible to Q_2 (or equivalently, if all problems in $W[t]$ are fpt-reducible to Q_2), and a parameterized problem Q_3 is $W[t]$-*complete* if Q_3 is in $W[t]$ and is $W[t]$-hard.

In particular, the problem WCS$[t]$ is a generic $W[t]$-complete problem for $t \geq 2$.[2]

We briefly review the related concepts in supply chain management, which has been the subject of a growing body of research literature. The readers are referred to [20, 22, 91] for detailed and systematic discussions, and [43, 44, 70, 94] for more recent progresses. The underlying structure of a supply chain model is a network consisting of various functional units (such as material suppliers, manufactures, storages, marketing/sales and retailers, and customers) and connections between different units (in the means of both material and information). A supply chain may have numerous tiers in the case of that substructure of manufactures forms a lengthy network itself [70]. *Supply chain management* involves the management of flows between and among the units in a supply chain to maximize total profitability [43]. The research in supply chain management includes the studies in *strategic*-, *tactical*-, and *operational*-level decisions [43]. In particular, tactical-level decisions, which is the subarea directly related to our current chapter, are concerned with medium-range planning efforts, such as production and distribution quantity planning among multiple existing facilities, system-wide inventory policies, and distribution frequency decisions between facilities.

B. Three-tier single product cover and $W[3]$-completeness

We follow the supply chain model studied in [90], which is a slight generalization of the model studied in [53]. The model is a 3-tier supply chain that consists of three kinds of units: (material) *suppliers*, (product) *manufacturers*, and *retailers*, such that:

1. A supplier can be linked to a manufacturer, and a manufacturer can be linked

[2] The corresponding definitions for the class $W[1]$ are somehow special and not directly related to our discussion, thus are omitted. The readers are referred to [26] for details.

to a retailer, standing for transportations/transactions between the units (link capacity is assumed unlimited);

2. A supplier can *provide* certain materials;

3. A manufacturer can *produce* a product if all needed materials for the product are provided by suppliers linked to the manufacturer;

4. A retailer has supply of a product if a manufacturer linked to the retailer produces the product.

Such a supply chain can be modeled by a directed graph $G = (S \cup M \cup R, E)$, where each unit is represented as a vertex in G and each directed edge in E represents a link between the corresponding units, here S is the set of all suppliers, M is the set of all manufacturers, and R is the set of all retailers. The objective of optimization studied in the current chapter on this model is to maximize the *channel profit* [20, 91], that is, to study the strategies that ensure that all retailers have supply of certain products they want to carry. In particular, we say that a product *covers* all retailers if all retailers have supply of that product.

Now suppose that we want to test the market of a new product at the widest range of customers, using as little experimental resource (i.e., suppliers) as possible and without overloading any supplier. For this, we assign at most one kind of material needed for the new product to each supplier and would like that the product covers all retailers. Obviously, the problem is directly related to the complexity of the product, i.e., the number k of different kinds of materials needed for the product. Formally, the problem can be formulated as the following parameterized problem:

3-SCM SINGLE-PRODUCT COVER:

> Let $G = (S \cup M \cup R, E)$ be a supply chain model, and k an integer, and suppose that we are going to produce a new product that requires k different kinds of materials. Is it possible to pick k suppliers, each for a different kind of material, to produce the new product, such that the product covers all retailers?

Before we prove our main result in this section, we first define the problem WEIGHTED SATISFIABILITY ON ANTIMONOTONE Π_3-CIRCUITS, shortly WCS$^-$[3]. The problem WCS$^-$[3] is a subproblem of the problem WCS[3] that requires that in the input pair (C, k) the Π_3-circuit C be antimonotone (i.e., all input gates of C be labeled by negative input literals). It is known that the problem WCS$^-$[3] is also $W[3]$-complete [26]. Thus, to prove the $W[3]$-completeness for the problem 3-SCM SINGLE-PRODUCT COVER, it suffices to derive fpt-reductions between WCS$^-$[3] and 3-SCM SINGLE-PRODUCT COVER.

Theorem VII.1 *The problem* 3-SCM SINGLE-PRODUCT COVER *is $W[3]$-complete.*

PROOF. As explained above, we first present an fpt-reduction from WCS$^-$[3] to 3-SCM SINGLE-PRODUCT COVER. Let (C, k) be an instance of WCS$^-$[3], where C is an antimonotone Π_3-circuit. Let g_0 be the output AND-gate of C (which is at level 0), L_1 be the set of OR-gates at level 1 in C (whose outputs are inputs to g_0), L_2 be the set of AND-gates at level 2 in C (whose outputs are inputs to gates in L_1), and L_3 be the set of input gates in C (which are inputs to gates in L_2 and are labeled by negative input literals).

Construct a 3-tier supply chain model $G = (S \cup M \cup R, E)$ as follows: (1) each retailer ρ_i in R corresponds to an OR-gate u_i in L_1; (2) each manufacturer μ_i in M corresponds to an AND-gate v_i in L_2; (3) each supplier σ_i in S corresponds to an input gate $\overline{x}_i$ in L_3. The vertices in G are connected in the following way: (1) there is a link from a manufacturer μ_i to a retailer ρ_j if and only if the corresponding AND-gate v_i is an input to the corresponding OR-gate u_j; and (2) there is a link from a supplier σ_i to a manufacturer μ_j if and only if the corresponding input gate $\overline{x}_i$ is *not* an input to the corresponding AND-gate v_j (note that C is an antimonotone circuit). This completes the description of the 3-tier supply chain model G. We prove that the circuit C has a satisfying assignment α of weight k if and only if we can pick k suppliers in the supply chain G, each for a different kind of material for a new product that needs k kinds of materials, so that the new product covers all retailers.

Suppose that the circuit C has a satisfying assignment α of weight k. Let X_k be the set of k variables in C that are assigned value 1 by α. Let S_k be the k suppliers corresponding to the k input variables in X_k. We show that we can pick the k suppliers in S_k, each for a different kind of material for the new product that needs k kinds of materials, such that the product covers all retailers in G. Consider any manufacturer μ_i in M. If μ_i has the supply for all k kinds of materials for the new product, i.e., if μ_i has links from all the k suppliers in S_k, then by the construction of the supply chain model G, the corresponding AND-gate v_i in C has no input from any input gate $\overline{x}_j$ where x_j is an input variable in X_k. Therefore, under the assignment α, all inputs to the gate v_i have value 1 and the output of v_i has value 1. On the other hand, if the manufacturer μ_i does not receive supply from a supplier σ_j in S_k, then the input gate $\overline{x}_j$ is an input to the AND-gate v_i, and under the assignment α, the output of gate v_i has value 0. In summary, the AND-gate v_i outputs value 1 if and only if the corresponding manufacturer μ_i has supply from all k suppliers in S_k and is able to produce the new product. Now, a retailer ρ_i has supply of the new product if and only if it has a link from a manufacturer μ_j that can produce the new product, which by the above analysis if and only if the corresponding AND-gate v_j in L_2 outputs value 1 under the assignment α. Since the retailer ρ_i has a link from a manufacturer μ_j if and only if the corresponding OR-gate u_i in C has input from the corresponding AND-gate v_j, we conclude that the retailer ρ_i has supply of the new product if and only if the corresponding OR-gate u_i in C outputs value 1 under the assignment α. Finally, since the output AND-gate g_0 of C is connected to all OR-gates in L_1, we conclude that the circuit C has value 1 if and only if all retailers have supply of the new product. In consequence, if α is a satisfying assignment for the circuit C, then picking the k suppliers in S_k results in the new product that covers all retailers in G.

Conversely, suppose there is a set S_k of k suppliers, each for a different kind of material for the new product such that the new product covers all retailers. We let X_k be the k input variables in the circuit C corresponding to the k suppliers in S_k. Let α be a weight-k assignment to C that assigns value 1 to the k variables in X_k and value 0 to all other input variables. Then following exactly the same reasoning as above, we can verify that the assignment α satisfies the circuit C.

This completes the analysis of the reduction from WCS$^-$[3] to 3-SCM SINGLE-PRODUCT COVER. The reduction is obviously an fpt-reduction. In conclusion, we have proved that the problem 3-SCM SINGLE-PRODUCT COVER is $W[3]$-hard.

To show that the problem 3-SCM SINGLE-PRODUCT COVER is in $W[3]$, it suffices to show that 3-SCM SINGLE-PRODUCT COVER is fpt-reducible to WCS$^-$[3]. The construction is very similar to the one described above: for an instance (G, k) of 3-SCM SINGLE-PRODUCT COVER, where $G = (S \cup M \cup R, E)$ is a supply chain model and k is an integer, we construct an instance (C, k) of WCS$^-$[3], where each level-1 OR-gate in C corresponds to a retailer in R, each level-2 AND-gate in C corresponds to a manufacturer in M, and each input gate in C (labeled by a negative literal) corresponds to a supplier in S. A level-1 OR-gate has an input from a level-2 AND-gate if and only if the corresponding retailer has a link from the corresponding manufacturer in G, and a level-2 AND-gate has an input from an input gate if and only if the corresponding manufacturer has *no* link from the corresponding supplier. Now by the exact method, we can verify that the circuit C has a satisfying assignment of weight k if and only if there are k suppliers, each for a different kind of material for the new product, such that the new product covers all retailers. In consequence, the problem 3-SCM SINGLE-PRODUCT COVER is in the class $W[3]$.

This proves that the problem 3-SCM SINGLE-PRODUCT COVER is $W[3]$-complete.

□

C. Three-tier multiple product cover and $W[4]$-completeness

To describe $W[4]$-complete problems, we consider a more general model of 3-tier supply chains by allowing a supplier to provider multiple kinds of materials, a manufacturer to produce multiple kinds of products, and a retailer to carry multiple kinds of products.

We first consider a problem that is concerned with the covering by a line of homogeneous (i.e., similar) products. Formally, let P be a given line of homogeneous products and let T be a set of materials, where each product π in P is associated with a set of materials in T that are needed for producing the product. In a 3-tier supply chain $G = (S \cup M \cup R, E)$, each supplier σ in S is associated with a list of materials in T that the supplier σ can provide, each manufacturer μ in M is associated with a list of products in P that the manufacturer μ can produce when necessary materials are provided by suppliers linked to μ, and each retailer ρ in R is associated with a suggested list of products in P that the retailer ρ is interested in carrying when the products are produced by the manufacturers linked to ρ. We are interested in the following problem in supply chain management: for a new line P

of homogeneous products, we want to use limited amount of resource (i.e., a small number of suppliers) to test the product market in the widest range of customers (i.e., make all retailers have supply of some of the new products). This is formulated as the following parameterized GENERAL 3-SCM H-PRODUCT-LINE COVER problem.

> Given a line P of homogeneous products, a general supply chain model $G = (S \cup M \cup R, E)$, and an integer k, is it possible to pick k suppliers for the products in P so that each retailer has supply of some products in its associated product list?

To study the complexity of this problem, we consider the problem WEIGHTED SATISFIABILITY ON MONOTONE Π_4-CIRCUITS, shortly WCS$^+[4]$, which is a subproblem of the problem WCS[4] with an additional constraint that in the input pair (C, k) the Π_4-circuit C be monotone (i.e., all input gates of C be labeled by positive input literals). It is known that the problem WCS$^+[4]$ is also $W[4]$-complete [26]. Thus, in order to prove the $W[4]$-completeness for GENERAL 3-SCM H-PRODUCT-LINE COVER, it suffices to present fpt-reductions between WCS$^+[4]$ and GENERAL 3-SCM H-PRODUCT-LINE COVER.

Lemma VII.2 *The problem* GENERAL 3-SCM H-PRODUCT-LINE COVER *is in* $W[4]$.

PROOF. We show how the problem GENERAL 3-SCM H-PRODUCT-LINE COVER is fpt-reducible to the problem WCS$^+[4]$.

Let (P, G, k) be an instance of GENERAL 3-SCM H-PRODUCT-LINE COVER, where $P = \{\pi_1, \ldots, \pi_h\}$ is a product line and each product π_i is associated with a set of materials needed for producing π_i, $G = (S \cup M \cup R, E)$ is a general 3-tier supply chain with the supplier set $S = \{\sigma_1, \ldots, \sigma_n\}$, the manufacturer set $M = \{\mu_1, \ldots, \mu_m\}$, and the retailer set $R = \{\rho_1, \ldots, \rho_t\}$. Let $T = \{\tau_1, \ldots, \tau_p\}$ be the set of different materials that are needed for the products in P (T can be obtained directly from the product list P). Each supplier σ_i is associated with a subset of T, indicating the materials that can be provided by σ_i; each manufacturer μ_i is associated with a subset of P, indicating the products that can be produced by μ_i when necessary materials are provided by the suppliers linked to μ_i; and each retailer ρ_i is associated with a subset of P, indicating the products that the retailer ρ_i is interested in carrying from the manufacturers linked to ρ_i.

We construct an instance (C, k) for WCS$^+[4]$, where the monotone Π_4-circuit C has the following structure:

(V0) The level-0 output AND-gate in C is g_0;

(V1) The set of level-1 gates in C consists of t OR-gates u_i, $1 \leq i \leq t$, corresponding to the t retailers in G;

(V2) For each manufacturer μ_i and each product π_j in the associated product list of μ_i, there is an AND-gate v_{ij} in level 2 in C;

(V3) For each manufacturer μ_i and each material τ_j that is needed for a product in the associated product list of μ_i, there is an OR-gate w_{ij} in level 3 in C;

(V4) the set of level-4 gates in C consists of n input gates labeled x_i, $1 \leq i \leq n$, respectively, corresponding to the n suppliers in G.

The gates in the circuit C are connected as follows:

(E1) all level-1 gates are inputs to the output gate g_0;

(E2) a level-2 gate v_{ij} is an input to a level-1 gate u_s if there is a link in G from the manufacturer μ_i to the retailer ρ_s in the supply chain G, and if the product π_j is contained in both associated product lists of μ_i and ρ_s;

(E3) a level-3 gate w_{ij} is an input to a level-2 gate v_{is} if the material τ_j is needed for the product π_s, and the product π_s is contained in the associated product list of the manufacturer μ_i;

(E4) an input gate labeled x_i is an input of a level-3 gate w_{js} if and only if the supplier σ_i can provide the material τ_s and if the supplier σ_i is linked to the manufacturer μ_j (note that by the construction, the material τ_s is needed for some product in μ_j).

This completes the description of the circuit C, which is obviously a monotone Π_4-circuit. We now show that (P, G, k) is a yes-instance for GENERAL 3-SCM H-PRODUCT-LINE COVER if and only if (C, k) is a yes-instance for WCS$^+$[4]. For this, we establish a one-to-one mapping between the subsets of k suppliers in G and the weight-k assignments to the circuit C, as follows: each subset S_k of k suppliers in G corresponds to the assignment $\phi(S_k)$ in C that assigns value 1 to an input variable x_i if and only if the supplier σ_i is in S_k. To prove the lemma, it suffices to show that $\phi(S_k)$ is a satisfying assignment for C if and only if picking the k suppliers in S_k will make all retailers in G have supply for some products in P. This can be verified by the following facts:

(F1) By rule (E4), a level-3 OR-gate w_{js} has value 1 under the assignment $\phi(S_k)$ if and only if the manufacturer μ_j has supply of material τ_s under the supplier selection S_k;

(F2) By rule (E3), a level-2 AND-gate v_{is} has value 1 under the assignment $\phi(S_k)$ if and only if the manufacturer μ_i has supply for all needed materials for product π_s under the supplier selection S_k, that is, if and only if μ_i can produce the product π_s;

(F3) By rule (E2), a level-1 OR-gate u_s has value 1 under the assignment $\phi(S_k)$ if and only if the retailer ρ_s has supply of some products in its associated product list under the supplier selection S_k.

Summarizing facts (F1), (F2), and (F3), we conclude that the weight-k assignment $\phi(S_k)$ satisfies the circuit C if and only if all retailers in G have supply for some products in P under the selection of the k suppliers in S_k. Since the mapping from S_k to $\phi(S_k)$ is a one-to-one mapping from the subsets of k suppliers in G to the weight-k assignments for the circuit C, this verifies that this reduction from GENERAL 3-SCM H-PRODUCT-LINE COVER to WCS$^+[4]$ is an fpt-reduction. In consequence, the problem GENERAL 3-SCM H-PRODUCT-LINE COVER is in the class $W[4]$. □

Now we verify the $W[4]$-hardness for the problem GENERAL 3-SCM H-PRODUCT-LINE COVER.

Lemma VII.3 *The problem* GENERAL 3-SCM H-PRODUCT-LINE COVER *is* $W[4]$-*hard.*

PROOF. As explained in the paragraph before Lemma VII.2, it suffices to present an fpt-reduction from the problem WCS$^+[4]$ to the problem GENERAL 3-SCM H-PRODUCT-LINE COVER.

Let (C, k) be an instance of WCS$^+[4]$, where C is a monotone Π_4-circuit. Let the output AND-gate of C be g_0, and suppose that in the circuit C there are t level-1 OR-gates u_i, $1 \leq i \leq t$, q level-2 AND-gates v_i, $1 \leq i \leq q$, m level-3 OR-gates w_i, $1 \leq i \leq m$, and n input gates labeled x_i, $1 \leq i \leq n$. We first perform a preprocessing on the circuit C as follows. If any gate g (at any level) has exactly the same input as another gate g' at the same level, then we "merge" these two gates into a single gate g'' of the same type, and let the output of g'' connect to the outputs of g and g' in the original circuit. It is easy to see that such a modification can be done in polynomial time and does not change the circuit function. With this preprocessing, we can assume, without loss of generality, that no two gates at the same level in the circuit have exactly the same input.

We construct an instance (P, G, k) for GENERAL 3-SCM H-PRODUCT-LINE COVER, where $G = (S \cup M \cup R, E)$, as follows. Each level-3 gate w_i corresponds to a different material τ_i, $1 \leq i \leq m$, and each input variable x_i corresponds to a supplier σ_i, $1 \leq i \leq n$ (thus, $S = \{\sigma_1, \ldots, \sigma_n\}$). A material τ_i is in the supplier σ_j (recall that each supplier is specified by a set of materials that can be provided by the supplier) if and only if the variable x_j is an input to the level-3 gate w_i. There are q products π_i in P, $1 \leq i \leq q$, such that a material τ_j is needed for the product π_i if and only if the level-3 gate w_j is an input to the level-2 gate v_i (thus, $P = \{\pi_1, \ldots, \pi_q\}$, note that by our preprocessing, all products in P require different subsets of materials). There are also q manufacturers μ_i, corresponding to the q level-2 gates v_i in C, $1 \leq i \leq q$ (thus, $M = \{\mu_1, \ldots, \mu_q\}$). For each i, the manufacturer μ_i is associated with the product set $\{\pi_i\}$ consisting of a single product π_i. Finally, there are t retailers ρ_i, corresponding to the t level-1 gates u_i in C, $1 \leq i \leq t$ (thus, $R = \{\rho_1, \ldots, \rho_t\}$). The product set associated with a retailer ρ_i consists of exactly those products π_j such that the level-2 gate v_j is an input of the level-1 gate u_i. The suppliers in S and the manufacturers in M are fully connected (i.e., every supplier is linked to every manufacturer), and a manufacturer μ_i is linked to a retailer ρ_j if and only if the level-2 gate v_i is an input to the level-1 gate u_j. This completes the description of the general 3-tier supply chain model G and the product set P.

As we did in Lemma VII.2, we establish a one-to-one mapping between the subsets of k suppliers in G and the weight-k assignments for the circuit C, by mapping a subset S_k of k suppliers to the assignment $\phi(S_k)$ such that $\phi(S_k)$ assigns value 1 to a variable x_i if and only if the supplier σ_i is in the subset S_k. Again we show that $\phi(S_k)$ is a satisfying assignment for the circuit C if and only if the selection of the k

suppliers in S_k makes all retailers in R to have supply for some products in P. This can be proved by verifying the following facts.

(F1) By our construction of the suppliers in S, and since the suppliers and the manufacturers in G are fully connected, a level-3 OR-gate w_i in C has value 1 under the assignment $\phi(S_k)$ if and only if the material τ_i can be provided (to all manufacturers) under the supplier selection S_k;

(F2) A level-2 AND-gate v_i has value 1 under the assignment $\phi(S_k)$ if all of its inputs in level 3 have value 1, equivalently, if all needed materials for product π_i are provided for the manufacturer μ_i under the supplier selection S_k. Therefore, a level-2 gate v_i has value 1 under the assignment $\phi(S_k)$ if and only if the manufacturer μ_i can produce the product π_i under the supplier selection S_k (note that π_i is the only product that can be produced by the manufacturer μ_i);

(F3) Finally, by our connections between the manufacturers and retailers in G, a retailer ρ_i has supply of a product π_j in P under the supplier selection S_k if and only if the manufacturer μ_j is linked to ρ_i and can produce the product π_j, equivalently, if and only if the level-2 gate v_j in C is an input of the level-1 OR-gate u_i and v_j has value 1 under the assignment $\phi(S_k)$. Therefore, the retailer ρ_i has supply of some products in P under the supplier selection S_k if and only if the level-1 gate u_i has value 1 under the assignment $\phi(S_k)$.

This completes the verification that the assignment $\phi(S_k)$ satisfies the circuit C if and only if every retailer in G has supply of some products in P. It clearly gives an fpt-reduction from WCS$^+$[4] to GENERAL 3-SCM H-PRODUCT-LINE COVER. In consequence, the problem GENERAL 3-SCM H-PRODUCT-LINE COVER is $W[4]$-hard. □

Combining Lemma VII.2 and Lemma VII.3, we get immediately,

Theorem VII.4 *The problem* GENERAL 3-SCM H-PRODUCT-LINE COVER *is* $W[4]$-*complete.*

The $W[4]$-completeness also provides precise complexity characterization for other computational problems in 3-tier supply chain management. For example, suppose now that a firm is interested in investigating the market for a set P of *non-homogeneous* products. The 3-tier supply chain is again given as a network of suppliers, manufacturers, and retailers, where each supplier is given as before and associated with a set of materials that can be provided by the supplier. Each manufacturer μ is associated with a set T_μ of materials and a set P_μ of products such that when all materials in T_μ are provided by suppliers linked to μ, the manufacturer μ can produce all products in P_μ. Finally, each retailer ρ is associated with a *requested* list of products that *must* be carried by the retailer ρ. This supply chain model gives a parameterized problem as follows:

> GENERAL 3-SCM PRODUCT-SET COVER:
>
> Given a product set P, a general supply chain model $G = (S \cup M \cup R, E)$ as described above, and an integer k, is it possible to pick k suppliers in S for materials so that every retailer in R has supply of *all* products in its associated product list?

The main difference between GENERAL 3-SCM H-PRODUCT-LINE COVER and GENERAL 3-SCM PRODUCT-SET COVER is that in the former model each retailer only needs to carry *some* of the products in its associated list while in the latter model each retailer must carry *all* products in its associated list.

The proof for the following theorem is similar to (actually, slightly simpler than) that for Lemma VII.2 and Lemma VII.3. We omit it and leave it to the interested readers.

Theorem VII.5 *The* GENERAL 3-SCM PRODUCT-SET COVER *problem is* $W[4]$*-complete.*

D. Computational lower bounds and inapproximability results

Theorem VII.1, Theorem VII.4 and Theorem VII.5 provide strong lower bounds for the complexity of the problems 3-SCM SINGLE-PRODUCT COVER, GENERAL 3-SCM H-PRODUCT-LINE COVER, and GENERAL 3-SCM PRODUCT-SET COVER.

Theorem VII.6 *For any recursive function f, the problem* 3-SCM SINGLE-PRODUCT COVER *cannot be solved in time $f(k)m^{O(1)}n^{o(k)}$ unless $W[2] = FPT$, and the problems* GENERAL 3-SCM H-PRODUCT-LINE COVER *and* GENERAL 3-SCM PRODUCT-SET COVER *cannot be solved in time $f(k)m^{O(1)}n^{o(k)}$ unless $W[3] = FPT$, where n is the number of suppliers and m is the size of the instance of the problems.*

PROOF. Suppose that the problem 3-SCM SINGLE-PRODUCT COVER could be solved in time $f(k)m^{O(1)}n^{o(k)}$, then by the fpt-reduction from WCS$^{-}[3]$ to 3-SCM SINGLE-PRODUCT COVER given in Theorem VII.1, it is easy to see that the problem WCS$^{-}[3]$ can also be solved in time $f(k)m^{O(1)}n^{o(k)}$, where m is the instance size and n is the number of input variables in the circuit. By Theorem 4.2 in [18], it would imply $W[2] = FPT$. The lower bounds for GENERAL 3-SCM H-PRODUCT-LINE COVER and GENERAL 3-SCM PRODUCT-SET COVER can be proved in the same way using the same theorem in [18]. □

Since it is generally believed that $W[t] \neq FPT$ for all $t > 0$, Theorem VII.6 provides a computational lower bound $f(k)m^{O(1)}n^{\Omega(k)}$ for the problems 3-SCM SINGLE-

PRODUCT COVER, GENERAL 3-SCM H-PRODUCT-LINE COVER, and GENERAL 3-SCM PRODUCT-SET COVER. Note that this is an asymptotically tight lower bound for the problems as the algorithm that exhaustively enumerates and examines all subsets of k suppliers in a problem instance solves the problems in time $O(m^2 n^k)$ trivially.

Theorem VII.6 further implies inapproximability results for certain optimization problems in 3-tier supply chain management. For this, we need to first review some related terminologies in approximation algorithms. The readers are referred to [4] for more detailed definitions and more comprehensive discussions.

An *optimization problem* Q consists of a set of *instances*, where each instance x is associated with a set of *solutions*. Each solution y of an instance x of Q is assigned an integral *value* $f_Q(x, y)$. The problem Q is a *maximization* (resp. *minimization*) problem if for each instance x of Q, we are looking for a solution of maximum (resp., minimum) value. Such a solution is called an *optimal solution* for the instance, whose value is denoted by $opt_Q(x)$.

An algorithm A is an *approximation algorithm* for an optimization problem Q if, for each instance x of Q, the algorithm A returns a solution $y_A(x)$ for x. The solution $y_A(x)$ has an *approximation ratio* r if it satisfies the following condition:

- $opt_Q(x)/f_Q(x, y_A(x)) \leq r$ if Q is a maximization problem
- $f_Q(x, y_A(x))/opt_Q(x) \leq r$ if Q is a minimization problem

The approximation algorithm A has an *approximation ratio* r if for any instance x of Q, the solution $y_A(x)$ constructed by the algorithm A has an approximation ratio bounded by r. A *polynomial time approximation scheme* (PTAS) for Q is an algorithm A' that on an instance x of Q and a real number $\epsilon > 0$, constructs a solution for x whose approximation ratio is bounded by $1+\epsilon$, and the running time of A' is bounded by a polynomial of $|x|$ for each fixed ϵ (see [4] for more detailed discussions on PTAS).

Now consider the following optimization problems in supply chain management:

> The 3-SCM MOST COMPLICATED PRODUCT COVER problem:
>
> Given a 3-tier supply chain G, we say a product P that requires k different materials is *viable* if we can select k suppliers in G, each for a different kind of material, such that P can be produced and all retailers in G have supply of P. Our goal is to find the largest such k, i.e. the most complicated product viable to G.
>
> the GENERAL 3-SCM MINIMUM-RESOURCE H-PRODUCT-LINE COVER problem:
>
> Given a line P of homogeneous products and a general 3-tier supply chain G (as defined in GENERAL 3-SCM H-PRODUCT-LINE COVER), select the minimum number of suppliers in G for the product line P, such that each retailer in G has supply of some products in its associated product list.
>
> and the GENERAL 3-SCM MINIMUM-RESOURCE PRODUCT-SET COVER problem:
>
> Given a set P of non-homogeneous products and a general 3-tier supply chain G (as defined in GENERAL 3-SCM PRODUCT-SET COVER), select the minimum number of suppliers in G for the product set P, such that each retailer in G has supply of all products in its product list.

Note that 3-SCM MOST COMPLICATED PRODUCT COVER is a maximization problem while GENERAL 3-SCM MINIMUM-RESOURCE H-PRODUCT-LINE COVER and GENERAL 3-SCM MINIMUM-RESOURCE PRODUCT-SET COVER are minimization problems.

An optimization problem Q can be parameterized using the following formulation [18]:

Definition VII.7 *The parameterized version of an optimization problem Q is defined as follows:*

1. *If Q is a maximization problem, then the parameterized version of Q is defined as $Q_{\geq} = \{(x, k) \mid x$ is an instance of Q and$opt_Q(x) \geq k\}$;*
2. *If Q is a minimization problem, then the parameterized version of Q is defined as $Q_{\leq} = \{(x, k) \mid x$ is an instance of Q and $opt_Q(x) \leq k\}$.*

Now we consider the parameterized versions of the problems 3-SCM MOST COMPLICATED PRODUCT COVER, GENERAL 3-SCM MINIMUM-RESOURCE H-PRODUCT-LINE COVER, and GENERAL 3-SCM MINIMUM-RESOURCE PRODUCT-SET COVER. The parameterized versions of these problems look slightly different from the corresponding problems in Theorem VII.6. For example, an instance (x, k) of GENERAL 3-SCM H-PRODUCT-LINE COVER asks whether we can pick *exact* k suppliers so that all retailers are covered by a given product line, while an instance (x, k) of the parameterized version of GENERAL 3-SCM MINIMUM-RESOURCE H-PRODUCT-LINE COVER asks whether we can pick *at most* k suppliers so that all retailers are covered by a given product line. However, a more careful examination shows that the two problems are in fact equivalent. In particular, if (x, k) is a yes-instance of the problem GENERAL 3-SCM H-PRODUCT-LINE COVER, then obviously (x, k) is also a yes-instance of the parameterized version of the problem GENERAL 3-SCM MINIMUM-RESOURCE H-PRODUCT-LINE COVER. On the other hand, suppose that (x', k) is a yes-instance of the parameterized version of the problem GENERAL 3-SCM MINIMUM-RESOURCE H-PRODUCT-LINE COVER, where $x' = (P, (S \cup M \cup R, E))$, P is a product line, and $(S \cup M \cup R, E)$ is a general supply chain model, as given in the description of the problem GENERAL 3-SCM H-PRODUCT-LINE COVER. Then there are k_1 suppliers in S for the given product line P, where $k_1 \leq k$, so that selecting these k_1 suppliers will make the product line P cover all retailers in R. Note that picking these k_1 suppliers plus any $k - k_1$ other suppliers in S will give k suppliers whose selection also makes the product line P cover all retailers in R. This shows that (x', k) is also a yes-instance for the problem GENERAL 3-SCM H-PRODUCT-LINE COVER. Thus, (x, k) is a yes-instance for the problem GENERAL 3-SCM H-PRODUCT-LINE COVER if and only if (x, k) is a yes-instance for the parameterized version of the problem GENERAL 3-SCM MINIMUM-RESOURCE H-PRODUCT-LINE COVER. This shows the equivalence between the two problems. Similarly, the equivalences can be derived between the problem 3-SCM SINGLE-PRODUCT COVER and the parameterized version of the problem 3-SCM MOST COMPLICATED PRODUCT COVER and between the problem GENERAL 3-SCM PRODUCT-SET COVER and the parameterized version of the problem GENERAL 3-SCM MINIMUM-RESOURCE PRODUCT-SET COVER.

Now we are ready for the following theorem.

Theorem VII.8 *For any recursive function f, the problem* 3-SCM MOST COMPLICATED PRODUCT COVER *has no PTAS of running time $f(1/\epsilon)m^{O(1)}n^{o(1/\epsilon)}$ unless $W[2] = FPT$, and the problems* GENERAL 3-SCM MINIMUM-RESOURCE H-PRODUCT-LINE COVER *and* GENERAL 3-SCM MINIMUM-RESOURCE PRODUCT-SET COVER *have no PTAS of running time $f(1/\epsilon)m^{O(1)}n^{o(1/\epsilon)}$ unless $W[3] = FPT$, where n is the number of suppliers and m is the instance size of the problems.*

PROOF. We give the detailed proof for the problem GENERAL 3-SCM MINIMUM-RESOURCE H-PRODUCT-LINE COVER. Suppose that this problem has a PTAS of running time $f(1/\epsilon)m^{O(1)}n^{o(1/\epsilon)}$ for a recursive function f, then by Theorem 5.1 in [16], its parameterized version can be solved in time $f(2k)m^{O(1)}n^{o(k)}$. Since the parameterized version of the problem GENERAL 3-SCM MINIMUM-RESOURCE H-PRODUCT-LINE COVER is equivalent to the problem GENERAL 3-SCM H-PRODUCT-

LINE COVER, this would imply that the problem GENERAL 3-SCM H-PRODUCT-LINE COVER can be solved in time $f(2k)m^{O(1)}n^{o(k)}$, which, by Theorem VII.6, would imply $W[3] = FPT$. The inapproximability for the problems 3-SCM MOST COMPLICATED PRODUCT COVER and GENERAL 3-SCM MINIMUM-RESOURCE PRODUCT-SET COVER can be proved using the same logic. □

Since it is commonly believed in parameterized complexity theory that $W[t] \neq FPT$ for all $t \geq 1$, Theorem VII.8 implies that even for a moderate error bound $\epsilon > 0$, any PTAS for the problems, if exists, will become impractical.

E. Final remarks

This chapter studies the complexity issues for certain computational problems arising from the research in supply chain management, and characterizes these problems in terms of parameterized completeness in higher levels in the W-hierarchy. The research contributes to both parameterized complexity theory and to the study of supply chain management. For parameterized complexity theory, we presented the first group of natural complete problems for the classes $W[3]$ and $W[4]$, which had no known natural complete problems except the generic complete problems WCS[3] and WCS[4] and their variations. For the study of supply chain management, to the authors' knowledge, our results provide first group of precise complexity characterizations for certain computational problems in the area, which derive directly strong computational lower bounds and inapproximability results for the problems. The hardness results of these problems will provide useful information in the study of supply chain management.

A supply chain model has its units classified into different kinds, which makes it natural to map the computation in the supply chain model to that of bounded depth circuits. However, the mapping is not always straightforward and in many cases must be with care. As we have seen in the current chapter, problems on 3-tier supply chains can either correspond to the class $W[3]$, which is associated with the satisfiability problem on Π_3-circuits of 3 levels, or correspond to the class $W[4]$, which is associated with the satisfiability problem on Π_4-circuits of 4 levels.

To further clarify this phenomenon, we consider a 4-tier supply chain model based on the *e-waste recycling system* studied by Nagurney and Toyasaki [74]. An e-waste recycling system consists of four kinds of units: the *waste-sources*, the *recyclers*, the *processors*, and the *markets*. A waste-source is linked to a recycler if the waste-source sends waste to the recycler. A recycler is linked to a processor if the recycler sends materials to the processor for processing and manufacturing. A processor is linked to a market if products produced by the processor are carried by the market. The problem we are concerned here is the *market pollution* problem. We assume that a waste-source may send harmful waste (probably on purpose) to recyclers without notifying the recyclers, and that a recycler has no knowledge about if it has collected harmful waste. Therefore, a recycler that collected harmful waste will deliver harmful materials to processors that, in turn, will produce dangerous products. Finally, a market will be polluted if it carries dangerous products. Now the question is if there exist a small number of waste-sources that can send harmful waste and pollute all markets. The problem is formulated as the following parameterized problem.

E-RECYCLING AND MARKET POLLUTION:

> Given an e-waste recycling system $H(S, R, P, M, T)$ and an integer k, where S is the set of waste-sources, R is the set of recyclers, P is the set of processors, M is the set of markets, and T is the set of connections between adjacent tiers (i.e. from S to R, from R to P, or from P to M).

Is there a collection W of k waste-sources such that if all waste-sources in W send harmful waste then all markets are polluted?

Theorem VII.9 E-RECYCLING AND MARKET POLLUTION *is* $W[2]$*-complete.*

PROOF. Consider the WEIGHTED SATISFIABILITY ON MONOTONE Π_2-CIRCUITS problem, shortly WCS$^+[2]$, which is a subproblem of the problem WCS$[2]$ with an additional constraint that in the input pair (C, k) the Π_2-circuit C be monotone (i.e., all input gates of C be labeled by positive input literals). It is known [26] that the problem WCS$^+[2]$ is $W[2]$-complete. Therefore, to prove the theorem, it suffices to present fpt-reductions between the problems WCS$^+[2]$ and E-RECYCLING AND MARKET POLLUTION.

Let (C, k) be an instance of the problem WCS$^+[2]$, where C is a monotone Π_2-circuit with input variables x_1, …, x_n and OR-gates g_1, …, g_m. We construct an e-waste recycling system $H(S, R, P, M, T)$, where

- the set S consists of n waste-sources σ_1, …, σ_n;
- the set R consists of m recyclers ρ_1, …, ρ_m, where a recycler ρ_i is linked to a waste-source σ_j if and only if the OR-gate g_i in C has the variable x_j as an input (note that the circuit C is monotone);
- the set P consists of m processors π_1, …, π_m, where for each i, the processor π_i is linked to the recycler ρ_i; and
- the set M consists of m markets μ_1, …, μ_m, where for each i, the market μ_i is linked to the processor π_i.

It is easy to verify that there is a weight-k assignment ϕ satisfying the circuit C (i.e., the assignment ϕ makes all OR-gates g_1, …, g_m in C have value 1) if and only if there are k waste-sources in the e-waste recycling system $H(S, R, P, M, T)$ that can pollute all markets in M. Therefore, this is an fpt-reduction from WCS$^+[2]$ to E-RECYCLING AND MARKET POLLUTION. In consequence, the problem E-RECYCLING AND MARKET POLLUTION is $W[2]$-hard.

To show that the problem E-RECYCLING AND MARKET POLLUTION is in $W[2]$, we reduce each instance $(H(S, R, P, M, T), k)$ of the problem to an instance (C, k) of WCS$^+[2]$ as follows. For each waste-source σ_i in S, the circuit C has an input variable x_i, and for each market μ_j in M, the circuit C has an OR-gate g_j. The variable x_i is an input to the OR-gate g_j in the circuit C if and only if there is a direct path in $H(S, R, P, M, T)$ from the waste-source σ_i to the market μ_j. Again it is routine to verify that this gives an fpt-reduction from E-RECYCLING AND MARKET POLLUTION to WCS$^+[2]$, which implies that the problem E-RECYCLING AND MARKET POLLUTION is in $W[2]$. □

Theorem VII.9 shows that the computational complexity of the problems in supply chain management does not directly depend on the number of tiers in the model but is more closely related to the actual applications. In particular, the research in supply chain management has opened an area in computational complexity and optimization, and provided very rich contexts for new large-scale optimization problems that are both theoretically interesting and practically important.

CHAPTER VIII

SUMMARY AND FUTURE RESEARCH

In this dissertation, we took a topological approach in designing efficient protocols for wireless sensor networks, developing effective parameterized algorithms for a set of well-known NP-hard problems and showing their applications in bioinformatics. The protocols and algorithms discussed in this dissertation can be improved in terms of overhead and efficiency. Furthermore researches in this dissertation are a good start for future research in the area of wireless sensor networking, ubiquitous computing and bioinformatics.

A. Dissertation summary

For wireless sensor networking, we studied the separability and the existence of a power efficient spanner for general wireless sensor networks. Based on these properties we derived efficient compact routing protocols for such networks. We also designed a novel routing algorithm based on face tracing that does not rely on any network model. The face tracing based routing algorithm can utilize the knowledge of node positions but still works when such information is not available. We developed robust planarization algorithms for unlocalized wireless sensor networks. Such planarization has applications in routing, data storage, and topology discovery. We present our sorting based data storage scheme that can achieve data load balance and supports ranged query in a natural way. We conduct extensive simulations to verify the excellency of our protocols. The simulation results showed that our routing schemes, topology discovery and data storage scheme have low overhead and outperform previous approaches.

For general computational optimization, we study the color coding technique.

We present the application in solving k-path problem. Our algorithm runs in time $O(12.2^k P(n))$ and improve previous algorithm significantly. Then we extend our color coding technique to solve the enumeration problems. We showed that certain techniques include branch and bound, color coding, and bounded treewidth can be generalized to solve the enumeration version of many NP-complete problems. At last we presented a polynomial formulation of the signaling pathway problem that can produce seemingly nice results for most cases in our simulations.

B. Future work

With the continuing advancement of the mobile communication technology and the wide spreading of personal computing devices, we are entering an era of ubiquitous computing [45, 47]. All kinds of information devices will be connected together to form a ubiquitous network where information and services can be accessed at any time, anywhere by anyone. Wireless sensor networks are very good examples of ubiquitous networks. The strong combination of collecting data, performing computations and communicating in sensor networks has become a strong force for scientific research and engineering. There are many questions in ubiquitous computing waiting to be answered.

To achieve the goal of ubiquitous computing, data should be stored properly within the network to enable efficient query and retrieval. However, unlike a LAN or the Internet, ubiquitous networks usually are not nicely structured. Furthermore, many nodes in a ubiquitous network could be wireless nodes that rely on battery power. This constraint limits the computational power and communication power of the nodes and brings new challenges to us. For example, we have to carefully balance the load between nodes to avoid a single node from being quickly drained out of power. Fundamental problems like routing, data security and network dynamics are other examples of the difficulties we have to deal with in ubiquitous computing.

1. Topological structure of ubiquitous networks

Ubiquitous networks are complex networks with inner structures hidden behind their topology. To discover these structures, one does not necessarily have to seek help from expensive devices (e.g., GPS). In many cases such hidden structures can be found algorithmically. The topology of wireless networks is closely correlated to the geometrical location of each node. The topological properties of the network can often tell us enough about the structures of the network to achieve certain goals (routing etc.). Based on such structures we can then develop algorithms and protocols that do not rely on the knowledge other than the network topology itself. Furthermore, knowledge of the topological features of the network can help improve the computation/communication process even when the nodes' positions are known. However, the study of the topological properties of the underlying graph model for general wireless networks has just started. There are many works assuming that the network is deployed in a 2-D space with idealized models. I will continue my study on general network models where the network is deployed in a 3-D space and the underlying graph model of the network is more complex and realistic.

2. Efficient routing protocols for ubiquitous computing

Routing is the building block for many applications in distributed networks. In the case of ubiquitous networks, an efficient routing algorithm is essential for data storage and information retrieval schemes. Geographic routing has been very successful in wireless sensor networks where the model is restricted to UDG or some special quasi-UDGs with the knowledge of node location information. There have been a few papers addressing the case of unlocalized general sensor networks. Most of them assume either very dense networks or very large scale networks. I will focus on the realistic network model of low density and different scales.

3. Balanced data storage in ubiquitous networks

Studying of load balancing in terms of the amount of data each node stores has not been done much for in-network storage in wireless networks. The amount of data stored in a node usually directly decides how often this node will be accessed in the future. To balance the energy consumption it is very desirable to have all the nodes store the same amount of data. Furthermore, efficient ranged query, if available, will increase the quality and performance of data service significantly for ubiquitous networks. Our current research provides a very good start of efficient schemes for high quality data service.

REFERENCES

[1] N. Alon, R. Yuster, and U. Zwick, "Color-coding," *Journal of the ACM*, vol. 42, no. 4, pp. 844–856, July 1995.

[2] K. Alzoubi, X. Li, Y. Wang, P. Wan, and O. Frieder, "Geometric spanners for wireless ad hoc networks," *IEEE Trans. Parallel and Distributed Systems*, vol. 14, no. 4, pp. 408–421, 2003.

[3] J. Aspnes, D. Goldenberg, and Y. R. Yang, "On the computational complexity of sensor network localization," in *Proc. 1st International Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS)*, Turku, Finland, July 2004, pp. 32–44.

[4] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, M. Marchetti-Spaccamela, and M. Protasi, *Complexity and Approximation.* New York, NY: Springer, 1999.

[5] L. Barrière, P. Fraigniaud, and L. Narayanan, "Robust position-based routing in wireless ad hoc networks with unstable transmission ranges," in *Proc. of the 5th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Commnunications(DIAL-M)*, Rome, Italy, 2001, pp. 19–27.

[6] P. Biswas and Y. Ye, "Semidefinite programming for ad hoc wireless sensor network localization," in *Proc. of the 3rd International Symposium on Information Processing in Sensor Networks (IPSN)*, Berkeley, CA, 2004, pp. 46–54.

[7] H. L. Bodlaender, "On linear time minor tests with depth-first search," *Journal of Algorithms*, vol. 14, no. 1, pp. 1–23, 1993.

[8] H. L. Bodlaender, "Dynamic programming on graphs with bounded treewidth," in *Proc. 15th Int. Colloq. Automata, Languages and Programming*, Aalborg, Denmark, 1998, pp. 105–118.

[9] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia, "Routing with guaranteed delivery in ad hoc wireless networks," in *Proceedings of the 3rd International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIAL-M)*, Seattle, WA, Aug. 1999, pp. 48–55.

[10] T. B. Brecht and C. J. Colbourn, "Lower bounds on two-terminal network reliability," *Discrete Appl. Math*, vol. 21, no. 3, pp. 185–198, Oct. 1988.

[11] H. Breu and D. G. Kirkpatrick, "Unit disk graph recognition is NP-hard," *Computational Geometry Theory and Applications*, vol. 9, no. 1-2, pp. 3–24, Jan. 1998.

[12] J. Bruck, J. Gao, and A. Jiang, "Localization and routing in sensor networks by local angle information," in *Proc. 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, Urbana-Champaign, IL, May 2005, pp. 181–192.

[13] J. Carter and M. Wegman, "Universal classes of hash functions," *Journal of Computer and System Sciences*, vol. 18, no. 2, pp. 143–154, Apr. 1979.

[14] M. Cesati, "Compendium of parameterized problems," Dept. Computer Science, Systems, and Industrial Eng.,University of Rome "Tor Vergata", 2006, Available at http://bravo.ce.uniroma2.it/home/cesati/research/compendium.pdf.

[15] J. Cheetham, F. Dehne, A. Rau-Chaplin, U. Stege, and P. Taillon, "Solving large FPT problems on coarse grained parallel machines," *Journal of Computer and*

System Sciences, vol. 67, no. 4, pp. 691–706, 2003.

[16] J. Chen, "Parameterized computation and complexity: a new approach dealing with NP-hardness," *Journal of Computer Science & Technology*, vol. 20, no. 1, pp. 18–37, Jan. 2005.

[17] J. Chen, D. Friesen, W. Jia, and I. Kanj, "Using nondeterminism to design efficient deterministic algorithms," *Algorithmica*, vol. 40, no. 2, pp. 83–97, 2004.

[18] J. Chen, X. Huang, I. Kanj, and G. Xia, "Linear fpt reductions and computational lower bounds," in *Proceedings of the 36th ACM Symposium on Theory of Computing (STOC)*, Chicago, IL, 2004, pp. 212–221.

[19] J. Chen, I. Kanj, and W. Jia, "Vertex cover: further observations and further improvements," *Journal of Algorithms*, vol. 41, no. 2, pp. 280–301, 2001.

[20] S. Chopra and P. Meindl, *Supply Chain Management: Strategy, Planning, and Operations*. Upper Saddle River, NJ: Prentice-Hall, 2001.

[21] B. N. Clark, C. J. Colbourn, and D. S. Johnson, "Unit disk graphs," *Discrete Mathematics*, vol. 86, no. 1-3, pp. 165–177, 1990.

[22] M. C. Cooper, D. M. Lambert, and J. D. Pagh, "Supply chain management: More than a new name for logistics," *The International Journal of Logistics Management*, vol. 8, no. 1, pp. 1–13, 1997.

[23] R. Davidson and D. Harel, "Drawing graphs nicely using simulated annealing," *ACM Transactions on Graphics*, vol. 15, no. 4, pp. 301–331, Oct. 1996.

[24] P. Desnoyers, D. Ganesan, and P. Shenoy, "Tsar: a two tier sensor storage architecture using interval skip graphs," in *Proc. 3rd ACM Conference on Embedded Networked Sensor Systems (SenSys)*, San Diego, CA, 2005, pp. 39–50.

[25] R. Diestel, *Graph Theory*, 2nd ed., ser. Graduate Texts in Mathematics 173. New York, NY: Springer-Verlag, 2000.

[26] R. Downey and M. Fellows, *Parameterized Complexity.* New York, NY: Springer, 1999.

[27] V. Dujmovic, M. Fellows, M. Hallett, M. Kitching, G. Liotta, C. McCartin, N. Nishimura, P. Ragde, F. Rosamond, and M. Suderman, "A fixed-parameter approach to two-layer planarization," *Algorithmica*, vol. 45, no. 2, pp. 159–182, 2006.

[28] P. Eades and S. Whitesides, "Drawing graphs in two layers," *Theoretical Computer Science*, vol. 131, no. 2, pp. 361–374, 1994.

[29] Q. Fang, J. Gao, and L. J. Guibas, "Landmark-based information storage and retrieval in sensor networks," in *Proc. 25th IEEE International Conference on Computer Communications (INFOCOM)*, Barcelona, Catalunya, Spain, 2006, pp. 1–12.

[30] U. Feige and J. Kilian, "Nonconstructive tools for proving polynomial-time decidability," *Journal of the ACM*, vol. 35, no. 3, pp. 727–739, 1988.

[31] M. R. Fellows, C. Knauer, N. Nishimura, P. Ragde, F. A. Rosamond, U. Stege, D. M. Thilikos, and S. Whitesides, "Faster fixed-parameter tractable algorithms for matching and packing problems," in *Proceedings 12th Annual European Symposium (Algorithms - ESA)*, ser. Lecture Notes in Computer Science, S. Albers and T. Radzik, Eds., vol. 3221. Bergen, Norway: Springer, Sept. 2004, pp. 311–322.

[32] G. Frederickson, "A distributed shortest path algorithm for a planar network," *Journal of Information and Computation*, vol. 86, no. 2, pp. 140–159, 1990.

[33] G. Frederickson, "Space efficient message routing in c-decomposable networks," *SIAM Journal on Computing*, vol. 19, no. 1, pp. 164–181, 1990.

[34] M. Fredman, J. Komlos, and E. Szemeredi, "Storing a sparse table with $o(1)$ worst case access time," *Journal of the ACM*, vol. 31, no. 3, pp. 538–544, July 1984.

[35] S. Funke, L. Guibas, A. Nguyen, and Y. Wang, "Distance-sensitive routing and information brokerage in sensor networks," in *Proc. 2006 International Conference on Distributed Computing in Sensor Systems (DCOSS)*, San Francisco, CA, 2006, pp. 234–251.

[36] S. Funke and C. Klein, "Hole detection or: "how much geometry hides in connectivity?"," in *Proc. 22nd Annual ACM Symposium on Computational Geometry (SCG)*, Sedona, AZ, 2006, pp. 377–385.

[37] S. Funke and N. Milosavljevic, "Network sketching or: "how much geometry hides in connectivity? -part ii"," in *Proc. Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, New Orleans, LA, 2007, pp. 958–967.

[38] K. Gabriel and R. Sokal, "A new statistical approach to geographic variation analysis," *Systematic Zoology*, vol. 18, no. 3, pp. 259–278, 1969.

[39] D. Ganesan, D. Estrin, and J. Heidemann, "Dimensions: Why do we need a new data handling architecture for sensor networks," in *Proc. of the ACM Workshop on Hot Topics in Networks*, Princeton, NJ, 2002., pp. 143–148.

[40] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker, "Complex behavior at scale: an experimental study of low-power wireless sensor networks," Los Angeles, CA, 2002, technical Report UCLA/CSD-TR 02-0013, UCLA.

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

[42] C. Gavoille, D. Peleg, S. Pèrennes, and R. Raz, "Distance labeling in graphs," *Journal of Algorithms*, vol. 53, no. 1, pp. 85–112, 2004.

[43] J. Genues and P. Pardalos, "Network optimization in supply chain management and financial engineering: an annotated bibliography," *Networks*, vol. 42, no. 2, pp. 66–84, 2003.

[44] J. Genues, P. Pardalos, and H. R. (Editors), *Supply Chain Management: Models, Applications, and Research Directions.* New York, NY: Springer, 2002, Kluwer Series in Applied Optimization **62**.

[45] A. Greenfield, *Everyware: the Dawning Age of Ubiquitous Computing.* Indianapolis, IN: New Riders Publishing, 2006.

[46] J. Gross and T. Tucker, *Topological Graph Theory.* New York, NY: Wiley-Interscience, 1987.

[47] U. Hansmann, L. Merk, M. Nicklous, and T. Stober, *Pervasive Computing: The Mobile World.* New York, NY: Springer, 2003.

[48] G. Hardy and E. Wright, *An Introduction to the Theory of Numbers*, 5th ed. Oxford: Oxford University Press, 1979.

[49] W. Jia, C. Zhang, and J. Chen, "An efficient parameterized algorithm for m-set packing," *Journal of Algorithms*, vol. 50, no. 1, pp. 106–117, 2004.

[50] I. A. Kanj and L. Perkovic, "Improved stretch factor for bounded-degree planar power spanners of wireless ad-hoc networks," in *Proc. 2nd International Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS)*, Venice, Italy, 2006, pp. 95–106.

[51] B. Karp and H. T. Kung, "GPSR: Greedy perimeter stateless routing for wireless networks," in *Proc. of the 6th Annual International Conference on Mobile Computing and Networking (MobiCom)*, Boston, MA, 2000, pp. 243–254.

[52] B. Kelley, R. Sharan, R. Karp, T. Sittler, D. Root, B. Stockwell, and T. Ideker, "Conserved pathways within bacteria and yeast as revealed by global protein network alignment," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 100, pp. 11 394–11 399, 2003.

[53] M. Khouja, "Optimizing inventory decisions in multi-stage multi-customer supply chain," *Transportation Research Part E*, vol. 39, no. 3, pp. 193–208, 2003.

[54] Y. Kim, R. Govindan, B. Karp, and S. Shenker, "Geographic routing made practical," in *Proc. of the 2nd USENIX/ACM Symposium on Networked System Design and Implementation (NSDI)*, Boston, MA, 2005, pp. 217–230.

[55] J. Kneis, D. Mölle, S. Richter, and P. Rossmanith, "Divide-and-color," in *Proc. 32nd International Workshop Graph-Theoretic Concepts in Computer Science (WG)*, Bergen, Norway, 2006, pp. 58–67.

[56] I. Koutis, "A faster parameterized algorithm for set packing," *Information Processing Letters*, vol. 94, no. 1, pp. 7–9, 2005. [Online]. Available:

http://dx.doi.org/10.1016/j.ipl.2004.12.005

[57] E. Kranakis, H. Singh, and J. Urrutia, “Compass routing on geometric networks,” in *Proc. 11th Canadian Conference on Computational Geometry (CCCG)*, Vancouver, BC, Canada, 1999, pp. 51–54.

[58] F. Kuhn, T. Moscibroda, and R. Wattenhofer, “Unit disk graph approximation,” in *Proc. Joint Workshop on Foundations of Mobile Computing (DIAL-M)*, Philadelphia, PA, 2004, pp. 17–23.

[59] F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger, “Geometric ad-hoc routing: Of theory and practice,” in *Proc. 22nd ACM Symposium on Principles of Distributed Computing (PODC)*, Boston, MA, 2003, pp. 63–72.

[60] F. Kuhn, R. Wattenhofer, and A. Zollinger, “Worst-case optimal and average-case efficient geometric ad-hoc routing,” in *Proc. 4th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, Annapolis, MD, 2003, pp. 267–278.

[61] F. Kuhn, R. Wattenhofer, and A. Zollinger, “Ad-hoc networks beyond unit disk graphs,” in *Proceedings of the 2003 Joint Workshop on Foundations of Mobile Computing (MobiCom)*, San Diego, CA, 2003, pp. 69–68.

[62] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. San Mateo, CA: M. Kaufmann Publishers, 1992.

[63] J. Li, J. Jannotti, D. DeCouto, D. Karger, and R. Morris, “A scalable location service for geographic ad-hoc routing,” in *Proc. the 6th Annual International Conference on Mobile Computing and Networking (MobiCom)*, Boston, MA, 2000, pp. 120–130.

[64] X. Li, Y. J. Kim, R. Govindan, and W. Hong, “Multi-dimensional range queries in sensor networks,” in *Proc. 1st ACM Conference on Embedded Networked Sensor Systems (SenSys)*, Los Angeles, CA, 2003, pp. 63–75.

[65] R. J. Lipton and R. E. Tarjan, “A separator theorem for planar graphs,” *SIAM Journal on Applied Mathematics*, vol. 36, no. 2, pp. 177–189, 1979.

[66] Y. Liu, S. Lu, J. Chen, and S.-H. Sze, “Greedy localization and color-coding: improved matching and packing algorithms,” in *Proceedings 2nd International Workshop on Parameterized and Exact Computation (IWPEC)*, Zurich, Switzerland, 2006, pp. 84–95.

[67] Y. Malka, S. Moran, and S. Zaks, “A lower bound on the period length of a distributed scheduler,” *Algorithmica*, vol. 10, no. 5, pp. 383–398, 1993.

[68] L. Mathieson, E. Prieto, and P. Shaw, “Packing edge disjoint triangles: a parameterized view,” in *Proceedings 1st International Workshop on Parameterized and Exact Computation (IWPEC)*, Bergen, Norway, 2004, pp. 127–137.

[69] Z. Michalewicz and D. B. Fogel, Eds., *How to Solve It: Modern Heuristics.* Berlin, Germany: Springer, 2000.

[70] H. Min and G. Zhou, “Supply chain modeling: past, present and future,” *Computers & Industrial Engineering*, vol. 43, no. 1-2, pp. 231–249, July 2002.

[71] B. Monien, “How to find long paths efficiently,” *Annals of Discrete Mathematics*, vol. 25, pp. 239–254, 1985.

[72] D. Moore, J. Leonard, D. Rus, and S. Teller, “Robust distributed network localization with noisy range measurements,” in *Proc. 2nd ACM Conference on*

Embedded Networked Sensor Systems (SenSys), Baltimore, MD, 2004, pp. 50–61.

[73] R. Motwani and P. Raghavan, Eds., *Randomized Algorithms.* New York, NY: Cambridge University Press, 1995.

[74] A. Nagurney and F. Toyasaki, "Reverse supply chain management and electronic waste recycling: A multitiered network equilibrium framework for e-cycling," *Transportation Research Part E: Transportation and Logistics Review*, vol. 41, no. 1, pp. 1–28, 2005.

[75] S. Nakano, "Planar drawings of plane graphs," *IEICE Transactions on Information and Systems*, vol. E83, no. 3, pp. 384–391, 2000.

[76] J. Newsome and D. Song, "Gem: Graph embedding for routing and data centric storage in sensor networks without geographic information," in *Proc. 1st ACM Conference on Embedded Networked Sensor Systems (SenSys)*, Los Angeles, CA, 2003, pp. 76–88.

[77] T. Nieberg, J. Hurink, and W. Kern, "A robust ptas for maximum weight independent sets in unit disk graphs," in *Proc. 30th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, Bad Honnef, Near Bonn, Germany, 2004, pp. 214–221.

[78] C. Papadimitriou and M. Yannakakis, "On limited nondeterminism and the complexity of the V-C dimension," *Journal of Computer and System Sciences*, vol. 53, no. 2, pp. 161–170, 1996.

[79] A. Pothen, "Graph partitioning algorithms with applications to scientific computing," in *Parallel Numerical Algorithms*, D. E. Keyes, A. H. Sameh, and

V. Venkatakrishnan, Eds. Kluwer Academic Press, 1997, pp. 323–368. [Online]. Available: citeseer.ist.psu.edu/pothen97graph.html

[80] E. Prieto and C. Sloper, "Looking at the stars," *Theoretical Computer Science*, vol. 351, no. 3, pp. 437–445, 2006.

[81] S. Ratnasamy, L. Yin, F. Yu, D. Estrin, R. Govindan, B. Karp, and S. Shenker, "GHT: A geographic hash table for data-centric storage," in *Proc. 1st ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*, Atlanta, GA, 2002, pp. 78–87.

[82] K. Ŕomer and F. Mattern, "The design space of wireless sensor networks," *IEEE Wireless Communications*, vol. 11, no. 6, pp. 54–61, 2004.

[83] R. Sarkar, X. Zhu, and J. Gao, "Double rulings for information brokerage in sensor networks," in *Proc. Annual International Conference on Mobile Computing and Networking (MobiCom)*, Los Angeles, CA, 2006, pp. 286–297.

[84] M. Schäfer, "Deciding the Vapnik-Cervonenkis dimension is Σ_3^p-complete," *Journal of Computer and System Sciences*, vol. 58, no. 1, pp. 177–182, 1999.

[85] M. Schäfer and C. Umans, "Completeness in the polynomial-time hierarchy: part i: a compendium," *SIGACT News*, vol. 33, no. 3, pp. 32–49, 2002.

[86] M. Schäfer and C. Umans, "Completeness in the polynomial-time hierarchy: part ii," *SIGACT News*, vol. 33, no. 4, pp. 22–36, 2002.

[87] J. Schmidt and A. Siegel, "The spatial complexity of oblivious k-probe hash functions," *SIAM Journal on Computing*, vol. 19, no. 5, pp. 775–786, Oct. 1990.

[88] J. Scott, T. Ideker, R. Karp, and R. Sharan, “Efficient algorithms for detecting signaling pathways in protein interaction networks,” *Journal of Computational Biology*, vol. 13, no. 2, pp. 133–144, Mar. 2006.

[89] B. Sheng, Q. Li, and W. Mao, “Data storage placement in sensor networks,” in *Proc. 7th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, Florence, Italy, 2006, pp. 344–355.

[90] J. Sheu, “Locating manufacturing and distribution centers: an integrated supply chain-based spatial interaction approach,” *Transportation Research Part E*, vol. 39, no. 5, pp. 381–397, Sept. 2003.

[91] D. Simchi-Levi, P. Kaminsky, and E. Simchi-Levi, *Designing and Managing the Supply Chain: Concepts, Strategies, and Case Studies.* Boston, MA: Irwin McGraw-Hill, 2000.

[92] A.-C. So and Y. Ye, “Theory of semidefinite programming for sensor network localization,” *Mathematical Programming: Series A and B*, vol. 109, no. 2, Jan. 2007.

[93] K. Sohrabi, B. Manriquez, and G. Pottie, “Near ground wideband channel measurement in 800-1000 MHz,” *IEEE Vehicular Technology Conference*, vol. 1, pp. 571–574, 1999.

[94] J. Song and d. Yao (Editors), *Supply Chain Structures: Coordination, Information, and Optimization.* Boston, MA: Kluwer Academic Publishers, 2001, Kluwer International Series in Operations Research and Management Science **24**.

[95] W.-Z. Song, X.-Y. Li, Y. Wang, , and O. Frieder, “Localized algorithms for

energy efficient topology in wireless ad hoc networks," *Mobile Networks and Applications*, vol. 10, no. 6, pp. 911–923, 2005.

[96] G. Toussaint, "The relative neighborhood graph of a finite planar set," *Pattern Recognition*, vol. 12, no. 4, pp. 261–268, 1980.

[97] Y. Wang, J. Gao, and J. S. Mitchell, "Boundary recognition in sensor networks by topological methods," in *Proc. Annual International Conference on Mobile Computing and Networking (MobiCom)*, Los Angeles, CA, 2006, pp. 122–133.

[98] F. Zhang, A. Jiang, and J. Chen, "Robust planarization of unlocalized sensor networks," in *Proc. 27th IEEE International Conference on Computer Communications (INFOCOM)*, Phoenix, Arizona, 2008, pp. 798–806.

[99] F. Zhang, H. Li, A. Jiang, J. Chen, and P. Luo, "Face tracing based geographic routing in nonplanar wireless networks," in *Proc. 26th IEEE International Conference on Computer Communications (INFOCOM)*, Achorage, AK, 2007, pp. 2243–2251.

VITA

The author, Fenghui Zhang, received his B.S. degree in mathematics from Fudan University, China in August 1997.

Mr. Zhang started working toward his Ph.D. degree in computer science in January 2004, under the supervision of Dr. Jianer Chen and Dr. Anxiao (Andrew) Jiang. Mr. Zhang's research interests include wireless sensor networking, bioinformatics, design and analysis of Algorithms, graph theory, computational optimization, and complexity theory.

Mr. Zhang's current address is: Department of Computer Science, Texas A&M University, College Station, TX 77843-3112.

The typist for this dissertation was Fenghui Zhang.