Stony Brook University

Distributed Algorithms for Online Coordination in Wireless Sensor Networks

A Dissertation Presented
by
Dengpan Zhou

to
The Graduate School
in Partial Fulfillment of the
Requirements
for the Degree of
Doctor of Philosophy
in
Computer Science
Stony Brook University

May 2012

Stony Brook University
The Graduate School

Dengpan Zhou

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

Jie Gao, Dissertation Advisor
Professor, Computer Science Department

Samir Das, Chairperson of Defense
Professor, Computer Science Department

Xianfeng Gu
Professor, Computer Science Department

Sangjin Hong
Professor, Department of Electrical and Computer Engineering

This dissertation is accepted by the Graduate School

Charles Taber
Interim Dean of the Graduate School

Abstract of the Dissertation

Distributed Algorithms for Online Coordination in Wireless Sensor Networks

by

Dengpan Zhou

Doctor of Philosophy

in

Computer Science

Stony Brook University

2012

With the rapid development of large-scale wireless sensor networks in the past few years, we expect the embedded sensors to be integrated smoothly with other mobile embedded devices. In this dissertation, we consider the following model of a hybrid network with both static and mobile nodes. There are pervasive static sensor nodes embedded in the environment to gather real-time data. The mobile nodes can be either robots with controlled mobility to aid the network operation and repair dysfunctional network components, or users of the sensor network that demand real-time knowledge gathered by the sensor nodes, or robots/users that use the sensor network as a communication infrastructure, or a mixture of the above. The specific scenarios include, but are not limited to, online resource management and allocation, maintaining group communication and coordination of mobile agents, and efficient and resilient routing schemes.

To solve these problems, we introduce a framework to manage the efficient and highly selective information flow between the sensor nodes and the mobile nodes. This framework involves the following components:

1. We extract a hierarchical well separated tree (HST) to approximate the shortest path metric of the static sensor network.
2. With the HST, we allow spontaneous, distributed matching between users that may emerge anywhere and the resources available in the network.

3. We also show that in the same framework, we can coordinate mobile users by maintaining an approximate minimum Steiner tree with modest communication cost.
4. By using two or multiple HSTs, we also show how to support low-stretch routing that is also resilient to in-transit link failures.

In addition to the above HST framework, we develop the compact conformal map for greedy routing in wireless mobile sensor networks. The map is only dependent on the network domain and is independent of the network connectivity. This is the first practical solution for using virtual coordinates for greedy routing in a sensor network and could be easily extended to the case of a mobile network.

Contents

List of Tables

List of Figures

Acknowledgements

I would like to recognize a number of people who have made my experience at Stony Brook so remarkable.

Firstly, I would like to express my sincere gratitude to my supervisor, Prof. Jie Gao, whose patience and kindness, as well as her academic experience, have been invaluable to me. She has provided me with wonderful guidance and continuous support over these years. With her enthusiasm, her inspiration, and her great efforts to explain things clearly and simply, she helped to make research fun for me. I also greatly appreciate her assistance in writing reports and presenting works.

Also I want to thank the rest of my thesis committee, Prof. Samir Das, Prof. Xianfen Gu, Prof. Sangjin Hong, for their encouragement, insightful comments, and hard questions.

I want to thank all my friends at Stony Brook who helped me in various ways. Without them my life at Stony Brook would not be so colorful and fun.

Finally, I want to thank my parents, my sister and brother for the support they provided me through my entire life, and for their understanding, endless patience and encouragement when it was most required.

Publications

1. Dengpan Zhou, Jie Gao, Opportunistic Processing and Query of Motion Trajectories in Wireless Sensor Networks, Proc. of the 28th Annual IEEE Conference on Computer Communications (INFOCOM'09), 1197-1205, April, 2009.
2. Jie Gao, Leonidas J. Guibas, Nikola Milosavljevic, Dengpan Zhou, Distributed Resource Management and Matching in Sensor Networks, Proc. of the 8th International Symposium on Information Processing in Sensor Networks (IPSN'09), 97-108, April, 2009.
3. Dengpan Zhou, Jie Gao, Maintaining Approximate Minimum Steiner Tree and k-center for Mobile Agents in a Sensor Network, Proc. of the 29th Annual IEEE Conference on Computer Communications (INFOCOM'10), 511-515, mini-conference, March, 2010.
4. Jie Gao, Dengpan Zhou, The Emergence of Sparse Spanners and Greedy Well Separated Pair Decomposition, Proc. of the the 12th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT'10), 50-61, June, 2010. Journal version under the title The Emergence of Sparse Spanners and Well-separated Pair Decomposition Under Anarchy appeared in Journal of Computational Geometry, Vol 3, No 1 (2012).
5. Jie Gao, Dengpan Zhou, Resilient and Low Stretch Routing Through Embedding into Tree Metrics, Proc. of the 12th Algorithms and Data Structures Symposium (WADS'11), 438-450, August, 2011.
6. Jie Gao, Xianfeng Gu, Siming Li, Wei Zeng, Dengpan Zhou, Compact Conformal Map for Greedy Routing in Wireless Mobile Sensor Networks, submitted(in alphabetical order).

7. Xiaotian Yin, Wei Han, Dengpan Zhou, Xianfeng Gu and Jie Gao, Decentralized Path Homotopy Detection Using Hodge Decomposition in Sensor Networks, submitted.

Chapter 1

Introduction

1.1 Introduction

Integrated low-power sensing devices will greatly help remote object monitoring and tracking in many different scenarios and environments. These sensors are empowered with the ability to coordinate among themselves for communication, information collecting , distributing and processing. One promising application is to deploy sensors in inhospitable physical environments for monitoring purposes, such as remote fire-prone forest area or security applications.

The development of fundamental algorithms for a hybrid system with intelligent sensors and mobile physical agents has broad impact of social values. Sensor network research in the past few years has matured to a certain level that large-scale deployments start to become possible. As the state of the art, networks in the size of hundreds of nodes are common practice; thousands of nodes start to appear, hundreds of thousands are expected to be done in a couple of years. As sensor networks scale and cover the physical domain that people live and work, using the network simply for passive monitoring and data collection does not fully reach the network's full capacity. We expect the embedded sensors to be integrated smoothly with other mobile embedded devices so as to provide real-time data collection, knowledge extraction, environment understanding and eventually evolve into an intelligent component of cyber physical system.

In this dissertation we consider the following model of a hybrid network with

both static and mobile nodes. There are pervasive static sensor nodes embedded in the environment that gather real-time data. The sensor nodes may communicate with other nodes (either sensor nodes or mobile nodes) within communication range. The mobile nodes may be either robots with controlled mobility to aid the network operation and repair dysfunctional network components, or users of the sensor network that demand real-time knowledge gathered by the sensor nodes, or robots/users that use the sensor network as a communication infrastructure, or a mixture of the above.

For a large-scale sensor network that largely relies on battery power, energy conservation is a must and balanced energy usage is an important performance metric. The algorithms that the sensors execute must be light-weight and decentralized that intelligently use network resources (computation, communication and energy mainly). Obviously one can have a centralized server to handle all the coordination necessary and supply a query interface for users of the network. But this centralized solution represents a single point of failure, is not resilient to attacks and is not efficient. First of all, users might be in a neighborhood in which data is generated. Indeed users are most interested in data in close proximity. A centralized architecture would require both the data in the network and queries from the users to be delivered to a remote server. Secondly, when users/agents move around, the solution for the new positions of the agents might be very similar to the previous solution and thus can be adjusted slightly. A centralized solution would require a new cycle of data gathering, computation of a new solution, and the dissemination of the solution to the users, which is clearly not power efficient. Thirdly, the help of an underlying sensor network may provide new insights to problems that are difficult to solve even in the centralized setting.

1.2 Motivation and Objective

This thesis work is motivated by the recent advance and maturity of static wireless sensor networks for environment monitoring and prosperity of seamless integrating physical users and/or autonomous robots into a hybrid intelligent framework. The focus is on the joint information processing, optimization and coordination of both static and mobile nodes to comprehend and act on the environment.

Let's list a few examples for application scenarios captured by this hybrid network.

1. Mobile robots as aids to sensor network operation. A set of robots are employed to manage a remote wireless sensor networks. The robots implement the management task, including collection and delivery of sensor data, network operational maintenance such as battery replacement, node relocation, etc. The robots need to coordinate among themselves in order to optimize for energy usage regarding motion planing and task assignment.
2. Online resource management and allocation. Wireless sensors are deployed in Downtown Manhattan to monitor street parking spots. A sensor at a spot can tell whether the spot is taken or freed up. Drives may communicate with a nearby sensor node to ask for parking recommendation. Multiple vehicles may arrive at the same time and they will need to be assigned to available parking spots immediately, without conflict. To coordinate among multiple emerging vehicles and arrive at a matching of vehicles and available spots with a minimum total travel distance, a communication efficient method is needed for the system.
3. Maintaining group communication and coordination of mobile agents. In a scenario, we have a set of rescue workers collaborating on emergency tasks in an environment in which wireless sensor nodes are densely deployed. An existing communication infrastructure might not be available in such a disaster recovery setting , the workers will rely on the underlying wireless sensor network for coordination and communication. One way to achieve this is to maintain a group communication structure, e.g., a spanning tree connecting all workers, in the sensor network for the mobile rescue workers to continuously exchange information and commands.

The above examples cover several different application categories. We are going to answer most of these questions in latter chapters. One common theme behind these problems is the tight coupling and frequent information between static monitoring sensor nodes and mobile actionable agents. The optimal design for data collection via static sensor networks, and the optimal coordination and planning of mobile agents have been separated topics of study in the sensor network community and robotics community respectively. However the considered optimization or coordination problem of the mobile agents is often known to be theoretically difficulty.

For example, maintaining a minimum Steiner tree for mobile agents or computing the traveling salesman tour for data mules are well known NP-hard problems. The biggest challenge is to solve the information brokerage problem, in which sensors detecting interesting local events and the mobile users seeking such information are not aware of one another.

Basically we are going to examine the challenges arising from the interaction of static wireless sensor networks and mobile agents. There are two basic tasks for such kinds of applications. First of all, we must have an efficient way to organize the mobile agents and keep them communicate with each other well. Secondly, depending on the specific application, we should coordinate the mobile agents efficiently. One of the key requirements for sensor networks is to support for very large numbers of unattended autonomous nodes and adapt to environment and task dynamics. To aid the coordination tasks, we ask how sensor data should be stored and processed in the network, and what type of distributed structures should be maintained, to best serve mobile interactive users. Some naive or simple greedy algorithms do not work well here. To aid the application, we introduce a special tree metric, the hierarchically well-separated tree, which approximates the original metric with distortion at most $O(\lg n)$, where n is the size of the network. Many problems are easier on trees, and it is easy to aggregate information on trees. Based on this special tree metric, we are able to solve the above problems in an easy and elegant way.

Routing is one of the most important and challenging tasks in wireless sensor networks. Due to the resource constraints of sensor nodes, efficient routing mechanisms are desirable for wireless sensor networks. To cope with the mobility nature of wireless sensor networks, we are interested in light-weighted routing algorithms that can quickly and efficiently respond to unpredictable topological changes.

1.3 Overview

In Chapter 2: We briefly discuss the tree metric and introduce a special tree metric, i.e., Hierarchically Well-separated Trees (HSTs). We review the centralized top-down algorithm for computing an HST. Then we present our bottom-up distributed algorithm for the same HST computation.

In Chapter 3: We consider a scenario in which there are *resources* at or near nodes of a network, which are either static (e.g. fire stations, parking spots) or mobile (e.g. police cars). Over time, *events* (fires, crime reports, cars looking for parking) arise one-by-one at arbitrary nodes, and need to be quickly matched to and served by an appropriate nearby resource, without knowledge of future requests, and without the ability to alter any decision once it has been made.

We develop distributed algorithms to direct available resources in the network to these events (or vice versa) in a coordinated fashion, so that no two resources are assigned to the same event, and the total distance of the events from their matched resources is minimized. The key idea is to extract, in a preprocessing stage, a *well-separated tree metric* that approximates the original network metric by a logarithmic distortion, allowing greedy matching algorithms to generate close to optimal matchings, and enabling communication-efficient probing-based algorithms for events to detect nearby available resources. The distributed matching algorithm requires no global coordination and achieves polylogarithmic performance ratio in both online and offline settings. Simulation experiments corroborate the theoretical results on solution quality and further evaluate the communication costs of our scheme in practice.

In Chapter 4: We study the problem of maintaining group communication between m mobile agents, tracked and helped by n static networked sensors. We develop algorithms to maintain a $O(\lg n)$-approximation to the minimum Steiner tree of the mobile agents such that the maintenance message cost is on average $O(\lg n)$ for each hop an agent moves. The key idea is to extract a 'hierarchical well-separated tree (HST)' on the sensor nodes such that the tree distance approximates the sensor network hop distance by a factor of $O(\lg n)$. We then prove that maintaining the subtree of the mobile agents on the HST uses logarithmic messages per hop movement. With the HST we can also maintain $O(\lg n)$ approximate k-center for the mobile agents with the same message cost. Both the minimum Steiner tree and the k-center problems are NP-hard and our algorithms are the first efficient algorithms for maintaining approximate solutions in a distributed setting.

In Chapter 5: Given a network, the simplest routing scheme is probably routing on a spanning tree. This method however does not provide good stretch — the route between two nodes can be much longer than their shortest distance, nor does it give

good resilience — one node failure may disconnect quadratically many pairs. In this chapter we use two trees to achieve both constant stretch and good resilience. Given a metric (e.g., as the shortest path metric of a given communication network), we build two *hierarchical well-separated trees* such that for any two nodes u, v, the shorter path of the two paths in the two respective trees gives a *constant* stretch of the metric distance of u, v, and the removal of any node only disconnect the routes between $O(1/n)$ fraction of all pairs. This result holds true as long as the metric follows certain geometric growth rate (the number of nodes within distance r is a polynomial function of r), which holds for many realistic network settings such as wireless ad hoc networks and Internet backbone graphs. Besides the theoretical results, we also evaluate the routing performance in wireless sensor networks and the Internet.

In Chapter 6: Motivated by mobile sensor networks as in participatory sensing applications, we are interested in developing a practical, lightweight solution for routing in a mobile network. While greedy routing is robust to mobility, location errors and link dynamics, it may get stuck in a local minimum, which then requires non-trivial recovery methods. We follow the approach taken by Sarkar et. al. [IPSN 2009] to find an embedding of the network such that greedy routing using the virtual coordinates guarantees delivery, thus eliminating the necessity of any recovery methods. Our new contribution is to replace the in-network computation of the embedding by a preprocessing of the domain before network deployment and encode the map of the network domain to virtual coordinate space by using a small number of parameters which can be pre-loaded to all sensor nodes. As a result, the map is only dependent on the network domain and is independent of the network connectivity. Each node can directly compute its virtual coordinates by applying the locally stored map on its geographical coordinates. This represents the first practical solution for using virtual coordinates for greedy routing in a sensor network and could be easily extended to the case of a mobile network. When nodes move around, by their GPS location or displacement from previous location each node updates it own virtual coordinates and there is no need of any in-network computation/communication for maintenance of the embedding. The paper describes algorithmic innovations as well as simulations and implementations on a real testbed to support our claims.

In Chapter 7: A spanner graph on a set of points in $\mathbb{R}^d$ provides shortest paths between any pair of points with lengths at most a constant factor of their Euclidean distance. A spanner with a sparse set of edges is thus a good candidate for network backbones, as desired in many practical scenarios such as the transportation network and peer-to-peer network overlays. In this paper we investigate new models and aim to interpret why good spanners 'emerge' in reality, when they are clearly built in pieces by agents with their own interests and the construction is not coordinated. Our main result is to show that the following algorithm generates a $(1+\varepsilon)$-spanner with a linear number of edges, constant average degree, and the total edge length a small logarithmic factor of the cost of the minimum spanning tree. In our algorithm, the points may build edges at an *arbitrary* order. When a point p checks on whether the edge to a point q should be built, it will build this edge if and only if there is no existing edge $p'q'$ with p' and q' at distances no more than $\frac{1}{4(1+1/\varepsilon)} \cdot |pq|$ from p, q respectively. Eventually when all points have finished checking edges to all other points, the resulted collection of edges forms a sparse spanner as desired. This new spanner construction algorithm has applications in the construction of and local routing on nice network topologies for peer-to-peer systems, when peers join and leave the network and has only limited information about the rest of the network.

As a side product, we show a simple algorithm for constructing linear-size well-separated pair decompositions that may be of interest on its own. A well-separated pair decomposition is a collection of subset pairs such that each pair of point sets is fairly far away from each other compared with their diameters and that every pair of points is 'covered' by at least one well-separated pair. Our algorithm selects an *arbitrary* pair of points that is not yet covered and puts a 'dumb-bell' around the pair as the well-separated pair, repeats this until all pairs of points are covered. At this point, we show only a linear number of pairs is generated, which is asymptotically optimal.

In Chapter 8: We study the problem of in-network processing and queries of trajectories of moving targets in a sensor network. The main idea is to exploit the spatial coherence of target trajectories for opportunistic information dissemination with no or small extra communication cost, as well as for efficient probabilistic queries searching for a given target signature in a real-time manner. Sensors near

a moving target are waken up to record information about this target and take the communication opportunities to exchange their knowledge with preceding and descending sensor nodes along the trajectory. Thus a moving target's information is naturally detected, recorded, and disseminated along its trajectory, as well as the motion trajectories that enter the sensor field afterwards.

We analyzed and through simulations tested the dissemination cost and query success rate for randomly generated data sets. Trajectories of reasonable length can be discovered by probabilistic in-network queries with high probability. Compared with the scheme without opportunistic dissemination, the in-network processing of trajectories, with modest cost on dissemination, allows substantially reduced query cost and delay.

1.4 Reference

Parts of the materials included in this dissertation have been or will be published in international conferences, and thus are under copyright. We list them in the publication section.

Chapter 2

Introduction to Hierarchically Well-separated Tree and Its Distributed Implementation

2.1 Introduction

Approximation algorithm [154] is suitable and attractive for a lot of practical applications, especially for important NP-hard problems. Approximation algorithm efficiently produces a sub-optimal solution. One of the techniques for good approximation algorithm is through approximating metric spaces.

Approximating metric spaces by more simple metric spaces has many algorithmic applications. One particular choice of such simple metrics is the tree metric, i.e. a metric defined by the shortest path distance on a tree containing the given point set. Many problems would be greatly simplified and easy on special tree metrics. For problems on general metrics, we could get a simplified version of the problem through embedding into tree metrics. The original problem is well approximated by solving the tree version if the tree metric is not far away from the original metric.

Given a metric (P,d) we embed it to a hierarchically well-separated tree (HST), defined as a rooted weighted tree such that: the weight of all edges between a node and its children are the same; the edge weights along any path from the root to a leaf are decreasing by a factor of α. In this paper we simply take $\alpha = 2$. The leaf

nodes of the HST are 1-to-1 mapped to nodes in P and internal nodes of the HST are also mapped to nodes of P although certain nodes may appear multiple times. The embedding of (P,d) into the tree metric leads to distortions of the metric distances. As discussed earlier, using a fixed tree one cannot avoid the worst case distortion of $\Omega(n)$. But if one build a randomized tree, chosen from a family of tree metrics, the *expected* distortion can be bounded by $O(\log n)$. Thus using this tree for routing one immediately obtains $O(\log n)$ stretch routing with low routing overhead. Approximating a metric with probabilistic hierarchical well-separated trees was first proposed by Bartal [21, 22], with the motivation that many problems are easier to solve on a tree than on a general graph. Later, Fakcharoenphol *et al.* [56] improved the distortion to $O(\log n)$ for any n node metric and this is tight.

Due to the distributed nature and increasing deployment of wireless sensor networks, distributed computing algorithms draws a lot of attention and raises a lot of interesting research topics in wireless sensor networks commnuity. The goal here is to design efficient distributed algorithm compared to a corresponding global algorithm. We could easily turn a global centralized algorithm into a distributed algorithm. We can always achieve this by assuming a central station, which simply centrally collect the distributed state, compute a global solution, and distribute the solution. However, the major drawback related with this simple routine is about the unreasonably high communication overhead and bad load balance. We have power restriction in sensor networks and sending and receiving messages are expensive operations in wireless sensor networks. And this simple scheme does not exploit the distributed nature and property of wireless sensor networks well. We aim at more reasonable distributed algorithm that minimize communication.

Combining these two well-established research areas leads to a promising and efficient approach for a lot of difficult problems arising from wireless sensor networks. Distributed approximation algorithms trade-off opitimality of the solution for the amount of resources consumed by the distributed algorithm.

2.2 Hierarchically Well-Separated Trees

In this section, we will give the definition of an α-hierarchically well separated tree (α-HST) and show how to compute the α-HST embedding of a constant

doubling dimension graph metric in a distributed way.

Definition 1 (α-HST). *A rooted weighted tree T is an α-HST if*

- *the weights of all edges between an internal node and its children are the same*
- *all root-to-leaf paths have the same hop-distance*
- *the edge weights along any such path decrease by a factor of α as we go down the tree*

Throughout this chapter we 'count levels from leaves to the root', i.e., we define the *level* of a node u in α-HST as the distance *in hops* from u to any of its leaves. In particular, all leaves have level 0. We define the level of a subtree to be the level of its root.

Fakcharoenphol *et al.* [56] give a centralized algorithm to compute, given an n-point metric (P,d), a random 2-HST T whose tree metric d_T $O(\log n)$-*probabilistically* approximates d. That is, for any $u,v \in P$, $d_T(u,v) \geq d(u,v)$ and $E[d_T(u,v)] \leq O(\log n) \cdot d(u,v)$, where the expectation is taken over random choices of the algorithm, equivalently over the distribution on the resulting trees. In this paper, we use P to denote the set of nodes in an undirected graph modeling the sensor network, and $d(u,v)$ denotes the minimum hop count between $u,v \in P$ in the sensor network. When we use a 2-HST to approximate the graph metric d, all the nodes in P are placed as the leaf nodes of the 2-HST. The distance between two nodes $u,v \in P$ on the tree T, $d_T(u,v)$, is defined as the sum of the distances along the paths from u,v to their lowest common ancestor in the 2-HST. The internal nodes in the HST are abstract nodes, although they might be labeled by some nodes of P, as will be explained later. We denote by $B(p,r)$ the ball centered at point p with radius r.

The algorithm of [56] is centralized and executes in a top-down fashion to partition the nodes in the network hierarchically. The HST naturally corresponds to this hierarchical partition of P. The nodes of P are the leaf nodes of the HST and each internal node in the HST corresponds to a cluster of nodes in the hierarchical partitioning. For completeness, we review this algorithm below. We fix a permutation $\pi : P \to \{1,2,\ldots,n\}$ of the nodes, chosen uniformly at random from the set of all permutations. We also fix a value β chosen uniformly at random from $[\frac{1}{2},1)$.

To get a 2-HST, we compute for each node u a $O(\log(n))$-dimensional *signature vector* $S(u)$, where the i-th element in the vector is

$$S(u)_i = \arg \min_{v \in B(u, 2^i\beta)} \pi(v), \tag{1}$$

for $i = 0$ to $m = \lceil \log D \rceil + 1$ with D the network diameter. That is, each node keeps the node with the smallest rank among all nodes within distance $2^i\beta$. For all nodes u, $S(u)_m$ is the node with rank 1. These signature vectors define the HST embedding of d. In particular, the leaves are nodes of P, the level-i ancestor of a node u is labeled $S(u)_i$, and the weights of all edges between level i and level $i-1$ is 2^i. The fact that this is indeed a 2-HST (in the sense of Definition 1) is not obvious; its proof appears in [56]. The top-down construction of the 2-HST in [56] is hard to implement in a distributed network.

2.2.1 Distributed algorithm to compute 2-HST

In this section we propose a bottom-up construction to compute a 2-HST in a distributed way, with total communication cost $O(n \log n)$. The HST construction is performed as preprocessing at the network initialization stage. Specifically, we will show how to compute signature vectors shown in (1) and the necessary information for each node to find path to its parent in a distributed way.

Our algorithm proceeds in a bottom-up fashion and computes the i-th element of the signature vector for every node in round i, for i starting at 0. If $S(u)_i = v$, we say that *u nominates v* at round i. Intuitively, as i increases, only the nodes with small indices can possibly be nominated and appear in the signature vector. Observe that if at level i a node u is not nominated by any other node, i.e., every node v already has some other node with index smaller than u within radius $2^i\beta$, the node u cannot possibly be nominated at level $i+1$. Thus we keep track of the subset of nodes that have been nominated and 'survive' round i and only these nodes will flood the network with maximum hop count (TTL) value of $2^{i+1}\beta$ in round $i+1$. We use P_i to denote the nodes that are nominated from the round i and thus are candidates to be nominated in round $i+1$.

At the beginning, every node is a candidate, that is $P_0 = P$. A fixed predetermined node ('leader') chooses a random β uniformly from $[\frac{1}{2}, 1)$ and distributes it to all nodes using a single global flood. In round $i+1$, the nodes of

P_i flood the network up to distance $2^{i+1}\beta$. The flooding packets are cached at the nodes receiving these packets until the end of the current round. Now every node u receives some flooding messages from the candidate nodes $P_i \cap B(u, 2^{i+1}\beta)$. During round $i+1$, node u maintains the identity of the node $v_{\min}$ with lowest rank that it has received. At the end of round $i+1$, u nominates $v_{\min}$ as its $(i+1)$-th level element in its signature vector for round $i+1$ (also its level-$(i+1)$ ancestor), i.e., $S(u)_{i+1} = v_{\min}$.

Each node records the information where it got the flooding message from. With this information, each node u can trace back to get the path to the ancestor $v_{\min}$ and report the nomination. By this process, each node knows its parent and its children in that level of the tree. At the end of each round, all nodes clear all flooding messages that are not traced back by other nodes. P_{i+1} consists of the nodes in P_i that are nominated by some nodes during round $i+1$. Continue the above process until there are no more candidates. After we compute the signature vector $S(u)$ for each node u, we could easily reconstruct the original HST from these signature vectors.

We remark that the 2-HST is computed and stored in a completely distributed manner. This is sufficient for the applications in the distributed matching problem.

2.2.2 Communication cost

In this section we prove that the total communication cost, measured by the total number of message transmissions, for the computation of the 2-HST described earlier is bounded by $O(n \log n)$, provided that the doubling dimension of our minimum hop metric is bounded by a constant. The constant doubling dimension metric has been used in prior work as an appropriate model of a sensor network metric, when the sensor nodes are densely and uniformly deployed in a geometric region [64].

First we formalize the intuition stated above, that the sets of candidates P_i become sparser as level i increases. As the construction algorithm is randomized, the following bounds are taken in expectation on random permutations and parameters β. Actually, communication cost analysis in this section holds for any fixed β (the fact that β is random is only important for the distortion bound).

Lemma 2. *If the doubling dimension is at most γ, any ball of radius 2^i contains at most $2^{6\gamma}$ elements of P_i in expectation, for all i.*

Proof. For convenience, let $2^{6\gamma} =: c$. The proof is by induction on i. The claim is true for the first round ($i = 0$) as every node sends a message to its 1-hop neighbors, and each node only responds by one message to the neighbor with minimum index.

Consider the ball $B(v, 2^i)$ for arbitrary v. Nodes of P_i in this ball may only be nominated (in the i-th round) by nodes in $B(v, 2^{i+1})$. The latter can be covered by a family $\mathcal{A}$ of $2^{2\gamma}$ balls of radius 2^{i-1}, so it suffices to prove that each ball of $\mathcal{A}$ nominates at most $2^{-2\gamma}c$ nodes in P_i.

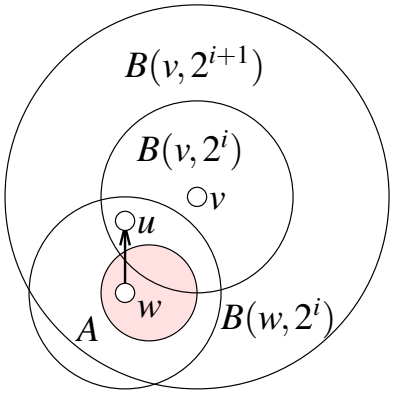


Figure 1. Any ball of radius 2^i contains at most $2^{6\gamma}$ elements of P_i in expectation, for all i. The arrow from w to u means node w nominates u in round i.

Let A be any of the covering balls, $A \in \mathcal{A}$. Denote by $|A|$ the number of nodes of P inside A. If a node in $P_i \cap B(v, 2^i)$ is nominated from A, it must be chosen from $P_{i-1} \cap B(v, 2^i)$. See Figure 1 for illustration. Since $B(v, 2^i)$ can be covered by at most 2^γ balls of radius 2^{i-1}, and by inductive hypothesis, there are at most $2^\gamma c$ choices for this nomination. That is, $|P_{i-1} \cap B(v, 2^i)| \le 2^\gamma c$. We claim that for a fixed choice u, the probability that u is nominated by a node in A is at most $\frac{1}{|A|}$. To see this, note that any $w \in A$ contains entire A inside its 2^i-ball, that is $A \subseteq B(w, 2^i)$. So if u is nominated by any node in A, then in particular, $\pi(u)$ is smaller than the ranks of *all* nodes in A, which is true for at most $\frac{1}{|A|}$-fraction of permutations. Hence in expectation, the number of nominations by A is $\frac{2^\gamma c}{|A|}$. On the other hand, it is also deterministically bounded by $|A|$, as each node in A can nominate only once. Hence

the number of nominations by A is bounded by $\min\{|A|, \frac{2^{\gamma}c}{|A|}\} \leq \sqrt{2^{\gamma}c}$. This is at most $2^{-2\gamma}c$ by our choice of c. □

Now we can prove the main result.

Theorem 3. *For a sensor network with n nodes, the total communication cost for constructing the 2-HST is $O(n \log n)$ in expectation, and each node uses expected storage for $O(\log n)$ node IDs.*

Proof. The initial flood (to distribute the value of β) takes $O(n)$ messages, because the maximum degree of the network is bounded. Now it suffices to prove that each node at each round receives $O(1)$ messages. The communication cost will then be only $O(n)$ in each round as there are n nodes in total. Since there are $O(\log n)$ rounds, the total number of messages is $O(n \log n)$.

The number of messages transmitted (propagated) by node u in round i is at most $|P_{i-1} \cap B(u, 2^i\beta)| \leq |P_{i-1} \cap B(u, 2^i)|$. Since $B(u, 2^i)$ can be covered by at most 2^{γ} balls of radius 2^{i-1}, each of which contains at most $2^{6\gamma}$ elements of P_{i-1} (by Lemma 2), we conclude that u transmits at most $2^{7\gamma} = O(1)$ messages in each round.

Total storage requirement consists of all next-hop pointers that encode the paths that realize edges of the HST. Notice that each such pointer can be charged to a unique flooding packet reception event. Hence the storage requirement for a given node u is at most the number of flooding messages that u received throughout the computation. Again, Lemma 2 implies that this number is $O(1)$ per round, or $O(\log n)$ in total. □

2.2.3 α-HST

The algorithm described above computes a 2-HST for a distributed network. It is not hard to convert a 2-HST to an α-HST for any value $\alpha \geq 2$. We remark that converting a 2-HST to an α-HST can also be done in a distributed setting. Basically at the end of our distributed algorithm, each node identifies its ancestors in the 2-HST T. Now we condense the tree T to an α-HST T' by removing some intermediate levels of T and re-connecting the nodes directly to their lowest ancestors that are not removed. In particular, the leaf nodes remain the same. For i increasing from 0, suppose we have constructed the tree T' up to level i. The nodes on level i of T' correspond to the nodes on level j on tree T. Now we proceed to build the

tree T' in level $i+1$. To do that, we take the lowest level j' in the tree T such that the distance $\sum_{h=j}^{j'-1} 2^h = 2^{j'} - 2^j$ is no greater than α^i and remove all the internal nodes on level $j+1, j+2, \cdots, j'-1$ of tree T. The nodes on level j' are nodes on level $i+1$ in tree T'. The nodes in level i connect to their corresponding ancestors on level $i+1$ by a single edge with weight α^i. See Figure 2 for an example. Notice that the tree metric T' again dominates the original metric as we are only relaxing the edge weights. In particular, the edge connecting a node on level i in T' to its parent has weight α^i, with $2^{j'} - 2^j \leq \alpha^i < 2^{j'+1} - 2^j$ by construction, where j is level i's corresponding level in T and j' is level $(i+1)$'s corresponding level in T.

The relaxation of the edge weights of T' will add at most a constant multiplicative factor on the distortion as shown below. For any two leaf nodes u, v on T, suppose their lowest common ancestor is located at level i on T' and level j'' on T. Clearly $j \leq j'' < j'$. Now we would like to bound $d_{T'}(u,v)$ by $d_T(u,v)$. First $d_T(u,v) = 2 \cdot \sum_{h=0}^{j''-1} 2^h = 2(2^{j''} - 1)$, $d_{T'}(u,v) = 2 \cdot \sum_{h=0}^{i-1} \alpha^h = 2(\alpha^i - 1)/(\alpha - 1)$. Since $\alpha^{i-1} < 2^{j+1}$. We have

$$d_{T'}(u,v) \leq \frac{2\alpha}{\alpha - 1} d_T(u,v) + \frac{4\alpha - 2}{\alpha - 1} \leq 7 d_T(u,v).$$

That is, the distortion for $d_{T'}$ is also bounded by $O(\log n)$.

This conversion can be executed by each node in the network examining its signature vectors and selecting subset of the elements in exactly the same way. In the end we have an α-HST that is again implicitly stored on the nodes in the network, and $O(\log n)$-probabilistically approximates the underlying network metric.

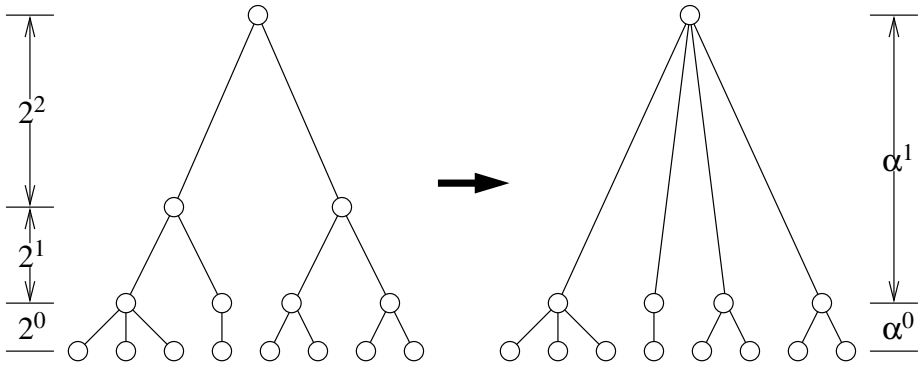


Figure 2. Convert a 2-HST to an α-HST for any $\alpha \geq 2$.

At the end we summarize the properties of the α-HST that will be useful later.

Lemma 4. *Suppose for level i on the α-HST T' corresponds to level $\ell(i)$ on the 2-HST T. Now we have:*

1. $2^{\ell(i+1)} - 2^{\ell(i)} \leq \alpha^i < 2^{\ell(i+1)+1} - 2^{\ell(i)}$.
2. *Each leaf node is within distance $(\alpha^i - 1)\beta/(\alpha - 1)$ from its i-level ancestor on T'.*

Proof. The first claim is due to the construction of the α-HST. For the second claim, we know each leaf node is within distance $2^j\beta$ of its j-level ancestor on the 2-HST. Now the distance from a leaf node to its i-level ancestor on T' is at most $2^{\ell(i)}\beta \leq [\alpha^{i-1} + 2^{\ell(i-1)}]\beta \leq (\alpha^i - 1)\beta/(\alpha - 1)$. □

2.2.4 HST on k resources

In Section 3.5 we will need to construct an HST only on the sub-metric of the original shortest-path metric which is induced by a given subset K of k nodes, let's call them *special* nodes for now. This can be done with the same communication and storage cost as before, by simply picking the permutation uniformly at random from the set in which the special nodes are k lowest-ranked nodes, and in the end taking the part of the tree containing special nodes (notice that this is still a tree, and that it has the same root. The parent of a special node is always special, due to the choice of ranks. Notice that the parent of a node could be itself if it has the lowest rank among all the nodes inside its proper neighborhood. Hence a breadth-first search from the root, stopped upon encountering a non-special node, is guaranteed to discovers special nodes.).

Why does this work? The main observation is that the execution of the algorithm restricted to K does not depend on non-special nodes at all. In fact, this simulates the execution of the original algorithm of [56] on the metric (K, d_K), where d_K is the restriction of the shortest path metric on K. Recall that the same construction works for *any* metric, as long as the $2^i\beta$-neighborhoods are determined with respect to the appropriate distance function. Since this is the case here, and the metric has k points, correctness follows.

2.3 Simulation

We implemented the 2-HST construction algorithm (Section 2.2) in MATLAB and tested in on 100 randomly generated networks. The networks were generated by perturbing n nodes of the $\sqrt{n} \times \sqrt{n}$ grid in the $[0,1]^2$ unit square, by 2D Gaussian noise of variance $0.3\sqrt{n}$, and connecting two resulting nodes if they are at most $\frac{3}{\sqrt{n}}$ apart (in Euclidean distance). We also made sure that all generated networks were connected.

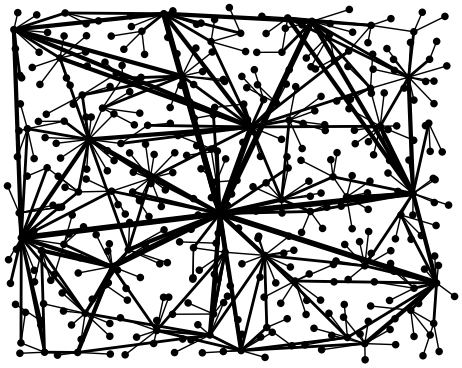


Figure 3. A typical HST on $n = 400$ nodes. Thicker lines represent higher-level edges.

Figure 3 illustrates a typical 2-HST layout on our test network, and Figure 4 shows a typical distribution of communication load and storage requirements. Communication is evenly distributed, as predicted by Lemma 2. Storage requirement is higher for nodes higher up in the hierarchy, but it grows slowly with the network size (Theorem 3). This result means that the hidden constant in Theorem 3 is in practice much smaller than predicted by our theoretical analysis.

Figure 5 shows how the communication cost and storage requirement change, on average, as a function of network size. The value for each size is an average of 100 independent trials. We conclude that, as predicted by Theorem 3, both per-node storage and per-node communication cost grow logarithmically with number of nodes.

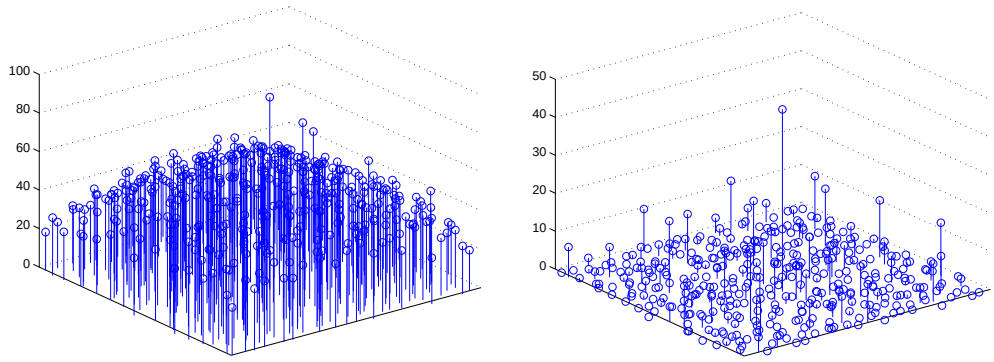


Figure 4. Communication cost (left) and storage requirement (right) in a perturbed grid network of $n = 400$ nodes.

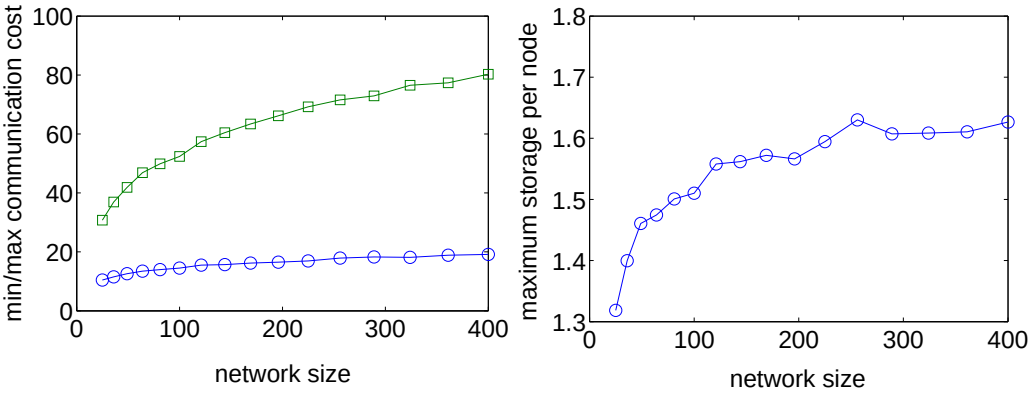


Figure 5. Per-node communication cost and storage requirement scale sublinearly with network size.

2.4 Conclusion

This chapter shows how to extract the hierarchically separated tree metric from a network in a distributed and communication efficient way. In the next chapter, we are going to implement the distributed resource management algorithms in this tree metric.

Chapter 3

Distributed Resource Management and Matching in Sensor Networks

3.1 Introduction

Recent advances in wireless sensor networks reveal the potential of such embedded network systems for revolutionizing the way we observe, interact with, and influence the physical world. Applications of sensor networks now extend beyond military deployments and the monitoring of animal or other natural habitats to places where humans work and live: homes, cars, buildings, roads, cities, etc. In these human spaces a sensor network serves users embedded in the same physical space as the network, allowing real-time data acquisition, situational understanding, event response and control, eventually leading to a fully intelligent living environment.

In this chapter we explore the challenge of using a network of embedded sensors in aiding information discovery and resource management so as to allow coordinated response to distributed emerging events. The embedded sensors serve two purposes: discovering/detecting the events of interest (e.g., a parking spot is left empty); and forming a supporting infrastructure for distributed resources/users to act on the detected events (e.g., help a vehicle to look for an empty parking spot). In the parking scenario, multiple vehicles can be consulting with the sensor network for available parking spots at the same time. The sensor network needs to resolve

competition and match vehicles with empty parking spots, in a way that avoids directing two cars to the same empty parking spot. In another scenario, emergency events detected by sensor nodes such as abnormalities, sensor battery outages, or local data overflows need to be handled immediately. Thus the sensor network faces the problem of directing surveillance vehicles/network helpers/data mules, one per event, to support real-time response and maintain network normal functioning.

When multiple events arise with multiple available resources to possibly act on them, there is an immediate need for coordinated and distributed resource management, to assign the resources to these events in a balanced yet efficient way. Here the *event* refers to, for example, a new vehicle querying for parking spots, or an emergency situation waiting for rescue efforts; and the *resource* refers to available parking spots or rescue teams. We model the resource management problem as follows: there are k available resources residing in the network, events may emerge at any time near any node, and multiple events can appear simultaneously. Both available resources and these emerging events are detected and tracked by their nearby sensor nodes. We would like to design a distributed matching mechanism to assign each event to a distinct available resource. Naturally, to reduce response delay and energy expense, an event should be matched to a nearby resource, with the distance measured either in the network metric (the number of hops in the network from the resource location to the event location) or other underlying metric (e.g., the Euclidean distance metric), depending on how the event is to be serviced. The quality of the solution is naturally represented by the total distance of the matching.

This resource management problem, in the centralized setting, is simply a classical minimum cost bipartite matching problem. Each resource i is connected to each event j with an edge of cost equal to the distance between them (in the appropriate sense). The optimal matching with the minimum total cost of the edges in the matching can be found by flow algorithms in $O(k^3)$ running time for k events and k resources [121]. However, implementation of the centralized min-cost matching solution is not feasible in the sensor network setting for two reasons. First this would require the collection of all the information on the resources and events at a central place to execute the flow algorithm, which we would like to avoid. In addition, the events may emerge anywhere anytime and often require immediate response. Thus we deal with an online setting in which resource commitments must be made before

we know the entire event sequence in the future. How to achieve the coordination needed in the matching algorithm in a distributed setting, with the events coming online, not aware of each other and not aware of the locations of available resources, is the main problem we will tackle in this chapter.

3.2 Overview

For distributed resource management, we need to solve the following two problems:

- low-cost resource discovery and aggregation: when an event emerges, how does it discover nearby available resources in a communication efficient way; and
- close to optimal matching algorithm: which resource should an event select to be matched with, and how to resolve competition, so that no two events select the same resource.

The two problems, *resource discovery* and *distributed matching*, are closely related. The solution for one may impact the solution for the other.

Let us first consider a typical setting when a set of events pop up at the same time. In a distributed setting where no sensor node is taking full control, the challenge for the resource discovery problem is to decide what information is to be sent, and to where. The naive solution of flooding the entire network with all the events allows each event to solve for the centralized min-cost matching solution, but is obviously too communication expensive. To reduce communication cost, one may use restricted flooding to discover the closest available resource from each event. In particular, the TTL of the flooding messages from an event doubles until a resource is found. The communication cost, measured in the number of message transmissions, is $O(\ell^2)$, if the closest resource is ℓ hops away. In this way, each event discovers the closest resource that has not yet been matched with others. The problem is that such a greedy matching algorithm may give a very poor approximation ratio of $\Theta(k^{\lg_2^3})$ to the min-cost matching of k events [132]. To get a better approximation ratio, one may adopt the 2-approximation algorithm for min-cost matching by the primal-dual method [73]. This would require a flooding from each

event and global coordination of the growth of each aggregated component, thus incurring a much higher total communication cost.

From the above discussion, it is clear that we need to find a solution that achieves a balance between a low communication cost for resource discovery and a good approximation ratio for distributed matching. Our solution is to define a tree metric from the underlying network metric (minimum hop count distance metric of the sensor network) so that

- the simple greedy algorithm in the tree metric gives a polylogarithmic approximation ratio (or competitive ratio in the online setting when the events appear one by one [26]);
- the tree structure allows easy probing based detection of nearby resources, thus significantly saving communication cost (when compared with flooding the network).

The tree metric we extract is based on a *hierarchical well-separated tree*. An α-hierarchically well-separated tree (α-HST) is defined as a rooted weighted tree such that: the weight of all edges between a node and its children are the same; the edge weights along any path from the root to a leaf are decreasing by a factor of α. The nodes of the HST are the nodes in the original communication network G, but a tree edge is virtual, i.e., it maps to a path in the original network.

This hierarchical well-separated tree is going to approximate the original graph metric G by a logarithmic distortion factor, in terms of the shortest path distance between any two nodes. In particular, what we propose to extract is one α-HST, chosen by a distribution $\mathcal{D}$ from a family $\mathcal{S}$ of α-HSTs such that for each HST T in the family $\mathcal{S}$, the distance $d_T(u,v)$ between any two nodes u,v in the tree T is greater than the shortest path length $d_G(u,v)$ in the original graph G, and that the expected distance between u,v, $E_{T\in\mathcal{S}}[d_T(u,v)D(T)]$, taken over the distribution $\mathcal{D}$ on the family $\mathcal{S}$ of trees, is no greater than $\beta \cdot d_G(u,v)$, where $D(T)$ is the probability of T in the family $\mathcal{S}$. Such a family of HSTs is said to β-probabilistically approximate the original metric G, with β being the (expected) distortion factor.

The hierarchical well-separated tree brings in two benefits to the distributed resource management problem. First, a HST is a special and simpler metric such

that simple greedy algorithm by recursively matching closest pairs on the *tree*, instead of the original graph, gives good performance for both off-line and online matching. In the off-line (but distributed) setting when k events appear at the same time, the greedy algorithm gives *optimal* min-cost matching on the HST. As the tree metric approximates the original graph metric by a logarithmic factor, this algorithm gives an $O(\log k)$ approximation ratio for the min-cost matching on the original communication graph. In the online setting when the events appear in any order and we compare the quality of the online matching with the off-line optimal solution (when the entire sequence of events is known), the simple greedy algorithm gives an $O(\log^2 k)$-competitive ratio compared with the optimal matching on the HST and consequently an $O(\log^3 k)$-competitive ratio in the original metric. Secondly, working on the hierarchical well-separated tree solves information discovery and aggregation easily. We adopt a similar probing scheme as in our previous sparse aggregation framework [69]. Each event sends probes from along the HST looking for the nearest unmatched resource.

In section 2.2 we describe a distributed algorithm for the extraction of the probabilistic HST from the sensor network metric. The construction of HST is a preprocessing step that is executed during the network setup phase. Nodes on the HST correspond to nodes in the network; each network node maps to at least one tree node, possibly more. Edges of the HST correspond to paths in the network. Each such path is stored as a (bidirectional) chain of next-hop pointers, one per node along the path. Each HST node records information required to associate its outgoing paths to its adjacent HST edges, as well as the weights of those edges in the HST. In summary, to store the HST in the network, a network node has to store the amount of information proportional to the number of paths that go through it, and the number of HST edges adjacent to HST nodes that it represents.

For a particular resource management problem, we can condense the HST to get a tree with the leaf nodes only on the available m resources with the total size of the tree being $O(m)$. For load balancing, we can build a few HSTs, with different random seeds, and rotate between them periodically. The construction of the HST assumes a random indexing of the nodes and proceeds in $O(\log n)$ rounds. At round i, only a subset of the nodes P_i 'survive' to be qualified as the nodes on the i-th level of the tree (counted from the leaf level of the tree). These nodes flood the network

with maximum hop count $\beta \cdot 2^i$, where β is a number chosen randomly between $1/2$ and 1 in the initialization phase by a preconfigured 'leader' node, who broadcasts the value to the rest of the network. Every node selects, among all the messages it received at this round, the node with minimum index to be its ancestor at level $i+1$ with an edge weight as 2^i. The nodes that are not nominated by any node will quit after round i. Intuitively as i increases, although the flooding radii increase, the number of candidates decreases. We show that the total communication cost for each round, in terms of the number of message transmissions, is bounded by $O(n)$, and the total communication cost for the construction of a HST is $O(n \log n)$, if the underlying communication graph has a constant doubling dimension. The doubling dimension of a metric space (X,d) is the smallest value ρ such that each ball of radius R can be covered by at most 2^ρ balls of radius $R/2$ [81]. If we place at most a constant number of sensor nodes inside any unit disk and the holes in the sensor networks are not very fragmenting, the communication graph has constant doubling dimension.

The construction of the HST in a distributed environment is interesting on its own and can potentially have more applications besides the distributed matching algorithm for resource management explained above — working on a tree is much easier than working on the original graph; thus many optimization problems admit algorithms of improved performance on a tree. The construction of the HST naturally extends the applicable domain of these algorithms to a sensor network setting with only a logarithmic factor loss.

3.3 Previous Work

The min-cost matching problem in a bipartite weighted graph can be solved optimally by the flow algorithm in a centralized setting. The greedy matching algorithm that matches closest pairs achieves a worst-case approximation ratio $\Theta(k^{\lg 3}_2)$ for k vertices on a general graph or under Euclidean metric [132], but is optimal on a HST metric [20]. For the online min-cost matching problem, the k vertices (blue vertices) on one side of the bipartite graph are given, the k vertices (red vertices) on the other side of the graph are revealed one by one. When a red vertex appears, it needs to be matched with a blue vertex immediately. The solution is compared

with the optimal matching when all the red vertices are given altogether and the performance degradation is captured by the competitive ratio of the online decision making procedure. Meyerson *et al.* [110] proposed the first randomized online algorithm that achieves a logarithmic competitive ratio of $O(\log^3 k)$, by using the probabilistic HST approximation of a general graph metric. Essentially each red node is matched to the closest blue node on the HST and ties are broken randomly. An improved algorithm by Bansal *et al.* [20] achieves a competitive ratio of $O(\log^2 k)$.

Approximating a metric with probabilistic hierarchical well-separated trees was first proposed by Bartal [21, 22], with the motivation that many problems are easier to solve on a tree than on a general graph. Later, Fakcharoenphol *et al.* [56] improved the distortion to $O(\log n)$ for any n node metric and this is tight.

This chapter borrows from these ideas. This previous works, however, were for a centralized environment. In this chapter the distributed algorithm design and the communication cost analysis for both (i) extracting a HST from the network metric and (ii) applying probing-based mechanisms on the HST for information discovery and resource management, in a resource constrained sensor network setting, are new.

The results in this chapter is also related with our previous work on aggregation of sparsely located events, none of them with knowledge of each other [69]. The idea is to use light-weight local probes (in the vertical and horizontal directions) from the simultaneously emerging events to achieve distance sensitive neighbor discovery with nearby events discovering each other first. Aggregation is performed when multiple events 'find' each other and at the same time an aggregation tree is formed to suppress the probing of all but one event being aggregated. The probes that survive in this aggregation will carry the aggregated information and propagate further to look for other possible events. The fact that aggregation is done naturally along a tree structure constructed by the individual probes substantially reduces the total communication cost for the group discovery of these events to only an $O(\log k)$ factor over the cost of the minimum spanning tree connecting all the events. The novelty in this chapter is the different tree metric (the HST) that we use and the benefits of the special tree metric for the distributed matching problem. We show in Section 3.4 that the HST can also be used for sparse aggregation with the same

communication cost.

3.4 Sparse aggregation with HST

We also remark that the 2-HST computed in Section 2.2 can be used to get an aggregation tree when spontaneously emerging resources spread out in the network are detected by their local sensors and no resource is aware of how many and where the other resources are. That is, it solves the same problem as in the sparse aggregation framework [69].

Suppose the a set K of k resources are detected simultaneously (i.e., at network setup), each node with new detection sends a message along the path in the tree towards the root. With the same assumption as in [69] that the messages travel with speed lower-bounded by V, these messages can be aggregated and pruned when they travel towards the root. In particular, an internal node u at level i will wait for a period of time $2^i \cdot V$ after the network setup — so that all messages from the resources in its subtree will be able to arrive at u for sure. After that u aggregates the messages it received and sends a message to its parent on the tree.

Eventually the process will result in a sparse aggregation tree $T(K)$ with root as the lowest common ancestor $LCS(K)$ of all the resources in K. The edges of $T(K)$ include the paths from the k resources to $LCS(K)$. From $LCS(K)$ one message travels to the root (as it does not know whether there are resources in other part of the tree and has to travel to the root to find that out). The total communication cost of the sparse aggregation is $O(|T(K)| + \ell)$, where $|T(K)|$ is the size of the sparse aggregation tree, ℓ is the distance from $LCS(K)$ to the root of the HST and is upper bounded by the network diameter. See Figure 6 for an example. The blue nodes are the resources. The thick edges are the edges traveled by aggregation messages.

We claim that the tree $T(K)$ has total size $O(\log k)$ times the smallest aggregation tree possible (i.e., minimum Steiner tree) in the original network (if the resources know each other in advance). In particular, $T(K)$ is the minimum Steiner tree of the resources in K on the HST metric. Suppose the minimum Steiner tree in the original network is $T'(K)$, we replace each edge xy on $T'(K)$ by the path between nodes xy on the HST, with a blowup factor of $O(\log k)$, using Section 2.2.4. Now the resulting network must have a size no smaller than the size

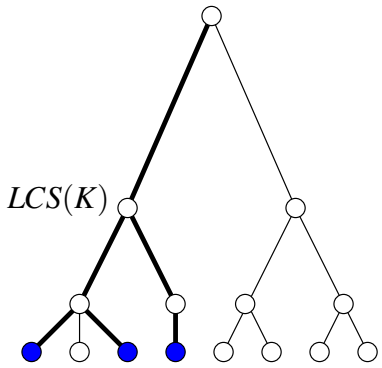


Figure 6. Sparse aggregation of k resources/events by an HST.

of $T(K)$ — as $T(K)$ is the minimum Steiner tree on the HST. This shows that $|T(K)| \leq O(\log k \cdot |MST|)$, where MST is the minimum Steiner tree in the original network.

We remark that the paper [69] requires a double rulings scheme supported in the underlying network, for the probe messages to meet and discover each other. Our solution does not require that and can work on any network with bounded doubling dimension, given that the 2-HST has been computed in the preprocessing phase.

3.5 Distributed Matching Algorithms

In this section we describe several algorithms for distributed bipartite matching problem, in online and offline settings. The main goal of the section is to show that, if preceded by a preprocessing phase in which an HST is computed as in Section 2.2, the matching algorithms have *output-sensitive communication cost*, in addition to providing good approximation/competitive ratio (in offline and online setting, respectively).

In all models we consider there are k resources whose number and locations do not change over time. The location of a given resource is initially unknown to any other node except the nearest sensor nodes that detect it. Also, we assume that there are exactly k requests, which can either be present in the beginning, or arrive

one by one. In any case, a perfect matching always exists. We remark that most of our results continue to hold if this is not the case.

We assume that packets traverse one hop per time unit, i.e., that appropriate algorithms are employed to handle low-level issues such as medium access, packet buffering and transmission scheduling. We also assume for simplicity that all nodes have enough local storage, and focus on minimizing communication cost. This is justified by the fact that with today's technology energy is a much more valuable resource than storage space.

3.5.1 Offline setting

First we consider the offline case, in which both resources and requests are present simultaneously (but the resources and the requests do not know each other). Bansal *et al.* [20] proved that the natural greedy algorithm — go through the set of requests in arbitrary order and match each request with the closest unmatched resource *in the* α*-HST tree metric*, breaking ties arbitrarily — gives as $O(\alpha \log k)$-approximation (in expectation) to the optimal solution *in the original metric*, for any $\alpha > 1$. In the following we use a 2-HST and describe a distributed implementation of this algorithm that uses a factor of $O(\log k)$ more packets than the total length of the returned matching (the sum of hop-distances of matched pairs).

We may assume without loss of generality that no node has both a resource and a request; such pairs can be immediately paired and removed from consideration. We allow multiple resources to occupy the same node, the same with requests.

The distributed algorithm proceeds in a similar way as the sparse aggregation algorithm described in subsection 3.4. Resources and events send their information up the tree. Internal nodes match the resources and events in its subtree whenever possible. Information on unmatched resources or events is propagated further up the tree until every event is matched. Specifically, leaves that have resources or requests send this information to their parents (at level 1). Note that empty leaves do nothing. Parents then locally compute an arbitrary matching between requests and resources they received and notify matched leaves. We recursively solve the problem with unmatched requests and resources, where we think of them as residing at level-1 nodes. The resulting matching is represented by a number of paths in the HST,

and corresponding paths that are constructed (by establishing 'next hop' pointers) in the underlying graph. Once this is done, level-1 nodes finish the algorithm by propagating each path to the leaf that initially holds the corresponding matched resource/request.

Notice that level-1 nodes have to wait at most one time unit for the packets coming from their children. This follows from out assumption on packet propagation speed, and 2-HST structure, i.e., the fact any leaf is within $2^1\beta < 2$ hops, hence at most one hop, away from its parent. If a packet from some child is not received within this time, it means that the child has no requests/resources to report. Similar time bounds hold in recursive calls.

Now we turn to bounding the communication cost. Clearly, no communication is ever 'wasted', i.e., if a node sends a packet to its parent reporting resources or requests (recall, never both), eventually that HST edge (and the corresponding path in the graph) will be used in the paths that match those resources/requests.

Theorem 5. *The expected communication cost of computing a matching of length l with our algorithm is $O(\log k) \cdot l$.*

Proof. The claim easily follows by observing that each edge computed by the algorithm may be a factor of $O(\log k)$ shorter in the original metric than in the HST metric, and by summing over all edges in the matching. □

3.5.2 Online setting

Now we turn our attention to the online setting in which requests come in one by one, and each request must be *irrevocably* matched to some resource before the next request arrives. This models fairly common applications of wireless networks in which decisions are time-critical, cannot be changed once they are made, and there is no *a priori* upper bound on the time between consecutive request arrivals.

We propose a distributed algorithm which never pays more than order of l^γ in communication cost for computing a matching of cost (length) l, where γ is the doubling dimension of the metric. Throughout this section we assume that the next request does not arrive until the current source is matched and all associated messages have been delivered. This assumption captures the application scenarios when event inter-arrival times are much greater compared with message propagation time.

One solution is based on the following randomized greedy algorithm proposed in a centralized setting by Meyerson *et al.* [110]. When a new request arrives, it is matched to the closest unmatched resource on an α-HST, breaking ties randomly, i.e., by choosing each of the tied resources with equal probability. They proved that this algorithm is $O(\alpha \log^2 k)$-competitive (in expectation) on the original metric, provided that $\alpha \geq 1 + 2\ln k$.

For a node u, let $T(u,i)$ be the unique level-i subtree that contains u. Also, for $i \geq 1$, define $N(u,i) = T(u,i) - T(u,i-1)$. Current request r can be matched to its randomly chosen closest unmatched resource by having the leaf that holds r gradually explore its neighborhood in the HST. Specifically, the exploration proceeds by levels: first $N(r,0) = \{r\}$, then $N(r,1)$, then $N(r,2)$ etc. It is essentially a post-order traversal of the HST, which starts from r and stops on encountering an unmatched resource. Notice that, with such organization, each $N(r,i)$ can be explored with a communication cost of at most twice the total length of all paths (in the underlying graph) that realize all edges of the subtree. This also includes the task of picking a uniformly random unmatched resource, if the subtree happens to contain any. For example, each resource, once found by exploration broadcast, can choose a random number uniformly from $[0,1]$, and then the minimum of all choices can be computed using sparse aggregation (as in Section 3.4). The 'trail' left by the minimum number's packet during aggregation leads to a uniformly random resource.

It is easy to see that the algorithm is correct (under the above assumption on request arrivals). It is also obvious that if r is matched to s after exploring levels up to i, i.e., entire $T(r,i)$, the communication cost involved is at most twice the total length of $T(r,i)$.

Lemma 6. *Consider an α-HST constructed as in Section 2.2 for an underlying metric with doubling dimension γ. Let T be a level-i subtree and let h be its height (root-to-leaf distance) in the HST metric. Total length of T (all its edges) in the HST metric is at most $C(\alpha,\gamma)h^\gamma$, where $C(\alpha,\gamma) = \frac{\alpha^\gamma 2^{5\gamma}}{\alpha^\gamma - \alpha}$.*

Proof. By the α-HST construction in subsection 2.2.3 and Lemma 2.2.3, the root of the subtree T is at level $\ell(i)$ at the corresponding 2-HST. Thus all leaves can be covered by a single ball of radius $2^{\ell(i)-1}\beta < 2^{\ell(i)-1}$ centered at the root of the subtree T. Now we consider a level j in the subtree T with $0 \leq j \leq i-1$ and

we would like to count the number of nodes at level j. By Lemma 2, any ball of radius $2^{\ell(j)}$ has at most $2^{6\gamma}$ level-j nodes. Hence the leaves can be covered by at most $2^{\gamma(\ell(i)-1-\ell(j))}$ balls of radius $2^{\ell(j)}$. Thus the number of level-j nodes is at most $2^{\gamma(\ell(i)-\ell(j)+5)}$. Each level-$j$ node has a single edge of length α^j connecting it to its parent. Furthermore, the total length of the tree is exactly the sum of all these edges. So, summing over j, we have

$$\begin{aligned}\sum_{j=0}^{i-1} 2^{\gamma(\ell(i)-\ell(j)+5)} \cdot \alpha^j &= 2^{\gamma(\ell(i)+5)} \sum_{j=0}^{i-1} \alpha^{-(\gamma-1)\ell(j)} \\ &\leq \frac{2^{\gamma(\ell(i)+5)}\alpha^\gamma}{\alpha^\gamma - \alpha} \\ &\leq \frac{\alpha^\gamma 2^{5\gamma}}{\alpha^\gamma - \alpha} \cdot \left(\frac{\alpha^i - 1}{\alpha - 1}\right)^\gamma.\end{aligned}$$

The height of the subtree is $1+\alpha+\alpha^2+\cdots+\alpha^{i-1} = \frac{\alpha^i-1}{\alpha-1}$. Thus the total length of T is bounded by $O(C(\alpha,\gamma)h^\gamma)$. □

This directly implies a bound of $O(C(\alpha,\gamma)l^\gamma)$ on the communication cost of computing a single matching edge of length l in HST metric.

We can now prove the main result about our online algorithm.

Theorem 7. *The expected communication cost for computing a matching of total length l is $O(l^\gamma)$.*

Proof. In expectation, a matched pair may be a factor of $O(\log k)$ closer in the original metric than in HST, yielding a ratio of $O(C(\alpha,\gamma)\log^\gamma k)$ for each edge. The same ratio holds for the total length, since $\sum_i x_i^\gamma \leq (\sum_i x_i)^\gamma$, for $\gamma \geq 1$. The claim follows by substituting $\alpha = \Theta(\log k)$. □

3.6 Simulations and Experiments

3.6.1 Approximation and competitive ratios

We implemented the algorithm described in Section 3.5.1 in this section. Since we know that it is communication-optimal in the 2-HST (Theorem 5), we only

compared optimal matching costs in the 2-HST and in the underlying graph. This is, of course, directly related to the distortion of the HST embedding.

We generated random perturbed grid networks from the same distribution as in the previous experiment. Network sizes range from 25 to 400, and the number of resources was kept fixed at $k = 10$. We sampled 100 networks for each network size. For each network, we chose k resources and k requests uniformly at random, and computed a 2-HST *with requests and resources as special nodes* (Section 2.2.4). Then we computed the communication cost of the matching computed by our offline algorithm, the length (in hops) of the same matching in the α-HST metric, and the length of the optimal matching on the same set of resources/requests (computed by the Hungarian algorithm [121], as implemented in [1]).

The following two figures show how the ratio of optimal matching costs (in 2-HST and the original metric) changes with the network size n, with the number of resources k fixed (Figure 7) and vice versa (Figure 8).

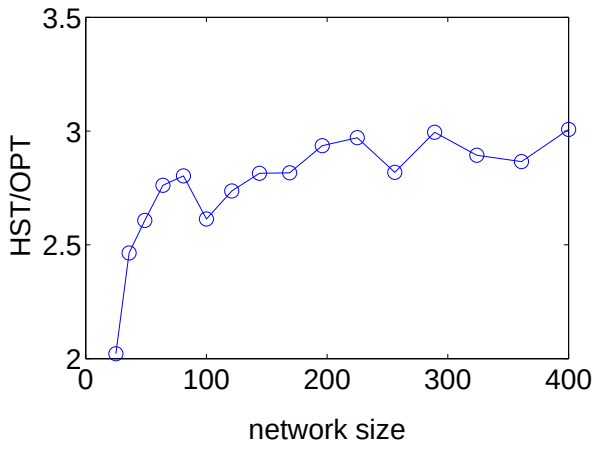


Figure 7. In offline setting, length of the optimal matching in the 2-HST and length of the optimal matching in the underlying metric are within a factor of 3 over a wide range of network sizes n, with fixed $k = 10$.

Finally, we consider the online algorithm of Section 3.5.2. We tested perturbed grid networks between 25 and 400 nodes, with 50 samples for each size according to the same distribution as above. For each sample we computed an α-HST for $\alpha = 1 + 2\ln k$ and ran the algorithm, recording the resulting communication cost, cost of the matching returned by the algorithm (in the HST metric), and optimal

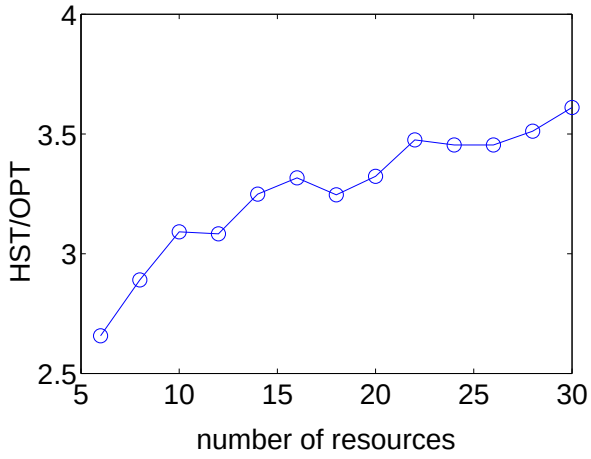


Figure 8. In offline setting, the ratio of optimal matching length becomes worse when the number of resources k increases, and $n = 225$ is fixed. This is because the network embeds into 2-HST with distortion that increases with k.

cost in the original metric.

Figures 9 and 10 show the ratio of the communication cost and costs of optimal matchings, one in the HST and the other in the original metric. In Figure 9 the number of resources k is fixed and network size changes. Figure 10 shows the opposite situation.

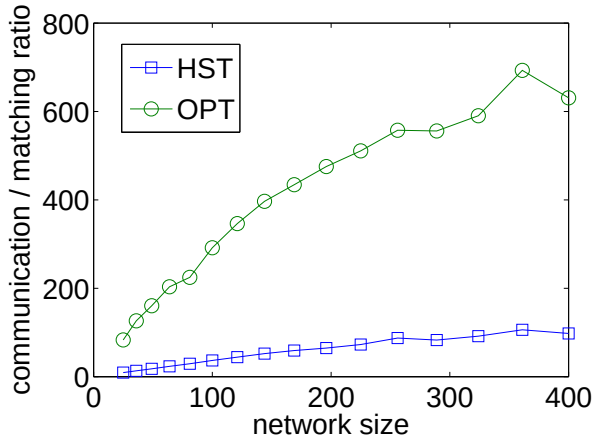


Figure 9. Communication overhead as a function of network size, with fixed $k = 15$.

Figure 11 shows that, as predicted by Theorem 7, communication cost grows

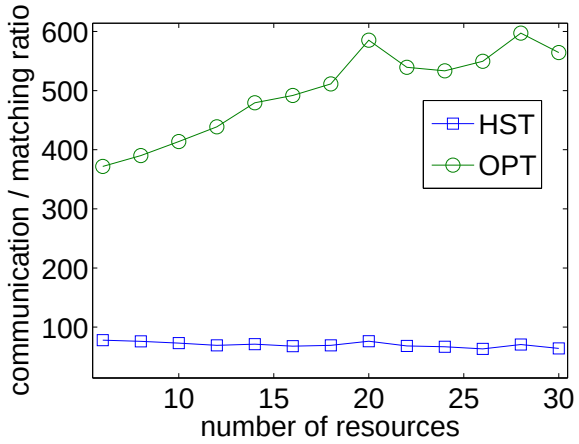


Figure 10. Communication overhead as a function of the number of resources, with fixed $n = 225$.

roughly quadratically with the cost of computed matching in the tree metric. This is expected, since we tested on two-dimensional geometric graphs.

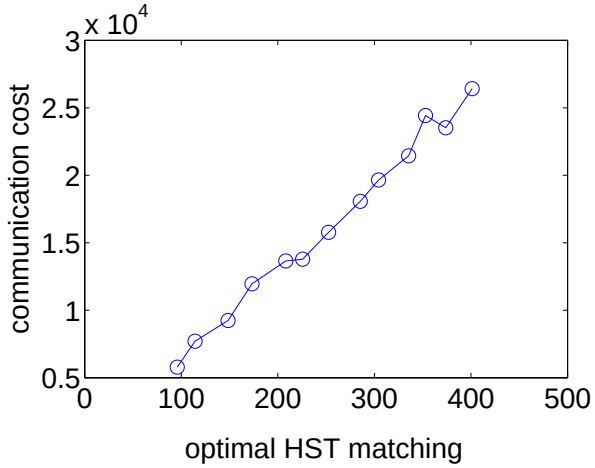


Figure 11. Communication cost of the online matching algorithm as a function of the cost of the computed matching in the α-HST metric, with a fixed number of $k = 15$ resources and variable network size n. The slope suggests quadratic dependence (linear regression yields a slope of about 2.37).

3.7 Conclusion

Resource management in a distributed setting is a challenging problem due to the lack of global coordination and knowledge of available resources/emerging events. However, the problem can be simpler when the underlying network metric is simpler (such as a tree). This chapter shows how to implement the distributed resource management algorithms in the hierarchical well-separated tree metric, for both offline and online scenarios. We would like to emphasize that the idea of working on a simpler metric can possibly be applied to other problems in a distributed setting to derive communication efficient algorithms with quality guarantees, such as k-server [23, 42], metric labeling [96], and tracking mobile sinks [19]. We will further explore this direction in the future.

Chapter 4

Maintaining Approximate Minimum Steiner Tree and k-center for Mobile Agents in a Sensor Network

4.1 Introduction

In this chapter we examine the challenges arising from a system of embedded static wireless nodes and a set of mobile agents acting in the same physical space. The agents could be either mobile robots or human users collaborating on a task. They do not necessarily have out-of-band communication channels among them and may have to resort to multi-hop routing in the static wireless network for inter-communication. The embedded sensor nodes provide real-time monitoring of the environment, possibly in-network processing to aid situation understanding, and interact with the mobile agents as a supporting infrastructure to help with both coordination and communication of the agents. This model captures many real-world scenarios, ranging from robots exploring a space, mules collecting sensor data, rescue team helping with disaster relief, etc.

We focus on two specific problems in this framework. The first is to maintain group communication of the mobile agents, in particular, a tree spanning the agents for continuous information exchange, decision making and order dissemination. We

denote by the sensor closest to an agent i a *proxy node* for i. To reduce communication delay, the communication tree is preferred to be the minimum Steiner tree of the agents in the sensor network, i.e., the tree with minimum total hops connecting the proxy nodes, possibly by using other non-proxy nodes as relay. It is well known that the minimum Steiner tree is a NP-hard problem [89], therefore we maintain at best an approximation to it. As agent i moves, its proxy node may hop to a neighboring node. In that case we will need to update the tree. The objective is to achieve small update cost with a good approximation ratio to the optimal minimum Steiner tree.

The second problem we study is the mobile k-center problem, that asks for k sensors as *centers* such that the maximum distance from each agent to its closest center is minimized. These centers naturally lead to spatial clustering or grouping of the agents, and map to obvious locations for control centers where information from the agents are aggregated and disseminated. In another scenario, when the agents need to physically gather together, the 1-center solution is the meeting point that minimizes the maximum travel distance. Again the k-center problem with k as a parameter is NP-hard [89]. We ask for the efficient maintenance of a good approximate solution.

4.1.1 Challenges

There are two fundamental challenges to allow such coordination and communication between mobile agents.

The first problem is *location management*, that is, the tracking and dissemination of the current location of the agents. In our setting we identify the location of an agent with its proxy sensor node. Obviously the proxy node is aware of the agent in its proximity. The difficulty is to inform other nodes/agents of the current location of an agent. We need to decide how frequently to disseminate the current agent location. This problem has been studied extensively in the literature. The earliest work is by Awerbuch and Peleg [19], in which the location information is updated to a carefully selected subset of nodes such that both location update and location query can be done in polylogarithmic time. Location services when all the nodes are mobile have also been developed in the last few years [5, 63, 103]. Although

being theoretically intriguing and inspiring, these schemes are still relatively heavy for real-world systems.

The second challenge is efficient maintenance of approximate minimum Steiner tree or approximate k-center of mobile agents, as efficient algorithms even in the centralized setting are lacking. Nothing is known for maintaining α-approximate minimum Steiner tree when $\alpha < 2$. The minimum spanning tree (MST) is a 2-approximation of the minimum Steiner tree. Yet the maintenance of MST is not perfectly solved even in the centralized kinetic data structure framework [79] (i.e., when the moving trajectories of all agents are given). When the nodes follow linear motion, there exists an algorithm for maintaining a minimum spanning tree when the distances are measured using the L_1 or L_∞ norms, or a $(1+\varepsilon)$-approximate Euclidean MST if the Euclidean distance is used [24]. But in this algorithm the number of updates of the data structure can be much higher than what is necessary (the number of necessary updates of the MST). In a slightly different setting where edge weights in a given graph change linearly, deterministic and randomized algorithms were designed [10]. These algorithms are very involved with heavy algorithmic techniques and are not practical. In the sensor network setting, a scheme called RoamHBA has been proposed to maintain group communication [58]. RoamHBA assumes that the current location of all agents are available (thus requiring a location management scheme as described above) and uses a heuristic algorithm to generate a tree. There is no guarantee on either the approximation to the MST nor the update communication cost when the agents move.

Regarding the k-center problem, the situation is not much better. In the centralized setting, the only known result is a kinetic algorithm for maintaining a 8-approximate k-center [68]. Nothing is known in the distributed setting to our knowledge.

4.1.2 Our approach

In short, we first preprocess the sensor network with $O(n \lg n)$ total messages. With the preprocessing no location management is needed for MST and k-center maintenance; the agents are not aware of the current location of other agents. Thus we save the communication cost necessary to update and query for the current agent

location. In particular, a $O(\lg n)$ approximate MST and k-center are maintained with an expected update cost of $O(\lg n)$ messages each time the proxy node of an agent hops to a neighboring node. Such results have not been achieved in past literature even when the location of the nodes are available.

In our preprocessing phase, we extract a tree metric on the sensor nodes called a *hierarchical well-separated tree*. An α-hierarchically well-separated tree H (α-HST) is defined as a rooted weighted tree such that: the weight of all edges between a node and its children are the same; the edge weights along any path from the root to a leaf are decreasing by a factor of α. The nodes of the HST are the nodes in the original sensor network G, but a tree edge is virtual, i.e., it maps to a path in the original network.

This hierarchical well-separated tree is going to approximate the original graph metric G by a logarithmic distortion factor, in terms of the shortest path distance between any two nodes. In particular, what we propose to extract is one α-HST, chosen by a distribution $\mathcal{D}$ from a family $\mathcal{S}$ of α-HSTs such that for each HST H in the family $\mathcal{S}$, the distance $d_H(u,v)$ between any two nodes u,v in the tree H is greater than the shortest path length $d_G(u,v)$ in the original graph G, and that the expected distance between u,v, $E_{H\in\mathcal{H}}[d_H(u,v)D(H)]$, taken over the distribution $\mathcal{D}$ on the family $\mathcal{H}$ of trees, is no greater than $\rho \cdot d_G(u,v)$, where $D(H)$ is the probability of H in the family $\mathcal{H}$. Such a family of HSTs is said to ρ-probabilistically approximate the original metric G, with ρ being the (expected) distortion factor.

The advantage of having the HST is that, the minimum Steiner tree on the HST metric is trivial — it is simply the connection of edges from all the agents/proxy nodes to their common ancestor on HST. As the HST is a $O(\lg n)$ approximation of the graph metric G, the minimum Steiner tree H computed on the HST metric is a $O(\lg n)$ approximation of the MST on the graph metric G. The question thus remains as how to maintain the minimum Steiner tree on the HST metric when the agents hop from node to node.

Similarly, for the k-center problem, working on a tree is much easier than working with a general graph metric. For $k = 1$, the common ancestor of all the nodes is the optimal center for the HST, and is a $O(\lg n)$ approximation to the 1-center solution of the original network G. For a general k, the optimal k-center for the HST will be taking the *lowest* level of the HST such that the number of nodes

with one or more agents in the subtree is no greater than k. Again this can be shown to be a $O(\lg n)$ approximate solution for the k-center in G.

In this chapter we show that with the HST constructed in the preprocessing phase, we can maintain the MST and the k-center of the mobile agents with expected communication cost of $O(\lg n)$ for each hop the agents move. Suppose one agent moves from node u to node v, we update the MST by taking the common ancestor w of u and v on the HST. Then only the edges on the path from u, v to w may possibly be changed. Clearly, if the node w is high up on the HST, the cost will be higher. We show that as the HST is built with a randomized algorithm the probability for w to be high on the HST is small so the expected cost is bounded by $O(\lg n)$.

We also demonstrate the experimental results for our scheme and compare with the only previous scheme RoamHBA [58].

4.1.3 Related work

Approximating a metric with probabilistic hierarchical well-separated trees was first proposed by Bartal [21, 22], with the motivation that many problems are easier to solve on a tree than on a general graph. Later, Fakcharoenphol *et al.* [56] improved the distortion to $O(\lg n)$ for any n node metric and this is tight. This previous works, however, were for a centralized environment. In an earlier work of ours [70], we developed a distributed algorithm to extract a HST in a sensor network setting. In [70], we also applied the HST to the problem of resource management and distributed matching. The application of HST on the minimum Steiner tree and k-center problem, as well as their maintenance when the agents are mobile, are new results.

4.2 Network Setup

Our following discussion is based on the assumption of a static sensor network consisting of n sensor nodes P and m mobile agents S. Agents are tracked by nearby sensor nodes called proxy nodes. We assume that every node is aware of the number of mobile agents, m.

A metric (P,d) has growth rate γ, if for any $v \in P$ there are $c_1 \cdot r^{\gamma-1} \leq f(r) \leq$

$c_2 \cdot r^{\gamma-1}$ nodes with distance exactly r from v, for some constant c_1 and c_2, $c_1 \leq c_2$. The doubling dimension of a metric space (P,d) is the smallest value λ such that every ball with radius r in P can be covered by λ balls of radius $r/2$. The doubling dimension of P is then defined to be $\dim(P) = \lg\lambda$. We call a metric to have a constant doubling dimension if its doubling dimension is bounded by a constant. It is not hard to see that a metric with bounded growth rate has constant doubling dimension, by a simple packing argument.

All sensors are uniformly distributed in a planar domain with nearby nodes directly communicating to each other. The communication network on the sensors is denoted as $G = (P,E)$. In our case, the network metric (P,d_G) is the minimum hop count metric in the sensor network. The vertices in the metric is the set of sensor nodes. The distance between two nodes is the minimum hop count value. In the following when we mention the "network metric" or the "original metric", we mean the minimum hop count metric (P,d_G). We assume that the minimum hop count metric of the communication graph has bounded growth rate of 2, and therefore constant doubling dimension.

4.3 HST review

In this section, we review the definition of α-hierarchically well separated tree (α-HST) and a distributed algorithm to compute the α-HST for a doubling dimension metric. The nodes of the HST are the nodes in the sensor network. The edges of the HST are virtual edges and the weights are redefined.

Definition 8 (α-HST). *A rooted weighted tree H is an α-HST if the weights of all edges between an internal node and its children are the same, all root-to-leaf paths have the same hop-distance, and the edge weights along any such path decrease by a factor of α as we go down the tree.*

Fakcharoenphol *et al.* [56] gives a centralized algorithm to compute a 2-HST metric d_H for an n-point metric (P,d_G). When we say the "tree metric", we mean the distance on the HST and denote it as d_H. d_H provides a $O(\lg n)$-probabilistically approximation on d_G, that is, for any $u,v \in P$, $d_H(u,v) \geq d_G(u,v)$, and $E[d_H(u,v)] \leq O(\lg n) \cdot d_G(u,v)$, where the expectation is taken over random

choices of the algorithm. The algorithm is as follows: We uniformly choose a random permutation $\pi : P \to \{1,2,\ldots,n\}$ of the nodes from the set of all permutations. We also fix a value β chosen uniformly from $[\frac{1}{2},1)$. For convenience, $B(u,r)$ denotes a ball with radius r centered at u, and D represents the network diameter. For each node u, we compute a $O(\lg n)$-dimensional *signature vector* $S(u)$, where the i-th element in the vector is

$$S(u)_i = arg \min_{v \in B(u,2^i\beta)} \pi(v)$$

for $i = 0$ to $\ell = \lceil \lg D \rceil + 1$. In other words, each node keeps the node with the smallest rank among all nodes within distance $2^i\beta$. $S(u)_\ell$ is the node with rank 1 for all nodes u. These signature vectors define the HST embedding of d. In particular, the leaves are nodes of P, the level i ancestor of a node u is $S(u)_i$, and the weights of all edges between level i and level $i-1$ is 2^i.

Since this algorithm is centralized, in [70] we provide a distributed implementation of the HST construction. The algorithm proceeds in a bottom-up fashion and compute the i-th element of the signature vector for every node in round i. The simple idea behind this is that, as i increases, only nodes with small ranks can be nominated and remained in the next round. The value of β, chosen uniformly at random from $[\frac{1}{2},1)$, is distributed to all nodes in the network by a single global flood. Initiatively, every node is a candidate, that is, $P_0 = P$. In round $i+1$, the nodes remaining in P_i flood the network up to distances $2^{i+1}\beta$. The flooding packets are cached at each node receiving these packets until the end of each round. Then each node u in the network will choose the node v_{min} with the lowest rank among all the nodes it receives in this round, and nominate it as its $(i+1)$-th level element for its signature vector, i.e. $S(u)_{i+1} = v_{min}$.

At the end of the algorithm, each node u keeps a signature vector $S_u(k)$, where $S_u(i)$ is u's i-th level ancestor. This information is enough for constructing the HST and convenient for our application.

As the algorithm proceeds, fewer and fewer nodes remain active though the flooding range increases. Therefore the total message cost is still near linear. We have proved in Chapter 2 the following lemma.

Lemma 9. *For a sensor network with n nodes, the total communication cost for*

constructing the 2-HST is $O(n \lg n)$ in expection, and each node uses expected storage space for $O(\lg n)$ node IDs.

For our applications, we list some of the properties of HST here.

Lemma 10 (α-HST properties). *Suppose the HST metric (H, d_H) corresponding to the original metric (P, d_G)*

1. *For any $u, v \in P$, $d_H(u,v) \geq d_G(u,v)$ and $E[d_H(u,v)] \leq O(\lg n) \cdot d_G(u,v)$.*
2. *For any $u \in P$, suppose its i-th level ancestor (from leaf to root) in H is u_i. We have $d_H(u_{i+1}, u_i) = \alpha \cdot d_H(u_{i-1}, u_i)$.*
3. *For $\forall i, j (1 \leq i, j \leq \ell, \ell$ is the HST height$)$, the distance between all the nodes in level i and j are of the same value.*

4.4 Maintaining approximate minimum Steiner tree

In this section, we have a static wireless sensor network with node set P. We want to show how to maintain a minimal Steiner tree for a set of agents S ($m = |S| \leq n = |P|$), which reside on the sensor nodes and may move in the sensor network. We will show how to construct an approximate minimal Steiner tree from the initial state, and how to maintain it with small update cost when some agent move from one sensor node to another.

4.4.1 Maintenance Algorithm

First we compute a 2-HST for the original metric network in a distributed way as stated in our previous section. Each node keeps a signature vector of the ancestors in each level. This process is implemented at the beginning of our application, and it will be used for all the following applications.

4.4.1.1 Compute approximate minimum Steiner tree

- Each agent is assigned to the nearest sensor node, denoted as the proxy node. A sensor node assigned by some agent sends a counter with the number of agents residing in that node to its parent.

- Each internal node generates a new counter by adding up the counters receiving from its children, and continues to send the counter to its parent.
- Finally, when the counter value equals m, this node will be the root of the minimal Steiner tree in the HST metric.
- After we get the minimal Steiner tree $H(S)$ on the HST H, then replace each edge uv of $H(S)$ with the shortest path between u, v. This gives us a Steiner tree $T(S)$ in the original metric.

Note that $T(S)$ is a logical tree on G. By replacing each edge of $H(S)$ with the shortest path in G we may end up with duplicate edges and possible cycles. One can remove duplicate edges or cut open the cycles to obtain a real Steiner tree on S in the graph G. This operation can only reduce the weight of the tree.

If the sensor nodes are not aware of the number of agents m in the network, we can solve this by sending notification to upper level until the HST root from the proxy nodes, then tracing back to the child of the lowest internal node u that gets only one notification from its children. And this node u is the root of the minimal Steiner tree.

Figure 12 shows the basic idea of the above algorithm. We will show the Steiner tree $T(S)$ is a $O(\lg n)$ approximation for the optimal minimal Steiner tree in the original metric later.

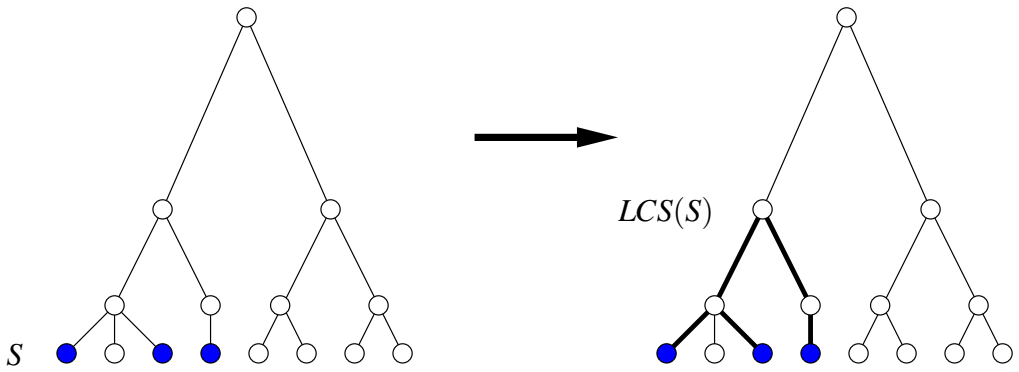


Figure 12. An example to show how to get minimal spanning tree on HST for a subset agents.

4.4.1.2 Maintenance under agent motion

As stated in our algorithm, when constructing a minimal Steiner tree on HST metric, each node will keep two types of information: a pointer to its parent in the minimal Steiner tree and a counter that counts the total number of agents located in the subtree rooted at this node.

When an agent moves from a proxy node p to a proxy node q, we have to update the minimal Steiner tree on the HST. The idea is that we only need to update the paths from p, q to their lowest common ancestor. In our next section, we show that the expected update cost is small.

- Both p and q send a notification upward until their lowest common ancestor u in the constructed HST.
- All the nodes along the path from p to u will decrease their counter by 1. And any node v with counter to be 0 will delete the edge to its parent in the tree.
- For those nodes on the path from q to u will increase their counter by 1. And add the path to the Steiner tree $H(S)$ if necessary (as well as the Steiner tree $T(S)$ in the original metric).

It is easy to see that, repairing the tree in this way is exactly the same as constructing from our previous algorithm. Thus we are able to maintain the minimum Steiner tree of the HST metric and the approximate ratio for the minimum Steiner tree in the original metric is kept.

4.4.2 Analysis and performance

We want to show that the minimal Steiner tree $T(S)$ extending from $H(S)$ is a $O(\lg n)$ approximation for the minimal Steiner tree in the original metric. And we show that the expected update cost is bounded.

Theorem 11. *For a set of agents S, the minimum Steiner tree $T(S)$ is a $O(\lg n)$ approximation for the optimal minimum Steiner tree $MStT(S)$ in the original graph G.*

Proof. For the tree $T(S)$, we use $w(T(S))$ to denote the total weight of all the edges of $T(S)$. Recall from the construction algorithm that $w(T(S)) \leq w(H(S))$,

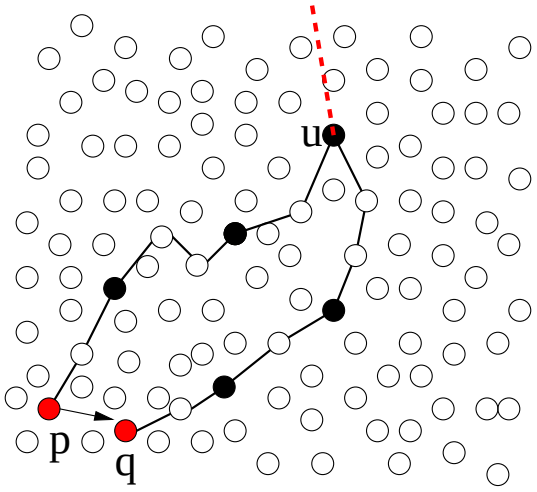


Figure 13. When a agent moving from p to q, we only repair the paths from p,q to their lowest common ancestor.

since the edge on the HST is always greater than the weight of the shortest path in G. So we just need to show $w(H(S)) \leq O(\lg n)w(MStT(S))$.

We construct a complete graph $G(S)$ on the agents only. The edge weight for two agents u,v is the minimum hop count value between u,v in the sensor network G. Now, the weight of the minimum Steiner tree $MStT(S)$ for the agents S in G is at least half of the weight of the minimum spanning tree $T'(S)$ of the agents in $G(S)$. Thus we only need to show $w(H(S))$ is no greater than $O(\lg n) \cdot w(T'(S))$, where $w(T'(S))$ is the weight of $T'(S)$.

We now 'lift' the minimum spanning tree $T'(S)$ to the HST metric H. For any edge uv in $T'(S)$, we take the shortest path on H, $P_H(u,v)$. The total weight of the path $P_H(u,v)$ is at most $O(\lg n) \cdot d_G(u,v)$, by the HST property. We will get a corresponding graph T'' in the HST metric. $w(T'') \leq O(\lg n) \cdot w(T'(S))$. T'' is a Steiner tree on H covering all the agents S. Thus its weight is at least the weight of the minimum Steiner tree on H, which is $H(S)$. This says $w(H(S)) \leq w(T'') \leq O(\lg n)w(T'(S)) \leq O(\lg n)w(MStT(S))$. Then we have $w(T(S)) \leq w(H(S)) \leq O(\lg n)w(MStT(S))$. □

For update cost, we may have to update a long path from leaf to the root. That is to say the update cost for 1 hop movement can be $O(D)$ in the worst case, where D is the diameter of G. But intuitively, due to the random rank permutation, the

probability for such kind of update will be quite small. In expectation the update cost is only $O(\lg n)$, as in the following theorem.

Theorem 12. *For a metric (P, d_G) with bounded growth rate and a HST H with metric (P, d_H), the expected update cost of the minimum Steiner tree on H for each hop an agent moves is bounded by $O(\lg n)$.*

Proof. Suppose an agent is located at a node p and moves to a neighboring node q. Then we want to compute the update cost for maintaining the minimal Steiner tree on the HST. Suppose the lowest common ancestor of p and q, $LCA(p,q)$, on the HST is at level i. Then the update cost will be bounded by $O(2^i)$. Now let's compute the probability that p,q's lowest common ancestor is at level i. Equivalently, p's level $i-1$ ancestor $S(p)_{i-1}$ is different from q's level $i-1$ ancestor $S(q)_{i-1}$ and p,q have the same level i ancestor $S(p)_i = S(q)_i$.

Let $D(p,r)$/$D(q,r)$ denote the disk with radius r centered at node p/q, $r = 2^{i-1}\beta$, where β is a constant parameter in the HST construction, $\beta < 1$. Recall in the HST construction the ancestor of p at level $i-1$ is the node with lowest rank inside the disk $D(p,r)$. Similarly, the ancestor of q at level $i-1$ is the node with lowest rank inside the disk $D(q,r)$. If p,q have different level $i-1$ ancestor, either $S(p)_{i-1}$ is inside $D(p,r) \setminus D(q,r)$ or $S(q)_{i-1}$ is inside $D(q,r) \setminus D(p,r)$. See Figure 14. As we assume a random permutation as the rank of the sensor nodes, then the probability that p,q have different level $i-1$ ancestors is

$$\frac{|D(p,r) \setminus D(q,r)|}{|D(p,r)|} + \frac{|D(q,r) \setminus D(p,r)|}{|D(q,r)|},$$

where $|D(p,r)|$ is the number of nodes inside $D(p,r)$.

For a metric (P, d_G) with growth rate γ, there are $c_1 \cdot r^{\gamma-1} \leq f(r) \leq c_2 \cdot r^{\gamma-1}$ nodes of P with distance exactly r from any node $v \in P$, c_1 and c_2 are some constants, $c_1 \leq c_2$. The number of nodes within distance r from v is at least $c_3 \cdot r^{\gamma}$, for some constant c_3. Thus we have

$$\begin{aligned} &\frac{|D(q,r) \setminus D(p,r)|}{|D(q,r)|} \leq \frac{|D(q,r) \setminus D(q,r-1)|}{|D(q,r)|} \leq \frac{c_2 \cdot r^{\gamma-1}}{c_3 \cdot r^{\gamma}} \\ =&\frac{c_2}{c_3} \cdot \frac{1}{r} \end{aligned}$$

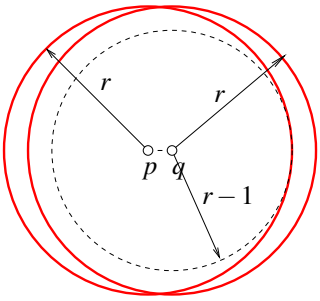


Figure 14. A figure to show when do we need update the lowest common ancestor at level i.

Similarly, we can show that

$$\frac{|D(p,r) \setminus D(q,r)|}{|D(p,r)|} \leq \frac{c_2}{c_3} \cdot \frac{1}{r}.$$

Thus we have,

$$\begin{aligned}
&\text{Prob}\{\text{the lowest common ancestor is at level } i\} \\
\leq &\frac{|D(p,r) \setminus D(q,r)|}{|D(p,r)|} + \frac{|D(q,r) \setminus D(p,r)|}{|D(q,r)|} \\
\leq &\frac{2c_2}{c_3} \cdot \frac{1}{r} = \frac{4c_2}{c_3\beta} \cdot \frac{1}{2^i}
\end{aligned}$$

The update cost of $H(S)$ if $LCA(p,q)$ is at ith level is proportional to 2^i. Thus the expected update cost for a node set P with bounded growth rate is

$$\begin{aligned}
&\sum_{i=1}^{\ell} \text{Prob}\{LCA(p,q) \text{ is at level } i\} \\
&\cdot \{\text{the tree update cost if } LCA(p,q) \text{ is at level } i\} \\
= &\sum_{i=1}^{\ell} O(2^i) \cdot \frac{4c_2}{c_3\beta} \cdot \frac{1}{2^i} = O(\lg n).
\end{aligned}$$

□

4.5 Maintaining approximate k-center

In this section, we give an algorithm to get an approximate solution for k-center problem.

Definition 13 (k-center). *Given a sensor network with metric (P, d_G), where d_G is taken on the shortest hop distance for any two node $p, q \in P$ in network G, for a set of agents (actually their proxy nodes) $S \subseteq P$ and an integer k, compute a node set $K = \{a_1, a_2, \ldots, a_k\} \subseteq P$ such that the maximum distance from any agent to its closest center,*

$$\max_{p \in S} \min_{a \in K} d(p, a),$$

is minimized.

We show that with the same data structure for maintaining the minimum Steiner tree on the HST, we also maintain a $O(\lg n)$ approximate solution for k-center problem for any k. As stated before, each node will keep two types of information: a pointer to its parent in the minimal spanning tree of S on H and a counter that counts the total number of agents located in the subtree rooted at this node.

For a set of agents S, the lowest common ancestor u of S will be a good candidate for the 1-center with a $O(\lg n)$ approximation. To find a good k-center solution, we simply take the lowest level on the HST such that the number of nodes with non-zero counter is no greater than k. These nodes are the k-centers. We show below that the approximation ratio is $O(\lg n)$ compared with the optimal k-center of G.

Theorem 14. *The k-center solution in the HST metric gives a $O(\lg n)$ approximation to the optimal k-center in G.*

Proof. We first prove the claim for $k = 1$. Suppose u is lowest common ancestor of S on the HST, i.e., the 1-center by our algorithm, and u is at level i. Suppose the height of u in the HST is h. Then for any agent $z \in S$, $d_G(z, u) \leq d_H(z, u) = h$. That is, the cost with u as the 1-center, denoted as $R(u)$, is at most h.

Suppose that OPT is the cost of the optimal 1-center problem in G. Suppose that x and y are the furthest pair in H. Then $OPT \geq \frac{1}{2} d_G(x, y)$. By Lemma 10 $d_G(x, y) \leq d_H(x, y) \leq O(\lg n) d_G(x, y)$. This means, $d_H(x, y) \leq O(\lg n) OPT$. Since u is the common ancestor of all the nodes in S, and x, y are the furthest pair. We know the lowest common ancestor for x, y must be u. Thus $d_H(x, y) = 2h$.

Putting everything together, we have $R(u) \leq h = d_H(x, y)/2 \leq O(\lg n) OPT$. This shows that the solution with u as the center has a cost at most $O(\lg n)$ the optimal cost in G.

Now we can extend the proof for k-center solution. Suppose we take the level i as the lowest level such that there are $\leq k$ nodes with at least one agent in their subtrees. We make these nodes as the k centers. The cost of the solution is at most the length of a path from the leaf to level i, say h. $h = 2^{i+1} - 1$. In addition, there are more than k nodes, $x_1, x_2, \cdots, x_f$, at level $i-1$, each has at least one agent in their subtree, $f > k$. The length of a path from the leaf to level $i-1$, h', is $2^i - 1$. The subtrees of x_i's are disjoint as well. We take one agent y_i from each subtree of x_i. Any two nodes y_i, y_j have their common ancestor at level i or higher. Thus $d_H(y_i, y_j) \geq 2h$, for any $1 \leq i, j \leq f$. By Lemma 10, $c \lg n \cdot d_G(y_i, y_j) \geq d_H(y_i, y_j)$ for some constant c. Thus $d_G(y_i, y_j) \geq 2h/(c \lg n)$. Since there are more than k agents that are pairwise of distance at least $2h/(c \lg n)$ apart. The optimal k-center solution must have at least two of the y_i's grouped to one cluster. That says, the optimal cost OPT is at least $h/(c \lg n)$. This means $h \leq O(\lg n)OPT$. □

When agents move, we do not explicitly maintain the level in which the centers stay on. In fact, if we use the structure for data aggregation at k centers, the HST and the minimum Steiner tree of the agents on H directly imply a solution for the aggregation. In particular, each agent sends its data upward along the HST tree. Internal node of the HST will take the data from the subtree and compute the aggregation. When an internal node has the aggregated data from all the agents in its subtree, it will report the aggregation to its parent on the HST. This way, the information is naturally aggregated on the internal nodes. For each level of the HST the aggregation result is at a subset of aggregation nodes, such that the distance from all agents to these aggregation nodes is not far away from the minimum possible.

4.6 Simulation

In this section, we implemented the above algorithms in MATLAB. The experiment focused on verifying our previous performance analysis. We also compared it with some existing algorithm. The cost in our simulation is counted as the hop distance.

4.6.1 Approximate minimal steiner tree construction

We implemented the approximate minimal Steiner tree algorithm here. The networks were generated by perturbing n nodes of the $\sqrt{n} \times \sqrt{n}$ grid in the $[0,1]^2$ unit square, by 2D Gaussian noise of standard deviation $\frac{0.3}{\sqrt{n}}$, and connecting two resulting nodes if they are at most $\frac{2}{\sqrt{n}}$ apart. Figure 15 is an example of the minimal Steiner tree from the above algorithm.

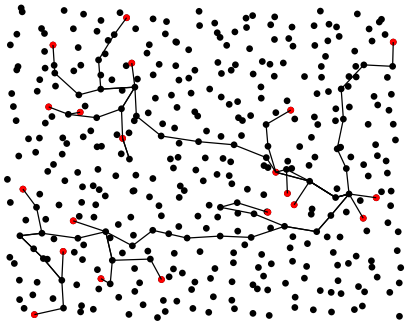


Figure 15. An example of the minimal Steiner tree computed from the HST. The agents are in red. The network size is 400, and the agent size is 20.

4.6.2 Cost comparison with MST

We implemented our algorithm for the approximate minimal Steiner tree with different network sizes and a fixed agent set size. We generated random perturbed grid networks in the same way as in the previous section. Network size ranged from 400 to 2500, and the agent set size was kept at 100. We sampled 100 networks for each network size. For each network, we randomly generated an agent set of size 100. And an approximate minimal Steiner tree was constructed with our algorithm. We also computed the shortest path between each pair in the agent set. Then we had a complete graph $G(S)$. The vertices were the agent set, and the weight for each edge was the shortest hop distance in the original network. Now we computed the minimal spanning tree in this complete graph. After that, we got a Steiner tree in the original network, whose total weight is at most twice the cost of the minimum Steiner tree of S. In our following discussion, we use MST to denote the cost of

this tree. Figure 16 is a minimal spanning tree example for the same data with Figure 15.

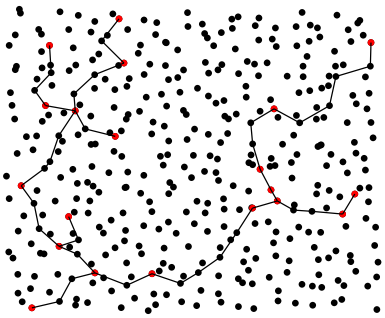


Figure 16. The minimal spanning tree of the agents.

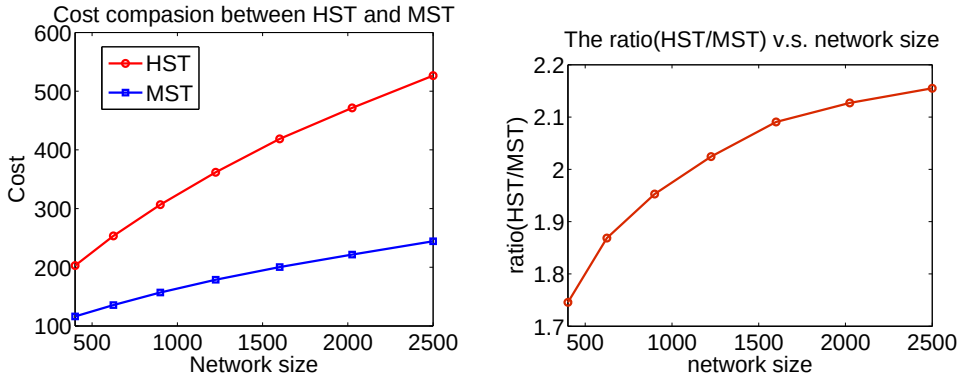
Figure 17(a) demonstrates the cost for our solution and the optimal spanning tree with different network size and a fixed number of agents. Figure 17(b) shows the ratio of our solution over MST. Figure 17(c) shows the ratio plotted in a log scale. The cost of our algorithm is not far away from the MST case, and is always within a factor of 2.2. The ratio increases slowly with the network size. According to Figure 17(c), the approximation ratio almost linearly depends on $\lg n$, where n is the network size.

We also implemented our algorithm for a fixed network size and varying agent set size. Now we fixed the network size to be 1000, and the agent set size ranged from 50 to 400.

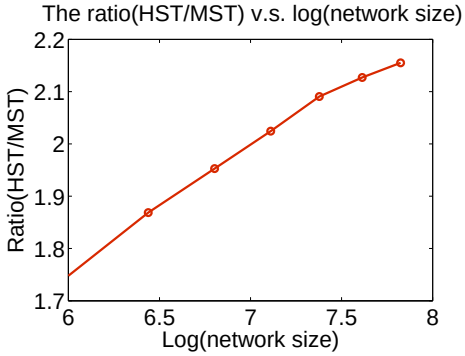
Figure 18(a) shows the cost of our solution and MST with different agent size when the network size n is fixed. According to Figure 18(b), the cost ratio between the HST and MST solution decreases when the number of agents increases.

4.6.3 Comparison with RoamHBA

In [58], the authors proposed RoamHBA to maintain group connectivity in sensor networks. In RoamHBA, the algorithm will choose a horizontal or vertical line across the network as the backbone. The backbone is constructed by choosing

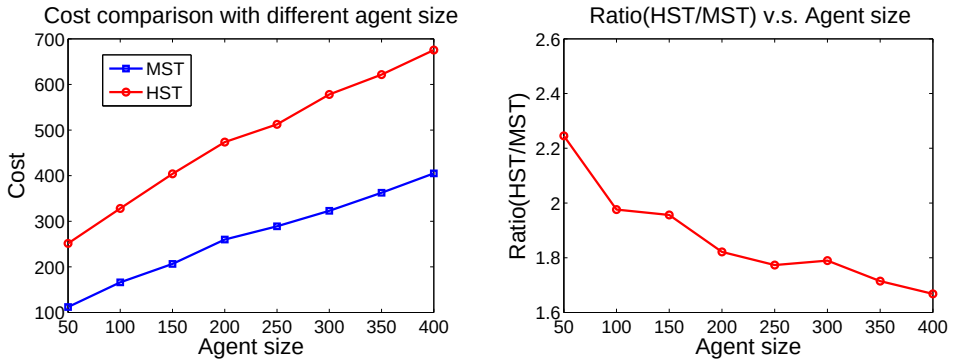


(a) Cost comparison for different network size (b) Cost ratio for different network size.



(c) Cost ratio for log(network size)

Figure 17. Cost comparison between HST and MST for different network size but the agent set size fixed to be 100.



(a) Cost comparison for different network size (b) Cost ratio for different network size

Figure 18. The cost comparison between the HST and MST solution, with different agent set size but the network size fixed to be 1000.

a median node with greedy forwarding in the desired direction. The rest of the agents connect to the backbone by greedy forwarding to the backbone. The greedy forwarding for node (x,y) is in the following way (assuming the horizontal direction): choose the neighbor that is nearest to point $(x+R,y)$ and $(x-R,y)$. Figure 19 shows an approximate minimal Steiner tree from RoamHBA from the same data as Figure 15 and Figure 16.

In this experiment, we generated random perturbed grid networks from the same distribution as in our previous experiment. Network size ranged from 400 to 2500, and the agent size was kept fixed at k = 100. We sampled 20 networks for each network size. For each network, we choose k nodes from the network as the agent set. Then we computed the approximate minimal Steiner tree from HST and RoamHBA respectively.

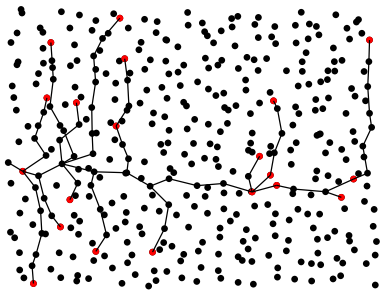


Figure 19. An example of the Steiner tree computed with RoamHBA.

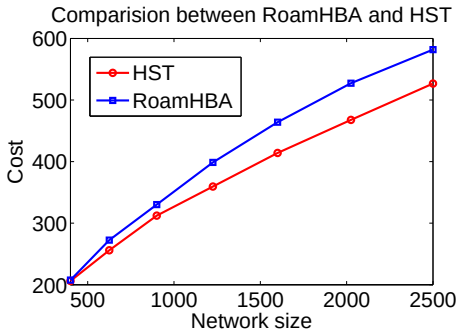


Figure 20. Cost comparison between HST solution and RoamHBA solution, with agent size fixed to be 100.

According to our simulation result in Figure 20, our algorithm is better than RoamHBA with respect to the cost of the tree. But when the agent size is small, RoamHBA may have some advantage.

4.6.4 Update cost

In this experiment, we generated random perturbed grid networks from the same distribution as in our previous experiment. Network size ranged from 400 to 2500, and the agent size was kept fixed at $k = 100$. We sampled 15 networks for each network size. For each network, we chose k nodes from the network as the agent set. And we set a random moving direction for each agent. In each time step, the agent moved to $r/2$ (r is the communication range.) away along its moving direction. When an agent hit the network boundary, it bounded back along the reflected direction. In our simulation, we let all the agent move 1 step one by one. And repeated this 100 times. The we took the average of each time step and each node for the update cost for that network size.

From Figure 21, we see that the update cost almost changes linearly with logarithm of the network size. In fact, the update cost for each movement step is on average only $6 \sim 7$.

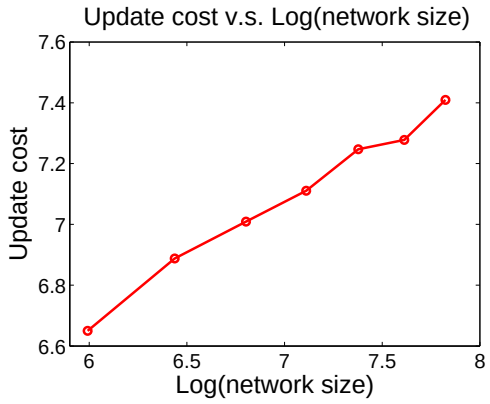
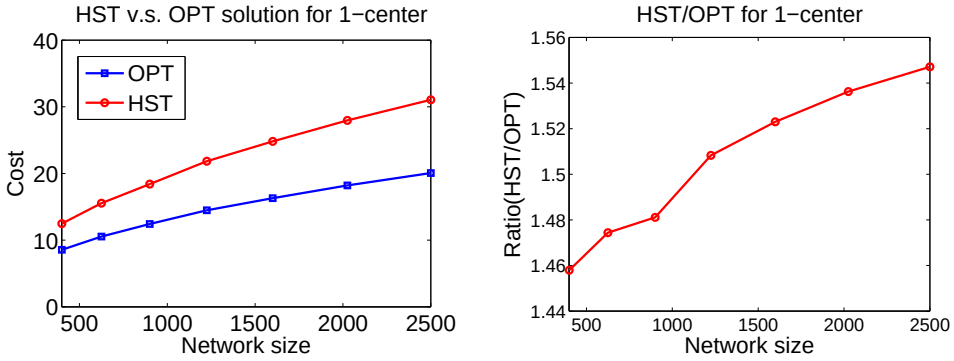


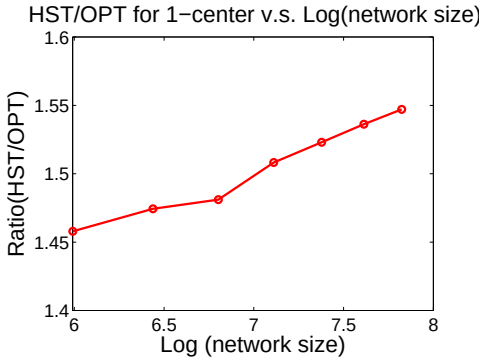
Figure 21. Update cost changes with log(network size) with the agent size fixed to be 100.

4.6.5 1-center

In this experiment, we implemented our algorithm for approximate 1-center solution. we generated random perturbed grid networks as our previous experiments. Network size ranged from 500 to 2500, and the agent size was kept fixed at m =100. We sampled 100 networks for each network size. For each network, we chose m agents randomly from the network nodes. Then we solved the 1-center problem with our algorithm. For comparison, we also computed the optimal solution for 1-center, and compared their cost.



(a) Cost comparison for different network size (b) Cost ratio for different network size



(c) Cost ratio for log(network size)

Figure 22. Cost comparison between HST and OPT for 1-center, with varied network size but agent size fixed to be 100.

Figure 22(a) gives the cost for our approximate 1-center solution and the optimal 1-center solution. Figure 22(b) is the ratio between them. The ratio increases

slowly with the network size. Figure 22(c) shows how the ratio changes with logarithm of the network size. We can see the nearly linear relationship between them. So our algorithm generates an $O(\lg n)$ approximation solution for the 1-center problem.

4.7 Conclusion

We show in this chapter that the hierarchical well-separated tree, extracted from the underlying network, can be useful in maintaining $O(\lg n)$ approximate solution for the minimum Steiner tree problem or the k-center problem. In particular, the agents do not need any information about the location of other nodes. The algorithm is also simple and distributed. This leads to applications such as maintaining group communication and aggregation nodes among a set of mobile agents.

Chapter 5

Resilient and Low Stretch Routing Through Embedding into Tree Metrics

5.1 Introduction

This chapter considers a fundamental problem of designing routing schemes that give low stretch and are resilient to node failures. We consider a metric (P,d) on n nodes (possibly as the shortest path metric of a given network) and examine routing schemes (together with proper routing tables) to direct a message to the destination. In particular, the result we present in this chapter is a routing structure, constructed in a distributed manner such that each node of P keeps routing information of size $O(\log n)$, the route discovered has constant stretch (e.g., a constant factor longer than the metric distance), and the routing structure is robust to node failures, where a single node failure will only disconnect $O(1/n)$ fraction of the routes between all possible pairs.

The technique we use in this chapter is through embedding into tree metrics. Given a metric (P,d), the simplest way to route is probably taking a spanning tree to guide message routing. This has a number of benefits, as a tree metric is a much simpler metric with many special features. For example, between any two vertices in a tree, there is a unique simple path connecting them, and the unique path can

be found in a local manner by first traversing up the tree towards the root, and traversing down the tree at the lowest common ancestor. There is a simple labeling scheme such that one can use routing table of $O(\log n)$ bits at each node to support routing on a tree [12].

However, routing on a spanning tree of the metric has a number of problems, in particular, the *poor stretch* and *lack of resilience*. The path on a tree might be much longer than the metric distance. Take the shortest path metric of a cycle of n vertices, any spanning tree will separate some pair of vertices, adjacent on the cycle, by distance $n-1$. To see this, consider one node u on the cycle as the root of the tree. If u has two or more subtrees, the nodes other than u are placed to the subtrees. There must be a pair of nodes, adjacent on the cycle, that are placed on different subtrees. Their distance on the spanning tree must go through the root and is at least $n-1$. Note that we can always choose a root with multiple subtrees. Thus the claim is true. That is, the distortion introduced by routing on a spanning tree is factor of $\Omega(n)$ of their true distance. A more serious problem of routing on a tree is due to the lack of robustness to node failures. If a node fails or decides not to cooperate and stops forwarding messages, the tree is broken into pieces and in the worst case quadratically many pairs have their paths disconnected.

In this chapter we use embedding into tree metrics for efficient, scalable routing, but address the shortcomings regarding stretch and resilience. Instead of using one tree, we use simply two trees. The basic idea is that if the route on the first tree has a poor stretch, the route on the second has a good stretch and one can always use the shorter path of the two. Regarding node failures, if a node u fails and the path between two nodes x, y is disconnected as it goes through u, the path connecting x, y in the second tree hopefully does not contain u and still remains valid. We briefly elaborate our technical approach and then relate to prior work.

5.1.1 Our Results

The tree embedding we use follows from the embedding of a general metric into tree metrics with low distortion. Given a metric (P,d) we embed it to a hierarchically well-separated tree (HST), defined as a rooted weighted tree such that: the weight of all edges between a node and its children are the same; the edge weights

along any path from the root to a leaf are decreasing by a factor of α. In this chapter we simply take $\alpha = 2$. The leaf nodes of the HST are 1-to-1 mapped to nodes in P and internal nodes of the HST are also mapped to nodes of P although certain nodes may appear multiple times. The embedding of (P,d) into the tree metric leads to distortions of the metric distances. As discussed earlier, using a fixed tree one cannot avoid the worst case distortion of $\Omega(n)$. But if one build a randomized tree, chosen from a family of tree metrics, the *expected* distortion can be bounded by $O(\log n)$. Thus using this tree for routing one immediately obtains $O(\log n)$ stretch routing with low routing overhead. Approximating a metric with probabilistic hierarchical well-separated trees was first proposed by Bartal [21, 22], with the motivation that many problems are easier to solve on a tree than on a general graph. Later, Fakcharoenphol *et al.* [56] improved the distortion to $O(\log n)$ for any n node metric and this is tight.

The results we prove in this chapter are mainly in three pieces

- Using two HSTs, randomly constructed with independent seeds, we show that the stretch can be improved to a constant. That is, for any two nodes x, y, between the two paths in the two HSTs respectively, one of them is short and is at most a constant factor of the metric distance between x, y.
- Regarding the resilience of using one HST for routing, we show that for any node failure, the number of pairs with their routes on the HST disconnected is at most a fraction of $O(\log \Delta/n)$ of all pairs, where Δ is the aspect ratio of (P,d), defined as the further pair distance versus the closest pair distance. When Δ is polynomial in n the bound is small as $O(\log n/n)$ but in the worst case when the aspect ratio is exponential the bound can be bad.
- Using two HSTs we substantially improve the routing resilience. We build two HSTs with random, independent seeds. In the case of a node failure, we show the number of pairs with their routes on both HSTs disconnected is at most a fraction of $O(1/n)$ of all pairs, thus removing the factor of $O(\log \Delta)$ compared with the case of a single HST.

In this chapter we focus on the robust of using the HSTs as the routing structures. Thus a node failure means that a node does not participate in the HST-based routing. It is different from a physical node failure – for example, if the metric

(P,d) is the shortest path metric of a given network G, then a node may appear on a number of shortest paths and the physical breakdown of a node will not only lead to failures of HST-based routing but more seriously change the metric. This is not the focus in our thesis.

The results hold for metrics with 'geometric growth', that is, the number of nodes within distance r from any node grows as a polynomial function of r, not exponential (as in the case of a balanced binary tree). Such a family of metrics appears in many real settings, either due to physical constraints such as in wireless networks and VLSI layout networks, or due to geographical constraints such as in peer-to-peer overlay networks [118, 125, 126]. In the next section we introduce the rigorous definitions and elaborate the precise assumptions for each of the results.

Last remark that in the case that (P,d) is the shortest path metric of a given network G, there is a distributed algorithm that constructs the the HST with a total number of messages bounded by $O(n \log n)$. In addition, each node is given a label of size $O(\log n)$ such that one can route on an HST using only the node label information. Thus the entire scheme of using one or multiple HSTs for robust, low-stretch and efficient routing can be implemented in a completely de-centralized manner.

5.1.2 Prior Work

The problem of routing is such a fundamental one that it has been studied extensively with numerous prior work. We only have the space to review some most relevant ones to our work.

5.1.2.1 Low stretch routing

The traditional routing methods as used for the Internet are essentially shortest path routing. Essentially each node keeps a routing table of size $O(n)$ to save the next hop on the shortest path for each destination. This is equivalent to maintaining n shortest path trees, rooted on every node. From this perspective, our approaches defines one or two global trees, rather than one tree per node. By doing so we can substantially reduce the size of the routing table from $O(n)$ to $O(\log n)$, while still keeping the routing stretch by a constant.

From a theoretical aspect, compact routing that minimizes the routing table size while achieving low stretch routing has been studied extensively [122]. There are two popular models in the literature, the *labeled routing model* and *name-independent routing*. In the labeled routing model [43, 51, 152], one is allowed to produce for each node a label (typically of polylogarithmic size) such that routing is done with the labels of the source and destination. In the name-independent model [8, 98], the nodes are given generic IDs that are independent of the routing scheme. Thus routing is inherently more difficult as the routing scheme needs to also find out where the node is. Generally speaking, the theoretical results in compact routing in a graph whose shortest path metric has a constant doubling dimension are able to obtain, with polylogarithmic routing table size, $1+\varepsilon$ stretch routing in the labeled routing scheme (see [35] and many others in the reference therein), and constant stretch factor routing in the name-independent routing scheme [6, 98] (getting a stretch factor of $3-\varepsilon$ will require linear routing table size [6]). The results here are all centralized constructions and aim to get the best asymptotic bounds. Our focus of using tree embedding is to obtain practical easy routing solutions with theoretical guarantee. Further, the compact routing schemes above have no consideration of robustness to node failures.

5.1.2.2 Resilient Routing

Routing methods that can recover from node or link failures receive a lot of interests recently. Path splicing [112], proposed for increasing routing reliability on the Internet, obtains robustness by using multiple metrics. Given a weighted graph, one perturbs the weights of the edges and produces a different shortest path tree rooted at each node. These multiple shortest paths trees are used in combination to generate a routing path in case of in-transit link failures. Traffic in the network can freely switch between different shortest path trees, which results in a large number of braided routing paths. The overhead of switching between different trees is done by just changing a few bits in the packet header. This supports fast recovery from link or node failure and ensures low end-to-end delay with minimum changes to the current Internet infrastructure. Our approach of using multiple tree metrics has certain similarity with path splicing. In fact we can also use multiple HSTs

to accommodate in-transit failures. Whenever the route by following one HST encounters a problem we can use the second HST to route towards the destination. The difference of our method is that we do not keep separate shortest path trees (or under perturbed metrics) rooted at each node. Thus our storage overhead is substantially better. We actually evaluate the routing performance of using two HSTs and using perturbed shortest paths in the simulation section. The observation is that we have roughly the same routing robustness, our stretch is a little higher but we substantially save on routing table size. Two random trees provide a lot of important properties in scalability and path diversity [75].

A similar idea to achieve routing resilience is to use the notion of 'protection routing' [101], where one looks for a set of routing trees, one for each destination, such that all nodes have a standby alternate next-hop available when the primary next-hop becomes unreachable. In some sense the protection routing makes it rigorous the use of multiple routing trees that 'protect' each other. But the computation of protection routing is NP-hard and the routing method is a centralized approach. There are heuristic efforts with related ideas for Internet routing such as fast re-routing [140], Loop-free alternate (LFA) [18], O2 [131], DIV-R [130] and MARA [119]. These methods have no theoretical guarantee.

We start by introducing the hierarchical separated trees (HSTs) and embedding into HSTs in Section 5.2. Our results for using two trees to achieve better stretch and resilience are presented in Section 5.3 and Section 5.4 respectively. We also evaluate the performance of using two HSTs for routing in two settings, representing wireless sensor networks and Internet backbone graphs.

5.2 Preliminaries

5.2.1 Metrics With Geometric Growth

Given a metric (P,τ), there is a distance function $\tau(u,v)$ for any two nodes $u,v \in P$ that satisfies triangular inequality. In the scenario that we are given a network $G = (P,E)$, τ is typically the length of the shortest path in G. An important family of metrics is the metrics with 'geometric growth'. Rigorously there are several definitions for capturing metrics with 'geometric growth'.

Let $B(p,r) = \{v \,|\, \tau(p,v) \leq r\}$ denote the radius r ball centered at p. In [92], a metric has bounded *expansion rate* (also called the KR-dimension, counting measure) k_1 if $|B(v,2r)| \leq k_1|B(v,r)|$ for a constant k_1; and in [81], a metric has bounded *doubling dimension* k_2 if $B(v,2r)$ is contained in the union of at most k_2 balls with radius r for a constant k; in [71, 104], a metric has upper bounded growth rate *growth rate* k_3 if for every $p \in V$ and every $r \geq 1$, $|B(p,r)| \leq \rho r^{k_3}$, for a constant ρ and k_3. A few sensor network papers [136, 158] consider a model when the growth rate is both upper and lower bounded, i.e., $\rho^- r^{k_4} \leq |B(p,r)| \leq \rho^+ r^{k_4}$ for a constant k_4, where $\rho^- \leq \rho^+$ are two constants. We denote the family of metrics with constant expansion rate, constant doubling dimension, constant upper bounded growth rate, and constant upper and lower bounded growth rate as $\mathcal{M}_{\text{expansion}}, \mathcal{M}_{\text{doubling}}, \mathcal{M}^+_{\text{growth}}, \mathcal{M}_{\text{growth}}$ respectively. It is not hard to see that

$$\mathcal{M}_{\text{growth}} \subseteq \mathcal{M}_{\text{expansion}} \subseteq \mathcal{M}_{\text{doubling}} \subseteq \mathcal{M}^+_{\text{growth}}.$$

See [71, 81] for more discussions. In terms of the results in this chapter the detailed definitions actually matter. In the following we will make it clear which definition is needed for each result.

5.2.2 Embedding into Tree Metrics

Given two metric spaces (X, d_X) and (Y, d_Y) an injective mapping $f : X \to Y$ is called an *embedding* of X into Y. We can scale up Y to make the embedding to be *non-contractive*, i.e., for any $u \neq v \in X$: $d_Y(f(u), f(v)) \geq d_X(u,v)$. We say Y *dominates* X. The distortion of the pair u, v is

$$\text{dist}_f(u,v) = \frac{d_Y(f(u), f(v))}{d_X(u,v)}.$$

The distortion of the embedding f is

$$\text{dist}(f) = \max_{u,v \in X} \text{dist}_f(u,v).$$

Given a metric (P,d), we embed it to a tree metric and use the tree metric to guide message routing. Ideally we want the route length to be close to the metric distance. That is, we'd like to embed P to a tree metric such that the distortion is

small. As shown in the introduction, it is not possible to get any distortion $o(n)$ using a single tree. However, it is known that for any metric (P,d), one can use randomization and choose a tree randomly from a family of trees such that the *expected* distortion is only $O(\log n)$. Such a tree is a type of a *hierarchical well-separated tree H*, as defined below.

Definition 15 (α-HST [21]). *A rooted weighted tree H is an α-HST if the weights of all edges between an internal node to its children are the same, all root-to-leaf paths have the same hop-distance, and the edge weights along any such path decrease by a factor of α as we go down the tree.*

In this chapter we focus on 2-HST. The leaves of T are the vertices in P, and the internal nodes are Steiner nodes. Fakcharoenphol, Rao and Talwar [56] have shown that for any metric (P,d) one can find a family of trees such that a randomly selected metric from the family has expected distortion of $O(\log n)$. Rigorously, we say that such a family $O(\log n)$-probabilistically approximate (P,d).

Definition 16 (γ-probabilistically approximation [56]). *Let $\mathcal{S}$ be a family of metrics over P, and let $\mathcal{D}$ be a distribution over $\mathcal{S}$. $(\mathcal{S},\mathcal{D})$ γ-probabilistically approximates a metric (P,d) if every metric in S dominates d, and for every pair of vertices $(u,v) \in P$, $E_{d' \in \mathcal{S}}[d'(u,v)] \leq \gamma \cdot d(u,v)$. A metric (P,d') dominates (P,d) means that $d'(u,v) \geq d(u,v)$ for all $u,v \in P$.*

Using a single tree to (probabilistically) approximate a general metric, the best (expected) distortion is $\Omega(\log n)$. Thus the bound in [56] is essentially tight. In the next section we show by selecting two trees from the family $\mathcal{S}$, and taking the shorter path from the two trees, the expected distortion can be improved to $O(1)$, if P has constant KR dimension. Here we first describe the FRT algorithm [56] to build one tree, uniformly randomly selected from the family $\mathcal{S}$. To build two trees we simply choose two trees uniformly randomly from $\mathcal{S}$, i.e., run the same algorithm twice with different random seeds.

5.2.3 Review of The FRT Algorithm

Without loss of generality, we assume that the smallest distance between any two vertices in P is 1 and the diameter of P is Δ. The aspect ratio is also Δ. Assume

$2^{\delta-1} < \Delta \leq 2^{\delta}$.

The FRT algorithm proceeds in a centralized manner by computing a hierarchical cut decomposition $D_0, D_1, \cdots, D_\delta$.

Definition 17 (Cut decomposition). *For a parameter r, a r-decomposition of a metric (P,d) is a partitioning of P into clusters, each centered at a vertex with radius r.*

Definition 18 (Hierarchical cut decomposition). *A hierarchical cut decomposition of (P,d) is a sequence of $\delta+1$ nested cut decompositions $D_0, D_1, \cdots, D_\delta$ such that*

- *$D_\delta = P$,* i.e.*the trivial partition that puts all vertices in a single cluster.*
- *D_i is a 2^i-cut decomposition, and a refinement of D_{i+1}. That is, each cluster in D_{i+1} is further partitioned into clusters with radius 2^i.*

To find the hierarchical cut decomposition, one first chooses a random permutation $\pi : P \to \{1,2,\cdots,n\}$ of the nodes. We use $\pi(i)$ to denote the node with rank i in the permutation. We also fix a value β chosen uniformly at random from the interval $[1,2]$. For each i, compute D_i from D_{i+1} as follows. First set β_i to be $2^{i-1}\beta$. Let S be a cluster in D_{i+1}. Each vertex $u \in S$ is assigned to the first (according to π) vertex v within distance β_i. We also say that u nominates v. Each child cluster of S in D_i then consists of the set of vertices in S assigned to the same center. We denote the center of a cluster C by center(C). Note that all clusters in D_i have radius $2^{i-1} \leq 2^{i-1}\beta \leq 2^i$. Remark that a node can nominate a center outside of its current cluster in D_{i+1} and one node can be the center for multiple clusters.

An alternative view of the hierarchical cut decomposition is to define for each node u a δ-dimensional *signature vector* $S(u)$. The i-th element in the vector is the lowest rank node within distance $2^i\beta$.

$$S(u)_i = \arg \min_{v \in B(u,2^i\beta)} \pi(v)\,, \tag{2}$$

where $B(p,r)$ is the collection of nodes within distance r from node p. A cluster at level i contains all the nodes with the same prefix $[1,i]$ of their signature vectors.

To turn the hierarchical cut decomposition to a 2-HST, the points of P are the leaf nodes of the HST and each internal node in the HST corresponds to a cluster

of nodes in the hierarchical partitioning. The refined clusters in D_{i-1} of a cluster C in D_i are mapped to children of C. The root corresponds to D_0. We can also use the center u of a cluster C as the representative node of C in the HST. Thus the root of the HST has $\pi(1)$ as its representative node. Denote by P_i the centers of the clusters in D_i. P_i is the set of node that are 'nominated' by others at level i.

The HST has $\delta+1$ levels, at $0,1,\cdots,\delta$. The level i has a number of internal nodes in the HST corresponding to P_i. The edge weight connecting a cluster C in D_i to its children clusters in D_{i-1} is 2^i, i.e., greater than the radius of the cluster C. Clearly the HST metric dominates (V,d), as one only relaxes the distances. For any two nodes u,v, suppose that they are first separated in different clusters in the decomposition D_i, i.e., their lowest common ancestor in the HST is at level $i+1$. In this case we have their distance on the tree to be $d_H(u,v) = 2\sum_{j=1}^{i} 2^j = 2^{i+2}$. Fakcharoenphol, Rao and Talwar [56] proved that $d_H(u,v) \leq O(\log n)d(u,v)$, in expectation over all random choices of β and π.

5.2.4 Distributed Implementation of the Tree Embedding

The algorithm for constructing 2-HST in [56] is centralized. In Chapter 2 a distributed algorithm to implement a 2-HST is proposed, when the metric (P,d) is the shortest path metric of an underlying network G. The hierarchical decomposition is replaced by a bottom-up restricted flooding from the centers P_i in round i. Notice that if a node is not nominated by anyone at level i, it will never be nominated for levels $j \geq i$ — if a node is not the lowest rank node of anyone else within distance $\beta 2^i$, it cannot be so for any node within distance $\beta 2^j \geq \beta 2^i$. Thus the set of nodes that are eligible to be nominated will be fewer and fewer as level goes up. Thus only nodes in P_i are candidates to be nominated in round $i+1$. $P_{i+1} \subseteq P_i$ and $P_0 = P$. Recall that P_i is the subset of nodes that have been nominated and 'survive' round i and only these nodes will need to flood up to distance $2^{i+1}\beta$ in round $i+1$. Each node u will maintain the signature vector and using the received messages from relevant nodes in P_i one can find the value $S_{i+1}(u)$. Notice that there are more nodes at lower levels and they flood up to a shorter distance; and there are fewer nodes at higher nodes that flood up to a longer distance.

With the property above, one can show that the total number of messages transmitted during the distributed construction of an HST can be bounded by $O(n \log n)$, if we consider a network of n vertices such that the shortest path metric has constant doubling dimension. In addition, each node is given a label of size $O(\log n)$ such that one can route on an HST using only the node label information. We refer to [70] for the details and simply emphasize here that efficient, distributed algorithms for constructing HSTs exist.

5.3 Constant Distortion Routing Using Two HSTs

Starting from this section we examine the properties of routing using *two* trees, instead of one. As shown earlier that using one HST, we can support distributed routing between any pair of nodes such that the expected path stretch is $O(\log n)$. Here we show that using two trees we can get $O(1)$ stretch for metrics with constant expansion rate. The reason that we use the family of metric $\mathcal{M}_{\text{expansion}}$ is because the result does not work for the larger family $\mathcal{M}_{\text{doubling}}$. We omit the lower bound construction due to page limitations. Recall that an n-point metric (P,d) has expansion rate k if $|B(p,2r)| \leq k \cdot |B(p,r)|$, where $B(p,r)$ is the set of points within distance r from the point $p \in P$ [92].

5.3.1 Constant Distortion Embedding in Two HSTs

For a given metric (P,d) with expansion rate k, we build two HSTs, H_1 and H_2 with the algorithm in [56]. For any two points u,v in P, we define the distance between them to be the minimum shortest path in the two trees. That is $d_H(u,v) = \min\{d_{H_1}(u,v), d_{H_2}(u,v)\}$.

Theorem 19. *For any metric (P,d) with expansion rate k and two HSTs H_1, H_2, there is a constant c such that for any two nodes $u,v \in P$,*

$$E[d_H(u,v)] = E[\min\{d_{H_1}(u,v), d_{H_2}(u,v)\}] \leq c \cdot k^4 \cdot d(u,v).$$

For two nodes $u,v \in P$, denote their lowest common ancestor (LCA) in H_i by $\text{LCA}_i(u,v)$, for $i = 1,2$. And denote $\text{LCA}(u,v) = \min\{\text{LCA}_i(u,v), i = 1,2\}$. Thus

$d_H(u,v) = 2^{i+2}$ if LCA(u,v) is at $i+1$. Now we have

$$E[d_H(u,v)] = \sum_{i=0}^{\delta-1} \text{Prob}\{\text{LCA}(u,v) \text{ is at level } i+1\} \cdot 2^{i+2}.$$

With the following Lemma that bounds the probability that LCA(u,v) is at $i+1$, we can prove the Theorem.

Lemma 20.

$$\begin{aligned} & \textit{Prob}\{\text{LCA}(u,v) \textit{ is at level } i+1\} \\ \leq & \begin{cases} 0, & \textit{if } 2^{i+2} < d(u,v); \\ 3k^4 \cdot d^2(u,v)/2^{2i-4}, & \textit{if } 2^{i-2} \geq d(u,v). \end{cases} \end{aligned}$$

With the above lemma, we can prove Theorem 19 easily. Suppose j^* is the smallest i such that $2^{i+2} \geq d(u,v)$,

$$\begin{aligned} E[d_H(u,v)] &= \textstyle\sum_{i=0}^{\delta} \text{Prob}\{\text{LCA}(u,v) \text{ is at level } i+1\} \cdot 2^{i+2} \\ &\leq \textstyle\sum_{i=j^*+4}^{\delta} [3k^4 \cdot \frac{d^2(u,v)}{2^{2i-4}}] \cdot 2^{i+2} + \sum_{i=j^*}^{j^*+3} 2^{i+2} \\ &\leq 2^7 \cdot 3k^4 \cdot d^2(u,v)/2^{i^*} + 14d(u,v) \\ &\leq (96k^4 + 14) \cdot d(u,v). \end{aligned}$$

To prove Lemma 20, we first evaluate the probability that in one tree, say, H_1, the probability that u,v have a lowest common ancestor at level j, $1 \leq j \leq \delta$.

Lemma 21.

$$\begin{aligned} & \textit{Prob}\{\text{LCA}_1(u,v) \textit{ is at level } i+1\} \\ \leq & \begin{cases} 0, & \textit{if } 2^{i+1} < d(u,v); \\ k^2 \cdot d(u,v)/2^{i-2}, & \textit{if } 2^{i-2} \geq d(u,v). \end{cases} \end{aligned}$$

Proof. First, if $w = \text{LCA}_1(u,v)$ is at level $i+1$, then $d(w,u) \leq \beta_{i-1} \leq 2^i$, $d(w,v) \leq \beta_{i-1} \leq 2^i$. By triangle inequality $d(u,v) \leq d(u,w) + d(w,v) \leq 2^{i+1}$. Thus in the first case of the lemma, the probability is 0. Suppose j^* is the smallest i such that $2^{i+2} \geq d(u,v)$. In the following we focus on the second case, i.e., $i \geq j^* + 4$.

If u,v belong to different clusters at level i, we say that the decomposition D_i *separates* u,v at level i. Thus LCA$_1(u,v)$ is at level $i+1$ if and only if D_i separates u,v and $D_j (j > i)$ does not. Thus,

$$\text{Prob}\{\text{LCA}_1(u,v) \text{is at level } i+1\} \leq \text{Prob}\{D_i \text{ separates } (u,v)\}.$$

Take this level i such that D_i separates u, v. There is a node w such that one of u, v is first assigned to w and the other is not. We say that w *settles* the pair u, v at level i. Such a node w is unique, as once the pair u, v is settled it won't be settled again. Thus we will consider the union of the probability for each node w of P to possibly settle u, v. If w settles u, v and u is assigned to w, we say w *cuts* u out. Summarizing the above, we have $\text{Prob}\{D_i \text{ separates } (u,v)\} = \sum_w \text{Prob}\{w \text{ settles } u,v\} = \sum_w \text{Prob}\{w \text{ cuts } u \text{ out}\} + \sum_w \text{Prob}\{w \text{ cuts } v \text{ out}\}$.

Let K_i^u be the set of nodes in P within distance 2^i to node u, and let $k_i^u = |K_i^u|$. We rank the node in K_i^u with increasing order of distance from u: $w_1, w_2, \cdots, w_{k_i^u}$. For a node w_s to cut u out of the pair u, v at level i, it must satisfy the following conditions:

1. $d(u, w_s) \leq \beta_i$.
2. $d(v, w_s) > \beta_i$.
3. w_s settles u, v.

Thus β_i must lie in $[d(u,w_s), d(v,w_s)]$. But we have $d(v,w_s) \leq d(v,u) + d(u,w_s)$ by triangle inequality. so the length of interval $[d(u,w_s), d(v,w_s)]$ is at most $d(u,v)$. Since we choose β_i uniformly from the range $[2^{i-1}, 2^i]$, the probability for β_i to fall into this interval is at most $d(u,v)/2^{i-1}$.

We also need to bound the probability that it is w_s that cut u out of the pair u, v, not others in K_i^u. First we note that the points that are very close to both u, v cannot possibly settle u, v. In fact, w_s must lie outside K_{i-2}^u for $i \geq j^* + 4$. Suppose otherwise, w_s is in K_{i-2}^u, and u is assigned to w_s, then v must be assigned to w_s too, by triangle inequality, $d(v,w_s) \leq d(v,u) + d(u,w_s) \leq 2^{i-2} + 2^{i-2} \leq 2^{i-1} \leq \beta_i$ (note that $i \geq j^* + 4$). Thus only those in $w_{k_{i-2}^u+1}, w_{k_{i-2}^u+2}, \cdots, w_{k_i^u}$ can separate u, v in level i. Since we have a random permutation on the node rank, the probability for w_s to be the first center assigned to u is at most $1/s$. Then the probability that u is cut out of the pair (u,v) at level i is bounded by

$$\sum_{s=k_{i-2}^u+1}^{k_i^u} \frac{1}{s} \cdot \frac{d(u,v)}{2^{i-1}} = \frac{d(u,v)}{2^{i-1}} \cdot (H_{k_i^u} - H_{k_{i-2}^u}),$$

where $H(m)$ is the harmonic function.

For a metric with expansion ratio k, we have $k_i^u \le k \cdot k_{i-1}^u \le k^2 \cdot k_{i-2}^u$. Then

$$H_{k_i^u} - H_{k_{i-2}^u} = \sum_{s=k_{i-2}^u+1}^{k_i^u} \frac{1}{s} < \sum_{s=k_{i-2}^u+1}^{k_i^u} \frac{1}{k_{i-2}^u} = \frac{k_i^u}{k_{i-2}^u} - 1 \le k^2.$$

Thus, we have $\text{Prob}\{D_i \text{ separates } (u,v)\} = d(u,v) \cdot \frac{k^2}{2^{i-2}}$, as required in the theorem. □

Now we are ready to prove Lemma 20.

First, if $\text{LCA}(u,v)$ is at level $i+1$, then at least in one tree the lowest common ancestor is at level $i+1$, the probability of which is 0 if $d(u,v) < 2^{i+2}$, as shown in Lemma 21. In the following we focus on the second case when $2^{i-2} \ge d(u,v)$.

If $\text{LCA}(u,v)$ is at level $i+1$, the first time (smallest level) that u,v belong to different clusters is i in one tree and is $j \ge i$ in another tree. Denote by $P_1(i)$ and $P_2(i)$ the probability that $\text{LCA}_1(u,v)$ and $\text{LCA}_2(u,v)$ are at level $i+1$ respectively.

$$\begin{aligned} & \text{Prob}\{\text{LCA}(u,v) \text{ is at level } i+1\} \\ = \;& P_1(i)\sum_{j=i+1}^{\delta} P_2(j) + P_2(i)\sum_{j=i+1}^{\delta} P_1(j) + P_1(i)P_2(i) \end{aligned}$$

By using Lemma 21. Now we have

$$\begin{aligned} & \text{Prob}\{\text{LCA}(u,v) \text{ is at level } i+1\} \\ \le \;& 2k^2\tfrac{d(u,v)}{2^{i-2}}\sum_{j=i+1}^{\delta}[k^2\tfrac{d(u,v)}{2^{j-2}}] + [k^2\tfrac{d(u,v)}{2^{i-2}}][k^2\tfrac{d(u,v)}{2^{i-2}}] \\ = \;& 3k^4 \cdot d^2(u,v)/2^{2i-4}. \end{aligned}$$

This finishes the proof.

5.3.2 Routing with Two HSTs

First we show how to route using one HST H. Recall that all nodes of P are leaf nodes of the HST H and the internal nodes map to the clusters in the hierarchical decomposition. When we route on the HST, we replace the cluster C by its center $\text{center}(C)$. An edge between a cluster C_1 and its parent cluster C_2, $C_1 \subseteq C_2$ with $C_1 \in D_i$ and $C_2 \in D_{i+1}$, is now realized by their centers $\text{center}(C_1), \text{center}(C_2)$. This edge $\text{center}(C_1), \text{center}(C_2)$ has length $d(\text{center}(C_1), \text{center}(C_2)) \le d(\text{center}(C_1), z) + d(\text{center}(C_2), z) \le 2^i + 2^{i+1} \le 2^{i+2}$, where $z \in C_1 \subseteq C_2$, which is at most twice the edge weight on the HST. To route

between two nodes $x, y \in P$, we take the leaf nodes corresponding to x, y in H and route along the unique path connecting the two nodes in H. The stretch is at most twice the stretch of the HST.

As shown in the previous subsection, if we build two HSTs, using different random seeds, one can obtain constant distortion by always using the shorter path of the two trees. In particular, we use the distributed algorithm as shown in [70] to find 2 HSTs, and obtain node labels for each of the tree. To route a message from a source to a destination node, we check each set of labels to see which tree gives a lower LCA (lowest common ancestor). That tree will provide a path with only constant stretch. We remark that the storage requirement for each node is very low, in the order of $O(\log n)$. In comparison, standard routing table approach build a shortest path tree rooted on each node, and the routing table size is thus $O(n)$. Our method is much more scalable as we benefit from having two global tree structure used by all the nodes.

5.4 Resilience to Node Failures Using Two HSTs

Using a tree metric to route is easy as there is a unique simple path connecting any two nodes and one can find the path easily. But such a routing method is not robust to failures. A link or node failure may disconnect the paths between quadratically many pairs. In this section we show that using two trees, instead of one, can improve the routing robustness substantially. For a pair of node u, v, if the path connecting them is disconnected on the first tree, it is still possible that there is a path between them on the second tree. Thus one can switch to the second tree for a backup route. Thus using two trees one can also recover from sudden, unforseen failures instantaneously, akin to the path splicing idea [112].

5.4.1 Robustness of One HST

We first examine the properties of a single HST in terms of node failure. When a node u fails, any path on the HST that uses a cluster with u as the center is disconnected. We examine how many such pairs there are. the worst case is that u is a center of a cluster near the root of the HST – this will leave big components and

$\Omega(n^2)$ number of pairs disconnected. For example, if the node $\pi(1)$ fails. However, since the construction of the HST uses random permutations (assuming the adversary has no control over the choice of this random permutation, as in standard settings of randomized algorithms), a single node failure is unlikely to be near the root. We show below that a single node failure only 'chops off' a set of nodes of size $O(\log\Delta)$, where Δ is the aspect ratio of the metric, i.e., the longest pairwise distance versus the shortest pairwise distance. Thus there are only $O(n\log\Delta)$ pairs whose paths on the HST are partitioned by a single node failure. Note that this is almost a factor of n off from the worst case. The following theorem works for any metric (P,d) with constant doubling dimension.

Theorem 22. *Given a node u and an HST, the expected number of nodes within clusters with u as center is $O(\log\Delta)$, where Δ is the aspect ratio of the metric (P,d) with constant doubling dimension.*

Proof. Suppose a node x is within a cluster with u as the center, say this cluster is at level i. Then we know that $d(u,x) \leq \beta 2^i$ and u is the highest rank node in $B(x,\beta 2^i)$. Now, take $\ell_u(x)$ as the lowest level j such that $d(u,x) \leq \beta 2^i$. Clearly, $\ell_u(x) \leq i$. Thus $B(x,\beta 2^{\ell_u(x)}) \subseteq B(x,\beta 2^i)$. That is, u is the lowest rank node at level $\ell_u(x)$ as well. The probability for that to happen is $1/|B(x,\beta 2^{\ell_u(x)})|$. Thus the probability that x is inside a cluster with u as center is no greater than $1/|B(x,\beta 2^{\ell_u(x)})|$.

Now, the expected number of nodes within clusters with u as center, denoted as W, is,

$$\begin{aligned} W &= \textstyle\sum_x \text{Prob}\{x \text{ is in a cluster with } u \text{ as the center}\} \\ &\leq \textstyle\sum_x 1/|B(x,\beta 2^{\ell_u(x)})| \\ &= \textstyle\sum_j \sum_{x\in B(u,\beta 2^j)\setminus B(u,\beta 2^{j-1})} 1/|B(x,\beta 2^j)| \\ &\leq \textstyle\sum_j \sum_{x\in B(u,\beta 2^j)} 1/|B(x,\beta 2^j)|. \end{aligned}$$

Now, recall that the metric (P,d) has constant doubling dimension γ. Thus we can cover the point set $B(u,\beta 2^j)$ by balls of radius $\beta 2^{j-1}$, denoted as sets $B_1, B_2, \cdots, B_m$, $m \leq 2^\gamma$. Since the points in B_j are within a ball with radius $\beta 2^{j-1}$, all the points within B_j are within distance $\beta 2^j$ of each other. That is, for a node $y \in B_i$, $B_i \subseteq B(y,\beta 2^j)$. Thus $|B_i| \leq |B(y,\beta 2^j)|$, where $y \in B_i$. Now we group the points of $B(u,\beta 2^j)$ first by

the balls they belong to, and then take the summation over the balls.

$$\begin{aligned} W &\leq \sum_j \sum_{x \in B(u, \beta 2^j)} 1/|B(x, \beta 2^j)| \\ &= \sum_j \sum_i^m \sum_{x \in B_i} 1/|B(x, \beta 2^j)| \\ &\leq \sum_j \sum_i^m \sum_{x \in B_i} 1/|B_i| \\ &= \sum_j \sum_i^m |B_i| \cdot 1/|B_i| \\ &= \sum_j m \leq 2^\gamma \delta = O(\log \Delta). \end{aligned}$$

□ Suppose that u is removed, then the route on the HST between any pair x, y where exactly one of x, y is in a cluster with u as the center, is broken. The above lemma shows that the total number of such pairs effected is bounded by $O(n \log \Delta)$, i.e., $O(\log \Delta / n)$ fraction of all pairs.

5.4.2 Robustness of Two Random HSTs

We now examine the robustness property of using two random HSTs and bound the number of pairs 'disconnected' in both of the trees, i.e., their routes by using both HSTs go through u. For this case we assume that (P, d) has both constant upper and lower bounded growth ratio. By using two trees we reduce the expected number of disconnected pairs from $O(n \log \Delta)$ to $O(n)$.

Theorem 23. *The number of pairs of nodes disconnected in two HSTs, constructed using independent random permutations, is a fraction of $O(1/n)$ of all pairs, for a metric (P, d) with both constant upper and lower bounded growth ratio.*

Proof. Take a pair of nodes x, y, the paths connecting the two in both trees are disconnected if and only if in each of the tree, exactly one node is in a cluster with u as center and another one is not in any cluster with u as center. Denote by $P_u(x)$ the probability that x is in a cluster with u as the center. $P_u(x) \leq 1/|B(x, \beta 2^{\ell_u(x)})|$. Thus the expected number of pairs of nodes disconnected after node u is removed is,

$$\begin{aligned} W_2 &= \sum_y \sum_x 4[P_u(x)]^2 [1 - P_u(y)]^2 \\ &\leq \sum_y \sum_x 4[1/|B(x, \beta 2^{\ell_u(x)})|]^2 \\ &= 4n \sum_j \sum_{x \in B(u, \beta 2^j) \setminus B(u, \beta 2^{j-1})} 1/|B(x, \beta 2^j)|^2 \\ &= 4n \sum_j (|B(u, \beta 2^j)| - |B(u, \beta 2^{j-1})|)/|B(x, \beta 2^j)|^2. \end{aligned}$$

If (P,d) has constant bounded growth ratio k, we know that $\rho^-\beta^k 2^{jk} \leq |B(x,\beta 2^j)| \leq \rho^+\beta^k 2^{jk}$ for constants $\rho^- \leq \rho^+$. Thus

$$\begin{aligned} W_2 &\leq 4n\sum_j [\rho^+\beta^k 2^{jk}]/[\rho^-\beta^k 2^{jk}]^2 \\ &= 4n\sum_j \rho^+/(\rho^-)^2 \cdot 1/(\beta^k 2^{jk}) \\ &= O(n). \end{aligned}$$

□

5.4.3 Robustness of Two HSTs With Reversed Rank

An alternative method to use two trees for robust routing is to construct the second tree to be as different as possible from the first tree. Recall that the HST is purely determined by the random parameters, the permutation π and the random parameter β. Here we first construct an HST H_1 using random permutation π_1, and then choose the second HST H_2 by using π_2, as the reverse of the permutation π_1. As an immediate consequence of that, suppose x is in a cluster with u as the center in H_1, then x can not be inside any cluster with u as center in H_2. This is because the rank of x is greater than u in π_1, and the rank of x must be smaller than the rank of u in π_2. Thus x can never nominate u in H_2. This says that the set of nodes 'chopped off' by the failure of u in H_1 will not be chopped off in H_2, ensuring robustness of routing. The theoretical analysis of this case is fairly complicated but we evaluate the method by simulations and it performs no worse than the two random HSTs.

5.5 Simulations

This section evaluates our two HSTs mechanism in terms of path stretch and reliability against node or link failures. We find that two HSTs provides low stretch path, and achieves high reliability.

5.5.1 Simulation setting

We run our simulation on two data sets. The first data set is a unit disk graph on a network of nodes deployed using perturbed grid model, a widely used model for wireless sensor networks. To be specific, the networks are generated by perturbing

n nodes of the $\sqrt{n} \times \sqrt{n}$ grid in the $[0,1]^2$ unit square, by 2D Gaussian noise of standard deviation $\frac{0.3}{\sqrt{n}}$, and connecting two resulting nodes if they are at most $\frac{2}{\sqrt{n}}$ apart. The average degree of the network generated in this way is about 5. The second data set is the Sprint backbone network topology inferred from Rocketfuel [147], which has 314 nodes and 972 edges.

5.5.2 Simulation methods

We first study the routing stretch by using 1 HST and 2 HSTs respectively assuming no node or link failures. We also examine the number of pairs disconnected when using one HST and two HSTs respectively. For the two HSTs, we carry out simulations for both random HSTs and a pair of reversed rank HSTs.

We then compare the path stretch with the path splicing approach when there are random link or node failures. For path splicing [112], we first create a graph based on random link weight perturbations by setting $L'(i,j) = L(i,j) + f_{ab}(degree(i) + degree(j)) \cdot Random(0, L(i,j))$, where $L(i,j)$ is the original link weight from node i to j, $f_{ab}(degree(i) + degree(j))$ is a linear function in $degree(i) + degree(j)$ ranging from a to b (in our simulation we choose $a = 0, b = 10$.). Then we get one splicing by build the shortest path tree for the perturbed graph for each node in the network. There are n spanning trees in total, rooted at each node. Repeat the above process to build another splicing. For routing with two HSTs, we conducted two sets of experiments using two randomly constructed HSTs, and a pair of HSTs with reversed rank. In both path splicing and routing with two HSTs, we route a message using one HST or one splicing, and switch to another HST or splicing instance when we encounter a link or node failure on the next hop. For both methods, the link or node has a random probability p to fail at any instance. We vary the parameter p to evaluate the robustness of the methods. In this probabilistic setting a message may, in the worst case, wander forever in the network. If both trees fail at some point or the total forwarding hops exceed some limit(We choose $5 \cdot n$ here, where n is the network size.)to avoid infinite loop, then the routing fails and the packet is discarded. The path stretch is computed only on the messages that are not discarded.

5.5.3 Summary of simulation results

Our observations from these experiments are:

- *Small path stretch.* In case of no failures, the path stretch by using two HSTs improves significantly over a single HST. In case of failures, using two HSTs gives worse stretch compared with path splicing, but with the benefit of significantly reducing the routing table size.
- *Extremely good resilience.* We evaluate the resilience of using two HSTs. In the case of one node failure, the average number of disconnected pairs is very small for both one HST and two HSTs, below 5%. The maximum number of disconnected pairs using one HST can be bad, roughly 85% but using two HSTs the number drops to below 10%, thus keeping a lot of pairs still connected. Using a pairs of reversed rank HSTs gives the best resilience as the two trees are nearly 'complementary' to one another. When nodes or links fail and we use 2 HSTs with path splicing, our routing performance, in terms of delivery rate, is nearly the same as that of using $2n$ spanning trees in the original path splicing paper.

5.5.4 Path Stretch

We randomly generate networks with size from 100 to 2025 and compare the path stretch by using a single HST and two HSTs. For the case of two HSTs, we always take the shorter path on the two trees. We draw the stretch as the average path stretch for all pairs in the network. For each network size, we sample 20 different networks and take the average value. Figure 23(a) shows the result. Using both random HTSs and a pair of reversed rank HSTs consistently decreases the path stretch by roughly a factor of 1.3 (from 3.3 to 2.5) on the unit disk graph data set. The path stretch for either a single HST or two HSTs has a very slow or nearly no increase when the network scales.

Then we study the average stretch for Sprint backbone topology from 2 HSTs and the original path splicing when each underlying link fails with probability p, where p changes from 0.01 to 0.1. Figure 23(b) displays the result. Since we are using only two trees for routing and the path splicing uses $2n$ trees altogether, it is

not surprising to see that path splicing gives better stretch than our method. But our method gives a stretch not much worse, roughly within a small constant factor of that, with the benefit of reducing the routing table size significantly by $O(n)$.

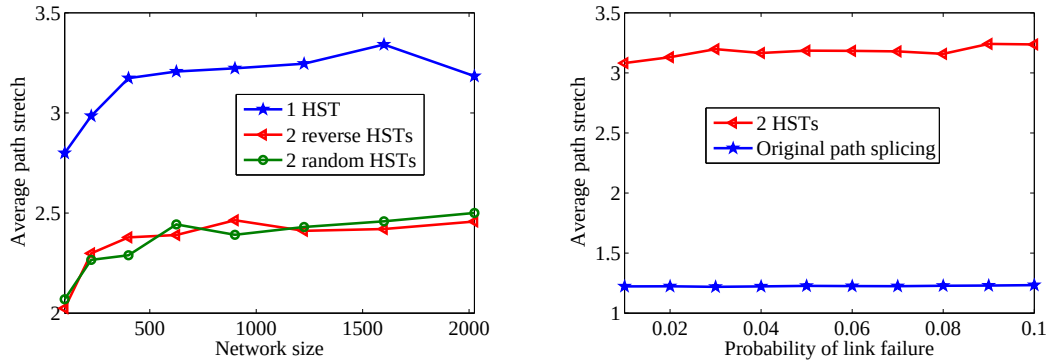


Figure 23. (a) Average path stretch using 2 HSTs V.S. 1 HST for the randomly generated network. (b) Path stretch using 2 HSTs v.s. path splicing on the Sprint topology with link failure.

5.5.5 Robustness to Node or Link Failures

To evaluate the reliability of our two HSTs scheme, we implement our algorithm in terms of node failure and link failure and compare with the path splicing method, described earlier.

5.5.5.1 Perturbed grid network

The random generated network size, n, varies from 100 to 2250. We run n instances, by letting each node fail and removing all its adjacent edges from the network. We then compute the fraction of disconnected pairs. Then we compute the average and the maximum disconnected fraction for the network. We repeat this process 20 times for each network size and take the average value. Figure 24 (a) and (b) shows the result for the average and maximum disconnected pair fraction, respectively. The average fraction of disconnected pairs for either one HST and two HSTs is much smaller than the maximum fraction and is never above 5%. This is easy to understand as in the worst case, removing the lowest rank node in the HST will result in large pairs of nodes disconnected. For the average case, using two

HSTs consistently reduces the disconnected pairs by half. For the maximum case, using two HSTs helps dramatically (from over 85% to below 10%) — when the lowest rank node is removed, it is unlikely to be the lowest rank node in the second tree, thus keeping a lot of pairs still connected. Using a pairs of reversed rank HSTs gives even better performance, reducing the worse case to be nearly below 2%. In fact, except the pairs that involve the failed node, all other pairs are still connected using at least one of the HSTs for single node failure in the perturbed grid network.

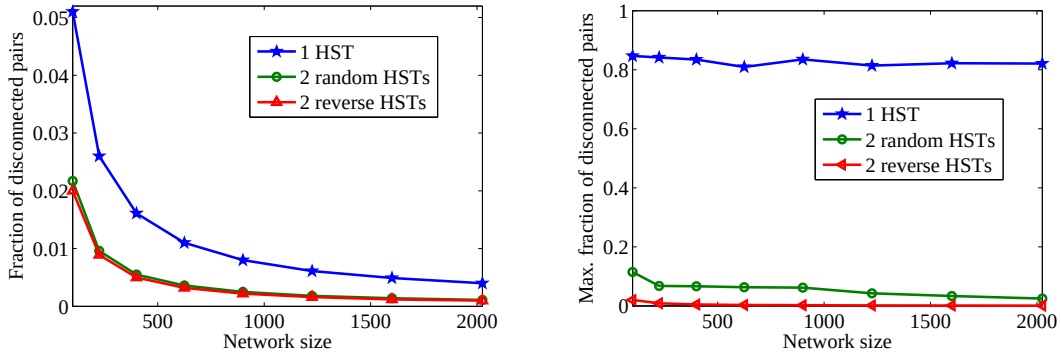


Figure 24. The fraction of disconnected pairs using 1 HST v.s. 2 HSTs. (a) average value. (b) maximum value.

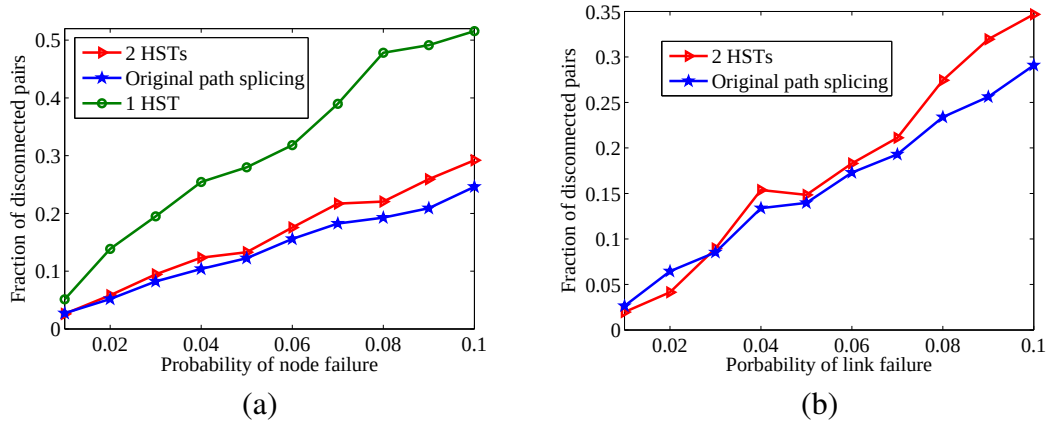


Figure 25. The fraction of messages that are not delivered to the destination on the Sprint network. (a) Random node failure. (b) Random link failure.

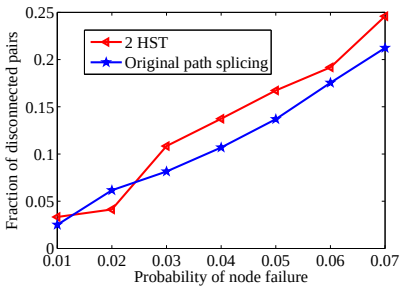


Figure 26. This simulation runs on randomly generated grid networks with 400 nodes. Each node fails with independent probability p. We sample 50 different networks for each value p to get the average fraction of failure pairs. For large p, path splicing achieves better than 2 HSTs. But the difference is mostly within 5%.

5.5.5.2 Sprint topology

In our second simulation, we use the Sprint backbone network topology and let each node in the network fail independently with probability p. We then compare with path splicing and evaluate the fraction of messages that eventually arrive at the destination. Figure 25 (a) shows the result. There is only slight difference between our 2 HSTs mechanism and the original path splicing algorithm, although we use much fewer spanning trees. There is even some advantage from our algorithm when the failure probability p is small. Of course we can see the great improvement from 2 HSTs over 1 single HST.

We conducted the simulations for link failure. To model the link failure, we remove each edge from from the network graph independently with probability p. For every node pair (i, j), we are trying to deliver a packet from i to j by switching between the 2 HSTs or 2 splicing instances. We compute the fraction of failed pairs. The above process is repeated 20 times to get the average fraction of disconnected pairs for each probability p.

Figure 25 (b) shows result for the Sprint backbone network topology. The result is similar to the case of node failure. Our 2 HSTs scheme only has slightly higher failure rate when $p \geq 0.03$ compared with path splicing mechanism.

5.6 Conclusion

This chapter presents a scheme of using two carefully constructed tree metrics for scalable, resilient, and low stretch routing in metrics with geometric growth. Both theoretical analysis and simulation validation demonstrate the clear advantage

of using such special tree metrics. We could exploit more applications of using multiple HSTs.

Chapter 6

Compact Conformal Map for Greedy Routing in Wireless Mobile Sensor Networks

6.1 Introduction

This chapter is motivated by applications of participatory sensing, in which participants such as human beings or vehicles carry sensors and may move around inside a domain of interest. We are particular interested in the case of participants densely populated in a fixed physical space. Examples include tourists in an amusement park, vehicles or pedestrians in a busy downtown area, college students on campuses, visitors at busy beaches or recreational parks/playgrounds. In these examples, the high density of participants ensures a minimum level of coverage of the domain, as well as a minimum level of quality guarantee for sensing data. The sensor data can go beyond traditional automatic measurements and may exploit voluntary user intervention and inputs. Many interesting applications can be formed in such a setting. Of particular interest to us is the spontaneous sensing opportunities due to incidental spatial proximity. For example, a Disneyland tourist may ask: "how long is the line at the Mickey's House?" Those tourists who are in line at Mickey's House happen to have such data, say by sensing the density of people around. A college student may wonder whether the pizza of his favorite flavor is

still available at the university cafe. At this particular moment, the students who are getting pizza at the cafe may conveniently help out by taking a photo. Exploiting the spatial proximity is one of the main ideas in participatory sensing. Such a system is able to provide data that is very selective and personalized, location specific and time-sensitive, the type of data that would be otherwise difficult to get, and not cost efficient to gather or archive on the web.

While it still remains a major open issue as how to give incentives to participants and how to bootstrap the system, our focus in this chapter is one critical component of how to support efficient routing for queries and answers. It remains an arguable issue whether existing communication infrastructures such as WiFi and 3G/4G connections can support the potentially large number of exchanges of these location-specific, fleeting information. The limited coverage of WiFi signals, the high cost and limited bandwidth of 3G links are nevertheless major hurdles to cross. Here we would like to explore the possibilities of relying on unlicensed spectrum with short-ranged wireless communications for delivering queries and data in such a highly mobile but dense network. Bandwidth aside, we would like to see whether it is possible to manage a mobile ad hoc network for reliable, efficient and low-cost routing for this particular scenario.

6.1.1 Prior Work on Routing in Mobile Networks

Routing in a wireless mobile network has been a long standing, active research problem for quite a number of years. While the traditional proposals of dynamic distance vector routing and on-demand routing suffer from high overhead of either maintaining the routing tables or discovering a route to the destination by flooding, geographical routing that relies on local operations based on the geographical positions appears to be an appealing solution. Unfortunately, although geographical routing works nicely in theory [27, 93] for a dynamic, mobile network, it fails in practice, as shown by experiments in real testbeds [95, 139]. Geographical routing has two components, the greedy routing in which a message is delivered to the neighbor closer to the destination, and the recovery method of face routing, which is executed when greedy routing gets stuck and delivers a message along the faces of a planarized graph. In a real world setting, there are a number of complications

that make geographical routing challenging. The node location is not always accurate. Wireless communication model does not follow the idealistic unit disk graph model and has various spatial and temporal radio irregularities [66, 159]. Although greedy routing is very robust to such real world issues [127], face routing fails. In practice, the planar subgraph extracted may become disconnected or still contain crossing edges. Face routing on such a graph may either get into a loop or fail to deliver the message although a path exists.

In the last few years, a number of alternative recovery methods have been proposed. In particular, quite a number of them suggest using virtual coordinates such that greedy routing always works. This completely eliminates the necessity of face routing and of course all the practical issues that come with it. This family of work includes both heuristic algorithms [127], centralized and theoretical constructions for 3-connected planar graphs [4, 13, 49, 53, 87, 120], embedding in high dimensional spaces [62], embeddings in hyperbolic spaces [53, 97, 109], embedding into circular domains (all the holes are circular) [133]. Most of existing work are mainly of theoretical interest. All of these are only for static networks. When nodes move and the network changes its topology, the virtual coordinates need to be recomputed or updated, which is highly non-trivial.

At last we note that since nodes are mobile so theoretically speaking delivery is not an issue — the source can simply hold the message and wait until it gets sufficiently close to the destination. Although being used in a number of theoretical models to study capacity issues in mobile networks [77], this routing scheme is completely impractical. Similarly, by using greedy routing on geographical locations or any virtual coordinates, in the case of a message getting stuck, the node holding the message could simply wait until mobility brings a new neighbor closer to the destination. Indeed message delivery could be ensured at the cost of high delay. In this chapter we investigate compact, lightweight schemes for obtaining virtual coordinates that will substantially help to reduce the delay, or, improve delivery rate when the maximum delay is fixed.

6.1.2 Pre-computed Compact Map for Guaranteed Greedy Routing

In this chapter we consider a domain R in which a dense collection of mobile nodes reside. Each node has a GPS unit or other localization schemes to find out its geographical position. A routing request is formed by specifying the destination's geographical location, rather than the destination's ID – this is because we are interested in finding the nodes near the specified location that have an opportunity to acquire the desirable data, instead of any particular node that may move elsewhere. The nodes may move freely within the domain but as often happens in these application settings the domain R is always nicely covered (with minimum density everywhere).

Our approach is to use essentially greedy routing for its simplicity and robustness to network dynamics, but avoid face routing completely. For that we would have to use virtual coordinates that will guarantee delivery by greedy routing. But different from all the previous designs for routing in a virtual space, we pre-compute the virtual coordinates for all points of the domain R, represent them a compact way by a mapping f, and encode the mapping f with each sensor node. With the help of the hard-coded mapping f, each node p can easily compute its virtual coordinate, by simply taking the value of f on its current GPS coordinates. We also get the virtual coordinates of the destination location as well as that of the neighbors. The next hop is chosen by the greedy rule in the virtual coordinate space.

We are motivated by the work by Sarkar *et al.* [133]. It uses Ricci flow to compute a map, that takes any triangulated domain with holes to a circular domain where all the holes are disks. Greedy routing inside a circular domain guarantees delivery. The computation of the mapping in [133] is carried in a distributed manner on the sensor network such that each node computes its own virtual coordinates when the algorithm converges. However, if the nodes move around, a node would need to get a new set of virtual coordinates. This means we need to run Ricci flow again to re-converge to the new coordinates. Besides, the destination's virtual coordinates can also change and we would need to use a location management scheme to update and maintain the current location information.

What is different in this chapter is that we directly compute the map of the

geometric domain R to a circular domain D. The shape of the domain R can be obtained through external sources such as a map and can be represented by a polygon. We will compute the mapping offline (on a centralized machine) and then use a small number of parameters to compactly represent the map. The number of parameters is the same as the complexity of describing the geometric domain R (e.g., the number of vertices of the polygon describing R), and is typically a small number in practice. There are numerous work on the simplification and approximation of a polygon [47]. By allowing reasonable approximation one can substantially reduce the complexity of a polygon. Thus the parameters for the conformal map can be preloaded and programmed on the sensor nodes before deployment and each node can by itself calculate the virtual coordinates with its current geographical location. This representation is compact and stable. It does not depend on the network topology and does not change when the nodes move around. Besides the application in the mobile setting, the new method is also the first practical solution for applying virtual coordinates in a static sensor network.

In the following we first review the basic background knowledge on discrete Ricci flow. Then we report the theory and algorithms for computing and encoding the compact conformal map. Finally we report results in our simulation and experiments. We implemented and tested the new method on a real testbed (the Orbit testbed [3]), a wireless network emulator with 400 nodes on a 20 by 20 grid. We compared routing using geographical locations and virtual coordinates, in both static and mobile network settings. Experimental results confirms that using the compact conformal map one can substantially improve the delivery rate and largely reduce the packet delay.

6.2 Discrete Ricci Flow

6.2.1 Ricci Flow Theory

Let S be a smooth surface with a Riemannian metric $\mathbf{g}$, $u : S \to \mathbb{R}$ be a smooth function defined on S, then

$$\tilde{\mathbf{g}} := e^{2u}\mathbf{g},$$

is also a Riemannian metric on S. We call $\tilde{\mathbf{g}}$ is *conformal* to $\mathbf{g}$ due to the fact that the angles among tangent vectors measured by $\tilde{\mathbf{g}}$ are equal to those measured by $\mathbf{g}$. Suppose $\mathbf{g}$ and $\tilde{\mathbf{g}}$ induce Gaussian curvature functions K and $\tilde{K}$ respectively, then the curvatures satisfy the following Yamabe equation

$$\tilde{K} = e^{-2u}(K - \Delta_{\mathbf{g}} u),$$

where $\Delta_{\mathbf{g}}$ is the Laplace-Beltrami operator induced by the original metric $\mathbf{g}$. For geodesic curvature along the boundary,

$$\tilde{k_g} = e^{-u}(k_g - \partial_n u),$$

where k_g represents the geodesic curvature. Yamabe equations can be solved by Hamilton's Ricci flow

$$\frac{dg_{ij}}{dt} = 2(\tilde{K} - K)g_{ij},$$

or equivalently

$$\frac{du}{dt} = 2(\tilde{K} - K). \tag{3}$$

On a continuous surface Ricci flow will converge to the solution exponentially fast, see [84] for the proof for high genus surfaces and [37] for genus zero surfaces.

6.2.2 Discrete Ricci Flow

In the discrete setting, one can define Ricci flow for a triangulated discrete surface. Suppose such a triangle mesh Σ has with vertex set V, edge set E and face set F.

We can define a Riemannian metric in the discrete setting by using the edge lengths on a mesh Σ: $l : E \to \mathbb{R}^+$, such that for a triangle face f_{ijk} with vertices v_i, v_j, v_k, the lengths of the three edges satisfy the triangle inequality: $l_{ij} + l_{jk} > l_{ki}$.

The Riemannian metric determines the corner angles of the triangles. Suppose we have a triangle f_{ijk} with edge lengths $\{l_{ij}, l_{jk}, l_{ki}\}$, and the angles against the corresponding edges are $\{\theta_k, \theta_i, \theta_j\}$ (see Figure 27). By the cosine law,

$$l_{ij}^2 = l_{jk}^2 + l_{ki}^2 - 2l_{jk}l_{ki}\cos\theta_k, \tag{4}$$

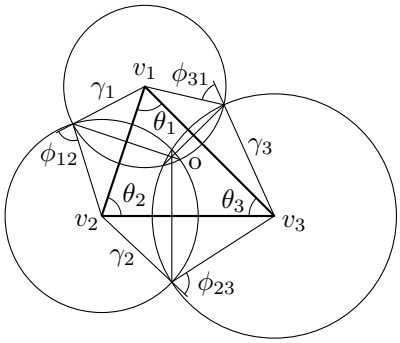


Figure 27. The circle packing metric.

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

$$K_i = \begin{cases} 2\pi - \sum_{f_{ijk} \in F} \theta_i^{jk}, & v_i \text{ is an interior vertex} \\ \pi - \sum_{f_{ijk} \in F} \theta_i^{jk}, & v_i \text{ is at boundary} \end{cases} \tag{5}$$

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

Recall that Ricci flow defines a conformal map, i.e., angle preserving. We now need to introduce the circle packing metric, proposed by [148, 153], to approximate the conformal deformation of metrics. Let us denote by Γ a function which assigns a radius γ_i to each vertex v_i, $\Gamma : V \to \mathbb{R}^+$. We also define a *weight* function $\Phi : E \to [0, \frac{\pi}{2}]$, by assigning a positive number $\Phi(e_{ij})$ to each edge e_{ij}. The pair of vertex radii and edge weight functions on a mesh Σ, (Γ, Φ), is called a *circle packing metric* of Σ. Figure 27 illustrates the circle packing metric. Each vertex v_i has a circle whose radius is γ_i. On each edge e_{ij}, an intersection angle ϕ_{ij} is defined by two circles of v_i and v_j, which intersect with or are tangent to each other. Two circle packing metrics (Γ_1, Φ_1) and (Γ_2, Φ_2) on the same mesh are *conformal equivalent*, if $\Phi_1 \equiv \Phi_2$ (i.e., the intersection angle of the neighboring circles is the same). Therefore, a conformal deformation of a circle packing metric only modifies the vertex radii γ_i's.

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

$$l_{ij}^2 = \gamma_i^2 + \gamma_j^2 + 2\gamma_i \gamma_j \cos \phi_{ij} \tag{6}$$

Let u_i to be $\log \gamma_i$ for each vertex. Let K_i' be the target curvature at vertex i. Then, the discrete Ricci flow is defined as follows.

$$\frac{du_i(t)}{dt} = (K_i' - K_i) \tag{7}$$

Computing desired metric with prescribed curvature K' is equivalent to minimizing the discrete Ricci energy. The discrete Ricci energy is strictly convex (namely, its Hessian is positive definite). The global minimum uniquely exists, corresponding to the metric that induces K'. The discrete Ricci flow converges to this global minimum [38].

The convergence rate of the discrete Ricci flow using Equation 7 is governed by the following theorem

Theorem 24 (Chow & Luo). *The Ricci flow (as in Equation 7) converges exponentially fast,*

$$|K_i' - K_i(t)| < c_1 e^{-c_2 t}, \tag{8}$$

where c_1, c_2 are two positive constants.

6.2.3 Discrete Algorithm

In our application we will first triangulate the interior of the domain R to be a triangular mesh Σ (irrelevant to the network topology). We will deform the shape to a circular domain Ω using an offline, centralized algorithm. We first define the edge length of the mesh Σ to be initially 1 everywhere. Then we define the circle packing metric by placing a disk of radius $1/2$ on all vertices. Thus disks at adjacent vertices are tangent to each other, i.e., intersection angle is zero.

To specify the target domain Ω, all the interior vertices have target curvature 0. For vertices on an inner boundary γ_i, suppose there are k vertices on γ_i, then each vertex has a target curvature as $-2\pi/k$. For vertices on an outer boundary, each vertex has target curvature $2\pi/m$, if there are m nodes on the outer boundary.

To run the discrete Ricci flow, each node v_i is associated with a disk, with radius e^{u_i}, where u_i is a scalar value. For simplicity, the length of each edge connecting v_i and v_j equals to $e^{u_i} + e^{u_j}$. That is, the two disks at v_i, v_j are tangent to each other. The corner angles of each triangle can be estimated using cosine law by

each node locally. The curvature can be computed by each node directly. Then u_i is modified proportionally to the difference between the target curvature and the current curvature. Once the curvature error is less than a given threshold, the process stops. The final edge length for edge ij is $e^{u_i} + e^{u_j}$.

Once the lengths for all edges in the triangular mesh are computed, we can flatten the triangular mesh in the plane to obtain the embedding. We can start from placing an arbitrary triangle with the specified edge length in the plane, and gradually attach the adjacent triangles. Since all interior vertices have curvature zero the triangles will not have any overlaps.

6.3 Background of Conformal Mapping

Given a domain R with an irregular shape and possibly holes, we would like to map it to a circular domain Ω, in which all boundaries are circular. Now we briefly go through the mathematical theory that describes this mapping.

To deform the domain, we are changing two measures, the *metric* that defines distances and the length of a curve, and the *curvature*, which describes how much a geometric object deviates from being flat in the case of a surface, or straight in the case of a curve. A point in the interior of a flat surface has curvature zero; a point on a surface with positive curvature locally has the shape of a hill/valley; a point on a surface with negative curvature locally has the shape of a saddle. Regarding the curvature of a point on the boundary of a surface, the curvature is zero if and only if the boundary locally at the point is straight. Points on a circular boundary of a planar domain have uniform curvature.

In our particular case we need to deform an irregular shaped planar domain R to a circular planar domain Ω. Since both R and Ω are flat, all points in the interior have zero curvature in both cases. But we need to change the curvature on all boundary points in R to be uniform (and thus being circular in Ω). Consider the simple case of deforming a simple polygon to a disk, as shown in Figure 28. For the boundary of a polygon, the points in the interior of boundary edges have curvature zero; and the vertices have non-zero curvature, defined as the *turning angles* at this vertex between the two adjacent edges. In particular, the curvature at vertex w_1 is $\beta_1\pi$. To make the curvatures at all these vertices the same (thus the polygon being

regular) we need to change the metric as well, i.e., stretching the polygon in certain ways.

The tool to compute the deformation is Ricci flow. In particular, Ricci flow deforms the Riemannian metric proportional to the local curvature, such that the curvature evolves according to the heat diffusion process. Eventually, curvature at all points converge and the surface has uniform curvature. In order to deform a surface to be a particular shape, we can specify the target curvature and define Ricci flow by deforming the metric according to the *difference* of the local curvature and the target curvature. When Ricci flow converges we get the shape with the target curvature. In our case we will specify Ω such that all points in the interior have zero curvature and points on the same (interior/outer) boundary have uniform (negative/positive) curvature. Ricci flow has been applied in the proof of Poincaré conjecture by Perelman in [124]. In the current work, we apply surface Ricci flow theory developed by Hamilton [84] and Chow [37]. The earlier work by Sarkar *et al.* [133] developed the distributed algorithm for implementing Ricci flow in a static sensor network.

The process of Ricci flow defines a map from the original surface R to the deformed surface Ω. This map is conformal. A conformal map between two surfaces preserves angles. For any two arbitrary curves γ_1, γ_2 on the surface S, a conformal map ϕ maps them to $\phi(\gamma_1)$, $\phi(\gamma_2)$ with the same intersection angle as that of γ_1, γ_2 on S.

In our application we assume that a set of wireless sensor nodes densely cover the domain R, which is represented by a polygon with possibly inner holes. We assume that the conformal mapping from the polygon R to a circular domain has been computed offline, say, by using Ricci flow methods. The details of the offline computation are presented in Section 6.2 and are essentially the same as in previous work [133]. In this chapter we show that the the conformal mapping can be *encoded* by a small number of parameters — the number of these parameters is the same as the complexity of describing R and in practice is small. These parameters are preloaded to all the sensors in the network such that each sensor can compute the virtual coordinates, i.e., coordinates under the mapping. The two sections below describe the encoding and the algorithm to compute the virtual coordinates with the encoded map respectively.

6.4 Schwarz-Christoffel Transformation Using Laurent Series

To encode a conformal map, we introduce the Schwarz-Christoffel formula and the Laurent series [48,50]. In the following we describe two methods for representing the conformal mapping for simply connected domains (i.e., domains without holes) and multiply connected domains (i.e., domains with holes) respectively.

6.4.1 Simply Connected Domain

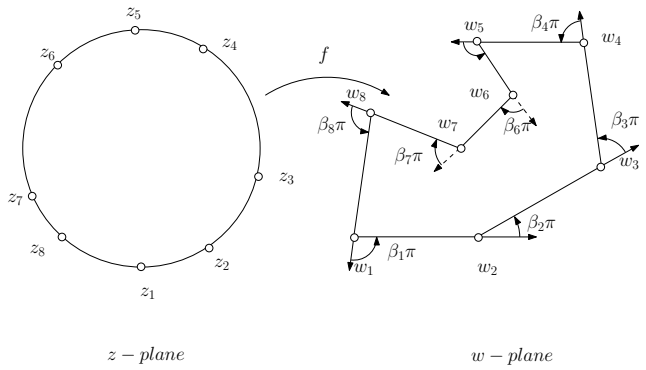


Figure 28. The Schwarz-Christoffel transformation for a simply connected domain. $f(z_i) = w_i$.

In the case of sensors in a simply connected domain R, as shown in Figure 28, we would like to find a mapping to a unit disk $\mathbb{D}$. A holomorphic function is a complex-valued function of one or more complex variables that is complex differentiable in a neighborhood of every point in its domain. We first look at the conformal mapping from the unit disk to a polygon R, a holomorphic function, $f : \mathbb{D} \to R$. The mapping we would like to encode is the inverse of f, $f^{-1} : R \to \mathbb{D}$.

Suppose the vertices of the polygon are $\{w_1, w_2, \cdots, w_n\}$. The Schwarz-Christoffel formula [50] describes the mapping of a disk onto the interior of a simple polygon.

$$f(z) = A \int^{z} \prod_{k=1}^{n} (\zeta - z_k)^{-\beta_k} d\zeta + B, \tag{9}$$

where the pre-image of w_k on the circle is z_k, i.e., $w_k = f(z_k)$; the polygon tangent turning angle at w_k is $\beta_k\pi$; A and B are two constants.

In our case, we need to compute the inverse $f^{-1} : R \to \mathbb{D}$. For convenience, we denote $g(w) = f^{-1}(w)$. Suppose $w_0 = f(0)$. For each $w \in P$, we choose a path connecting w_0 and w, then

$$g(w) = f^{-1}(w) = \int_{w_0}^{w} \frac{dg(w)}{dw} dw + 0.$$

where the derivative is given by

$$\frac{dg(w)}{dw} = \left(\frac{df(z)}{dz}\right)^{-1} = \prod_{k=1}^{n} (z - z_k)^{\beta_k} = \prod_{k=1}^{n} (g(w) - z_k)^{\beta_k}.$$

Therefore we can calculate $g(w)$ as:

$$g(w) = \int_{w_0}^{w} \prod_{k=1}^{n} (g(\tau) - z_k)^{\beta_k} d\tau + 0. \tag{10}$$

This is a typical ODE (ordinary differential equation) problem, and can be easily solved by Runge-Kutta method [30]. Therefore, each sensor only requires the parameters $\{z_k, \beta_k\}_{k=1}^{n}$ and the constants A, B. With such information every sensor can solve the ODE and obtains its own location. The number of parameters stored on each sensors is the size of the complexity of the description of the sensor domain R. If we approximate R with simpler polygon we can reduce the storage requirement accordingly.

The evaluation of the conformal map is equivalent to do a path integration, the shorter the path is, the faster the evaluation is. Therefore in the mobile network setting we can update the virtual coordinates of a mobile node based on its current location, the past location and the past virtual coordinates. This will be faster. Suppose at the k-th step, a node's position is w_k, its virtual coordinates is z_k. Then at the $k+1$-th step, if the position is w_{k+1}, the virtual coordinate z_{k+1} can be updated as

$$z_{k+1} = \int_{\gamma} f'(\zeta) d\zeta,$$

where γ is the path connecting w_k to w_{k+1}. Since w_{k+1} is often close to w_k, therefore, we can update the node location much faster than the direct computation.

6.4.2 Multiply Connected Domain

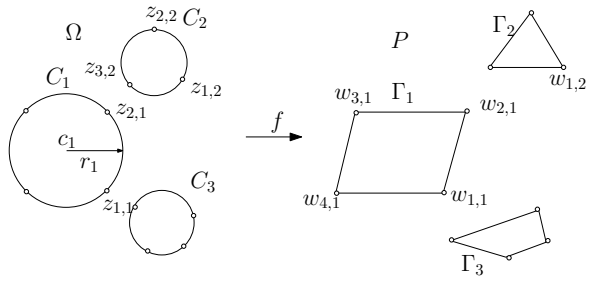


Figure 29. The Schwarz-Christoffel transformation for a multiply connected domain. $f(z_{i,j}) = w_{i,j}$.

The formulation for conformal mappings for multiply connected domains is very similar. Recall that by using Ricci flow on the polygon R (with $m-1$ holes) describing the domain of interest, we map it to a circular domain Ω with $m-1$ circular interior holes. Now we apply a reflection on Ω to make it an unbounded domain with m circular holes. A reflection through a circle maps the interior of the circle to the exterior. Suppose p is a point inside the circle $C_j(c_j, r_j)$, then it is reflected to q, where

$$q = \frac{p - c_j}{|p - c_j|^2} r_j^2 + c_j$$

In particular, we can reflect the circular domain Ω with respect to the outer boundary and project it to an unbounded domain with m circular holes. This extra reflection is also conformal. For the ease of description, we will assume from now on that Ω is an unbounded domain Ω with m circular holes.

Similar to the simply connected case, there is a description of a conformal mapping f from an unbounded domain Ω with m circular holes to an unbounded domain R with m polygonal holes. Again what we want to use is the inverse of f, say $g = f^{-1}$.

As shown in Figure 29, let C_j be the j-th circular hole, with center c_j and radius r_j, which is mapped to a polygonal hole Γ_j, $f(C_j) = \Gamma_j$. The vertices of Γ_j are $\{w_{1,j}, w_{2,j}, \cdots, w_{k_j,j}\} \subset \Gamma_j$. Their pre-images are $\{z_{1,j}, z_{2,j}, \cdots, z_{k_j,j}\} \subset C_j$.

On the circular domain Ω we will apply reflections with respect to the circles. See Figure 30. A circle C_j is reflected through circle C_i to get a circle C_{ij}. A circle

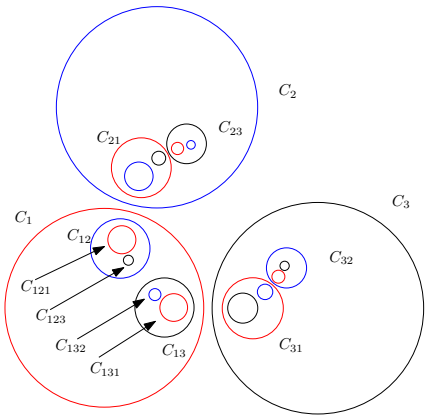


Figure 30. Circle reflection.

C_k is reflected through the circle C_{ij} to get a circle C_{ijk}. More general, a circle C_{i_n} is reflected through the circle $C_{i_1 i_2,\cdots i_{n-1}}$ to get $C_{i_1 i_2,\cdots,i_n}$. The multi-index $i_1 i_2,\cdots,i_n$ tracks the sequence of reflections. The set of all multi-indices with length n is denoted as σ_n.

By the Schwarz-Christoffel formula [48], the conformal mapping $f:\Omega\to R$ has the formula, called the Laurent series:

$$f(z)=A\int^z\prod_{j=1}^{m}\prod_{k=1}^{K_j}\left[\prod_{\substack{n=0\\ \nu\in\sigma_n}}^{\infty}\left(\frac{\zeta-z_{k,\nu j}}{\zeta-s_{\nu j}}\right)\right]^{\beta_{k,j}}d\zeta+B. \tag{11}$$

where $z_{k,\nu j}$ are reflections through circles of prevertices $z_{k,j}$; $s_{\nu j}$ are reflections of circle centers $s_j=c_j$; ν is a multi-index tracking reflections; A and B are two constants.

Similar to the simply connected domain case, the inverse of f can be evaluated by solving an ODE. In practice, instead of using infinitely many reflections, one can reflect only N times, for a reasonable N. Therefore, the Laurent series could be

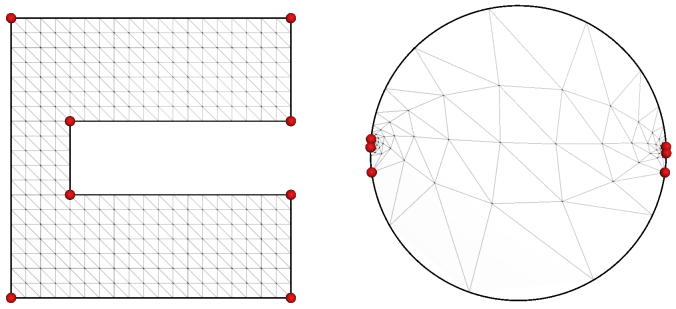
approximated by:

$$f(z) = A \int^{z} \prod_{j=1}^{m} \prod_{k=1}^{K_j} \left[\prod_{\substack{n=0 \\ \nu \in \sigma_n}}^{N} \left(\frac{\zeta - z_{k,\nu j}}{\zeta - s_{\nu j}} \right) \right]^{\beta_{k,j}} d\zeta + B. \tag{12}$$

The prevertices $\{z_{k,j}\}$, the circle domain $C_j(c_j, r_j)$ are computed using Ricci flow method offline (explained in the appendix). Then these values are preloaded to all the sensors with the tangent turning angle $\{\beta_{k,j}\}$ and the constants A, B. Each sensor can be computed through the conformal mapping $f^{-1} : P \to \Omega$ by solving an ODE. $f(z)$ can be approximated by Equation 12, where in practice N is less than 5.

6.5 Examples

In this section, we elaborate the computation of the virtual coordinates using specific examples. These two examples are what we used in the following experiments.



(a) vertices on $w-plane$ (b) prevertices on $z-plane$

Figure 31. Prevertices of a simply connected polygon vertices used for evaluating Schwartz-Christoffel transformation.

w	β	z
1 + 1i	0.5	-0.98993 + 0.14155i
20 + 1i	0.5	-0.99478 + 0.10202i
20 + 8i	0.5	-0.99480 + 0.10184i
5 + 8i	-0.5	-0.99978 - 0.02078i
5 + 13i	-0.5	0.97883 - 0.20466i
20 + 13i	0.5	0.99872 - 0.05063i
20 + 20i	0.5	0.99873 - 0.05041i
1 + 20i	0.5	1.00000 + 0.00000i
A	-0.16623648 + 2.5343952i	
B	3.15 + 10.65i	

Note: w - polygon domain, z - disk domain, $\beta\pi$ - tangential turning angle, A - constant scalar in transformation, B - conformal center on w.

Table 1. Parameters of Schwarz-Christoffel transformation.

6.5.1 Simply Connected Domain

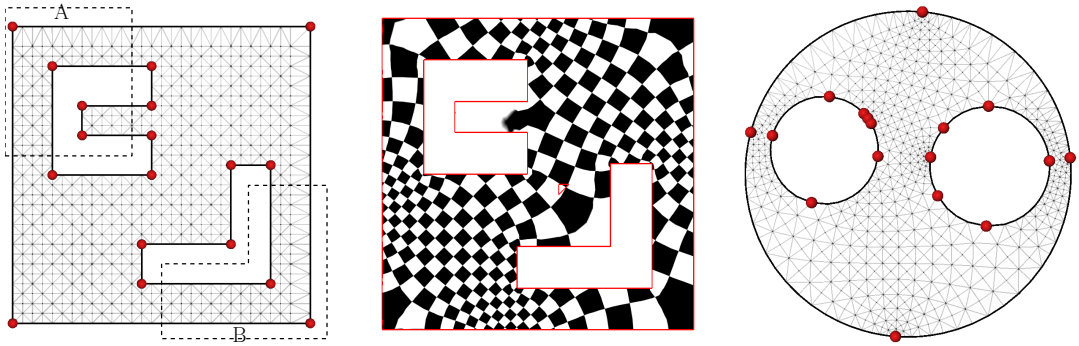
Suppose the sensors move within a simply connected domain, i.e., with one exterior polygonal boundary, as shown in Figure 31(a). The images of the polygon vertices (prevertices) under the conformal mapping onto the unit circle has been computed already using Ricci flow method, as shown in Figure 31(c). Alternatively, the prevertices can be computed using the method based on complex analysis [50].

Table 1 illustrates the coefficients for the Schwarz-Christoffel transformation, where w are the coordinates of the polygon vertices, z are the prevertices on the unit disk domain, A is a constant scalar, B is the conformal center which is mapped to the origin of the unit disk, β is the tangential turning angle of the input polygon boundary. The prevertex for the interior sensors can be computed by Equation 10 using the obtained coefficients above.

6.5.2 Multiply Connected Domain

Suppose the mobile sensors move within a multiply connected domain, e.g., with inner holes, as shown in Figure 32(a). We first use Ricci flow to conformally map the domain to a circle domain, where exterior boundary is mapped to a unit disk, the inner holes are mapped to circular hoes, as shown in Figure 32(b). Through

this initial mapping, we obtain the coefficients in Equation 12, as shown in Table 2. The prevertex for the interior sensors can be evaluated by solving an ODE to compute the inverse function of Equation 12 with the obtained coefficients.



(a) input domain on w-plane (b) checker-board texture mapping (c) conformal mapping on z-plane

Figure 32. Ricci flow for conformal mapping a multiply connected polygon.

6.6 Experimental Results

In this section, we compare the proposed virtual-coordinate based greedy routing with the traditional greedy routing based on the original geographical locations in both static and mobile network settings. We did not implement any recovery scheme (e.g., face routing [93]) when greedy routing (based on whatever coordinates) fails at a local minimum. The reasons for such consideration are the following: first, such recovery schemes are more complicated and are usually developed for an idealistic setting; second, for mobile networks it makes less sense to try to recover from local minimum as nodes move around and a packet may only get stuck temporarily; third, in our applications for participatory sensing and opportunistic query, it is not so critical that every message must be delivered – a scheme with sufficiently high success ratio that emphasizes on efficiency and low overhead will be more desirable.

The experiments are performed by both computer simulations and real testbed emulations. The computer simulation is based on a computer generated network

w	β	z
0	0.5	-0.0769204 -0.997173i
30 + 1i	0.5	0.994966 + 0.100535i
30 + 30i	0.5	0.0887658 + 0.995941i
30i	0.5	-0.966776 + 0.256636i
4 + 15i	-0.5	-0.592622 - 0.175069i
4 + 26i	-0.5	-0.832789 + 0.236657i
14 + 26i	-0.5	-0.482482 + 0.47661i
14 + 22i	-0.5	-0.270788 + 0.371865i
7 + 22i	0.5	-0.246845 + 0.342856i
7 + 19i	0.5	-0.246771 + 0.342755i
14 + 19i	-0.5	-0.226475 + 0.311561i
14 + 15i	-0.5	-0.184719 + 0.109243i
13 + 4i	-0.5	0.479915 - 0.319252i
13 + 8i	-0.5	0.182277 - 0.133036i
22 + 8i	0.5	0.138806 + 0.104056i
22 + 16i	-0.5	0.218467 + 0.283291i
26 + 16i	-0.5	0.494831 + 0.417102i
26 + 4i	-0.5	0.867876 + 0.0811398i

(c_1, r_1)	(-0.507853 + 0.159686 i, 0.327225)
(c_2, r_2)	(0.513089 + 0.054973 i, 0.358639)
A	-0.21564355 + 2.27355436i
B	16.45 + 11.65i

Note: w - polygon domain, z - circle domain, $\beta\pi$ - tangential turning angle, (c_k, r_k) - center and radius of hole k on circle domain, A - constant scalar in transformation, B - conformal center on w.

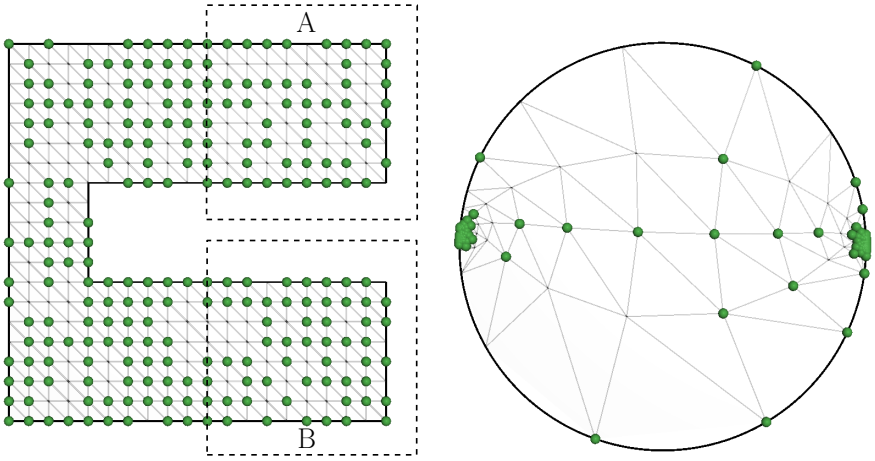
Table 2. Parameters of Laurent series.

with 1000 nodes. The emulation is deployed on Orbit, an open accessed testbed available at WINLAB, Rutgers University [3]. The performance comparisons are based on evaluating the delivery rate, the number of hops and the end-to-end delay. To make it clear, hops, delay and the delivery rate in this experiment are defined as follows.

- **Delay:** Specify how long it takes to deliver a packet from source to destination.
- **Hops:** Represent the number of nodes the packets visited from source to destination.
- **Delivery rate:** Measure the fraction of successfully delivered source and destination packets.

6.6.1 Simulation Results

In this experiment, we implement the computer based simulations for both static and mobile networks.



(a) active nodes on orbit (b) virtual coordinates

Figure 33. Active nodes on orbit.

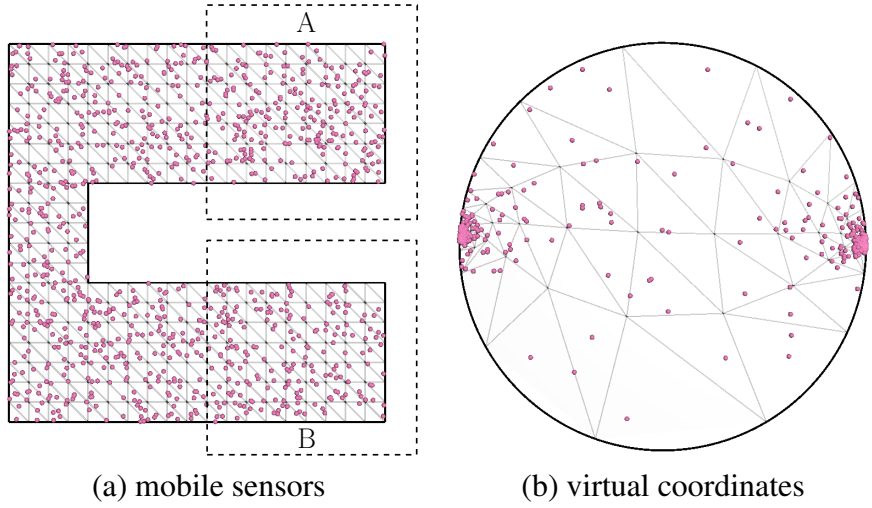


Figure 34. Mobile sensors at one snapshot.

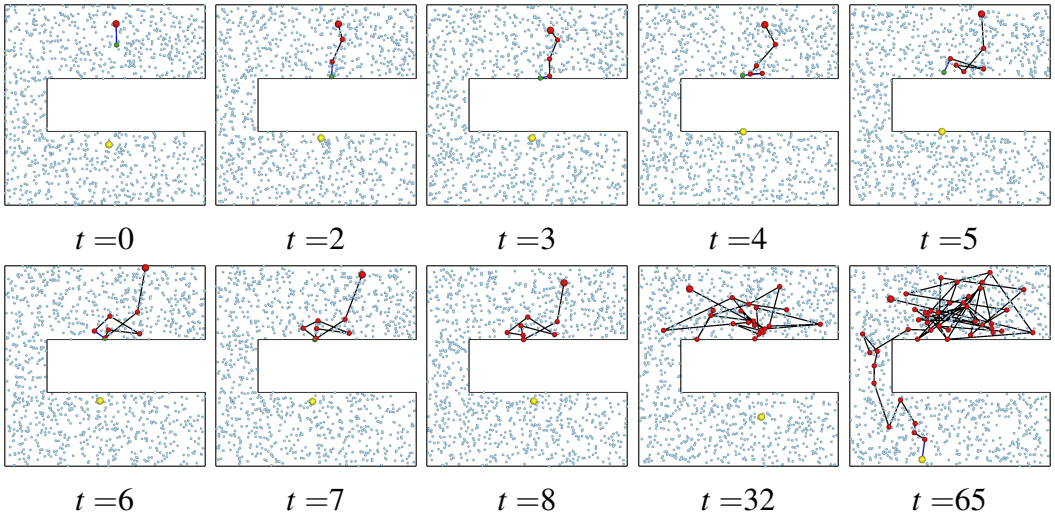


Figure 35. Greedy routing for mobile sensor network using real coordinates. Red node with larger size is the starting sensor, yellow node is the destination. The green node on each frame shows the routing sensor at time t. Greedy routing will get stuck at time 8 and 32.

6.6.1.1 The Experimental Setup

The computer simulation is based on the computer generated network of 1000 nodes uniformly distributed inside a polygon with the boundary of [1,1], [20,1],

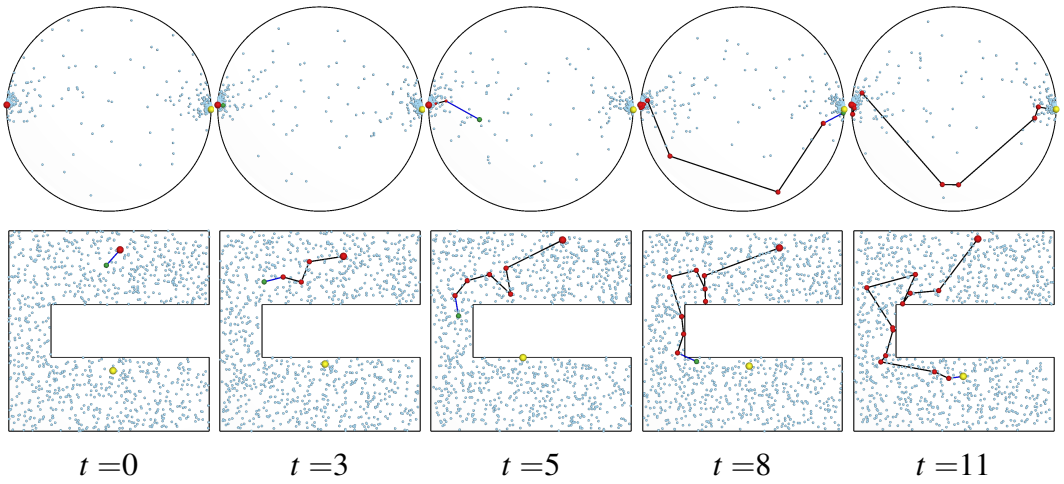


Figure 36. Greedy routing for mobile sensor network using virtual coordinates. 1st row: routing paths on real network; 2nd row: routing paths on virtual domain. Red node with larger size is the starting sensor, yellow node is the destination. The green node on each frame shows the routing sensor at time t. Greedy routing on virtual domain won't get stuck.

[20,8], [5,8], [5,13], [20,13], [20,20], [1,20], as showed in Figure 34 (a). The communication range of nodes follows the Unit Disk Graph (UDG) model, that is, if two nodes are within distance of one unit, then they can communicate with each; otherwise, they can not communicate with each other. In our experiments, we set the unit distance as 2. Under this setting, the average degree of each node in the resulted communication graph is 7 (or 7 neighbors for each node on average). The corresponding conformal mapping result of the network under our algorithm is shown in Figure 34 (b).

In our simulations, the MAC layer is assumed to be ideal – there is no transmission failure. Experiments that consider such failures are presented in our testbed implementation later.

All the results here are based on 400 pairs of delivery attempts. The 400 pairs of source and destination are chosen in two different ways. 1) Uniformly and randomly selected from the entire network. 2) Randomly chosen from the special areas A and B in Figure 34 (a) respectively, under which greedy routing based on geographical locations would have difficulty in delivering the packets, particularly in static setting. The 400 randomly chosen pairs are the same for the two algorithms

Uniformly and randomly selected pairs		
Methods	delivery rate	hops
Geographical locations	275/400	5.6
Virtual coordinates	400/400	8.52
Specially chosen pairs		
Methods	delivery rate	hops
Geographical locations	0/400	0
Virtual coordinates	398/400	17.23

Table 3. Performance comparison for computer simulation under static setting.

for fairness.

6.6.1.2 Static setting

In this setting nodes are stationary. We assume that the TTL of packets is high enough such that packets can be delivered as long as there are valid routes. Thus, the only reason a packet could not be delivered is due to the problem of local minimum. For our algorithm, virtual coordinates from the mapping are used to select the neighbor closest to the destination. Note that the network topology is the same for both algorithms. With the above network setting, we show the simulation results in Table 3.

We can see from the results that the delivery rate of our method is much better than greedy routing with geographical location, especially for the pairs in the specially chosen areas, for which greedy routing using geographical locations always gets stuck. For our method, among the 400 pairs, only two of them cannot be delivered. To understand this, recall that our conformal map is done on deforming the *continuous* geometric domain while greedy routing is performed in the *discrete* network. This is different from the previous work [133] in which the conformal map is computed directly on the network topology and thus delivery is guaranteed. Instead, we may encounter in rare cases a delivery failure due to local minimum. But the strength of our method is due to the extremely light weight implementation that would make the family of Ricci flow based methods possible for real world applications.

The number of hops listed in the table is the average value of hops for successfully delivered message. The hops for our methods is higher than that of greedy

routing with geographical locations, simply because our method can successfully deliver more messages, including the far apart pairs in the special area where greedy routing could not.

6.6.1.3 Mobile setting

To evaluate our algorithm for mobile setting, we adopt the Random Waypont (RWP) mobility model [29]. RWP is a commonly used synthetic model for mobility. In this model, each node i moves from its current waypoint p_i to the next one p_i'. The destination, speed and direction are all chosen randomly and independently of other nodes. The next waypoint for each node i is uniformly distributed over the disk with radius r centered at its current waypoint. And we randomly set the velocity moving from p_i to p_i' to be random number v_i drawn from the uniform distribution $[a,b]$. Whenever node i arriving its next waypoint p_i', it will stay there for a random time duration t_i, which is drawn from the uniform distribution $[0,T]$. After time t_i, node i continues its random waypoint walk following previous scheme. Basically this is a simple model that describes the movement pattern of mobile users, and how their location and velocity change over time. During the movement, we sample one snapshot with location information for all the nodes after time T_j. T_j follows the uniform distribution $[c,d]$. That means sample the time axis per T_j to generate different snapshots. But we generate new value for T_j after every snapshot. We can think this T_j as the retrying time for a packet to be delivered when it was got stuck in the last configuration.

Specifically, in the experiment, we set the moving disk radius r for all nodes to be 2. The moving speed v_i for each node i is drawn from $[0.5,2.5]$ uniformly and independently. When node i comes to stay stage, stay duration t_i is uniformly drawn from $[0,2]$. Finally, the sample time duration is uniformly distributed in the range $[1,3]$.

At each time slot/snapshot, the node holding a packet examines its neighbors to see if anyone is closer to the destination. If so, the message is delivered to the corresponding neighbor. If not, the packet remains at the current node for another try in the next time slot. Notice that in the next snapshot, the node with the packet moves to a different location and may have a new set of neighbors. Here is an example of the scenario: at time T_i, node n_p sends the packet to node n_q. For the

next time slot T_{i+1}, n_q moves to a new location and will calculate the next node to deliver based on this new location and all the neighbors' locations at time T_{i+1}. Since we assume that user motion can be arbitrary/unpredictable, a neighbor that is currently closer to the destination may actually move away. Thus even greedy routing based on a perfect set of virtual coordinates may not always deliver all packets successfully.

We use TTL to control how long the packets could wander in the network. So TTL equals the maximum number of snapshots that a packet can remain alive. After that a packet is dumped from the network and the packet is considered undeliverable. When a packet is delivered, we measure the delay as the number of snapshots experienced when the packet arrives at the destination. We measure the number of hops of a successfully delivered packet as the number of relay nodes on the path from its source to the destination. The initial locations of the mobile nodes, that is, the locations at snapshot 0, are the same as the static setting above. Also, the 400 pairs of source and destination are chosen as the same as the static setting so that we can examine the benefit brought by node mobility. When the node with the packet is within transmission range of the destination location, the packet is considered delivered successfully. Figure 35 shows greedy routing with original coordinates and the message gets stuck in some middle snapshots. Figure 36 demonstrates the message arrives in 12 snapshots without any stuck with our virtual coordinates greedy routing.

We record the delay, delivery rate and hop count for routing tests under different TTL setup. The results are concluded in Table 4.

We can observe from the results that the delivery rate of greedy routing using geographical locations when TTL equals to 50 is already higher than that of the static setting, for uniformly and randomly chosen source and destinations. Indeed this is due to node mobility and verifies that mobility can improve delivery. As mobility may bring a new neighbor closer to the destination for a packet stuck at a local minimum, the delivery rate also increases with maximum TTL. However, we can see that such improvement of delivery rate is at the expense of prolonged delay. For source and destination pairs in the regions that were difficult to reach one another, the delivery rate is substantially improved when mobility is allowed.

Greedy routing using geographical locations on uniformly randomly chosen pairs			
TTL	delivery rate	hops	delay
10	238/400	5.15	5.15
20	286/400	6.28	6.29
30	295/400	6.69	6.76
50	302/400	7.37	7.51
100	317/400	9.87	10.32
150	326/400	12.51	13.47
200	338/400	17.39	19.36
Greedy routing using geographical locations on specially chosen pairs			
Methods	delivery rate	hops	delay
10	0/400	0	0
20	2/400	15.5	16.5
30	15/400	21.73	23.13
50	35/400	28.74	30.51
100	98/400	51.72	57.88
150	129/400	64.50	73.43
200	178/400	87.04	100.61
Greedy routing using virtual coordinates on uniformly randomly chosen pairs			
TTL	delivery rate	hops	delay
10	231/400	5.28	5.28
20	379/400	8.43	8.43
30	400/400	9.12	9.12
Greedy routing using virtual coordinates on specially chosen pairs			
Methods	delivery rate	hops	delay
10	0	0	0
20	254/400	16.50	16.50
30	400/400	18.40	18.40

Table 4. Performance of the methods under mobile setting of computer simulation.

The same observation is true for our method as well. Nevertheless with virtual coordinates our delivery rate is already pretty high. So we could deliver the messages with much smaller delay and hops. In particular, notice that in our experiments the values for hops and delay are always the same for our method. This means our method never gets stuck — in every time slot we are able to find a neighbor that is closer to the destination; the message is never held at a node for more than one time unit. This explains the reduced delay and the high delivery rate. For

the messages that are indeed delivered in greedy routing method using geographical locations, the delay and hop counts experienced are similar to that in our methods. This basically says that for messages that start at positions whose straight line paths to the destination are not 'blocked', and they can be delivered fairly quickly and easily, using both methods. But for messages that get stuck, using geographical locations they will need to wait for a long time for the node holding them to move out of that region; using our methods they have no problem to find good neighbors.

In conclusion, comparing greedy routing with geographical locations and with our virtual coordinates, we have a higher delivery rate with much lower delay. This is because the virtual coordinates do a better job at directing packets to the correct directions.

6.6.2 Emulation on Orbit

In this section, we demonstrate results from emulation experiments on a real testbed to validate our algorithm and compare to greedy routing with geographical locations. The emulation is deployed on the publicly accessible Orbit testbed [3]. This testbed consists of 400 nodes (standard Linux PCs), and it supports both wired and wireless experiments. The nodes are placed in a two-dimensional rectangular grid with 1 meter spacing and wireless antennas mounted on the sides.

6.6.2.1 Experimental Setup

The network we created on Orbit has the same boundary as what we used in the simulations. There are 340 nodes in this C-shape area. Among them 233 nodes (shown in Figure 33 (a)) of the grid are active nodes on Orbit. The rest nodes are artificially removed from our experiments. In our experiments, not all nodes selected are available to use and nodes may go down for no reason. So we do not have uniformly distributed nodes per our observation. The corresponding map of the network and virtual coordinates are shown in Figure 33 (b).

The network topology is created with a unit disk graph model with communication range of 2.5. For two nodes within communication range, we used the wired communication channel on Orbit for packet transmission in the experiments, since the wireless radios on Orbit were set up with very large communication range,

essentially covering the entire network. UDP is chosen as the communication protocol between the nodes. Although the communication graph is selected from a unit disk graph model, the packet is not always delivered even if the sender and receiver are within transmission range. This is observed from our experiments and could be observed from the results reported later.

Each of the node is equipped with two functions. 1) Initialize a packet as the sender and send it to the next node. 2) Listen and forward the message to the next node. The two functions are designed as threads, so the nodes can have the two functions at the same time. This simulates the scenario of a node in full duplex mode. We utilized a link table to filter packets at the application layer so that each node only receives the messages from its neighbors within communication range.

Since the local clocks at the Orbit nodes are not consistent, we cannot directly take the difference of the packet departure and arrival time. Instead, we work around this issue by sending a confirmation from the destination back to the source. Therefore the delay reported in our result actually includes this extra 'acknowledgement' step. Notice that since this extra step is added to both methods the comparison is still fair. The value for delay reported later is the median value.

6.6.2.2 Static setting

40 pairs of source and destination are chosen among the above 233 nodes for routing. The nodes' geographical coordinates are their grid coordinates. Same as in computer simulation, the pairs are chosen in two different ways. 1) uniformly and randomly selected source and destination pairs. 2) Special areas selected pairs. In case (2), sources are chosen from area A in Figure 33, and destinations are chosen from area B, where the packet is difficult to deliver under greedy routing algorithm using geographical locations.

Each node can obtain the locations of other nodes from the global node location table. And nodes can get their neighbor information from the global link table. To run our experiments, we issue 40 pairs of packets through the central controller. The sender initializes a packet with destination information and sends it to one of the neighbors according to the greedy algorithm. Since all the nodes listen in the network. If a packet is sent to a node, the node catches it. The node will then forward the packet to the next node until the packet arrives at the destination.

Uniformly randomly chosen pairs			
Methods	delivery rate	hops	delay (ms)
Geographical locations	27/40	5.8	25.669
Virtual coordinates	40/40	8.23	29.631
Specially chosen pairs			
Methods	delivery rate	hops	delay (ms)
Geographical locations	0/40	0	0
Virtual coordinates	38/40	16	33.270

Table 5. Performance of the methods under static setting of Orbit emulation.

The choice of which neighbor to send packet to depends on the node's location and connection specified in the global node locations table and global link table. The experiment setups for both greedy routing with geographical coordinates and virtual coordinates are the same except the different set of coordinates.

We compare delay, hops and delivery rate under the two methods, with randomly chosen pairs and specially chosen pairs. The results are listed in Table 5.

We simulate the experiment with all the same setting on the computer, and the delivery rate for virtual coordinates routing under special chosen area is 40/40. Notice that some of the delivery rate under our simulations (the ideal setting) is higher than the results we get from Orbit experiments. The reasons are 1) the Orbit network may experience unknown stability issues and packets may get lost for no reason (since UDP is used); 2) two packets may arrive at a node at the same time. Thus due to competition (similar to wireless interference) one of them may be lost; 3) the nodes may be go up and down during the experiments Nevertheless this represents a practical setting in which transmission failures may as well occur.

Under the same network condition, our method consistently works better than greedy routing with geographical locations, especially for pairs in the special chosen areas. The results are similar to the results obtained in our computer simulations.

6.6.2.3 Mobile setting

The nodes on Orbit are static. In order to mimic a mobile network on Orbit, we generate a mobile network scenario using simulations and then map the mobile nodes in our simulations onto the Orbit nodes. A mobile node is mapped to the

Greedy routing using geographical locations on uniformly randomly chosen pairs			
TTL	delivery rate	hops	delay (ms)
10	21/40	5.20	31.215
20	24/40	6.12	32.275
30	25/40	6.71	33.130
200	29/40	10.56	43.873
Greedy routing using geographical locations on specially chosen pairs			
TTL	delivery rate	hops	delay (ms)
10	0/40	0	0
20	0/40	0	0
30	1/40	12	93.978
200	10/40	50.72	629.985
Greedy routing using virtual coordinates on uniformly randomly chosen pairs			
TTL	delivery rate	hops	delay (ms)
10	21/40	5.22	31.582
20	33/40	8.27	32.232
30	37/40	9.01	32.775
Greedy routing using virtual coordinates on specially chosen pairs			
TTL	delivery rate	hops	delay (ms)
10	0/40	0	0
20	21/40	14.28	52.536
30	32/40	16.21	71.700

Table 6. Performance of the methods under mobile setting of Orbit.

closest Orbit node. In this case, we can use the movement of nodes from computer simulation to emulate the mobility of nodes on Orbit. Thus, the nodes move in the same way as reported earlier and the actual transmission are done on Orbit.

40 pairs of packet are chosen randomly uniformly and from the special area same as the above experiments respectively. Different from computer simulation, the delay on real testbed of Orbit is the real time difference between the message sent and received. The results are reported in Table 6.

The observation that mobility helps to improve delivery rate still holds for the testbed experiments, for both geographical greedy routing and our method. Again our method gives much smaller delay and fewer number of hops. We may notice

that there could be a large deviation from the hop counts and packet delay, in particular for geographical greedy routing for specially selected pairs. As in geographical greedy routing, many packets go through some small number of critical nodes. In our simulations we assume that all these packets go through with ease – an unrealistically ideal case. But on Orbit, when an Orbit node is called by several packets (invoked frequently), a queue is formed and this could substantially increase the delay.

6.6.3 Networks With Holes

We also run simulations in a domain with holes for both static and mobile network setting.

6.6.3.1 The Experimental Setup

In the simulations, we throw 1000 nodes uniformly and randomly inside a domain with holes as showed in Figure 32 (a). 1000 nodes are uniformly distributed inside the domain. The communication range of the nodes also follows the Unit Disk Graph (UDG) model. The average degree of each node is 7 when we set the unit distance as 2.8 when. The corresponding conformal mapping results of the network under our algorithm is shown in Figure 32 (a).

We run routing tests for 400 pairs of source and destinations. The 400 pairs of source and destination are also chosen in two different ways mentioned above, they are randomly and uniformly selected from the special areas A as source and B as destination in Figure 32 (a) respectively. The 400 randomly chosen pairs are the same for the two algorithms for fairness.

In the experiments, we also include additional 222 static sensor nodes on the hole boundaries to the network, in addition to the 1000 nodes in the interior. This improves the performance of our algorithm, as shown later. To understand this, recall that our conformal map is applied for the *continuous* geometric domain, while the sensor network is a discrete network that may not always cover the entire domain nicely. In particular, there may not be sensors on the circular hole boundaries and thus the real 'hole' in the sensor network may have tiny sawtooth-type variations. This may create cases when a packet can locally get stuck on such a varied

Uniformly randomly chosen pairs without additional boundary nodes		
Methods	delivery rate	hops
Geographical locations	315/400	6.63
Virtual coordinates	374/400	7.40
Specially chosen pairs without additional boundary nodes		
Methods	delivery rate	hops
Geographical locations	73/400	15.6
Virtual coordinates	136/400	15.54
Uniformly randomly chosen pairs with additional boundary nodes		
Methods	delivery rate	hops
Geographical locations	326/400	6.64
Virtual coordinates	400/400	7.62
Specially chosen pairs with additional boundary nodes		
Methods	delivery rate	hops
Geographical locations	0/400	0
Virtual coordinates	400/400	15.58

Table 7. Performance of the methods under static setting of computer simulation with holes.

hole boundary. Placing additional fixed sensor nodes on the hole boundaries greatly help to reduce the number of such scenarios.

6.6.3.2 Static Setting

We present the simulation results in Table 7. We can see from the results that our observations still hold – the delivery rate of our method is much better than greedy routing with geographical location. Placing additional nodes on the domain boundary also helps to further improve the delivery rate of our algorithm. What is interesting (and a bit surprising) is that placing these additional boundary nodes may make geographical greedy routing worse, especially for the specially chosen pairs! Because additional boundary nodes make sure that geographical greedy routing definitely direct the messages to the 'pockets' in the domain and get stuck there.

6.6.3.3 Mobile setting

In the mobile setting, the movement of nodes follows random waypoint mobility model [29], the same as the previous simulations but with different parameters. Specifically, we set the moving disk radius r for all nodes to be 3. The moving speed v_i for each node i is drawn from $[1,4]$ uniformly and independently. When node i comes to stay stage, stay duration t_i is uniformly drawn from $[0,3]$. Finally, the sample time duration for the next snapshot is uniformly distributed in the range $[1,4]$.

The results are concluded in Table 8 and Table 9. For the mobile network setting including the static boundary nodes always helps our proposed methods.

6.7 Conclusion

This chapter describes a compact way to represent a conformal map and applies it to routing in both static and mobile networks. As the first practical solution to using virtual coordinates with greedy routing, we expect to see implementations of our method in real world applications.

Greedy routing using geographical locations on uniformly randomly chosen pairs, without boundary nodes			
TTL	delivery rate	hops	delay
10	230/400	5.15	5.15
20	341/400	7.60	7.66
30	356/400	8.19	8.35
50	369/400	9.16	9.50
100	378/400	10.42	11.15
150	385/400	11.91	13.29
200	387/400	12.57	14.23
Greedy routing using geographical locations on specially chosen pairs, without boundary nodes			
Methods	delivery rate	hops	delay
10	0/400	0	0
20	82/400	14.96	15.21
30	106/400	16.39	16.75
50	142/400	22.03	23.99
100	200/400	32.32	38.02
150	279/400	48.14	61.80
200	316/400	57.63	74.94
Greedy routing using virtual coordinates on uniformly randomly chosen pairs, without boundary nodes			
TTL	delivery rate	hops	delay
10	247/400	5.35	5.36
20	394/400	8.14	8.16
30	398/400	8.29	8.32
50	399/400	8.34	8.37
Greedy routing using virtual coordinates on specially chosen pairs, without boundary nodes			
Methods	delivery rate	hops	delay
10	5/400	9	9
20	266/400	15.03	15.10
30	389/400	17.25	17.40
50	398/400	17.59	17.78

Table 8. Performance of the methods without boundary nodes participation under mobile setting of computer simulation with holes.

Greedy routing using geographical locations on uniformly randomly chosen pairs, with boundary nodes			
TTL	delivery rate	hops	delay
10	242/400	5.21	5.21
20	352/400	7.41	7.52
30	358/400	7.60	7.80
50	367/400	8.04	8.57
100	376/400	8.71	10.26
150	382/400	9.17	12.09
200	385/400	9.50	13.40
Greedy routing using geographical locations on specially chosen pairs, with boundary nodes			
Methods	delivery rate	hops	delay
10	0/400	0	0
20	65/400	14.58	15.05
30	72/400	15.10	15.82
50	119/400	20.78	26.13
100	187/400	25.00	43.17
150	273/400	29.27	67.44
200	311/400	31.58	80.85
Greedy routing using virtual coordinates on uniformly randomly chosen pairs, with boundary nodes			
TTL	delivery rate	hops	delay
10	258/400	5.44	5.44
20	397/400	7.91	7.91
30	400/400	8.02	8.02
Greedy routing using virtual coordinates on specially chosen pairs, with boundary nodes			
Methods	delivery rate	hops	delay
10	3/400	9	9
20	340/400	15.28	15.28
30	400/400	16.24	16.24

Table 9. Performance of the methods with boundary nodes participation under mobile setting of computer simulation with holes.

Chapter 7

The Emergence of Sparse Spanners and Well-Separated Pair Decomposition Under Anarchism

7.1 Introduction

A geometric graph G defined on a set of points $P \subseteq \mathbb{R}^d$ with all edges as straight line segments of weight equal to their length is called a *Euclidean spanner*, if for any two points $p, q \in P$ the shortest path in G has length at most $t \cdot |pq|$ where $|pq|$ is the Euclidean distance. The factor t is called the *stretch factor* of G and the graph G is called an t-spanner. Spanners with a sparse set of edges provide good approximations to the pairwise Euclidean distances and are good candidates for network backbones. Thus, there has been a lot of work on the construction of sparse Euclidean spanners in both centralized [52, 115] and distributed settings [123].

In this chapter we are interested in the emergence of good Euclidean spanners formed by uncoordinated agents. Many real-world networks, such as the transportation network and the Internet backbone network, are good spanners — one can typically drive from any city to any other city in the U.S. with the total travel distance at most a small constant times their straight line distance. The same thing happens with the Internet backbone graph. However, these large networks are not owned or built by any single authority. They are often assembled with pieces built

by different governments or different ISPs, at different points in time. Nevertheless altogether they provide a convenient sparse spanner. The work in this chapter is motivated by this observation of the lack of coordination in reality and we would like to interpret why a good Euclidean spanner is able to 'emerge' from these agents incrementally.

Prior work that attempt to remove centralized coordination has been done, as in the network creation game [11, 41, 55, 91, 111], first introduced by Fabrikant *et al.* [55] to understand the evolution of network topologies maintained by selfish agents. A cost function is assigned to each agent, capturing the cost paid to build connections to others minus the benefit received due to the resulting network topology. The agents play a game by minimizing their individual costs and one is interested in the existence and the price of anarchy of Nash equilibria. Though being theoretically intriguing, there are two major open questions along this direction. First, the choice of cost functions is heuristic. Almost all past literatures use a unit cost for each edge and they deviate in how the benefit of 'being connected to others' is modeled. There is little understanding on what cost function best captures reality yet small variation in the cost function may result in big changes in the network topologies at Nash equilibria. There is also not much understanding of the topologies at Nash equilibria, some of them are simplistic topologies such as trees or complete graphs, that do not show up often in the real world. It remains open whether there is a natural cost model with which the Nash equilibrium is a sparse spanner.

The game theoretic model also has limitations capturing the reality: selfish agents may face deadlines and have to decide on building an edge or not immediately; once an edge is built, it probably does not make sense to remove it (as in the case of road networks); an agent may not have the strategies of all other agents making the evaluation of the cost function difficult. In this chapter, we take a different approach and ask whether there is any simple rule, with which each agent can determine on its own, and collectively build and maintain a sparse spanner topology without any necessity of coordination or negotiation. The simple rule serves as a 'certificate' of the sparse spanner property that warrants easy spanner maintenance under edge dynamics and node insertion. We believe such models and good algorithms under these models are worth further exploration and this chapter makes a

first step along this line.

7.1.1 Our contribution

We consider in this chapter the following model that abstracts the scenarios explained earlier. There are n points in the plane. Each point represents a separate agent and may consider to build edges from itself to other points. These decisions can happen at different points in time. When an agent p plans on an edge pq, p will only build it if there does not exist a 'nearby' edge $p'q'$ in the network, where $|pp'|$ and $|qq'|$ are within $\frac{1}{4(1+1/\varepsilon)} \cdot |p'q'|$ from p and q respectively. This strategy is very intuitive — if there is already a cross-country highway from Washington D.C. to San Francisco, it does not make economical sense to build a highway from New York to Los Angeles. We assume that each agent will eventually check on each possible edge from itself to all other points, but the order on who checks which edge can be *completely arbitrary*. With this strategy, the agents only make decisions with limited information and no agent has full control over how and what graph will be constructed. It is not obvious that this strategy will lead to a sparse spanner. It is not clear that the graph is even connected.

The main result in this chapter is to prove that with the above strategy executed in *any* arbitrary order, the graph built at the end of the process is a sparse spanner:

- The stretch factor of the spanner is $1+\varepsilon$.
- The number of edges is $O(n)$.
- The total edge length of the spanner is $O(|\text{MST}| \cdot \log \alpha)$, where α is the *aspect ratio*, i.e., the ratio of the distance between the furthest pair and the closest pair, and $|\text{MST}|$ is the total edge length of the minimum spanning tree of the point set. The $\log \alpha$ factor could be improved to $O(\log n)$ by using a different technique.
- The degree of each point is $O(\log \alpha)$ in the worst case and $O(1)$ on average.

To explain how this result is proved, we first obtain as a side product the following *greedy* algorithm for computing a well-separated pair decomposition. A pair of two sets of points, (A,B), is called *s-well-separated* if the smallest distance between any two points in A,B is at least s times greater than the diameters of A and B.

An *s-well-separated pair decomposition* (s-WSPD for short) for P is a collection of s-well-separated pairs $\mathcal{W} = \{(A_i, B_i)\}$ such that for any pair of points $p, q \in P$ there is a pair $(A, B) \in \mathcal{W}$ with $p \in A$ and $q \in B$. The size of an s-WSPD is the number of point set pairs in $\mathcal{W}$. Well-separated pair decomposition (WSPD) was first introduced by Callahan and Kosaraju [34] and they developed algorithms for computing an s-WSPD with linear size for points in $\mathbb{R}^d$. Since then WSPD has found many applications in computing k-nearest neighbors, n-body potential fields, geometric spanners and approximate minimum spanning trees [14, 15, 31–34, 54, 78, 102, 114].

So far there are three algorithms for computing optimal size WSPD, in [34], [85] and in [68]. All three of them use a hierarchical organization of the points (e.g., the fair split tree in [34], the compressed quadtree in [85] and the discrete center hierarchy in [68]) and output the well-separated pairs in a recursive way. In this chapter we show the following simple algorithm also outputs an s-WSPD with linear size. We take an *arbitrary* pair of points p, q that is not yet covered in any existing well-separated pair, and consider the pair of subsets $(B_r(p), B_r(q))$ with $r = |pq|/(2s+2)$ and $B_r(p)$ ($B_r(q)$) as the set of points of P within distance r from p (q). Clearly $(B_r(p), B_r(q))$ is an s-well-separated pair and all the pairs of points (p', q') with $p' \in B_r(p)$ and $q' \in B_r(q)$ are covered. The algorithm continues until all pairs of points are covered. We show that, no matter in which order the pairs are selected, the greedy algorithm will always output a linear number of well-separated pairs. Similarly, this algorithm can be executed in an environment when coordination is not present, while the previous algorithms (in [34, 68]) cannot.

WSPD is deeply connected to geometric spanners. Any WSPD will generate a spanner if one puts an edge between an arbitrary pair of points p, q from each well-separated pair $(A, B) \in \mathcal{W}$ [14, 15, 102, 114]. The number of edges in the spanner equals the size of $\mathcal{W}$. In the other direction, the deformable spanner proposed in [68] implies a WSPD of linear size. The connection is further witnessed in this chapter: our spanner emergence algorithm implies a WSPD generated under anarchism. Thus the well-separated pairs and spanner edges are in one-to-one correspondence.

Last, this chapter focuses on the Euclidean case when the points are distributed in the plane. The basic idea extends naturally to points in higher dimensions as well as metrics with constant doubling dimensions [81] (thus making the results

introduced in previous section applicable in non-Euclidean settings), as the main technique involves essentially various forms of geometric packing arguments. Sparse spanners and WSPD exist for metrics with constant doubling dimension as shown in [85] and [68] in a centralized setting. Ours uses distributed construction.

7.1.2 Applications

The results can be applied in maintaining network overlay topologies with good properties for P2P file sharing applications [106]. Such P2P overlay networks are often constructed in a distributed manner without centralized control, to achieve robustness, reliability and scalability. One important issue is reducing routing delay by making the overlay topology aware of the underlying network topology [39, 100, 128, 155, 156]. But all these existing algorithms are heuristics without any guarantee. A spanner graph would be a good solution for the overlay construction, yet there is no centralized authority in the P2P network that supervises the spanner construction and the peers may join or leave the network frequently. The work in this chapter initiates the study of the emergence of good spanners in the setting when there is little coordination between the peers and the users only need a modest amount of incomplete information of the current overlay topology.

It has been shown that the delay on the Internet has geometric growth properties [118, 125]. Thus we can apply our spanner construction for a P2P network under the model that the delay function has constant doubling dimension. We show that the spanner can be constructed under a proper model of the P2P network such that only $O(n \log \alpha)$ messages need to be delivered. The spanner topology is implicitly stored on the nodes with each node's storage cost bounded by $O(\log \alpha)$. With such partial information stored at each node, there is a local distributed algorithm that finds a $(1+\varepsilon)$-stretch path between any two nodes.

7.1.3 Related work

In the vast amount of prior literature on geometric spanners, there are three main ideas: Θ-graphs, the greedy spanners, and the WSPD-induced spanners [115]. Please refer to the book Geometric Spanner Networks for a nice survey [115]. We will review two spanner construction ideas that are most related to our approach.

The first idea is the path-greedy spanner construction [36, 44–46]. All pairwise edges are ordered with non-decreasing lengths and checked in that order. An edge is included in the spanner if the shortest path in the current graph is longer than t times the Euclidean distance, and is discarded otherwise. Variants of this idea generate spanners with constant degree and total weight $O(|\text{MST}|)$. This idea cannot be applied in our setting as edges constructed in practice may not be in non-decreasing order of their lengths. The second idea is to use the gap property [36] — the sources and sinks of any two edges in an edge set are separated by a distance at least proportional to the length of the shorter of the two edges and their directions differ no more than a given angle. The gap-greedy algorithm [16] considers pairs of points, again in order of non-decreasing distances, and includes an edge in the spanner if and only if it does not violate the gap property. The spanner generated this way has constant degree and total weight $O(|\text{MST}|)$. Compared with our algorithm, our strategy is a relaxation of the gap property in the way that the edges in our spanner may have one of their endpoints arbitrarily close (or at the same points) and we have no restriction on the direction of the edges and the ordering of the edges to be considered. The proof for the gap greedy algorithm requires heavy plane geometry tools and our proof technique only uses packing argument and can be extended to the general metric setting as long as a similar packing argument holds. To get these benefits our algorithm has a slightly worse upper bounds on the spanner weight by a logarithmic factor.

Prior work on compact routing [6, 7, 74, 98, 144] usually implies a $(1+\varepsilon)$-spanner explicitly or implicitly. Again, these spanners are constructed in a coordinated setting.

An extended abstract of this chapter was presented in 2008 and also published in [72]. During the review of this thesis version, we became aware of the independent work at around the same time by Smid [145], which introduced the 'weak gap property' for spanner construction. The spanner constructed from our algorithm satisfies the weak gap property. Therefore, the properties resulting from weak gap property can be applied here directly. Specifically, it says any directed graph satisfying the weak gap property has $O(n)$ edges and total weight $O(\lg n \cdot |\text{MST}|)$. This corresponds to our conclusion on the linear size of the spanner edges and improves our result on the weight of the constructed spanner from $O(\lg \alpha \cdot |\text{MST}|)$

to $O(\lg n \cdot |\text{MST}|)$. The proof techniques and methodologies in the two proofs are different however.

7.1.4 Organization

In the rest of the chapter we first elaborate the spanner construction in an uncoordinated manner and then show the connection of the spanner with the greedy WSPD. We then show the good properties of both the greedy WSPD and our spanner. At the end, we describe how to apply the spanner in the P2P setting to support low-storage spanner representation and efficient local low-stretch routing.

7.2 Spanner construction under anarchism

Given n points in $\mathbb{R}^d$, each point represents an agent. As explained in the introduction, we consider the following algorithm for constructing a sparse spanner with stretch factor s in an uncoordinated way. For any point p, denote by $B_r(p)$ the collection of points that are within distance r from point p, i.e., inside the ball with radius r centered at p.

7.2.1 Spanner construction algorithm

Each point/agent p checks to see whether an edge from itself to another point q should be constructed or not. At this point there might be some edges already constructed by other agents. The order of which agent checks on which edge is completely *arbitrary*. Specifically, p performs the following operations:

Check where there is already an edge $p'q'$ such that p and q are within distance $\frac{|p'q'|}{2(s+1)}$ from p', q' respectively. If so, p does not build the edge to q. Otherwise, p will build an edge to q.

This incremental construction of edges is executed by different agents in a completely uncoordinated manner. We assume that no two agents perform the above strategy at exactly the same time. Thus when any agent conducts the above process, the decision is based on the current network already constructed. The algorithm terminates when all agents finish checking the edges from themselves to all

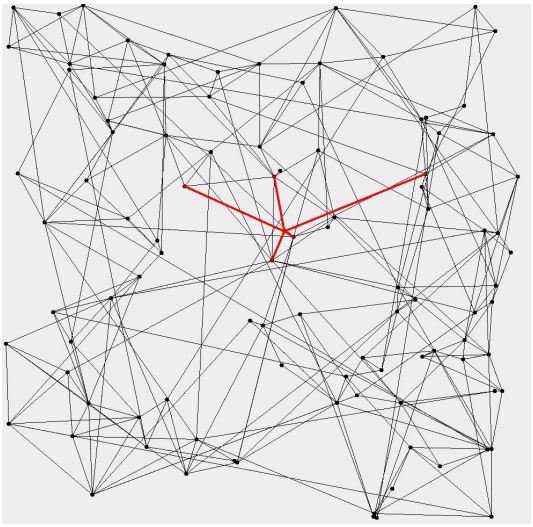


Figure 37. A greedy spanner example for 100 points with aspect ratio $\alpha = 223$, the average degree is 6.5, and the stretch is 3.4.

other points. In this chapter we first study the properties of the constructed graph G by these uncoordinated behaviors. We will discuss later in Section 7.5 a proper complexity model for the uncoordinated construction in a distributed environment and also bound the computing cost of this spanner. An spanner example generated from the above algorithm is shown in Figure 37. Throughout the construction the following invariant is maintained by the graph G.

Lemma 25. *1. For any edge pq that is not in G, there is another edge $p'q'$ in G such that $|pp'| \leq |p'q'|/(2s+2)$, $|qq'| \leq |p'q'|/(2s+2)$.*

2. For any two edges pq, $p'q'$ in the constructed graph G, suppose that pq is built before $p'q'$, then one of the following is true: $|pp'| > |pq|/(2s+2)$ or $|qq'| > |pq|/(2s+2)$.

To show that the algorithm eventually outputs a good spanner, we first show the connection of G with the notion *well-separated pair decomposition*.

Definition 26 (Well-separated pair). *Let $s > 0$ be a constant, and a pair of sets of points A, B are well-separate with respect to s (or s-separated), if $d(A,B) \geq s \cdot \max(diam(A), diam(B))$, where $diam(A)$ is the diameter of the point set A, and $d(A,B) = \min_{p \in A, q \in B} |pq|$.*

Definition 27 (Well-separated pair decomposition). *Let $t > 0$ be a constant,and P be a point set. A well-separated pair decomposition (WSPD) with respect to t of P is a set of pairs $\mathcal{W} = \{\{A_1, B_1\}, \ldots, \{A_m, B_m\}\}$, such that*

1. *$A_i, B_i \subseteq P$, and the pair sets A_i and B_i are s-separated for every i.*
2. *For any pair of points $p, q \in P$, there is at least one pair (A_i, B_i) such that $p \in A_i$ and $q \in B_i$.*

Here m is called the size of the WSPD.

In our construction of G, it is not so hard to see that a well-separated pair decomposition is actually implied.

Theorem 28. *From the uncoordinated construction of the graph G, we can build the following s-WSPD $\mathcal{W}$: for each edge pq in G, include in the well-separated pair decomposition the pair $(B_r(p), B_r(q))$, with $r = |pq|/(2s+2)$. The size of the WSPD is the number of edges in G.*

Proof. First each pair $(B_r(p), B_r(q))$ is an s-well-separated pair. Obviously, $d(B_r(p), B_r(q)) \geq |pq| - 2r$, and $\mathrm{diam}(B_r(p)), \mathrm{diam}(B_r(q)) \leq 2r$. One can then verify that $d(B_r(p), B_r(q)) \geq s \cdot \max(\mathrm{diam}(B_r(p)), \mathrm{diam}(B_r(q)))$.

We now show that any point p, q is included in one well-separated pair. If the edge pq is in the graph the claim is true obviously. Otherwise, there is an edge $p'q'$ in G such that $|pp'| \leq |p'q'|/(2s+2)$, $|qq'| \leq |p'q'|/(2s+2)$, by Lemma 25. This means that $p \in B_{r'}(p')$ and $q \in B_{r'}(q')$ with $r' = |p'q'|/(2s+2)$. This finishes the proof. □

7.2.2 Algorithm for well-separated pair decomposition

The above theorem shows the connection of the uncoordinated graph G with a WSPD $\mathcal{W}$. In fact, the way to compute the WSPD $\mathcal{W}$ via the construction of G is equivalent to the following algorithm that computes an s-WSPD, where s is the separation parameter of the well-separated pairs, $s > 1$.

1. Choose an arbitrary pair (p, q), not yet covered by existing well-separated pairs in $\mathcal{W}$.

2. Include the pair of point sets $B_r(p)$ and $B_r(q)$ in the WSPD $\mathcal{W}$, with $r = |pq|/(2+2s)$.
3. Label the point pair (p_i, q_i) with $p_i \in B_r(p)$ and $B_r(q)$ as being covered.
4. Repeat the above steps until every pair of points is covered.

With the s-WSPD $\mathcal{W}$, the uncoordinated construction of the graph G is in fact by taking an edge from each and every well-separated pair in $\mathcal{W}$ — the simple rule in Lemma 25 prevents two edges from the same well-separated pair in $\mathcal{W}$ to be constructed. It is already known that for any well-separated pair decomposition, if one edge is taken from each well-separated pair, then the edges will become a spanner on the original point set [14, 15, 102, 114]. For our specific greedy s-WSPD, we are able to get a slightly better stretch, since our well separated pairs are centered on two points and we are picking the edge between these two points as the spanner edge. The previous schemes choose an arbitrary edge in each well separated pair.

Theorem 29. *Graph G constructed from the greedy s-WSPD is a spanner with stretch factor $(s+1)/(s-1)$.*

Proof. Denote by $\pi(p,q)$ the shortest path length between p,q in the graph G. We show that $\pi(p,q) \leq \beta \cdot |pq|$ for any $p,q \in P$, with $\beta = (s+1)/(s-1)$. We prove this claim by induction on the distance between two points p,q. Take p,q as the closest pair of P. Then any s-WSPD will have to use a singleton pair (p,q) to cover the pair (p,q) if $s \geq 1$. Otherwise, say (P,Q) is an s-well-separated pair that covers (p,q), and $|P| > 1$. Then $\text{diam}(P) > |pq|$, and $d(P,Q) = |pq|$. This contradicts the fact that $d(P,Q) \geq s \cdot \text{diam}(P)$. Therefore the edge pq is included in G for sure and $\pi(p,q) = |pq|$.

Now suppose that for all pairs of nodes x,y with Euclidean distance $|xy| \leq \ell$, we have $\pi(x,y) \leq \beta \cdot |xy|$. Now we consider the pair of nodes p,q with the smallest distance (among all remaining pairs) that is still greater than ℓ. (p,q) is covered by an s-well-separated pair $(P,Q) \in \mathcal{W}$, where $P = B_r(p')$ and $Q = B_r(q')$ with $r = |p'q'|/(2s+2)$ and $p'q'$ an edge in G. Now we argue that $|pp'| \leq \ell$. Otherwise, $|pq| \geq d(P,Q) \geq s \cdot \text{diam}(P) \geq s \cdot |pp'| > |pp'|$. So we should have selected the pair (p,p') instead of (p,q). Similarly, $|qq'| \leq \ell$. Thus by induction hypothesis $\pi(p,p') \leq \beta \cdot |pp'|$, $\pi(q,q') \leq \beta \cdot |qq'|$. By triangle inequality, we have

$\pi(p,q) \leq \pi(p,p') + |p'q'| + \pi(q,q') \leq \beta \cdot (|pp'| + |qq'|) + |p'q'| \leq 2\beta \cdot r + |p'q'|$. On the other hand, we know by triangle inequality that $|pq| \geq |p'q'| - 2r = \frac{s}{s+1} \cdot |p'q'|$. Combining everything we get that $\pi(p,q) \leq (\frac{\beta}{s+1} + 1) \cdot \frac{s+1}{s} \cdot |pq| = \beta|pq|$, with $\beta = (s+1)/(s-1)$. This finishes the proof. □

The above theorem shows that the uncoordinated construction is indeed a s-panner. To make the stretch factor as $1+\varepsilon$, we just take $s = 1 + 2/\varepsilon$ in our spanner construction. We also want to show that the spanner is sparse and has some other nice properties useful for our applications. For that we will first show that the greedy WSPD algorithm will output a linear number of well-separated pairs, in the next section.

7.3 A greedy algorithm for well-separated pair decomposition

To show that the WSPD by the greedy algorithm has a linear number of pairs, we actually show the connection of this WSPD with a specific WSPD constructed by the deformable spanner [68], in the way that at most a constant number of pairs in $\mathcal{W}$ is mapped to each well-separated pair constructed by the deformable spanner. To be consistent, in the following description, the greedy WSPD is denoted by $\mathcal{W}$ and the WSPD constructed by the deformable spanner is denoted by $\hat{\mathcal{W}}$.

7.3.1 Deformable spanner and WSPD

In this section, we review the basic definition of the deformable spanner and some related properties, which will be used in our own algorithm analysis in the next subsection.

Given a set of points P in the plane, a set of *discrete centers* with radii r is defined to be the maximal set $S \subseteq P$ that satisfies the *covering* property and the *separation* property: any point $p \in P$ is within distance r to some point $p' \in S$; and every two points in S are of distance at least r away from each other. In other words, all the points in P can be covered by balls with radii r, whose centers are exactly those points in the discrete center set S. And these balls do not cover other discrete centers.

We now define a hierarchy of discrete centers in an recursive way. S_0 is the original point set P. S_i is the discrete center set of S_{i-1} with radii 2^i. Without loss of generality we assume that the closest pair has distance 1 (as we can scale the point set and do not change the combinatorial structure of the discrete center hierarchy). Thus the number of levels of the discrete center hierarchy is $\log \alpha$, where α is the aspect ratio of the point set P, defined as the ratio of the maximum pairwise distance to the minimum pairwise distance, that is, $\alpha = \max_{u,v \in P} |uv| / \min_{u,v \in P} |uv|$. Since a point p may stay in multiple consecutive levels and correspond to multiple nodes in the discrete center hierarchy, we denote by $p^{(i)}$ the existence of p at level i. For each point $p^{(i-1)} \in S_{i-1}$ on level i, it is within distance 2^i from at least one other point on level $i+1$. Thus we assign to $p^{(i-1)}$ a *parent* $q^{(i)}$ in S_i such that $|p^{(i-1)}q^{(i)}| \leq 2^i$. When there are multiple points in S_i that cover $p^{(i-1)}$, we choose one as its parent arbitrarily. We denote by $P(p^{(i-1)})$ the parent of $p^{(i-1)}$ on level i. We denote by $P^{(i)}(p) = P(P^{(i-1)}(p))$ the *ancestor* of p at level i.

The deformable spanner is based on the hierarchy, with all edges between two points u and v in S_i if $|uv| \leq c \cdot 2^i$, where c is a constant equal to $4 + 16/\varepsilon$.

Now we will restate some important properties of the deformable spanner that will be useful in our algorithm analysis.

Lemma 30 (Packing Lemma [68]). *In a point set $S \subseteq R^d$, if every two points are at least distance r away from each other, then there can be at most $(2R/r+1)^d$ points in S within any ball with radii R.*

Lemma 31 (Deformable spanner properties [68]). *For a set of n points in R^d with aspect ratio α,*

1. *For any point $p \in S_0$, its ancestor $P^{(i)}(p) \in S_i$ is of distance at most 2^{i+1} away from p.*
2. *Any point $p \in S_i$ has at most $(1+2c)^d - 1$ edges with other points of S_i.*
3. *The deformable spanner $\hat{G}$ is a $(1+\varepsilon)$-spanner G with $O(n/\varepsilon^d)$ edges.*
4. *$\hat{G}$ has total weight $O(|MST| \cdot \lg \alpha / \varepsilon^{d+1})$, where $|MST|$ is the weight of the minimal spanning tree of the point set S.*

As shown in [68], the deformable spanner implies a well-separated pair decomposition $\hat{\mathcal{W}}$ by taking all the 'cousin pairs'. Specifically, for a node $p^{(i)}$ on

level i, we denote by P_i the collection of points that are descent of $p^{(i)}$ (including $p^{(i)}$ itself), called the *descendants*. Now we take the pair (P_i, Q_i), the sets of descendants of a *cousin pair* $p^{(i)}$ and $q^{(i)}$, i.e., $p^{(i)}$ and $q^{(i)}$ are *not* neighbors in level i but their parents are neighbors in level $i+1$. This collection of pairs constitutes a $\frac{4}{\varepsilon}$-well-separated pair decomposition. The size of $\hat{\mathcal{W}}$ is bounded by the number of cousin pairs and is shown to be $O(n/\varepsilon^d)$.

7.3.2 Greedy well-separated pair decomposition has linear size

With the WSPD $\hat{\mathcal{W}}$ constructed by the deformable spanner, we now prove that the greedy WSPD $\mathcal{W}$ has linear size as well. The basic idea is to map the pairs in $\mathcal{W}$ to the pairs in $\hat{\mathcal{W}}$ and show that at most a constant number of pairs in $\mathcal{W}$ map to the same pair in $\hat{\mathcal{W}}$.

Theorem 32. *The greedy s-WSPD $\mathcal{W}$ has size $O(ns^d)$.*

Proof. Suppose that we have constructed a deformable spanner DS with $c = 4(s+1)$ and obtained an s-well-separated pair decomposition (WSPD) of it, call it $\hat{\mathcal{W}}$, where $s = c/4 - 1$. The size of $\hat{\mathcal{W}}$ is $O(ns^d)$. The deformable spanner has stretch $1 + 4/s$. Now we will construct a map that takes each pair in $\mathcal{W}$ and map it to a pair in $\hat{\mathcal{W}}$.

Each pair $\{P, Q\}$ in $\mathcal{W}$ is created by considering the points inside the balls $B_r(p), B_r(q)$ with radii $r = |pq|/(2+2s)$ around p, q. Now we consider the ancestors of p, q in the spanner DS respectively. There is a unique level i such that the ancestor $u_i = P^{(i)}(p)$ and $v_i = P^{(i)}(q)$ do not have a spanner edge in between but the ancestor $u_{i+1} = P^{(i+1)}(p)$ and $v_{i+1} = P^{(i+1)}(q)$ have an edge in between. The pair u_i, v_i is a cousin pair by definition and thus the decedents of them correspond to an s-well-separated pair in $\hat{\mathcal{W}}$. We say that the pair $(B_r(p), B_r(q)) \in \mathcal{W}$ maps to the descendant pair $(P_i, Q_i) \in \hat{\mathcal{W}}$.

By the discrete center hierarchy (Lemma 31), we show that,

$$|pq| \geq |u_i v_i| - |p u_i| - |q v_i| \geq |u_i v_i| - 2 \cdot 2^{i+1} \geq (c-4) \cdot 2^i.$$

The last inequality follows from that fact that u_i, v_i do not have an edge in the spanner and $|u_i v_i| > c \cdot 2^i$. On the other hand,

$$|pq| \leq |p u_{i+1}| + |u_{i+1} v_{i+1}| + |q v_{i+1}| \leq 2 \cdot 2^{i+2} + c \cdot 2^{i+1} = 2(c+4) \cdot 2^i.$$

The last inequality follows from the fact that u_{i+1}, v_{i+1} have an edge in the spanner and $|u_{i+1}v_{i+1}| \leq c \cdot 2^{i+1}$. Similarly, we have

$$c \cdot 2^i < |u_i v_i| \leq |u_i u_{i+1}| + |u_{i+1} v_{i+1}| + |v_i v_{i+1}| \leq 2 \cdot 2^{i+1} + c \cdot 2^{i+1} = 2(c+2) \cdot 2^i.$$

Therefore the distance between p and q is $c' \cdot |u_i v_i|$, where $(c-4)/(2c+4) \leq c' \leq (2c+8)/c$.

Now suppose two pair $(B_{r_1}(p_1), B_{r_1}(q_1))$, $(B_{r_2}(p_2), B_{r_2}(q_2))$ in $\mathcal{W}$ map to the same pair u_i and v_i by the above process. Without loss of generality suppose that p_1, q_1 are selected before p_2, q_2 in our greedy algorithm. Here is the observation:

1. $|p_1 q_1| = c'_1 \cdot |u_i v_i|$, $|p_2 q_2| = c'_2 \cdot |u_i v_i|$, $r_1 = |p_1 q_1|/(2+2s) = c'_1 \cdot |u_i v_i|/(2+2s)$, $r_2 = c'_2 \cdot |u_i v_i|/(2+2s)$, where $(c-4)/(2c+4) \leq c'_1, c'_2 \leq (2c+8)/c$, and r_1, r_2 are the radii of the balls for the two pairs respectively.
2. The reason that (p_2, q_2) can be selected in our greedy algorithm is that at least one of p_2 or q_2 is outside the balls $B(p_1), B(q_1)$, by Lemma 25. This says that at least one of p_2 or q_2 is of distance r_1 away from p_1, q_1.

Now we look at all the pairs (p_ℓ, q_ℓ) that are mapped to the same ancestor pair (u_i, v_i). The pairs are ordered in the same order as they are constructed, i.e., p_1, q_1 is the first pair selected in the greedy WSPD algorithm. Suppose r_{min} is the minimum among all radius r_i. $r_{min} \geq c/(2c+8) \cdot |u_i v_i|/(2+2s) = |u_i v_i|/(4s+8)$. We group these pairs in the following way. The first group H_1 contains (p_1, q_1) and all the pairs (p_ℓ, q_ℓ) that have p_ℓ within distance $r_{min}/2$ from p_1. We say that (p_1, q_1) is the representative pair in H_1 and the other pairs in H_1 are *close* to the pair (p_1, q_1). The second group H_2 contains, among all remaining pairs, the pair that was selected in the greedy algorithm the earliest, and all the pairs that are close to it. We repeat this process to group all the pairs into k groups, $H_1, H_2, \cdots, H_k$. For all the pairs in each group H_j, we have one representative pair, denoted by (p_j, q_j) and the rest of the pairs in this group are close to it.

We first bound the number of pairs belonging to each group by a constant with a packing argument. With our group criteria and the above observations, all p_ℓ in the group H_j are at most r_{min} away from each other. This means that the q_ℓ's must be far away — the q_ℓ's must be at least distance r_{min} away from each other, by Lemma 25. On the other hand, all the q_ℓ's are descendant of the node v_i, so $|v_i q_\ell| \leq 2^{i+1}$ by Theorem 31. That is, all the q_ℓ's are within a ball of radius

2^{i+1} centered at v_i. By the packing Lemma 30, the number of such q_ℓ's is at most $(2 \cdot 2^{i+1}/r_{min}+1)^d \leq (2 \cdot 2^{i+1}(4s+8)/|u_i v_i|+1)^d \leq (4(s+2)/(s+1)+1)^d$. This is also the bound on the number of pairs inside each group.

Now we bound the number of different groups, i.e., the value k. For the representative pairs of the k groups, $(p_1,q_1),(p_2,q_2),\cdots,(p_k,q_k)$, all the p_i's must be at least distance $r_{min}/2$ away from each other. Again these p_i's are all descendant of u_i and thus are within distance 2^{i+1} from u_i. By a similar packing argument, the number of such p_i's is bounded by $(4 \cdot 2^{i+1}/r_{min}+1)^d \leq (8(s+2)/(s+1)+1)^d$. So the total number of pairs mapped to the same ancestor pair in $\hat{\mathcal{W}}$ will be at most $(4(s+2)/(s+1)+1)^d \cdot (8(s+2)/(s+1)+1)^d = (O(1+1/s))^d$. Thus the total number of pairs in W is at most $O(ns^d)$. This finishes the proof. □

7.4 Size, degree and weight of the uncoordinated spanner

With the result that the greedy WSPD has linear size in the previous section and the connection of the greedy WSPD with the uncoordinated spanner construction in Section 7.2, we immediately get the following theorem.

Theorem 33. *The uncoordinated spanner with parameter s is a spanner with stretch factor $(s+1)/(s-1)$ and has $O(ns^d)$ number of edges.*

Proof. The number of edges in the spanner is the same as the size of the greedy WSPD $\mathcal{W}$ with the same parameter s constructed by selecting the same set of edges in the same order. □

Theorem 34. *The uncoordinated spanner has a maximal degree of $O(\lg\alpha \cdot s^d)$ and average degree $O(s^d)$.*

Proof. With the same argument as in Theorem 32, each pair (p,q) built in the uncoordinated spanner construction maps to a pair of ancestors $(P^{(i)}(p),P^{(i)}(q))$ in the deformable spanner that is a cousin pair. Consider all the edges of p in G, (p,q_ℓ), that map to the same ancestor pair $(P^{(i)}(p),P^{(i)}(q))$. By a similar argument, all the q_ℓ's must be at least distance r_{min} away from each other (since all these pairs

have p as the first element in the pair). Thus we have the number of such edges is bounded by $(4(s+2)/(s+1)+1)^d$. The cousin pairs associated with $P^{(i)}(p)$ is at most 5^d times the number of adjacent edges of $P^{(i+1)}(p)$, and is bounded by $5^d \cdot [(8s+9)^d - 1]$ (by Theorem 31). Since there are $\lfloor \lg \alpha \rfloor$ levels, the total number of edges associated with the node p is at most $\lfloor \lg \alpha \rfloor \cdot 5^d \cdot [(8s+9)^d - 1] \cdot (4(s+2)/(s+1)+1)^d$. Then the maximal degree of the spanner is $O(\lg \alpha \cdot s^d)$.

Since the spanner has total $O(ns^d)$ edges, the average degree is $O(s^d)$. □

Theorem 35. *The uncoordinated spanner has total weight $O(\lg \alpha \cdot |MST| \cdot s^{d+1})$.*

Proof. Again we use the mapping of the uncoordinated spanner edges to the cousin pairs in the deformable spanner *DS*, as in Theorem 32. We also use the same notation here. Consider all the edges (p_ℓ, q_ℓ) that map to the same ancestor cousin pair (u_i, v_i). We now map them to the edge between the parents of this cousin pairs, i.e., edge $u_{i+1}v_{i+1}$ in *DS*. The pair (u_{i+1}, v_{i+1}) has at most 5^{2d} number of cousin pairs. Thus at most $(4(s+2)/(s+1)+1)^d \cdot (8(s+2)/(s+1)+1)^d \cdot 5^{2d} = (O(1+1/s))^d$ edges in G are mapped to one edge in *DS*.

Now we will bound the length of an edge pq in G and the edge $u_{i+1}v_{i+1}$ in *DS* it maps to. From the proof of Theorem 32, we know that $(c-4) \cdot 2^i \leq |pq| \leq 2(c+4) \cdot 2^i$, where $c = 4(s+1)$. In addition, $|u_{i+1}v_{i+1}| \leq 2c \cdot 2^i$ as $u_{i+1}v_{i+1}$ is an edge in *DS*, and $|u_{i+1}v_{i+1}| \geq |u_iv_i| - |u_{i+1}u_i| - |v_{i+1}v_i| \geq c \cdot 2^i - 2 \cdot 2^{i+1} = (c-4) \cdot 2^i$. Thus, $(c-4)/(2c) \leq |pq|/|u_{i+1}v_{i+1}| \leq 2(c+4)/(c-4)$.

We now bound the total weight of the spanner G. We group all the edges by the spanner edge in *DS* that they map to. Thus we have the total weight of G is at most $2(c+4)/(c-4) \cdot (O(1+1/s))^d$ the weight of *DS*. By Lemma 31, the weight of *DS* is at most $O(\lg \alpha \cdot |\mathrm{MST}| \cdot s^{d+1})$. Thus the weight of G is at most $O(\lg \alpha \cdot |\mathrm{MST}| \cdot s^{d+1})$. □

We remark here that independently Smid shows that the weight of the spanner is bounded by $O(\log n \cdot |\mathrm{MST}| \cdot s^{d+1})$ by using the weak gap property in [145].

7.5 Spanner construction and routing in P2P networks

The uncoordinated spanner construction can be applied for peer-to-peer system design, to allow users to maintain a spanner in a distributed manner. For that, we will first extend our spanner results to a metric with constant doubling dimension. The doubling dimension of a metric space (X,d) is the smallest value γ such that each ball of radius R can be covered by at most 2^γ balls of radius $R/2$ [81].

Theorem 36. *For n points and a metric space defined on them with constant doubling dimension γ, the uncoordinated spanner construction outputs a spanner G with stretch factor $(s+1)/(s-1)$, has total weight $O(\gamma^2 \cdot \lg\alpha \cdot |MST| \cdot s^{O(\gamma)})$ and has $O(\gamma^2 \cdot n \cdot s^{O(\gamma)})$ number of edges. Also it has a maximal degree of $O(\gamma \cdot \lg\alpha \cdot s^{O(\gamma)})$ and average degree $O(\gamma \cdot s^{O(\gamma)})$.*

Proof. The proof follows almost the same as those in the previous section. The deformable spanner can be applied for metrics with constant doubling dimension [68]. Whenever we use a geometric packing argument, we replace by the property of metrics of constant doubling dimension. □

7.5.1 Distributed construction.

Now we would like to discuss the model of computing for P2P overlay design as well as the construction cost of the uncoordinated spanner. We assume that there is already a mechanism maintained in the system such that any node x can obtain the distance to any node y in $O(1)$ time. For example, this can be done by a TRACEROUTE command executed by x to the node y. We also assume that there is a service answering near neighbor queries: given a node p and a distance r, return the neighbours within distance r from p. Such an oracle is often maintained in distributed file sharing systems. Various structured P2P system support such function with low cost [106]. Even in unstructured system such as BitTorrent, the Ono plugin is effective at locating nearby peers, with vanishingly small overheads [2].

The spanner edges are recorded in a distributed fashion so that no node has the entire picture of the spanner topology. After each edge pq is constructed, the

peers p, q will inform their neighboring nodes (those in $B_r(p)$ and $B_r(q)$ with $r = |pq|/(2s+2)$) that such an edge pq exists so that they will not try to connect to one another. We assume that these messages are delivered immediately so that when any newly built edge is informed to nodes of relevance. The number of messages for this operation is bounded by $|B_r(p)| + |B_r(q)|$. The amount of storage at each node x is proportional to the number of well-separated pairs that include x. The following theorem bounds the total number of such messages during the execution of the algorithm and the amount of storage at each node.

Theorem 37. *For the uncoordinated spanner G and the corresponding greedy WSPD $\mathcal{W} = \{(P_i, Q_i)\}$ with size m, each node x is included in at most $O(s^d \lg \alpha)$ well-separated pairs in $\mathcal{W}$. Thus, $\sum_{i=1}^{m} (|P_i| + |Q_i|) = O(ns^d \cdot \lg \alpha)$.*

Proof. For each point x, each pair set it belongs to will be mapped to some ancestor pair in some level in the corresponding deformable spanner. Let's consider the set pair (P_k, Q_k) that are mapped to level i by selecting the point pair (p, q), where $x \in P_k$. Suppose the ancestor pair is $(p^{(i)}(p), p^{(i)}(q))$. If x is the descendent of $p^{(i)}(p)$, there are only a constant number of such pair sets mapped to $(p^{(i)}(p), p^{(i)}(q))$ according to our previous argument, which means x will be covered only in a constant number of pair sets corresponding to this level. If x is not the descendent of $p^{(i)}(p)$, then x may be mapped to different ancestor pairs $(p^{(i)}(p), p^{(i)}(q))$. The corresponding radius with (p, q) is $r = |pq|/(2+2s) \leq (1+4/c)2^{i+2}$. In this case, $2^{i+1} < |p^{(i)}(p)x| \leq 2^{i+1} + r \leq 2^{i+1} + (1+4/c)2^{i+2} = (1+2/c)2^{i+2}$. According to Lemma 31, different $p^{(i)}(p)$ in level i must be at least 2^i away from each other. With packing argument, there can be at most $(((1+2/c)2^{i+1})^d - (2^{i+1})^d)/(2^{i-1})^d = (4+8/c)^d = (2/(s+1)+4)^d - 2^{2d} = O(s^d)$ different such $p^{(i)}(p)$. So x can only be covered in a constant number of different pairs in level i which are not mapped to x's ancestor. Combining the above argument, there are only $O(s^d)$ pair sets that cover x in each level i. And there are only $\lg \alpha$ different levels. So each node x is included in at most $O(s^d \lg \alpha)$ well-separated pairs in $\mathcal{W}$.

As there are only linear number of pairs, the last statement follows immediately. □

7.5.2 Distributed routing.

Although the spanner topology is implicitly stored on the nodes with each node only knows some piece of it, we are actually able to do a distributed and local routing on the spanner with only information available at the nodes such that the path discovered has maximum stretch $(s+1)/(s-1)$. This can be useful when we would like to operate on the overlay topology only in a P2P application for security concerns.

In particular, for any node p who has a message to send to node q, it is guaranteed that (p,q) is covered by a well-separated pair $(B_r(p'),B_r(q'))$ with $p \in B_r(p')$ and $q \in B_r(q')$. By the construction algorithm, the edge $p'q'$, after constructed, is informed to all nodes in $B_r(p') \cup B_r(q')$, including p. Thus p includes in the packet a partial route with $\{p \rightsquigarrow p', p'q', q' \rightsquigarrow q\}$. The notation $p \rightsquigarrow p'$ means that p will need to first find out the low-stretch path from p to the node p' (inductively), from where the edge $p'q'$ can be taken, such that with another low-stretch path to be found out from q' to q, the message can be delivered to q. This way of routing with partial routing information stored with the packet is similar to the idea of source routing [151] except that we do not include the full routing path at the source node. By the same induction as used in the proof of spanner stretch (Theorem 29), the final path is going to have stretch at most $(s+1)/(s-1)$. The number of hops is bounded by $d^{1/(1+\lg s)}$.

Theorem 38. *For any two points p and q, $d = |pq|$, the distributed routing scheme gives a path with at most $h(d) = d^{1/(1+\lg s)}$ hops between p and q.*

Proof. For any pair (p,q), it will be covered by a pair set (P,Q), which is created by a pair (p',q'). Then there is an edge between p' and q'. The path $p \rightsquigarrow q$ between p and q can be found in this way: $p \rightsquigarrow p', p'q', q' \rightsquigarrow q$. Obviously $|pp'| \leq r \leq |pq|/(2s)$. We can get the recurrence $h(|pq|) = h(|pp'|)+1+h(|qq'|) \leq h(|pq|/(2s))+1+h(|pq|/(2s))$, that is, $h(d) = 2h(d/(2s))+1$. Solve this recurrence we get $h(d) = 2^{\lg_{2s} d+1} = 2^{\lg d/\lg(2s)+1} = 2d^{1/(1+\lg s)}$. □

7.5.3 Nearest neighbor search.

By maintaining the spanner each node actually keeps its nearest neighbor automatically. Recall that each point x keeps all the pairs (p,q) that create a 'dumb-bell' pair set covering x. Then we claim, among all these p, one of them must be the nearest neighbor of x. Otherwise, suppose y is the nearest neighbor of x, and y is not one of p. But in the WSPD, (x,y) will belong to one of the pair set (P_i,Q_i), which correspond to a pair (p',q'). Then there is a contradiction, as $|xp'| < |xy|$ implies that y is not the nearest neighbour of x. Thus one's nearest neighbor is locally stored at this node already. According to Theorem 37, x will belong to at most $O(s^d \lg \alpha)$ different pair sets. So the nearest neighbor is already locally stored and by searching in worst case $O(s^d \lg \alpha)$ time one can find the nearest neighbor without any communication.

7.6 Conclusion

This chapter aims to explain the emergence of good spanners from the behaviors of agents with their own interests. The results can be immediately applied to the construction of good network overlays by distributed peers with incomplete information. For our future work we would like to explore incentive-based overlay construction [59]. One problem faced in the current P2P system design is to reward peers that contribute to the network maintenance or service quality and punish the peers that try to take free rides [60,61,82,83,137,138]. We would like to extend the results in this chapter and come up with a spanner construction with different quality of service for different peers to achieve fairness — those who build more edges should have a smaller stretch to all other nodes and those who do not build many edges are punished accordingly by making the distances to others slightly longer.

Chapter 8

Opportunistic Processing and Query of Motion Trajectories in Wireless Sensor Networks

8.1 Introduction

The development of wireless sensor networks has enabled the possibility of large-scale and long-term environmental monitoring. One driving application for such technologies is the tracking of targets with embedded sensor nodes in an unobtrusive manner, with the targets being rare animals (e.g., Petrel birds on great duck island [108, 150]) in habitat monitoring, or vehicles/humans in security applications. There has been a lot of prior work on tracking moving targets by distributed sensors (see [80, 94, 143, 157] and references therein). A few tracking systems have also been deployed and evaluated in real testbeds [17, 86, 142].

A closely related problem for target tracking is the design of the interface with which the target trajectories are accessed by the users. In habitat monitoring, the most adopted approach has the sensors that detect a target record the detection event in the data logger or report it to a base station, where the target trajectories are assembled from the individual detections for post-experiment analysis. The deployment of such a monitoring network is often one-time experiment with the goal of collecting a sufficient amount of scientific data.

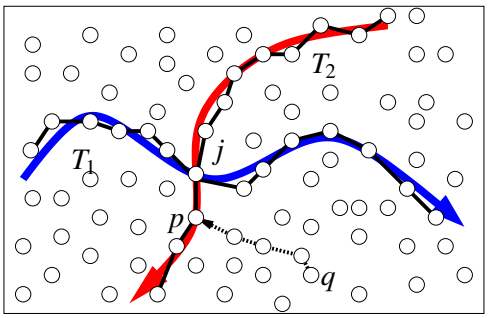


Figure 38. Opportunistic information dissemination: sensors waken up by a moving target will record this detected event and help to disseminate information about other trajectories they have learned so far to the descending sensor nodes. In this case, target T_2 enters the sensor field after target T_1. The nodes in the trajectory T_2 after the junction node j will learn the information of both T_1 and T_2. A query message from q that visits one such node (p) is able to discover T_1 as well.

In this chapter we are mainly motivated by the deployment of sensor nodes for long-term monitoring in a civil environment with the goal of creating an intelligent environment to support real-time sensing, situation understanding and knowledge extraction. The detected data needs to be processed in a timely manner, and accessed by users residing in the same physical space, sometimes with stringent delay requirements. The approach of reporting detection events to a base station in this case has not fully explored the potential of collaborative information processing enabled by a large number of networked sensor nodes. Further, storing all the data at a central server has the problem that the central server represents a single point of failure, and the communication cost for trajectory reporting for each individual detection can be high when the number of moving targets is large.

8.1.1 Our approach

In this chapter we explore the potential of an orthogonal approach and employ a low-cost processing and storage scheme for target trajectories in the network, as well as an in-network query algorithm to retrieve the detected trajectories. As a target moves, nearby sensors are triggered to record the signal signature (e.g., visual or acoustic signatures) and the time stamp when the target was detected. Users live in the same physical space and may inject queries to the network searching for the existence of a particular target and its recent moving trails:

- *Existence query:* was there a red-color truck moving in the field in the past half an hour?
- *Trajectory report query:* Report the trajectory of the red-color truck if it ever appeared in the past half an hour.

The problem we study in this chapter involves how to store and process the moving trajectories and how to answer queries efficiently with low delay for these moving targets.

We exploit two naturally available opportunities to help with efficient in-network trajectory query: (i) the continuity of the motion trajectory suggests that the detection of one sensor on the trajectory allows us to locally follow the trajectory and discover the entire path; (ii) the hand-over of nearby sensors that detect the same target provides communication opportunities to exchange their respective knowledge of previously detected targets. Thus, as a motion trajectory for a given target 'ages', i.e., as more motion trajectories are detected afterwards, this target trajectory becomes easier to find as its information is possibly widely disseminated along the trajectories that newly develop. See the example in Figure 38. The trajectory T_2 arrives later than trajectory T_1. As a sensor near T_2 is waken up anyway to record the signal signature of T_2, it also communicates with the nearby precedent and descent sensors for trajectory hand-over, and exchanges the signal signatures of target trajectories each has learned so far. In particular, after the junction node j located near both trajectories is triggered by T_2, the information j knows (in this case both T_1 and T_2) will be carried along the nodes near trajectory T_2 afterwards. When the trajectory T_2 exits the region of interest, we may also disseminate along the trajectory T_2 backward the information accumulated all time along to aid trajectory query with only a multiplicative factor 2 on the total communication cost. We call this *opportunistic information dissemination*.

From a sensor node's point of view, its task on information storage and processing is extremely simple. When it is waken up by a target in its neighborhood it will be activated to record the information of this particular target, as well as the information carried over by the precedent sensor node along this trajectory. As time goes by the information a target knows will accumulate. As trajectories that are too old are becoming increasingly not interesting in many applications, we can also associate an expiration time for each target trajectory. Trajectories that are too

old will be removed from the storage space of a node and a node only keep the K newest trajectories sorted by their time stamps.

To query for a target with a given acoustic signature, we again exploit the spatial and temporal coherence of the motion trajectories. The query algorithm is probabilistic and initiates query messages, each traveling along a random direction from the query node (similar to rumor routing [28] and implemented by greedy geographical routing, e.g., in [99, 116]), until one of them has discovered a node with information about the given target trajectory, or when the number of query trials exceeds a threshold, the trajectory in question is either not existing, too short or too new. In general, the number of query messages issued depends on the length of the trajectory to be searched (a trajectory that cuts corners is unlikely to be discovered by a random query path), and its age (defined as the number of trajectories that enter the sensor field afterwards; as more trajectories develop, the chance to be discovered increases).

To summarize, both the target detection and target query are locally processed in the network and do not assume a central server. Unlike the approach of gathering trajectory knowledge at a central server in which the communication cost increases with the number of targets, here more target trajectories will provide more opportunities and reduce the cost for trajectory queries. This makes the in-network processing and query scheme attractive in the scenarios with 'heavy traffic' and those when real-time access of the trajectories are desirable.

In this chapter we analyzed the probability that a random query discovers a given trajectory, with respect to its length and age (under opportunistic information dissemination). We also evaluated the performance of the system, in terms of the query success rate and communication cost, as well as storage requirement. The results show the effectiveness of opportunistic information dissemination. A trajectory with a reasonable length/appearance can be discovered on average with only a couple of queries as long as there have been a few trajectories that helped to disseminate it.

8.1.2 Related work

Data storage and retrieval in a distributed sensor network is a major architectural component and has attracted a lot of interest in recent years. Existing data discovery approaches can be classified as data-centric routing [9, 88, 90, 107] and data-centric storage [129]. In data-centric routing, limited preprocessing is undertaken and the query actively searches in the network for relevant data. Data is aggregated along the routing paths back to the user to reduce communication cost [40, 117, 141]. In data-centric storage, the discovered data is preprocessed and stored in the distributed sensors. Partial aggregates can also be evaluated to facilitate complex multi-dimensional queries [65, 67, 76, 135].

Our scheme belongs to the data-centric storage category. The major difference in this chapter is to exploit the spatial structure in the data itself – for both efficiently querying the trajectory and for helping to disseminate knowledge around without incurring additional communication costs.

The query algorithm is in spirit similar with rumor routing [28], and double rulings schemes [57,105,134,149]. In double rulings scheme, a data source (i.e., the sensor node with detected information) will take some specially designed routing paths to disseminate the data availability information. A query from a node will take another special routing path searching for the available data indices. In our setting information about the data source is naturally carried by the target itself and thus the dissemination stage is simple and implicit. The rigorous analysis of how information is disseminated in the network will also apply to rumor routing and supplement the experimental study in [28].

Remotely related to the work here, the idea of exploiting existing communication opportunities also appears in other settings such as opportunistic routing to improve network throughput [25].

In the remaining of the chapter we first present the analytical results in calculating the query success probability under opportunistic information dissemination. The analysis is complemented by simulations in section 8.3.

8.2 Trajectory Queries

We have a sensor field in which the sensors are used to track moving targets. Each target possibly enters and exits anywhere in the field. A sensor near the motion trajectory is waken up, say, by a high acoustic signal in its vicinity. These sensors then record a signal signature specifying the characteristics of the target detected. Naturally all the sensors near or on the trajectories detect the target and store the target signature. By movement continuity, the sensors are laid out along a curve connecting the entrance and exit of the target in the sensor field. We explore schemes on how these trajectories are stored and pre-processed and how a query node can efficiently extract a target trajectory given its signature.

8.2.1 Probabilistic queries

We first analyze in-network query without opportunistic information dissemination. When there is no preprocessing, only the sensor nodes on the trajectory have knowledge about the target. Thus the problem is to figure out one of these sensor nodes (for existence query), and possibly follow the trajectory (for trajectory report query). We explore a probabilistic query scheme in which the query node sends a query message along a random line, hoping that this message will hit one node on the trajectory of interest. In particular, for a line ℓ through the query point q, two query messages are sent out traveling in opposite directions, one along each ray from q. A query message will return when it hits a node on the trajectory or when it hits the sensor network boundary, in which case the query returns and does not discover the trajectory.

The query node can send out multiple such random queries to increase the chance that one of them discovers the data. The probability that a random query finds out the given trajectory, as well as the average number of query messages needed to eventually discover it with a sufficiently high probability, depend greatly on the specific shape of the trajectory. If a trajectory cuts the corner of the sensor field, i.e., the exit node is very close to the entrance node on the sensor field boundary, it is more difficult to discover it.

In the following analysis we start with a simple case when the target moves along a straight line and always enters and exits the sensor field at some boundary

nodes, i.e., we assume no target appears or disappears inside the sensor field. Then we move on to the case of an arbitrary trajectory.

8.2.1.1 Query on straight-line trajectories

We analyze the cost of a query initiated at a node distributed uniformly randomly in the sensor field. This scenario captures the *average-case* query cost for all possible positions.

A query message travels along a random direction. Since the analysis is all probabilistic, we will need to define rigorously what we mean by a 'random line' [146]. Here we will analyze three possible ways to generate a random query. In the first definition, a query travels along a line connecting the query node q with a random point w on the sensor field boundary, with w uniformly distributed along the boundary. We call this model the *random boundary point model*. In the second definition, a query travels along a line with angle θ counter-clockwise from the positive x-axis, where θ is taken uniformly randomly from 0 to 2π. This model is called the *random angle model*. In the third definition, a query travels along a random chord, defined as the chord connecting two random boundary nodes. This is called the *random chord model*.

Of course, different ways to define a random line will lead to different probability measures but we will show below that they differ by only a constant factor. In our algorithm we implemented the random angle model as it only uses local knowledge. When the sensor nodes know their locations, or have a local compass, the query can be routed with only local knowledge, in the same way as compass routing [99]. In the analytical result for the opportunistic information dissemination (subsection 8.2.2) we use the random chord model as that makes the analysis easier.

In what follows we assume the sensors are deployed uniformly randomly in a disk. We define the *effective length* of a trajectory by the distance between the entrance and exit nodes along the sensor field boundary. Without loss of generality we can assume the total length of the sensor field boundary is 1. We assume the length of the trajectory between 0 and $\frac{1}{2}$, due to symmetry. These assumptions will be relaxed later for a generic probabilistic analysis in section 8.2.1.2.

Now we will analyze the probability that a random line intersects with the trajectory and the total query cost in order to guarantee that the trajectory is discovered with probability at least $1-\varepsilon$, by using the above two different ways of generating a random query.

Random boundary point model. Given a motion trajectory of length ℓ, suppose that the query starts from a given query node q. A random query is selected to follow the line through q and a random point on the sensor network boundary. We now bound the probability that this query finds the desired trajectory.

Lemma 39. *The probability of a random query (in the random boundary point model) intersecting the given motion trajectory with effective length ℓ is $p_1(\ell) = 2\ell - 2\ell^2 + (\ell - 1/2)\frac{\sin 2\pi\ell}{\pi}$.*

Proof. As Figure 39 (i) shows, if the query point locates in area A, then the probability that it generates a query that hits the given trajectory is $1-\ell$, otherwise the probability is ℓ. The probability that the query point locates in area A is $(\pi r^2 \cdot \frac{2\pi\ell}{2\pi} - \frac{1}{2} \cdot 2r\sin\pi\ell \cdot r\cos\pi\ell)/(\pi r^2) = \ell - \frac{\sin 2\pi\ell}{2\pi}$. So the probability that a random query point hits the given trajectory is $(1-\ell)(\ell - \frac{\sin 2\pi\ell}{2\pi}) + \ell(1 - (\ell - \frac{\sin 2\pi\ell}{2\pi})) = 2\ell - 2\ell^2 + \frac{\ell \sin 2\pi\ell}{\pi} - \frac{\sin 2\pi\ell}{2\pi}$. □

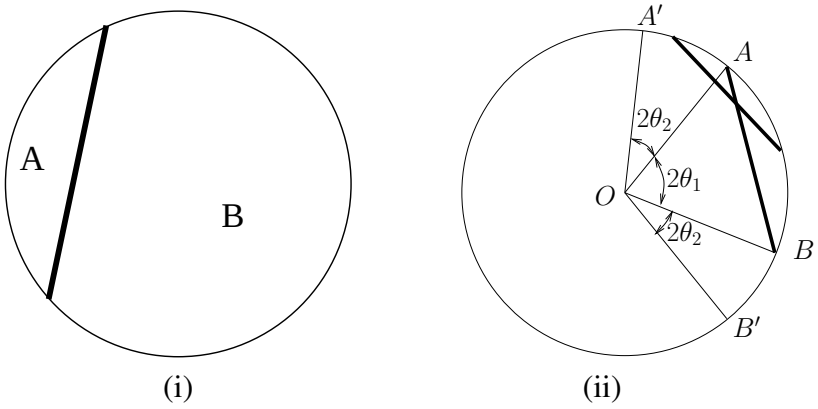


Figure 39. (i) When the query point locates in A or B, it has different probability to send a query intersecting the given trajectory. (ii) Two chords have an intersection.

Theorem 40. *Given a trajectory with length ℓ and a number $\varepsilon \in (0,1)$, with $K_1(\ell,\varepsilon) \geq \frac{1}{\ell}\ln 1/\varepsilon$ query lines under the random boundary point model, the probability to hit the given trajectory is greater than $1-\varepsilon$.*

Proof. The probability that all the k random query lines do not hit the given trajectory is $(1-p_1(\ell))^k$, where $p_1(\ell)$ is as defined in Lemma 39. So the probability that at least one random query lines hit that trajectory is $1-(1-p_1(\ell))^k$. We need $1-(1-p_1(\ell))^k \geq 1-\varepsilon$, that is $(1-p_1(\ell))^k \leq \varepsilon$, i.e., $(1-(2\ell-2\ell^2+\frac{\ell\sin 2\pi\ell}{\pi}-\frac{\sin 2\pi\ell}{2\pi}))^k \leq \varepsilon$. As $\ell \leq \frac{1}{2}$, $\frac{\ell\sin 2\pi\ell}{\pi}-\frac{\sin 2\pi\ell}{2\pi} = \frac{\sin 2\pi\ell}{\pi}(\ell-\frac{1}{2}) \leq 2\ell(\ell-\frac{1}{2})$.

We want $\varepsilon \geq (1-2\ell+2\ell^2-2\ell(\ell-1/2))^k = (1-\ell)^k$. So $k \geq \frac{\ln 1/\varepsilon}{\ln 1/(1-\ell)}$, as $\ln\frac{1}{1-\ell} > \ell$, then $\frac{\ln 1/\varepsilon}{\ln 1/(1-\ell)} \leq \frac{1}{\ell}\ln\frac{1}{\varepsilon}$. Then $K_1(\ell,\varepsilon) \geq \frac{1}{\ell}\ln\frac{1}{\varepsilon}$ is enough to meet the probability requirement. □

Random angle model. In the second model, the query line is chosen as the line through a given query point with a random angle relative to the positive x-axis. Again, we consider the expected cost with respect to a random query point. We start with a technical lemma.

Recall that the *central angle* of a chord AB with two endpoints A,B on the circle boundary is the angle $\angle AOB$ where O is the center of the circle. The *inscribed angle* is exactly half of the central angle and is in between 0 and $\pi/2$. The following lemma is independent of the definition of a random line.

Lemma 41. *The probability that two random chords with inscribed angles θ_1 ($0 \leq \theta_1 \leq \frac{\pi}{2}$) and θ_2 ($0 \leq \theta_2 \leq \frac{\pi}{2}$) intersect within the circle is Prob$\{$intersection$|\theta_1,\theta_2\} = \frac{2}{\pi}\min(\theta_1,\theta_2)$. Specifically, when one chord (say the one with half angle θ_1) is given, then the probability is Prob$\{$intersection$|\theta_1\} = \frac{2}{\pi}\int_0^{\theta_1}\theta_2 f(\theta_2)d\theta_2 + \frac{2}{\pi}\int_{\theta_1}^{\pi/2}\theta_1 f(\theta_2)d\theta_2$, where $f(\theta)$ is the probability density function of the random variable θ.*

Proof. Without loss of generality, suppose $\theta_1 > \theta_2$. Once chord 1, say AB, has been placed, chord 2 will intersect chord 1 if and only if one endpoint falls inside the arc AB and another endpoint falls inside AA' or BB' (see Figure 39 (ii)). The probability for this to happen is $\frac{2\theta_2}{\pi}$. Thus we have obtained

Prob$\{$intersection$|\theta_1,\theta_2\} = \frac{2}{\pi}\min(\theta_1,\theta_2)$. When chord 1 is given, the probability that a second random chord intersects chord 1 is Prob$\{$intersection$|\theta_1\} = \frac{2}{\pi}\int_0^{\pi/2}\min(\theta_1,\theta_2)f(\theta_2)d\theta_2 = \frac{2}{\pi}\int_0^{\theta_1}\theta_2 f(\theta_2)d\theta_2 + \frac{2}{\pi}\int_{\theta_1}^{\pi/2}\theta_1 f(\theta_2)d\theta_2$, where $f(\theta)$ depends on the probabilistic model we use to define a 'random chord'. □

Now we will consider the case that a random chord is generated by dropping a point at random in the circle and drawing a chord through that point in a randomly chosen direction.

Lemma 42. *Assume that the random point falls at a distance r from the center of the circle, and the random chord is selected by taking a randomly chosen direction through that point. Now the chord intercepts an arc of inscribed angle θ. Under this model, the density function $f(\theta|r) = \frac{2r\sin\theta}{\pi\sqrt{r^2-\cos^2\theta}}$, and $f(\theta) = \frac{4}{\pi}\sin^2\theta$, where $0 \le \theta \le \frac{\pi}{2}$.*

Proof. Refer to [146][Page 139]. □

From Lemma 41 and Lemma 42, we can get the following theorem. The proofs are omitted.

Lemma 43. *For a given trajectory with length ℓ, and a query point randomly placed inside the disk, generate a random query by choosing a direction randomly, then the probability that the query will hit the trajectory is $p_2(\ell) = 2(\pi\theta_1 - \theta_1^2 + \sin^2\theta_1)/\pi^2$, where $\theta_1 = \pi\ell$.*

Proof. The trajectory of length ℓ has an inscribed angle $\theta_1 = \pi\ell$. By Lemma 41, the probability that this trajectory is intersected by a random query is $p_2(\ell)$ and substitute $f(\theta)$ by $\frac{4}{\pi}\sin^2\theta$. Then $p_2(\ell) = \frac{2}{\pi}\int_0^{\theta_1}\theta_2 f(\theta_2)d\theta_2 + \frac{2}{\pi}\int_{\theta_1}^{\pi/2}\theta_1 f(\theta_2)d\theta_2$
$= \frac{2}{\pi}\int_0^{\theta_1}\theta_2\frac{4}{\pi}\sin^2\theta_2 d\theta_2 + \frac{2}{\pi}\int_{\theta_1}^{\pi/2}\theta_1\frac{4}{\pi}\sin^2\theta_2 d\theta_2$
$= 2(\pi\theta_1 - \theta_1^2 + \sin^2\theta_1)/\pi^2$. □

Theorem 44. *Given a trajectory with length ℓ and ε ($0 < \varepsilon < 1$), with $K_2(\ell,\varepsilon) \ge \frac{1}{\ell}\ln 1/\varepsilon$ query lines under the random angle model, the probability to hit the given trajectory is greater than $1-\varepsilon$.*

Proof. Suppose $\theta_1 = \pi\ell$. Similar to Theorem 40, for k queries to find the trajectory T, we need

$$k \geq \frac{\ln(1/\varepsilon)}{\ln(1/(1-2(\pi\theta_1-\theta_1^2+\sin\theta_1^2)/\pi^2))}.$$

The right hand side is at most $\frac{\ln 1/\varepsilon}{2(\pi\theta_1-\theta_1^2+\sin^2\theta_1)/\pi^2} \leq \frac{\ln 1/\varepsilon}{(\pi\theta_1)/\pi^2} = \frac{1}{\ell}\ln 1/\varepsilon$. So $K_2(\ell,\varepsilon) \geq \frac{1}{\ell}\ln 1/\varepsilon$ is enough to meet the probability requirement. □

Random chord model. In the random chord model, the query is taken as a random chord with its two endpoints chosen uniformly random on the network boundary.

Lemma 45. *Given a trajectory with effective length ℓ, a random query under the random chord model will discover it with probability $p_3(\ell) = 2\ell - 2\ell^2$.*

Proof. $p_3(\ell) = 1 - (\ell^2 + (1-\ell)^2) = 2\ell - 2\ell^2$. □

Theorem 46. *Given a trajectory with length ℓ and a number $\varepsilon \in (0,1)$, with $K_3(\ell,\varepsilon) \geq \frac{1}{\ell}\ln 1/\varepsilon$ query lines under the random chord model, the probability to hit the given trajectory is great than $1-\varepsilon$.*

The proof is similar to Theorem 44 and omitted.

8.2.1.2 Query on trajectories of arbitrary shape

In general, a motion trajectory can have any shape. And a target may start (appear) or stop (disappear) anywhere in the sensor field. We can use the same random query to search for such trajectories. The probability that a random query message discovers a target trajectory also depends on its shape and length.

To analyze the cost of such random queries, we will make use of standard geometric probability. A query starts from the query node and follows a random line with angle θ counter-clockwise from the positive x axis (i.e., the random angle model). We define the length of a trajectory as the perimeter of the convex hull of the trajectory. In a degenerate case when the trajectory is a straight line segment, the length is defined as twice its Euclidean length.

Specifically, the trajectory T has length ℓ. A random line G is represented by two parameters r and θ, where r is the distance from the origin to the line, and

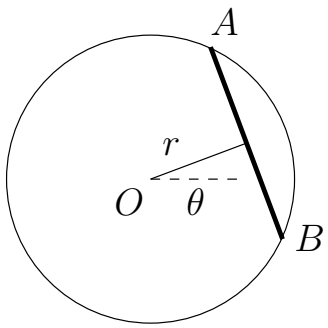


Figure 40. A random line with distance r from the origin and angle θ between its normal and the positive x-axis.

θ is the angle formed by the normal to the line and the x-axis. Both r and θ are uniformly randomly chosen among their respective ranges. As we will see later, that this definition of a random query line also gives a similar bound on the query success probability. See Figure 40.

By geometric probability the collection of lines that intersect a curve is represented precisely by the length of the curve [146].

Lemma 47 ([146]). *The measure of the set of straight lines that intersect a curve C is its length ℓ. That is $\int_{G\bigcap T\neq\emptyset} dG = \int_{G\bigcap T\neq\emptyset} dr\wedge d\theta = \int_0^{2\pi} rd\theta = \ell$.*

Now suppose T_1 is a chord of Euclidean length ℓ_1 inside a region $\mathcal{R}$. Then the chord is a degenerate rectangle with perimeter $2\ell_1$. When $\mathcal{R}$ is a disk with radius R, then $\ell_1 = 2R\sin\pi\ell$, where ℓ is the effective length of the chord.

Lemma 48 ([146]). *The probability that a random line intersecting $\mathcal{R}$ with effective length ℓ intersects with a trajectory T_1 of length ℓ_1, is given by $\frac{\int_{G\bigcap T_1} dG}{\int_{G\bigcap\mathcal{R}} dG} = \frac{2\ell_1}{\ell}$. Especially, when $\mathcal{R}$ is a disk, the probability is $\frac{2\sin\pi\ell}{\pi}$.*

Now we can extend the theorem to the general case.

Theorem 49. *Given a convex boundary field with perimeter L, a trajectory with effective length ℓ and a constant ε ($0<\varepsilon<1$), the number of random query lines to hit the given trajectory with probability great than $1-\varepsilon$ is $\frac{L}{2\ell}\ln\frac{1}{\varepsilon}$.*

The proof is similar to Theorem 44.

Despite various models of random queries, the analysis arrives at very similar results. This will be useful in the next section as we analyze the scheme of using motion trajectories themselves to disseminate information.

8.2.2 Opportunistic information propagation

In many tracking applications the targets being tracked come in the monitored field at different points in time. We can make use of the temporal diversity and perform opportunistic information propagation. To simplify the analysis, we assume that the targets come in the field one by one. When a new target moves in the field, some of the sensors nodes on its trajectory may have already had stored signatures of other targets that have passed by. Thus a sensor node waken up by a new target should take the opportunity to propagate the information it has recorded so far. Naturally as a target enters the field, the node along the trajectory will hand over the knowledge it has learned to the next node on the trajectory. After a target exits the monitored field, a backward propagation may also be conducted such that all the nodes on the trajectory share the union of the signal signatures of them. In the analysis for simplicity we assume the backward propagation. In our simulations we do not conduct backward propagation. By using opportunistic information propagation, a target signature is known not only to nodes along its trajectory but also possibly carried to other nodes by the targets that come in afterwards. The longer a signature exists in the system, the wider it gets propagated in the sensor field. We thus give each motion trajectory an *age* which is the number of targets that come in after itself, but before the query is initiated.

Suppose there are two trajectories A and B that show up in the monitored field sequentially. They have effective lengths ℓ_A and ℓ_B respectively. Say a node sends a random query along the line C. Then there are two possibilities for the query to discover the trajectory A, either because C intersects with A directly or because C intersects with B which intersected with A and helped to propagate information about A.

Here is an interesting tradeoff — the older a trajectory is, the easier to query

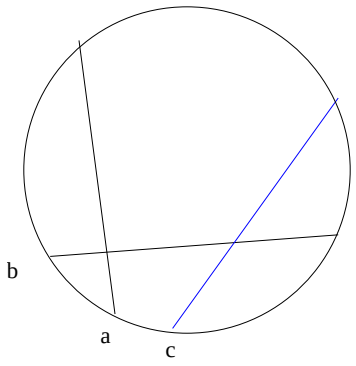


Figure 41. The query line C crosses another trajectory B, and B crosses the trajectory A.

for it; yet a detection long time ago was not going to be interesting from many real-time applications' point of view. The number of trajectories kept at each node is also bounded by the storage requirement. In the following analysis we will evaluate the minimum 'lifetime' of a trajectory, that is, how old a trajectory needs to be, in order to keep the query success rate high. Thus a sensor node would only record the newest k trajectories it has learned, where k is taken as the lifetime above or as application requires.

We remark that as our queries are also probabilistic, we can also take the communication opportunities provided by queries issued in the network to further spread information around. In this case a query is simply regarded as a 'fake' target trajectory and can be treated the same as the other real ones.

8.2.2.1 Technical lemmas

We start by some technical lemmas to be used later. We assume that each trajectory has a random entrance point and a random exit of the sensor field. It can be modeled as the chord connecting two endpoints selected uniformly randomly on the sensor field boundary. Recall that the effective length of a trajectory is defined with respect to its entrance and exit points on the sensor network boundary, which partition the sensor network boundary into two segments, one is longer than the other. The effective length is taken as the normalized length of the shorter segment and is always smaller than $1/2$.

Corollary 50. *Two random trajectories intersect with each other with probability* $2/3$.

Proof. We will integrate over $p_3(\ell)$ for $0 \le \ell \le 1/2$ and obtain $2\int_{\ell=0}^{1/2} p_3(\ell)d\ell = 2/3$. □

Lemma 51. *Given n random chords, the probability that they form a connected network is at least* $1 - O(1/n)$.

Proof. Each random chord connects two random points on the network boundary. Denote by the random endpoints as x_i, y_i. And the collection of these endpoints form a cyclic permutation of $2n$ points on the network boundary. If the trajectories do not form a connected network, one can partition the set of endpoints into 2 groups with $2k$ and $2n-2k$ endpoints each, consecutively distributed along the sensor network boundary, $1 \le k \le n-1$, such that no chord chooses its endpoints in different groups. Now we calculate the probability for this to happen:

$$\begin{aligned} & \text{Prob}\{n \text{ random trajectories are not connected}\} \\ \le\ & f(n) = \textstyle\sum_{k=1}^{n-1} \binom{n}{k} / \binom{2n}{2k} \\ =\ & \frac{1}{2n-1} \cdot [\sum_{k=1}^{n-2} \binom{n-1}{k} / \binom{2n-2}{2k} \cdot (2n-2k-1) + 1] \\ \le\ & \frac{1}{2n-1} \cdot [f(n-1) \cdot n + 1]. \end{aligned}$$

The last inequality uses the symmetry of the series $\binom{n}{k}/\binom{2n}{2k}$ in terms of k and sums up the first element with the last element, the 2nd element with the second last element and so on. We can check that if $f(n-1) \le \frac{c}{2n-3}$, for $c \ge 5$ and $n \ge 4$,

$$\begin{aligned} f(n) &\le \tfrac{1}{2n-1} \cdot [f(n-1) \cdot n + 1] \\ &\le \tfrac{1}{2n-1} \cdot [\tfrac{cn}{2n-3} + 1] \le \tfrac{c}{2n-1}. \end{aligned}$$

Thus with probability at least $1 - O(1/n)$ the random trajectories are connected. □

Corollary 52. *Given n random chords, the probability that at least* $n-h$ *of them form a connected network is at least* $1 - O(1/n^h)$.

Proof. We calculate the probability $q(n,h)$ as the probability that the largest connected component has size at most $n-h$. With a similar argument as the previous lemma, if the largest connected component has size $\le n-h$, we can partition the

endpoints into 2 groups with $2k$ and $2n-2k$ with $h \le k \le n-h$ such that each chord chooses its endpoints within one group only. We have

$$\begin{aligned} q(n,h) &= \text{Prob}\{\#\text{ largest connected component } \le n-h\} \\ &\le f(n,h) = \textstyle\sum_{k=h}^{n-h} \binom{n}{k} / \binom{2n}{2k} \\ &= f(n-1,h) \cdot \frac{n}{2n-1} + \binom{n}{h} / \binom{2n}{2h}. \end{aligned}$$

For large n and a small constant h, we have $\binom{n}{h} \approx \frac{n^h}{h!}$, $\binom{2n}{2h} \approx \frac{n^{2h}}{(2h)!}$. One can then verify that $f(n,h) = O(1/n^h)$. Thus the probability that at least there is a connected network of $n-h$ chords is $1 - O(1/n^h)$.

□

Given a set of n connected random trajectories, define the *separation length* of this set as the maximum distance between the adjacent endpoints on the network boundary, normalized by the total length of the perimeter of the sensor network.

Lemma 53. *Given n connected random trajectories, the separation length is in expectation $\frac{1}{2n}$ and $\frac{c \ln n}{n}$ with probability at least $1 - O(1/n)$, for a constant c.*

Proof. The analysis follows from the standard random sampling techniques [113]. A short remark is that the endpoints are randomly selected on the network boundary. We can ignore the fact that the trajectories are connected because this only determines the combinatorics of how the endpoints are paired up to n chords. □

Lemma 54. *For n random chords that form a connected trajectory network E, any random chord intersects the union of these chords with probability at least $1 - O(\ln^2 n/n)$.*

Proof. Given that the random chords form a connected network, the endpoints of these chords partition the network boundary into $2n$ segments ℓ_i, $1 \le i \le 2n$. $\sum_{i=1}^{2n} \ell_i = 1$. Therefore, the probability that a random chord does not intersect this trajectory network is $1 - \sum_{i=1}^{2n} \ell_i^2$. With probability $1 - O(1/n)$, $\ell_i \le \frac{c \ln n}{n}$ for all i (denoted by event A). Therefore the probability that a random chord will intersect

the trajectory network is

$$\begin{aligned} & p_4(n) = \text{Prob}\{\text{a random chord intersects } E\} \\ \geq\ & \text{Prob}\{\text{a random chord intersects } E|A\}(1-O(1/n)) \\ =\ & (1-\textstyle\sum_{i=1}^{2n}\ell_i^2)\cdot(1-O(1/n)) \\ \geq\ & 1-O(\ln^2 n/n). \end{aligned}$$

□

8.2.2.2 Opportunistic information dissemination

We will now analyze how the trajectories that come after a particular trajectory T can help to disseminate information about T. Notice that now the sequence of the trajectories showing up matters in how information gets propagated in the network. In particular, we define a *dissemination network* of n trajectories such that the i-th trajectory comes after the $(i-1)$-th trajectory and intersects with one of the previous trajectories. We now analyze the probability that a dissemination network of n trajectories is formed and how it helps to disseminate the information about a random trajectory T such that a random query will be able to retrieve T with probability $1-\varepsilon$, for $0<\varepsilon<1$. In this subsection we use the random chord model for a random query.

Consider f trajectories entering the field one by one after a trajectory T. We examine how large f should be in order to form a dissemination network with n trajectories to help disseminate information about T. We will divide the process into epoches such that after epoch 1 we have another trajectory T_1 that intersects T. The union of the trajectories T_1 and T is the dissemination network N_1 after epoch 1. Similarly, after epoch i we have a trajectory T_i that intersects with the current trajectory network N_{i-1} and the trajectory network N_i is updated to include T_i as well. Denote by f_i the number of trajectories in epoch i and take $f=\sum_{i=1}^{n} f_i$ as the total number of trajectories. Now we calculate the expected number of f_i. Recall that a trajectory will intersect a trajectory network of size $i-1$ with probability at least $1-c\ln^2 i/i$, for a constant c, by Lemma 54. Thus the expected number of trajectories in epoch i would be at most $1/(1-c\ln^2 i/i)$. Therefore the expected

total number of trajectories to obtain a network of size n is

$$E(f) = \sum_{i=1}^{n} E(f_i) \leq \sum_{i=1}^{n} 1/(1 - c\ln^2 i/i) = n + o(n).$$

We also calculate the variance of f. As f_i's are independent of each other and each f_i has a geometric distribution of probability at least $1 - c\ln^2 i/i$, then $V(f_i) \leq \frac{c\ln^2 i/i}{(1-c\ln^2 i/i)^2}$.

$$\begin{aligned} & V(f) = \textstyle\sum_{i=1}^{n} V(f_i) \leq \sum_{i=1}^{n} (c\ln^2 i/i)/(1 - 2c\ln^2 i/i) \\ = \;& c\ln^3 n + o(\ln^3 n). \end{aligned}$$

Now we can calculate the probability that f is greater than $(1+\delta)n$, by Chebyshev inequality:

$$\text{Prob}\{f \geq n + \delta n\} \leq 1/(\delta n/V(f))^2 = c^2 \ln^6 n/(\delta^2 n^2).$$

Now we can summarize the main result for opportunistic information dissemination, by combining the analysis above and Lemma 54.

Theorem 55. *Assuming the trajectories are random, a trajectory T with age a can be discovered by a random query with probability $1 - O(\ln^2 a/a)$.*

Proof. By the analysis above, the trajectories after T will form a dissemination network of $a/(1+\delta)$ trajectories with probability at least $1 - c^2\ln^6 a(1+\delta)^2/(\delta^2 a^2)$, for a constant $\delta < 1$. Further, the probability that a query will find out information about T from the dissemination network is at least $1 - O(\ln^2 a/a)$, from Lemma 54. Therefore the probability that a random trajectory with age a will be discovered with probability $1 - O(\ln^2 a/a)$. □

Corollary 56. *Assuming the trajectories are random, the number of queries to discover a random trajectory with age a with probability $1 - \varepsilon$ is $\Omega(\frac{\ln(1/\varepsilon)}{\ln a})$.*

Proof. The probability that k random queries do not discover the trajectory T with age a is $(O(\ln^2 a/a))^k \leq \varepsilon$. Thus $k \geq \Omega(\ln(1/\varepsilon)/\ln a)$. □

During the opportunistic dissemination process, each node has to store all the trajectory information that the trajectory carries. So the storage space will increase

with time. Here we will give an analysis on how long should the information of a trajectory be kept for the purpose of opportunistic dissemination. That is the expiration time for a given trajectory.

For a trajectory with length ℓ, the probability that another random trajectory will intersect with it is $p = 2\ell(1-\ell)$. Suppose its age is a. We want to calculate the proper value of age a, so that the trajectory with length ℓ can be discovered in constant times of queries after age a. The expected number of queries to discover a trajectory with length ℓ after age a is

$$\begin{aligned} N(\ell,a) &= \text{Prob}\{\exists \text{ one trajectory that hits } T\} \cdot c_1 + \\ &\quad \text{Prob}\{\nexists \text{ trajectory that hits } T\} \cdot 1/p \\ &= [1-(1-p)^a] \cdot c_1 + (1-p)^a \cdot 1/p, \end{aligned}$$

for a constant c_1. We want $N(\ell,a)$, that is, $(1-p)^a \cdot 1/p$ to be a constant. That means $(1-p)^a = c \cdot p$, for a constant c. Now we have $a = \frac{\ln 1/(cp)}{\ln 1/(1-p)} \leq \frac{\ln 1/\ell}{2\ell(1-\ell)}$. So for small ℓ, we need $a \approx \frac{c' \ln 1/\ell}{\ell}$, where c' is a constant.

From the above analysis, a trajectory does not need to be kept forever for the purpose of opportunistic dissemination. Each trajectory with (normalized) length ℓ only needs to be kept for an age of $\frac{c' \ln 1/\ell}{\ell}$. In this way, the storage requirement at each node will be reduced significantly, yet still have the trajectories to be discovered with a small number of queries.

8.3 Simulations

Our previous analysis focuses on the asymptotic bound of the number of queries for a given target. The simulation results below give an idea of the number of queries needed in a practical setting, as well as the tradeoff of storage requirement versus the query cost. We evaluate the opportunistic dissemination and in-network query scheme by simulations on random trajectories, and compare it with the scheme without opportunistic dissemination (i.e., only the nodes near the target trajectory record information about it). In particular, we evaluate the following three important measures:

- The number of queries issued to search for a given trajectory with respect to its length and age.

- The total communication cost comparison between the opportunistic dissemination scheme and the scheme without opportunistic dissemination. Note that the cost includes both the preprocessing cost and the query cost.
- The storage requirement at each node.

In the simulation, we generate a network with sensor nodes uniformly randomly distributed in a circular field with radius 25. We use a unit disk graph model with a communication range of 1. The average degree of the sensor network is about 7. The trajectories are randomly generated within the sensor field with the entrance and exit points taken randomly on the sensor field boundary. The nodes within distance 1 from the trajectory are waken up to record and disseminate information about the trajectory. In all our following simulations, we generate 500 random trajectories before starting the query.

Queries are initiated at nodes randomly chosen inside the sensor field. For a particular query, the query messages follows a randomly chosen direction until either it visits some sensor node that contains information about the desired trajectory, in which case the information is returned back to the query node; or it hits the sensor field boundary, in which case the query is not successful and another query is generated. Routing of a query is done by greedy geographical routing.

In the simulation we focus on trajectory existence query that answers whether or not a trajectory with a given ID was present or not. With trajectory report queries, we need to record not simply an ID but also the detailed trajectory information.

8.3.1 Query number v.s. age and length

We examine the relationship of the number of queries to find a trajectory with its age and length. We randomly choose a query point for each trajectory and repeat this process for 100 times. We divide the age and length into small ranges and take the average query number for all the trajectories with the (age, length) combination falling into the same bucket. Here the trajectory length is measured by its Euclidean length, normalized by the network size. The age of a trajectory is the number of trajectories detected after it but before the query is issued. In this experiment we assume each node keeps every trajectory it detects or exchanges from its neighbors during opportunistic dissemination.

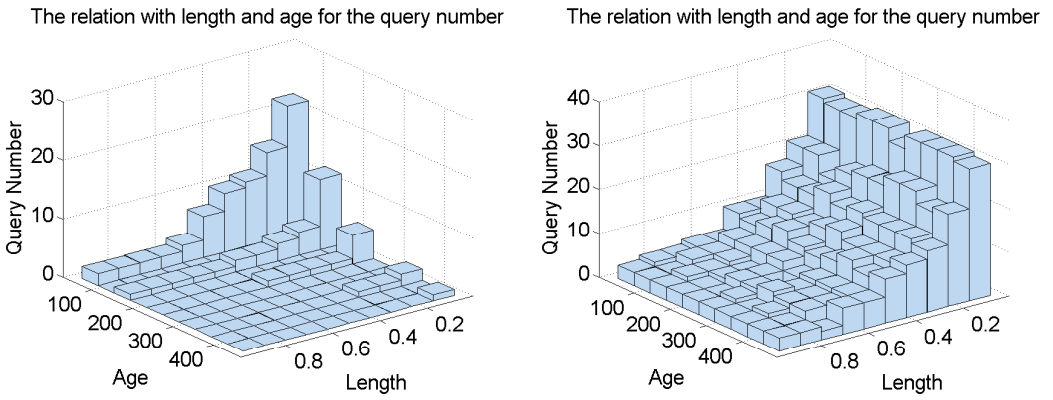


Figure 42. Left: The query number for a trajectory with different lengths and different ages with opportunistic dissemination. Right: The query number for a trajectory with different lengths and different ages without opportunistic dissemination.

From Figure 42, in opportunistic dissemination scheme, the query number will be reduced greatly with the increasing of age or length. When either the age or length is large enough, the query number is close to 1. Without opportunistic dissemination, the query number just relates to the length, and can be much larger than that of the trajectory with similar length in opportunistic dissemination scheme.

8.3.2 Required storage size

In this simulation, we will evaluate the affect of the storage size on the query number. Here, storage size means how many trajectory each node will keep. There is a tradeoff between the preprocess cost and the query cost, which is greatly affected by the storage size.

We have different ways to choose which trajectories will be kept depending on different requirement. For example, if we choose to keep the most frequently queried trajectory, it will help to distribute the popular trajectories, which may dominate the total communication cost and will help to reduce the total communication cost.

In our simulation, we simply choose the most recent k trajectories, where k is the storage size. But each node will always keep the trajectories that it detects.

From Figure 43, and the left Figure 42, we can see the affect of the storage size on the query number. With an increasing storage size, the query number for

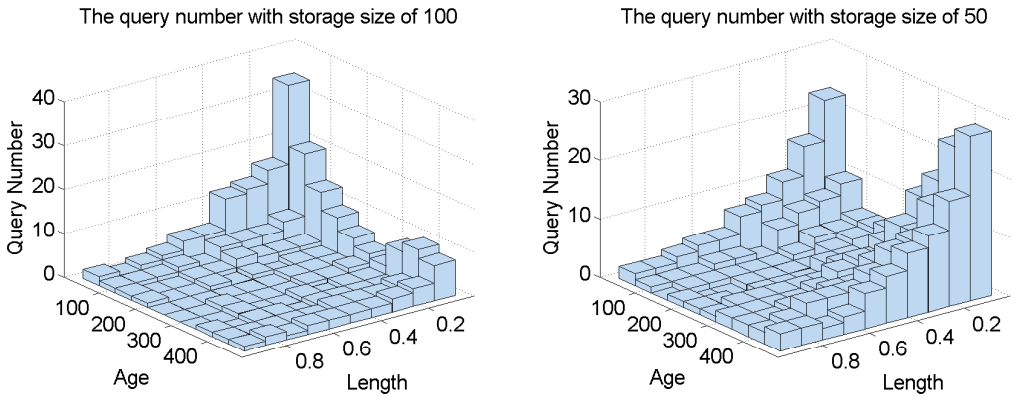


Figure 43. Left: The query number for a trajectory with different lengths and different ages with opportunistic when the storage size is 100. Right: The query number for a trajectory with different lengths and different ages with opportunistic when the storage size is 50.

most trajectories will be reduced. When the storage size decreases, old trajectories will be removed from some nodes, so cost for the old enough trajectories will be increased greatly.

The best storage size chosen in the scheme will depend on users' requirement. In our simulation, 100 (or around 100) seems to be a reasonable storage size for queries of the past 500 trajectories. As we are typically interested in querying recent trajectories, so a small storage size will work out fine. If we hope to query old trajectories, then we need larger storage size. The best storage size will vary significantly as the topologies and density of sensor nodes change. In practice one can use an adaptive algorithm to gradually reduce the storage size while still meet the delay requirement. This remains as future work.

8.3.3 Communication cost

We compare the communication cost in the situations with and without opportunistic dissemination. If we just consider the query cost, obviously it is better to adopt the opportunistic dissemination scheme. However, we have to pay additional cost for opportunistic dissemination, which is called the preprocessing cost. There is no preprocessing cost without the opportunistic dissemination, but the query cost is higher. So we will evaluate the the overall communication cost including both the preprocessing cost and the query cost for a certain query frequency.

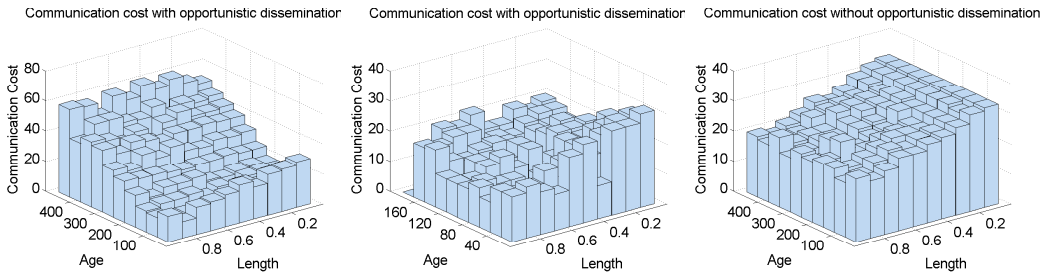


Figure 44. Left: The average communication cost by querying each trajectory 100 times under opportunistic dissemination. Middle: The average communication cost by querying each trajectory(with age less than 200) 100 times under opportunistic dissemination. Right: The average communication cost by querying each trajectory 100 times with no opportunistic dissemination.

We use the summed packet size to evaluate the communication cost. We adopt 36 bytes as the overhead for packet header from TinyOS. We assume 1 byte cost for each trajectory ID stored at the sensor node during the information exchange. For convenience, we consider the communication cost as the packet size divided by 36. Under this assumption, each packet adds 1 to the communication cost without opportunistic dissemination. With opportunistic dissemination, the packet may contain k prior trajectories and we divide the cost equally among the k trajectories. This is the additional cost that each trajectory has to pay to have its information disseminated. In the query stage, each packet contributes 1 to the cost for the given query trajectory. So the total communication cost is the sum of the preprocess cost and the total query cost (query frequency times the cost per query). From previous section, we know that storage size of 100 works well for our simulation. So in our following simulation, we set the storage size to be 100.

Figure 44 shows the results. The oldest trajectories have high communication cost because they were disseminated with a lot of preprocessing cost but were not queried on time before such information is flushed out by new trajectories. For the trajectories with middle-ranged ages, the communication cost is much smaller in opportunistic scheme than that without opportunistic dissemination. Intuitively, the benefit of dissemination (i.e., lower communication cost) shows up with increasing query frequency. When the query frequency is higher than 200, the total communication cost is always smaller for opportunistic scheme. If we just consider the most recent 200 trajectories as in the middle Figure 44, after 100 queries, the average

communication cost has already been less than that without opportunistic dissemination. While without opportunistic dissemination, the average cost will keep the same no matter how many queries are issued for a given trajectory.

8.4 Conclusion

We have presented in this chapter the exploitation of the spatial and temporal coherence of motion trajectories in the domain of in-network storage and query. As sensors have scarce resources, the use of existing opportunities in the detection itself in managing and query for the detection data, is expected to be useful in a more general domain of data management in wireless sensor networks.

In this chapter, we mainly address the discovery of existence of a certain target. In real life some queries are aggregate queries. For example, how many trucks have passed in the past hour. The same framework can be used to answer aggregate queries in a relatively naive way – use a random query to return all the trajectories detected and compute the aggregate. Advanced mechanism for answering aggregate queries of the motion data remains as future work. Our current algorithm and analysis is based on the assumption that the trajectories start and end randomly. We leave it as future work to extend the analysis to the setting when the entry and exit points are clustered in a small region.

Chapter 9

Conclusion

In this thesis, we have presented light-weighted and decentralized algorithms for all kinds of information processing, optimization, coordination of both static and mobile nodes to comprehend and act on the environment. The major motivation behind this is the significant battery and computational limitations of the current generation of wireless nodes and large scale of wireless sensor networks. With the recent advance and maturity of static wireless sensor networks for environment monitoring and prosperity of seamless integrating physical users and/or autonomous robots into a hybrid intelligent, we look for more intelligent and elegant way to solve these complicated problems.

Due to the limited resource restriction, it is not a smart way for each node to keep the entire topology and connection information in order to communicate with remote nodes. We proposed a decentralized hierarchically well-separated trees (HST) framework for wireless sensor networks, which makes a lot of problems easier to solve. This structure sparsely represents the entire network and provide overview on the networks with different granularity. It could serve as a backbone for information brokage purpose when we need to find some remote partners. We also provide a way to make almost guaranteed greedy routing efficiently under dynamic wireless sensor networks, which is based only on local and destination information.

Wireless network deployments continue to increase in commercial and military setting environment. How to intelligently coordinate mobile and static nodes, handle mobile users and dynamic topologies continues to be a challenge task. There

are a lot of new and exciting algorithmic problems and application motivations behind this. Our algorithms and solutions provide an initial exploration towards this direction. We look forward to applying and extending the light-weighted and decentralized mechanisms presented here to more mobility related problems.

Bibliography

[1] http://www.mathworks.com/matlabcentral/fileexchange/6543/.

[2] http://www.aqualab.cs.northwestern.edu/projects/Ono.html.

[3] Orbit testbed. http://www.orbit-lab.org/.

[4] Some results on greedy embeddings in metric spaces. In *Proc. of the 49th Annual Symposium on Foundations of Computer Science*, pages 337–346, October 2008.

[5] I. Abraham, D. Dolev, and D. Malkhi. LLS: a locality aware location service for mobile ad hoc networks. In *DIALM-POMC '04: Proceedings of the 2004 joint workshop on Foundations of mobile computing*, pages 75–84, 2004.

[6] I. Abraham, C. Gavoille, A. V. Goldberg, and D. Malkhi. Routing in networks with low doubling dimension. In *Proc. of the 26th International Conference on Distributed Computing Systems (ICDCS)*, July 2006.

[7] I. Abraham and D. Malkhi. Compact routing on euclidian metrics. In *PODC '04: Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 141–149, New York, NY, USA, 2004. ACM.

[8] I. Abraham and D. Malkhi. Name independent routing for growth bounded networks. In *SPAA '05: Proceedings of the seventeenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 49–55, 2005.

[9] W. Adjie-Winoto, E. Schwartz, and H. Balakrishnan. The design and implementation of an intentional naming system. In *SOSP '01: Proceedings*

of the eighteenth ACM symposium on Operating systems principles, pages 186–201, December 1999.

[10] P. K. Agarwal, D. Eppstein, L. J. Guibas, and M. R. Henzinger. Parametric and kinetic minimum spanning trees. In *FOCS '98: Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, page 596, Washington, DC, USA, 1998. IEEE Computer Society.

[11] S. Albers, S. Eilts, E. Even-Dar, Y. Mansour, and L. Roditty. On nash equilibria for a network creation game. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 89–98, New York, NY, USA, 2006. ACM.

[12] S. Alstrup, C. Gavoille, H. Kaplan, and T. Rauhe. Nearest common ancestors: a survey and a new distributed algorithm. In *SPAA '02: Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures*, pages 258–264, 2002.

[13] P. Angelini, F. Frati, and L. Grilli. An algorithm to construct greedy drawings of triangulations. In *Proc. of the 16th International Symposium on Graph Drawing*, pages 26–37, 2008.

[14] S. Arya, G. Das, D. M. Mount, J. S. Salowe, and M. Smid. Euclidean spanners: short, thin, and lanky. In *Proc. 27th ACM Symposium on Theory Computing*, pages 489–498, 1995.

[15] S. Arya, D. M. Mount, and M. Smid. Randomized and deterministic algorithms for geometric spanners of small diameter. In *Proc. 35th IEEE Symposium on Foundations of Computer Science*, pages 703–712, 1994.

[16] S. Arya and M. Smid. Efficient construction of a bounded-degree spanner with low weight. *Algorithmica*, 17:33–54, 1997.

[17] J. Aslam, Z. Butler, F. Constantin, V. Crespi, G. Cybenko, and D. Rus. Tracking a moving object with a binary sensor network. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 150–161, 2003.

[18] A. Atlas and A. Zinin. Basic specification for ip fast reroute: Loop-free alternates. September 2008.

[19] B. Awerbuch and D. Peleg. Concurrent online tracking of mobile users. In *SIGCOMM '91: Proceedings of the conference on Communications architecture & protocols*, pages 221–233, New York, NY, USA, 1991. ACM.

[20] N. Bansal, N. Buchbinder, A. Gupta, and J. S. Naor. An $O(\log^2 k)$-competitive algorithm for metric bipartite matching. In *Proceedings of the 15th Annual European Symposium (ESA'07)*, pages 522–533, 2007.

[21] Y. Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *FOCS '96: Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, page 184, Washington, DC, USA, 1996. IEEE Computer Society.

[22] Y. Bartal. On approximating arbitrary metrices by tree metrics. In *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 161–168, New York, NY, USA, 1998. ACM.

[23] Y. Bartal and A. Rosen. The distributed k-server problem – a competitive distributed translator for k-server problems. In *Proceeding of IEEE Symposium on Foundations of Computer Science*, pages 344–353, 1992.

[24] J. Basch, L. J. Guibas, and L. Zhang. Proximity problems on moving points. In *SCG '97: Proceedings of the thirteenth annual symposium on Computational geometry*, pages 344–351, New York, NY, USA, 1997. ACM.

[25] S. Biswas and R. Morris. Opportunistic routing in multi-hop wireless networks. *SIGCOMM Comput. Commun. Rev.*, 34(1):69–74, 2004.

[26] A. Borodin and R. El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 2005.

[27] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. *Wireless Networks*, 7(6):609–616, 2001.

[28] D. Braginsky and D. Estrin. Rumor routing algorithm for sensor networks. In *Proc. of the 1st ACM Int'l Workshop on Wireless Sensor Networks and Applications (WSNA)*, pages 22–31, September 2002.

[29] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proc. of the ACM/IEEE International Conference on Mobile Computing and Networking*, pages 85–97, 1998.

[30] J. C. Butcher. *Numerical Methods for Ordinary Differential Equations*. New York: John Wiley & Sons, 2002.

[31] Callahan and Kosaraju. Faster algorithms for some geometric graph problems in higher dimensions. In *Proc. 4th ACM-SIAM Symposium on Discrete Algorithms*, pages 291–300, 1993.

[32] P. B. Callahan. Optimal parallel all-nearest-neighbors using the well-separated pair decomposition. In *Proc. 34th IEEE Symposium on Foundations of Computer Science*, pages 332–340, 1993.

[33] P. B. Callahan and S. R. Kosaraju. Algorithms for dynamic closest-pair and n-body potential fields. In *Proc. 6th ACM-SIAM Symposium on Discrete Algorithms*, pages 263–272, 1995.

[34] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *J. ACM*, 42:67–90, 1995.

[35] H. T.-H. Chan, A. Gupta, B. M. Maggs, and S. Zhou. On hierarchical routing in doubling metrics. In *SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 762–771, 2005.

[36] B. Chandra, G. Das, G. Narasimhan, and J. Soares. New sparseness results on graph spanners. *Internat. J. Comput. Geom. Appl.*, 5:125–144, 1995.

[37] B. Chow. The ricci flow on the 2-sphere. *J. Differential Geom.*, 33(2):325–334, 1991.

[38] B. Chow and F. Luo. Combinatorial ricci flows on surfaces. *Journal Differential Geometry*, 63(1):97–129, 2003.

[39] Y. Chu, S. Rao, S. Seshan, and H. Zhang. Enabling conferencing applications on the internet using an overlay muilticast architecture. *SIGCOMM Comput. Commun. Rev.*, 31(4):55–67, 2001.

[40] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate aggregation techniques for sensor databases. In *ICDE '04: Proceedings of the 20th International Conference on Data Engineering*, pages 449–460, 2004.

[41] J. Corbo and D. Parkes. The price of selfish behavior in bilateral network formation. In *PODC '05: Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, pages 99–107, New York, NY, USA, 2005. ACM.

[42] A. Coté, A. Meyerson, and L. Poplawski. Randomized k-server on hierarchical binary trees. In *STOC '08: Proceedings of the 40th annual ACM symposium on Theory of computing*, pages 227–234, New York, NY, USA, 2008. ACM.

[43] L. J. Cowen. Compact routing with minimum stretch. In *SODA '99: Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 255–260, 1999.

[44] G. Das, P. Heffernan, and G. Narasimhan. Optimally sparse spanners in 3-dimensional Euclidean space. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 53–62, 1993.

[45] G. Das and G. Narasimhan. A fast algorithm for constructing sparse Euclidean spanners. *Internat. J. Comput. Geom. Appl.*, 7:297–315, 1997.

[46] G. Das, G. Narasimhan, and J. Salowe. A new way to weigh malnourished Euclidean graphs. In *Proc. 6th ACM-SIAM Sympos. Discrete Algorithms*, pages 215–222, 1995.

[47] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 1997.

[48] T. DeLillo and E. Kropf. Numerical computation of the schwarz-christoffel transformation for multiply connected domains. *SIAM J. on Scientific Computing*, 33:1369–1394, 2011.

[49] R. Dhandapani. Greedy drawings of triangulations. In *SODA '08: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 102–111, 2008.

[50] T. A. Driscoll and L. N. Trefethen. *Schwarz-Christoffel Mapping*, volume 8. Cambridge University Press, 2002.

[51] T. Eilam, C. Gavoille, and D. Peleg. Compact routing schemes with low stretch factor. *J. Algorithms*, 46(2):97–114, 2003.

[52] D. Eppstein. Spanning trees and spanners. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 425–461. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[53] D. Eppstein and M. T. Goodrich. Succinct greedy graph drawing in the hyperbolic plane. In *Proc. of the 16th International Symposium on Graph Drawing*, pages 14–25, 2008.

[54] J. Erickson. Dense point sets have sparse Delaunay triangulations. In *Proc. 13th ACM-SIAM Symposium on Discrete Algorithms*, pages 125–134, 2002.

[55] A. Fabrikant, A. Luthra, E. Maneva, C. H. Papadimitriou, and S. Shenker. On a network creation game. In *PODC '03: Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 347–351, 2003.

[56] J. Fakcharoenphol, S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *STOC '03: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 448–455, New York, NY, USA, 2003. ACM.

[57] Q. Fang, J. Gao, and L. J. Guibas. Landmark-based information storage and retrieval in sensor networks. In *The 25th Conference of the IEEE Communication Society (INFOCOM'06)*, pages 1–12, April 2006.

[58] Q. Fang, J. Li, L. Guiba, and F. Zhao. Roamhba: maintaining group connectivity in sensor networks. In *IPSN '04: Proceedings of the 3rd international symposium on Information processing in sensor networks*, pages 151–160, New York, NY, USA, 2004. ACM.

[59] J. Feigenbaum and S. Shenker. Distributed algorithmic mechanism design: recent results and future directions. In *DIALM '02: Proceedings of the 6th international workshop on Discrete algorithms and methods for mobile computing and communications*, pages 1–13, New York, NY, USA, 2002. ACM.

[60] M. Feldman and J. Chuang. Overcoming free-riding behavior in peer-to-peer systems. *SIGecom Exch.*, 5(4):41–50, 2005.

[61] M. Feldman, K. Lai, I. Stoica, and J. Chuang. Robust incentive techniques for peer-to-peer networks. In *EC '04: Proceedings of the 5th ACM conference on Electronic commerce*, pages 102–111, New York, NY, USA, 2004. ACM.

[62] R. Flury, S. Pemmaraju, and R. Wattenhofer. Greedy routing with bounded stretch. In *Proc. of the 28th Annual IEEE Conference on Computer Communications (INFOCOM)*, April 2009.

[63] R. Flury and R. Wattenhofer. MLS: an efficient location service for mobile ad hoc networks. In *MobiHoc '06: Proceedings of the seventh ACM international symposium on Mobile ad hoc networking and computing*, pages 226–237, 2006.

[64] S. Funke, L. J. Guibas, A. Nguyen, and Y. Wang. Distance-sensitive information brokerage in sensor networks. In *Proceedings of the second IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 234–251, 2006.

[65] D. Ganesan, D. Estrin, and J. Heidemann. DIMENSIONS: Why do we need a new data handling architecture for sensor networks. In *Proc. ACM SIGCOMM Workshop on Hot Topics in Networks*, pages 143–148, 2002.

[66] 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. Technical Report UCLA/CSD-TR 02-0013, UCLA, 2002.

[67] J. Gao, L. Guibas, J. Hershberger, and L. Zhang. Fractionally cascaded information in a sensor network. In *Proc. of the 3rd International Symposium on Information Processing in Sensor Networks (IPSN'04)*, pages 311–319, April 2004.

[68] J. Gao, L. Guibas, and A. Nguyen. Deformable spanners and their applications. *Computational Geometry: Theory and Applications*, 35(1-2):2–19, 2006.

[69] J. Gao, L. J. Guibas, J. Hershberger, and N. Milosavljević. Sparse data aggregation in sensor networks. In *Proc. of the International Conference on Information Processing in Sensor Networks (IPSN'07)*, pages 430–439, April 2007.

[70] J. Gao, L. J. Guibas, N. Milosavljevic, and D. Zhou. Distributed resource management and matching in sensor networks. In *Proc. of the 8th International Symposium on Information Processing in Sensor Networks (IPSN'09)*, April 2009.

[71] J. Gao and L. Zhang. Tradeoffs between stretch factor and load balancing ratio in routing on growth restricted graphs. *IEEE Transactions on Parallel and Distributed Computing*, 20(2):171–179, February 2009.

[72] J. Gao and D. Zhou. The emergence of sparse spanners and greedy well-separated pair decomposition. In *Proc. of the the 12th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT'10)*, pages 50–61, June 2010.

[73] M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24(2):296–317, 1995.

[74] L.-A. Gottlieb and L. Roditty. Improved algorithms for fully dynamic geometric spanners and geometric routing. In *SODA '08: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 591–600, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.

[75] N. Goyal, L. Rademacher, and S. Vempala. Expanders via random spanning trees. In *SODA '09: Proceedings of the Nineteenth Annual ACM -SIAM Symposium on Discrete Algorithms*, pages 576–585, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.

[76] B. Greenstein, D. Estrin, R. Govindan, S. Ratnasamy, and S. Shenker. DIFS: A distributed index for features in sensor networks. In *Proceedings of First IEEE International Workshop on Sensor Network Protocols and Applications*, pages 163–173, Anchorage, Alaska, May 2003.

[77] M. Grossglauser and D. Tse. Mobility increases the capacity of adhoc wireless networks. *IEEE/ACM Transactions on Networking*, 10(4):477–486, August 2002.

[78] J. Gudmundsson, C. Levcopoulos, G. Narasimhan, and M. Smid. Approximate distance oracles for geometric graphs. In *Proc. 13th ACM-SIAM Symposium on Discrete Algorithms*, pages 828–837, 2002.

[79] L. Guibas. Kinetic data structures. In D. Mehta and S. Sahni, editors, *Handbook of Data Structures and Applications*, pages 23–1–23–18. Chapman and Hall, CRC, 2004.

[80] L. J. Guibas. Sensing, tracking and reasoning with relations. *IEEE Signal Processing Magazine*, 19(2):73–85, March 2002.

[81] A. Gupta, R. Krauthgamer, and J. R. Lee. Bounded geometries, fractals, and low-distortion embeddings. In *FOCS '03: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, pages 534–543, 2003.

[82] A. Habib and J. Chuang. Incentive mechanism for peer-to-peer media streaming. In *Proc. of the 12th IEEE International Workshop on Quality of Service (IWQoS'04)*, June 2004.

[83] A. Habib and J. Chuang. Service differentiated peer selection: An incentive mechanism for peer-to-peer media streaming. *IEEE Transactions on Multimedia*, 8(3):610–621, June 2006.

[84] R. S. Hamilton. Three manifolds with positive ricci curvature. *Journal of Differential Geometry.*, 17:255–306, 1982.

[85] S. Har-Peled and M. Mendel. Fast construction of nets in low dimensional metrics, and their applications. *SIAM J. Comput.*, 35(5):1148–1184, 2006.

[86] T. He, S. Krishnamurthy, L. Luo, T. Yan, L. Gu, R. Stoleru, G. Zhou, Q. Cao, P. Vicaire, J. A. Stankovic, T. F. Abdelzaher, J. Hui, and B. Krogh. Vigilnet: An integrated sensor network system for energy-efficient surveillance. *ACM Trans. Sen. Netw.*, 2(1):1–38, 2006.

[87] X. He and H. Zhang. On succinct convex greedy drawing of 3-connected plane graphs. In *Proceedings of the ACM-SIAM symposium on Discrete algorithms*, pages 1477–1486, January 2011.

[88] J. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, and D. Ganesan. Building efficient wireless sensor networks with low-level naming. In *Proceedings of the Symposium on Operating Systems Principles*, pages 146–159, October 2001.

[89] D. S. Hochbaum, editor. PWS Publishing Company, Boston, MA, 1997.

[90] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *ACM Conf. on Mobile Computing and Networking (MobiCom)*, pages 56–67, 2000.

[91] T. Jansen and M. Theile. Stability in the self-organized evolution of networks. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 931–938, New York, NY, USA, 2007. ACM.

[92] D. Karger and M. Ruhl. Find nearest neighbors in growth-restricted metrics. In *Proc. ACM Symposium on Theory of Computing*, pages 741–750, 2002.

[93] B. Karp and H. Kung. GPSR: Greedy perimeter stateless routing for wireless networks. In *Proc. of the ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 243–254, 2000.

[94] W. Kim, K. Mechitov, J.-Y. Choi, and S. Ham. On target tracking with binary proximity sensors. In *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, page 40, 2005.

[95] Y.-J. Kim, R. Govindan, B. Karp, and S. Shenker. On the pitfalls of geographic face routing. In *DIALM-POMC '05: Proceedings of the 2005 joint workshop on Foundations of mobile computing*, pages 34–43, 2005.

[96] J. Kleinberg and Éva Tardos. Approximation algorithms for classification problems with pairwise relationships: metric labeling and markov random fields. *J. ACM*, 49(5):616–639, 2002.

[97] R. Kleinberg. Geographic routing using hyperbolic space. In *Proceedings of the 26th Conference of the IEEE Communications Society (INFOCOM'07)*, pages 1902–1909, 2007.

[98] G. Konjevod, A. W. Richa, and D. Xia. Optimal-stretch name-independent compact routing in doubling metrics. In *PODC '06: Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 198–207, 2006.

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

[100] M. Kwon and S. Fahmy. Topology-aware overlay networks for group communication. In *NOSSDAV '02: Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, pages 127–136, New York, NY, USA, 2002. ACM.

[101] K.-W. Kwong, L. Gao, R. Guerin, and Z.-L. Zhang. On the feasibility and efficacy of protection routing in ip networks. In *Proc. of IEEE INFOCOM'10*, March 2010.

[102] C. Levcopoulos, G. Narasimhan, and M. H. M. Smid. Improved algorithms for constructing fault-tolerant spanners. *Algorithmica*, 32(1):144–156, 2002.

[103] J. Li, J. Jannotti, D. Decouto, D. Karger, and R. Morris. A scalable location service for geographic ad-hoc routing. In *Proceedings of 6th ACM/IEEE International Conference on Mobile Computing and Networking*, pages 120–130, 2000.

[104] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15:215–245, 1995.

[105] X. Liu, Q. Huang, and Y. Zhang. Combs, needles, haystacks: balancing push and pull for discovery in large-scale sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 122–133, 2004.

[106] K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *Communications Surveys & Tutorials, IEEE*, pages 72–93, 2005.

[107] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: a tiny aggregation service for ad-hoc sensor networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):131–146, 2002.

[108] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 88–97, New York, NY, USA, 2002. ACM Press.

[109] P. Maymounkov. Greedy embeddings, trees, and euclidean vs. lobachevsky geometry. manuscript, 2006.

[110] A. Meyerson, A. Nanavati, and L. Poplawski. Randomized online algorithms for minimum metric bipartite matching. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 954–959, New York, NY, USA, 2006. ACM.

[111] T. Moscibroda, S. Schmid, and R. Wattenhofer. On the topologies formed by selfish peers. In *PODC '06: Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 133–142, New York, NY, USA, 2006. ACM.

[112] M. Motiwala, M. Elmore, N. Feamster, and S. Vempala. Path splicing. *SIGCOMM Comput. Commun. Rev.*, 38(4):27–38, 2008.

[113] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[114] G. Narasimhan and M. Smid. Approximating the stretch factor of Euclidean graphs. *SIAM J. Comput.*, 30:978–989, 2000.

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

[116] B. Nath and D. Niculescu. Routing on a curve. *SIGCOMM Comput. Commun. Rev.*, 33(1):155–160, 2003.

[117] S. Nath, P. B. Gibbons, S. Seshan, and Z. R. Anderson. Synopsis diffusion for robust aggregation in sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 250–262, 2004.

[118] E. Ng and H. Zhang. Predicting Internet network distance with coordinates-based approaches. In *Proc. IEEE INFOCOM*, pages 170–179, 2002.

[119] Y. Ohara, S. Imahori, and R. V. Meter. Mara: Maximum alternative routing algorithm. In *Proc. IEEE INFOCOM*, 2009.

[120] C. H. Papadimitriou and D. Ratajczak. On a conjecture related to geometric routing. *Theor. Comput. Sci.*, 344(1):3–14, 2005.

[121] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall, Englewood Cliffs, NJ, 1982.

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

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

[124] G. Perelman. The entropy formula for the ricci flow and its geometric applications. Technical Report arXiv.org, November 11 2002.

[125] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proc. ACM Symposium on Parallel Algorithms and Architectures*, pages 311–320, 1997.

[126] P. Raghavan and C. D. Thompson. Provably good routing in graphs: regular arrays. In *Proceedings of the 17th annual ACM Symposium on Theory of Computing*, pages 79–87, 1985.

[127] A. Rao, C. Papadimitriou, S. Shenker, and I. Stoica. Geographic routing without location information. In *Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 96–108, 2003.

[128] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-aware overlay construction and server selection. In *Proceedings of the 21th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'05)*, volume 3, pages 1190–1199, 2002.

[129] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker. GHT: A geographic hash table for data-centric storage in sensornets. In *Proc. 1st ACM Workshop on Wireless Sensor Networks ands Applications*, pages 78–87, 2002.

[130] S. Ray, R. Guérin, K.-W. Kwong, and R. Sofia. Always acyclic distributed path computation. volume 18, pages 307–319, February 2010.

[131] C. Reichert, Y. Glickmann, and T. Magedanz. Two routing algorithms for failure protection in ip networks. In *Proc. ISCC*, 2005.

[132] E. M. Reingold and R. E. Tarjan. On a greedy heuristic for complete matching. *SIAM J. Computing*, 10(4):676–681, November 1981.

[133] R. Sarkar, X. Yin, J. Gao, F. Luo, and X. D. Gu. Greedy routing with guaranteed delivery using ricci flows. In *Proc. of the 8th International Symposium on Information Processing in Sensor Networks (IPSN'09)*, April 2009.

[134] R. Sarkar, X. Zhu, and J. Gao. Double rulings for information brokerage in sensor networks. In *Proc. of the ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 286–297, September 2006.

[135] R. Sarkar, X. Zhu, and J. Gao. Hierarchical spatial gossip for multi-resolution representations in sensor networks. In *Proc. of the International Conference on Information Processing in Sensor Networks (IPSN'07)*, pages 420–429, April 2007.

[136] R. Sarkar, X. Zhu, and J. Gao. Spatial distribution in routing table design for sensor networks. In *Proc. of the 28th Annual IEEE Conference on Computer Communications (INFOCOM'09), mini-conference*, April 2009.

[137] S. Schosser, K. Böhm, R. Schmidt, and B. Vogt. Incentives engineering for structured p2p systems - a feasibility demonstration using economic experiments. In *EC '06: Proceedings of the 7th ACM conference on Electronic commerce*, pages 280–289, New York, NY, USA, 2006. ACM.

[138] S. Schosser, K. Böhm, and B. Vogt. Indirect partner interaction in peer-to-peer networks: stimulating cooperation by means of structure. In *EC '07: Proceedings of the 8th ACM conference on Electronic commerce*, pages 124–133, New York, NY, USA, 2007. ACM.

[139] K. Seada, A. Helmy, and R. Govindan. On the effect of localization errors on geographic face routing in sensor networks. In *IPSN '04: Proceedings of the third international symposium on Information processing in sensor networks*, pages 71–80, 2004.

[140] M. Shand and S. Bryant. Ip fast reroute framework. June 2009.

[141] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri. Medians and beyond: New aggregation techniques for sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 239–249, 2004.

[142] N. Shrivastava, R. M. U. Madhow, and S. Suri. Target tracking with binary proximity sensors: fundamental limits, minimal descriptions, and algorithms. In *SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 251–264, 2006.

[143] J. Singh, U. Madhow, R. Kumar, S. Suri, and R. Cagley. Tracking multiple targets using binary proximity sensors. In *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, pages 529–538, New York, NY, USA, 2007. ACM Press.

[144] A. Slivkins. Distance estimation and object location via rings of neighbors. In *PODC '05: Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, pages 41–50, 2005.

[145] M. Smid. *The Weak Gap Property in Metric Spaces of Bounded Doubling Dimension*, pages 275–289. Efficient Algorithms, Springer-Verlag, Berlin, Heidelberg, 2009.

[146] H. Solomon. *Geometric Probability*. CBMS-NSF Regional Conference Series in Applied Mathematics. Society for Industrial Mathematics, 1987.

[147] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson. Measuring isp topologies with rocketfuel. *IEEE/ACM Trans. Netw.*, 12(1):2–16, 2004.

[148] K. Stephenson. *Introduction To Circle Packing*. Cambridge University Press, 2005.

[149] I. Stojmenovic. A routing strategy and quorum based location update scheme for ad hoc wireless networks. Technical Report TR-99-09, SITE, University of Ottawa, September, 1999.

[150] R. Szewczyk, J. Polastre, A. Mainwaring, and D. Culler. Lessons from a sensor network expedition. In *Proceedings of the First European Workshop on Sensor Networks (EWSN)*, January 2004.

[151] A. S. Tanenbaum. *Computer networks (3rd ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[152] M. Thorup and U. Zwick. Compact routing schemes. In *SPAA '01: Proceedings of the thirteenth annual ACM symposium on Parallel algorithms and architectures*, pages 1–10, 2001.

[153] W. P. Thurston. *Geometry and Topology of Three-Manifolds*. Princeton lecture notes, 1976.

[154] V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, Berlin, Germany, 2001.

[155] W. Wang, C. Jin, and S. Jamin. Network overlay construction under limited end-to-end reachability. In *Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'05)*, volume 3, pages 2124–2134, March 2005.

[156] X. Zhang, Z. Li, and Y. Wang. A distributed topology-aware overlays construction algorithm. In *MG '08: Proceedings of the 15th ACM Mardi Gras conference*, pages 1–6, New York, NY, USA, 2008. ACM.

[157] F. Zhao, J. Shin, and J. Reich. Information-driven dynamic sensor collaboration. *IEEE Signal Processing Magazine*, 19(2):61–72, 2002.

[158] D. Zhou and J. Gao. Maintaining approximate minimum steiner tree and k-center for mobile agents in a sensor network. In *Proc. of the 29th Annual IEEE Conference on Computer Communications (INFOCOM'10)*, March 2010.

[159] G. Zhou, T. He, S. Krishnamurthy, and J. A. Stankovic. Impact of radio irregularity on wireless sensor networks. In *MobiSys '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 125–138, 2004.