Stony Brook University

Integrating Mobile Agents and Distributed Sensors in Wireless Sensor Networks

A Dissertation presented

by

Jiemin Zeng

to

The Graduate School

in Partial Fulfillment of the

Requirements

for the Degree of

Doctor of Philosophy

in

Computer Science

Stony Brook University

May 2016

Stony Brook University

The Graduate School

Jiemin Zeng

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

Joseph Mitchell - Chairperson of Defense
Professor, Department of Applied Mathematics and Statistics

Esther Arkin
Professor, Department of Applied Mathematics and Statistics

Matthew P. Johnson - External Member
Assistant Professor
Department of Mathematics and Computer Science at Lehman College
PhD Program in Computer Science at The CUNY Graduate Center

This dissertation is accepted by the Graduate School

Charles Taber
Dean of the Graduate School

Abstract of the Dissertation

Integrating Mobile Agents and Distributed Sensors in Wireless Sensor Networks

by

Jiemin Zeng

Doctor of Philosophy

in

Computer Science

Stony Brook University

2016

As computers become more ubiquitous in the prominent phenomenon of the Internet of Things, we encounter many unique challenges in the field of wireless sensor networks. A wireless sensor network is a group of small, low powered sensors with wireless capability to communicate with each other. The goal of these sensors, also called nodes, generally is to collect some information about their environment and report it back to a base station. Depending on the nature of the nodes or the environment, they may be placed in a deterministic pattern, deployed randomly, or mobile (i.e. cars on a highway). Due to the varying nature of these types of networks, they need specialized algorithms tailored to their scenarios and optimized to make efficient use of the limited power resources of the nodes. In this report, we provide upper and lower bounds for the following six different wireless sensor network problems.

- **Spanner on Imprecise Points:** We consider the construction of a Euclidean spanner for imprecise points where we take advantage of prior, inexact knowledge of our input.
- **Distributed Mobile Sensor Coverage:** We consider the problem of covering a domain by mobile sensors and the design of an efficient schedule that reduces unnecessary sensor overlap and energy consumption.
- **The Data Gathering Problem:** A data mule tours around a sensor net-

work and helps with network maintenance such as data collection and battery recharging/replacement. We assume that each sensor has a fixed data generation rate and a capacity (upper bound on storage size). If the data mule arrives after the storage capacity is met, additional data generated is lost. We aim to schedule a tour for the mule such that it optimally services the network.

- **The Shortest Separating Cycle Problem:** Given a set of pairs of points in the plane, the goal of the minimum length separating cycle problem is to find a simple tour of minimum length that separates the two points of each pair on different sides.
- ***r*-gather Clustering:** Given a set of points in Euclidean space and a value r, the aim of the r-gather problem is to cluster the points into groups of at least r points each such that the largest diameter of the clusters is minimized.
- **Geographical Routing in 3D:** Given a graph on vertices embedded in $\mathbb{R}^3$, the objective is to add the minimum number of *virtual edges* such that greedy routing succeeds for any source and destination pair.

Dedication Page

I would like to dedicate this work to the following people whose patience and guidance have made this thesis possible: my advisor, Jie Gao; my father, Zhaobang Zeng; my mother Jia Ma; and my boyfriend, Brandon Cieniewicz.

Table of Contents

List of Figures

List of Tables

Acknowledgments

We would like to acknowledge the support of the NSF under the grants CNS-1116881, CNS-1217823, DMS-1221339, CCF-12-16689, and CCF-1017539 .

Chapter 1

Introduction

As computers become more ubiquitous in the prominent phenomenon of the Internet of Things, we encounter many unique challenges in the field of wireless sensor networks. A wireless sensor network is a group of small, low powered sensors with wireless capability used to communicate with each other. These sensor nodes may contain a variety of different types of sensor hardware such as cameras, microphones, temperature sensors, etc. The goal of these nodes is generally to collect some information about their environment and report it back to a base station.

The nature of the sensor nodes are adapted to their environment and target function. In applications where nodes are densely deployed to monitor an area such as heat sensors distributed in a forest to detect the origin of forest fires or temperature sensors placed in an office building to monitor the indoor climate, static nodes may suffice [80, 120].

Other applications may see the need for mobile nodes, where either the sensors themselves are mobile or the sensors are attached to a moving object. Autonomous mobile robots, also know as mules, may have advantages in a sensor network over static nodes in that they can cover more area with fewer nodes or can mobilize to fill a coverage hole if a node goes offline. Sensors are also often attached to moving entities for applications such as monitoring patients' statuses in a hospital, tracking livestock on a farm, or detecting the speed of cars on a highway to observe traffic [44, 104, 145]. Finally, nodes in a sensor network do not have to be all mobile or static. Mixed sensor networks can have mules that can help collect data from the static sensor nodes or help manage the network (recharging batteries, upgrading software, etc).

In many of these applications, the sensors used are tailored to the specific project and the goals of the network architects. However, the prominence of smartphones

have provided an opportunity to take advantage of a deployment of sensors much larger than any possible planned sensor network. Although smartphones are mainly designed for user communication and social interactions, they also provide an platform for communities to sense and contribute sensory information to form a body of knowledge.

The unique nature of networks with mobile nodes brings new potentials but also new challenges. When the sensors are mobile, maintaining an objective as the nodes move around is difficult, especially if the sensors are attached to entities whose movement is unpredictable. It requires a method to detect critical events and methods to update the nodes in a way that is ideally more efficient than executing algorithms intended for static nodes in intervals. For sensor networks with mules, mule route planning needs to accommodate unique objectives that may not occur in other fields. Above all, any algorithm designed for wireless sensor networks must make efficient use of the limited power resources of the nodes.

In this report, we explore six different wireless sensor network scenarios and codify their unique challenges into formal problems. For each problem, we prove hardness constraints and provide algorithms with provable guarantees.

Spanner on Imprecise Points[1] An s-spanner on a set S of n points in $\mathbb{R}^d$ is a graph on S where for every two points $p, q \in S$, there exists a path between them in G whose length is less than or equal to $s \cdot |pq|$ where $|pq|$ is the Euclidean distance between p and q. We consider the construction of a Euclidean spanner for imprecise points where we take advantage of prior, inexact knowledge of our input. In particular, in the first phase, we preprocess n d-dimensional balls with radius r that are approximations of the input points with the guarantee that each input point lies within its respective ball. In the second phase, the specific points are revealed and we quickly compute a spanner using data from the preprocessing phase. We can compute (or update) the $(1+\varepsilon)$-spanner in time $O(n(r+\frac{1}{\varepsilon})^d \log(r+\frac{1}{\varepsilon}))$ after $O(n(r+\frac{1}{\varepsilon})^d \log \alpha)$ preprocessing time where α is the ratio between the furthest and the closest pair of points. Our algorithm does not have any restrictions on the distribution of the points. It is the first such algorithm with linear (update) running time.

Distributed Mobile Sensor Coverage[2] We consider the problem of covering a domain by mobile sensors and the design of an efficient schedule that reduces unnecessary sensor overlap and energy consumption. This problem is motivated

[1] This work was done in collaboration with Jie Gao.

[2] This work was done in collaboration with Esther Ezra and Jie Gao.

by emerging participatory sensing applications as well as other coverage problems involving mobile nodes. The problem of minimizing the total energy consumption while maintaining the same level of coverage guarantee is NP-hard. We develop distributed algorithms achieving a constant approximation factor when sensors have unit disk sensing ranges, and a $(1+\varepsilon)$ approximation factor when the sensors also have constant bounded density. For all these algorithms the communication cost is asymptotically bounded by the cost of simply maintaining the direct neighborhood of each sensor. The constant approximation distributed algorithm can be generalized for the k-coverage problem, when each point of interest has to be covered by at least k sensors.

The Data Gathering Problem[3] We consider the fundamental problem of scheduling data mules for managing a wireless sensor network. A data mule tours around a sensor network and can help with network maintenance such as data collection and battery recharging/replacement. We assume that each sensor has a fixed data generation rate and a capacity (upper bound on storage size). If the data mule arrives after the storage capacity is met, additional data generated is lost. We formulate several fundamental problems for the best schedule of single or multiple data mules and provide algorithms with provable performance. First, we consider using a single data mule to collect data from sensors, and we aim to maximize the data collection rate. We then generalize this model to consider k data mules. Additionally, we study the problem of minimizing the number of data mules such that it is possible for them to collect all data, without loss. For the above problems, when we assume that the capacities of all sensors are the same, we provide exact algorithms for special cases and constant-factor approximation algorithms for more general cases. We also show that several of these problems are NP-hard. When we allow sensor capacities to differ, we have a constant-factor approximation for each of the aforementioned problems when the ratio of the maximum capacity to the minimum capacity is constant.

The Shortest Separator Problem[4] Given a set of pairs of points in the plane, the goal of the shortest separating cycle problem is to find a simple tour of minimum length that separates the two points of each pair to different sides. In this section we prove hardness of the problem and provide approximation algorithms under various settings. The problem is hard to approximate for a small constant factor and is APX-hard even in very restricted scenarios such as when pairs are unit length

[3]This work was done in collaboration with Gui Citovsky, Jie Gao, and Joseph S. B. Mitchell.

[4]This work was done in collaboration with Esther M. Arkin, Jie Gao, Adam Hesterberg, and Joseph S. B. Mitchell.

apart. We provide a polynomial algorithm when all pairs are unit length apart with horizontal orientation inside a square board of size $2 - \varepsilon$. We provide constant approximation algorithms for unit length horizontal or vertical pairs or constant length pairs on points laying on a grid. For pairs with no restriction we have an $O(\sqrt{n})$-approximation algorithm and an $O(\log n)$-approximation algorithm for the shortest separating planar graph.

r-gather Clustering[5] Given a set of n points $P = \{p_1, p_2, \ldots, p_n\}$ in Euclidean space and a value r, the aim of the r-gather problem is to cluster the points into groups of at least r points each such that the largest diameter of the clusters is minimized. While results for this problem are tight for points in a general metric, we want to determine if we can exploit geometry and find better results in the Euclidean setting. We explore this problem with both static and mobile points. For the static setting, we show lower bounds that are close to 2 but do not close the gap. For the dynamic setting, we define three different variations that allow no, k or unlimited changes in the clustering through the entire time period. We show a lower bound of 2 when the points cannot change clusters and apply the lower bounds for the static setting to all variations of the dynamic setting. In addition, we present 2-approximation algorithms for the variations with no or k reclusterings. When we allow an unlimited amount of reclusterings, we show that there can be up to $O(n^3)$ clusters changes. Finally, we provide a distributed 4-approximation algorithm.

Geographical Routing in 3D[6] Existing geographical routing algorithms are mainly for wireless sensor nodes in 2D settings with a number of strong assumptions on wireless communication model or deployment density. We present new ideas that try to push geographical routing for real world network settings and for sensors in general 3D space. Given a graph on vertices embedded in $\mathbb{R}^3$, we would like to add the minimum number of *virtual edges* such that greedy routing succeeds for any source and destination pair. The virtual edges are created by enhanced communication range and when that is not possible, implemented as a path locally stored at the routing table of the nodes. We show that finding the minimum number of virtual edges is actually polynomially solvable. The maximum number of virtual edges is no greater than 5 for 2D and at most 11 for 3D, for *arbitrary directed graphs*. Thus the result would apply for real world network when the links do not necessarily follow any particular communication model and can be asymmetric and dynamic. In our

[5]This work was done in collaboration with Esther M. Arkin, Jie Gao, Matthew P. Johnson, Joseph S. B. Mitchell, and Rik Sarkar.

[6]This work was done in collaboration with Jie Gao, Adam Hesterberg, and Rob Johnson.

simulations the number of virtual edges is actually a lot less than the proved upper bound. The total routing table state size is also at a modest level compared to the worst case bound.

Chapter 2

Spanner on Imprecise Points

2.1 Introduction

While in classical computational geometry all input values are assumed to be accurate, in the real world this assumption does not always hold. Aside from measurement errors or human errors that are always inherent to any data, there are a few scenarios in emerging applications in which only approximate locations are known and from time to time more accurate positions are given. GPS units are generally power hungry and are typically not kept on all the time to save energy. It is common practice to only turn on GPS periodically and, in between the snapshots, interpolate or extrapolate the device's positions based on the speed of the user. In another application, a user's location data might be perturbed in an attempt to protect user privacy. Here, a random nearby location, instead of the user's true location, is reported to location based services, as in [149]. These are the major motivating applications for this problem.

Models on Imprecise Points The ways to model imprecise input can vary greatly based on the interpretation of the nature of uncertainty the data may contain. There are models that are tailored to address input data imprecision or limitations of data representation, where the exact values may never be known [78, 105, 114, 124]. Other approaches involve computing the boundary cases based on the bounds of the input [114] and computing a distribution of solutions when given a probability distribution of the input set [105]. Yet another case assumes a superset of the input is known [32].

A popular model assumes that some knowledge of the location of each input point is known ahead of time. Initially, instead of a point, a region where each input point

may lie is known. The shape (lines [62], circles/balls [2, 52, 81, 106], fat regions [31, 139], etc.) of the region can vary as well as constraints such as the amount of overlapping regions allowed. Algorithms that operate in this model have two phases. The first is a preprocessing phase where the input regions are processed in such a way as to aid the second phase. The second phase, also called the query phase, occurs when the precise locations of the input points are known and the result is computed usually using some structure from the preprocessing phase.

Spanners and Proximity Queries In this section we are interested in the maintenance of a Euclidean spanner and proximity queries using the spanner. Given a set P of n points in $\mathbb{R}^d$, a Euclidean $(1+\varepsilon)$-spanner defines a graph G (often a sparse graph with a linear number of edges) on the points, each edge weighted by the Euclidean length, such that the shortest path between any two points u, v in the graph is at most $1+\varepsilon$ times the Euclidean distance of u, v. Thus one can consider the spanner as a sparse backbone that approximates all of the pairwise distances. For this reason, a spanner can be used to answer many proximity queries such as closest pair, approximate nearest neighbor (given a point q, find a point $p \in P$ that is at most $(1+\varepsilon) \cdot d$ away from q, where d is the distance from q to its closest point in P), and approximate clustering.

Spanners have been studied for many years. Numerous algorithms have been developed to compute a spanner, see the survey [58] or the recent book [116]. Almost all previous work assume that the precise positions of the points are known. We present our results below and compare with the only previous work on spanners for imprecise inputs.

Our Results In the model our algorithm operates in, an imprecise point p is defined to be a disk or a d-dimensional ball centered at a point $\dot{p}$ with radius r. The assumption is that initially, the only information of an input point is $\dot{p}$ and r and that later, the precise location of the point $\hat{p}$ (located somewhere within the corresponding ball) is revealed. Our algorithm preprocesses a set $S = \{p_1, p_2, \ldots, p_n\}$ of n imprecise points in $O(n(r+\frac{1}{\varepsilon})^d \log \alpha)$ time such that when $\hat{S} = \{\hat{p}_1, \hat{p}_2, \ldots, \hat{p}_n\}$ is available, we can compute a $(1+\varepsilon)$-spanner in $O(n(r+\frac{1}{\varepsilon})^d \log(r+\frac{1}{\varepsilon}))$ time where d is the dimension of the input. Without loss of generality we scale our input such that the distance between the closest pair of centers is 1 and thus α is the diameter of the centers of the imprecise points. It is important to note that our algorithm will accept input sets with overlapping points of any depth. In the applications that we are interested in, r is typically a small constant.

Related Work Previous work on algorithms for imprecise points can be classified by objective (convex hull [62], triangulation [81, 139], Delaunay triangulation [31, 52, 106], etc) and shape of imprecise regions (lines [62], circles/balls [2, 52, 81, 106], fat regions [31, 139], etc.). Most of the algorithms aim for linear or almost linear running time for the second phase, when the precise locations are given. Notice that a linear running time is the best we can hope for, as one would need linear time to simply scan the input.

Our construction algorithm modifies a dynamic spanner which has been well studied before [3, 68, 72, 122]. However, since previous work focuses on optimizing the insertion/deletion of single points, the time needed to update the entire spanner for all imprecise points is superlinear.

The only known previous work for spanners on imprecise points is done by Abam et. al. [2]. They provided an algorithm to construct a $(1+\varepsilon)$-spanner from a set of n imprecise points in $O(n/\varepsilon^d)$ time after $O(n \log n)$ preprocessing time, in which the imprecise points are assumed to stay in n *pairwise disjoint* unit balls in d-space. In the preprocessing phase, the well-separated pair decomposition of the centers of the balls is computed, in time $O(n \log n)$ [34]. When the specific points are known, the spanner is constructed by creating an edge between every pair of well-separated sets. Since there are $O(n/\varepsilon^d)$ edges in the spanner, this phase takes $O(n/\varepsilon^d)$ time. They also provide a variation for inputs of different sizes, however this version runs in $O(n \log n/\varepsilon^{2d})$ time in the second phase after $O(n \log n)$ preprocessing.

In comparison with our results, we remark that the above algorithm requires that the imprecise regions are disjoint. If the closest pair of centers is defined to have distance 1, the input requires r to be at most $1/2$. The version with varying sized imprecise regions allows higher values of r but the update cost has an extra d on the exponent. Our algorithm accepts imprecise points with *overlapping* regions and allows the user a degree of control over the running time by trading efficiency with the size of the imprecise region.

2.2 Preliminary Information

The algorithm we present constructs a $(1+\varepsilon)$-spanner or more specifically the deformable spanner (DefSpanner) as described in [68]. A DEFSPANNER is a specific $(1+\varepsilon)$-spanner construction that is designed to be easily modified and updated. More specifically, for a set of points S in $\mathbb{R}^d$, the DEFSPANNER G is made up of a hierarchy of levels $G_{\lceil \log_2 \alpha \rceil} \subseteq G_{\lceil \log_2 \alpha \rceil - 1} \subseteq \cdots \subseteq G_i \subseteq G_{i-1} \subseteq \cdots \subseteq G_0 = S$ where the top level $G_{\lceil \log_2 \alpha \rceil}$ contains only one point and the bottom level G_0 contains all points in S.

The aspect ratio α of S is defined as the ratio of the distance between the furthest and closest pair of points in S. Since we scale all values such that the distance between the closest pair of imprecise centers is 1, α is also the diameter. Note that r is scaled as well. There is no restriction for the distance between the closest pair of precise points.

Any set G_i is a maximal subset of G_{i-1} where for any two points $p, q \in G_i$, $|pq| \geq 2^i$. Let $p^{(i)}$ denote the node p in level i given that $p \in G_i$. We say a point $p^{(i)}$ covers a point $q^{(i-1)}$ if $|pq| \leq 2^i$. While a point $q^{(i-1)}$ may be qualified to be covered by several points in the level above, we arbitrarily designate one of these points (say $p^{(i)}$) as its parent by $p^{(i)} = P(q^{(i-1)})$ and say that $q^{(i-1)}$ is the child of $p^{(i)}$. The exception is if $q \in G_i$, then $P(q^{(i-1)}) = q^{(i)}$. We denote $P^k(p^{(i)}) = P(P^{k-1}(p^{(i)}))$ to be the ancestor of $p^{(i)}$ at k levels higher and define $P^0(p^{(i)}) = p^{(i)}$. When p has no superscript, we assume that $P(p) \neq p$ or intuitively, that p is in the highest level it resides in. We also denote $C_{i-1}(p) = \{q \in G_{i-1} | P(q) = p\}$ as the set of children of $p^{(i)}$ in level $i-1$. By a packing argument (shown in [68]), $|C_{i-1}(p)| \leq 5^d$. Note that for any point p, $|P^i(p^{(0)})p| \leq 2^{i+1}$. See Figure 2.1(a) for an illustration of these rules.

The edges of a DefSpanner of a set S are determined by connecting all nodes within distance $c \cdot 2^i$ in all levels, where i is the level and c is a constant. Such pairs are called neighbors at level i. The neighbors of p at level i are denoted by $N_i(p)$. As shown in [68], $|N_i(p)| \leq (1+2c)^d - 1$. The total number of edges in a DefSpanner is less than $2(1+2c)^d n$. Note that two nodes can be neighbors in multiple levels and that an edge is always constructed between every parent-child pair. The family tree metaphor is extended to define cousins as two points whose parents are neighbors. If two points are neighbors, they are also cousins as their parents must be neighbors as well.

A path between two nodes in a DefSpanner that is less than or equal to $1+\epsilon$ times their euclidean distance can be found by traversing up the hierarchy from both nodes until a mutual link is found. This is illustrated in Figure 2.1(b). We refer to [68] for a proof and more details.

Now we state a property of a DefSpanner that will be useful later.

Lemma 2.2.1. *For a node $p \in G_i$, let $q \in G_{i+1}$ be another node such that $|pq| \leq (c-1)2^{i+1}$. The nodes $P^j(p^{(i)})$ and $P^{j-1}(q^{(i+1)})$ must be neighbors (or the same node) for all $j \geq 1$, i.e., the ancestors of p and q must be neighbors (or converge) in all levels above i.*

Proof: It is given that $|pq| \leq (c-1)2^{i+1}$. By the definition of a DefSpanner, $|P^j(p^{(i)})p| \leq 2^{i+j+1} - 2^{i+1}$ and $|P^{j-1}(q^{(i+1)})q| \leq 2^{i+j+1} - 2^{i+2}$. Thus, $|P^j(p^{(i)})P^{j-1}(q^{(i+1)})| \leq |P^j(p^{(i)})p| + |pq| + |P^{j-1}(q^{(i+1)})q| \leq 2^{i+j+1} - 2^{i+1} + (c-1)2^{i+1} + 2^{i+j+1} - 2^{i+2} =$

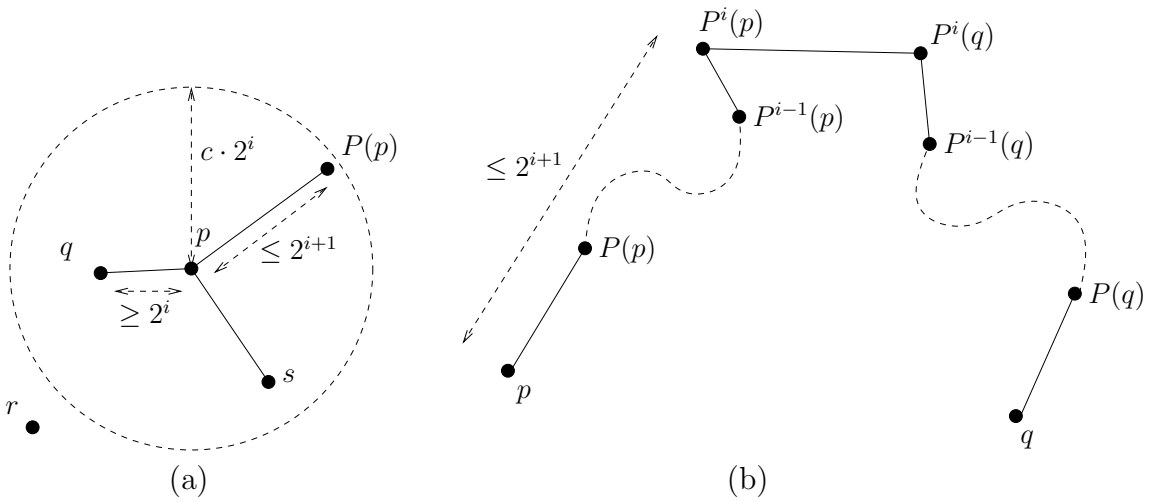


Figure 2.1. (a) The neighbors of p in level i include q, s, and $P(p)$. $P(p)$ must also be in level $i+1$. Since we have denoted p's parent to be another node, we know that p is not in level $i+1$. This diagram does not specify if q, r, or s are also in level $i+1$. (b) A path between p and q that fulfills the $(1+\epsilon)$-spanner property. There is no mutual link between $P^{i-1}(p)$ and $P^{i-1}(q)$.

$4 \cdot 2^{i+j} + (c-4)2^{i+1} \leq 4 \cdot 2^{i+j} + (c-4)2^{i+j} = c \cdot 2^{i+j}$. Therefore $P^j(p^{(i)})$ and $P^{j-1}(q^{(i+1)})$ must be neighbors or the same node in level $i+j$. □

2.3 Algorithm Description

Let $S = \{p_1, p_2, \ldots, p_n\}$ be a set of imprecise points where for p_i, $\dot{p}_i$ is the center, r is the radius, and $\hat{p}_i$ is an instance of p_i, i.e., $|\dot{p}_i\hat{p}_i| \leq r$. Let $\dot{S} = \{\dot{p}_1, \dot{p}_2, \ldots, \dot{p}_n\}$ and $\hat{S} = \{\hat{p}_1, \hat{p}_2, \ldots, \hat{p}_n\}$.

Preprocessing In the preprocessing phase, we are initially given $\dot{S}$ and r. We create a DefSpanner $\dot{G}$ with the point set $\dot{S}$ as in [68]. Here we choose $c = 2r + \frac{16}{\varepsilon} + 4$. The running time is $O(n(r + \frac{1}{\varepsilon})^d \log \alpha)$.

Spanner Construction When the true positions of the points are revealed, we will update the spanner. We construct a DefSpanner $\hat{G}$ for $\hat{S}$ by inserting the points one by one into $\hat{G}$. That is, $\hat{G}$ is initially empty and we gradually create the hierarchy by expanding downwards.

An ordered list is maintained of the remaining points yet to be inserted into the spanner. When we begin, this list contains all of the nodes in the order they appear in the hierarchy in $\dot{G}$, from highest to lowest. So all nodes in level i are placed right

before all nodes in level $i-1$. Parents in $\dot{G}$ will always appear before their children. After all nodes in level i have been inserted, we create level $i-1$ before adding the next node. Here each node at level i is repeated at level $i-1$ with itself as the parent. As we continue to insert nodes into $\hat{G}$, we may add levels to $\hat{G}$ that are beyond the range of levels in $\dot{G}$ – if the exact locations are closer than 1 for instance. During the insertion process for a node, we may find that it is necessary to move the node up or down the hierarchy. If a node is demoted to l, a lower level, l does not exist yet. Instead of creating more levels immediately, the point is removed from $\hat{G}$ and is reinserted into the list of remaining points right before all the other nodes in level l. Therefore, we wait until all levels above l are filled before inserting this node.

The algorithm terminates when all points are inserted. In some sense we are rebuilding the hierarchy for $\hat{G}$ by referring to the hierarchy of $\dot{G}$. We would like to reuse the structure of $\dot{G}$ as much as possible. We elaborate the procedure below.

We assume that $\hat{p}$ is at the beginning of the list, and we want to insert $\hat{p}$ into $\hat{G}$. We insert $\hat{p}$ into the bottom level of $\hat{G}$. Let's call this level i. There are three phases in our algorithm as described below.

Step 1: Check for demotions. First, we compare the distances between $\hat{p}$ and all of its neighbors in $\dot{G}$ that have been inserted in $\hat{G}$. If $|\hat{p}\hat{q}| < 2^i$, where $\hat{q}$ is a neighbor of $\hat{p}$, then $\hat{p}$ is too close to $\hat{q}$ and violates one of the properties of a DefSpanner. Therefore, we must move $\hat{p}$ to a lower level. We can calculate this level to be l, where $2^l \leq |\hat{p}\hat{q}| < 2^{l+1}$ and $\hat{q}$ is the closest node to $\hat{p}$. As discussed earlier, this level does not exist yet, so we must delay the insertion of $\hat{p}$. Meanwhile, we denote $c(\hat{p}) = \hat{q}$, the potential parent of $\hat{p}$.

When we try to reinsert a demoted node such as $\hat{p}$, we still need to check that it does not need to be demoted again. Since demoted nodes are reinserted into $\hat{G}$ before other nodes in the same level, a node can only be demoted by another previously demoted node. Therefore, we only need to test the previously demoted cousins of $\hat{p}$ (through $c(\hat{p})$) for possible demotion. We can strategically place pointers to keep track of these previously demoted nodes.

Step 2: Find a parent. Now we determine the parent of $\hat{p}$. If $c(\hat{p})$ has been defined for $\hat{p}$, then we assign $P(\hat{p}) = c(\hat{p})$. As mentioned in step 1, in this case $\hat{p}$ was demoted from a higher level and is now reinserted into level i so $|\hat{p}c(\hat{p})| \leq 2^{i+1}$. If $c(\hat{p})$ is not defined, we need to search for the parent of $\hat{p}$. First we define the point $u_{\hat{p}}$, the starting point for parent search, and prove a property of it. For this purpose we refer to the spanner $\dot{G}$. Let $u_{\hat{p}} = \hat{q}$ where $\dot{q} = P(\dot{p})$ in $\dot{G}$. Notice that we must have at least attempted to insert $\hat{q}$ in the new spanner. If $\hat{q}$ is not in level $i+1$ ($\hat{q}$ may have been demoted), then we let $u_{\hat{p}} = P(\hat{q})$. If $\hat{q}$ has not been inserted yet (an insertion was attempted but $\hat{q}$ was demoted and has not been reinserted yet), then

let $u_{\hat{p}} = c(\hat{q})$.

Lemma 2.3.1. *Let $\hat{p}$ be a node inserted in to $\hat{G}$ at level i. $P(\hat{p}) \in N_j(P^{j-i-1}(u_{\hat{p}}))$ for some $j \geq i+1$ or does not exist. That is, the parent of $\hat{p}$ is a neighbor of $u_{\hat{p}}$, or a neighbor of an ancestor of $u_{\hat{p}}$, or does not exist (in which case $\hat{p}$ belongs at the top of the hierarchy).*

Proof: Define $\hat{q}$ such that $\dot{q} = P(\dot{p})$ in $\dot{G}$. We know that $u_{\hat{p}}$ may be one of three nodes, $\hat{q}$, $P(\hat{q})$, or $c(\hat{q})$. We know that $|\hat{p}\hat{q}| \leq 2^{i+1} + 2r$, $|\hat{p}P(\hat{q})| \leq |\hat{p}\hat{q}| + |\hat{q}P(\hat{q})| \leq 2^{i+1} + 2r + 2^{i+1}$, and $|\hat{p}c(\hat{q})| \leq |\hat{p}\hat{q}| + |\hat{q}c(\hat{q})| \leq 2^{i+1} + 2r + 2^{i+1}$. Since all of these distances are less than or equal to $(c-2)2^{i+1}$, we can invoke Lemma 2.2.1. Therefore, for all cases of $u_{\hat{p}}$, $P(\hat{p})$ and $P^{j-i-1}(u_{\hat{p}})$ must be neighbors for some $j \geq i+1$ or $\hat{p}$ is at the top and $P(\hat{p})$ does not exist. □

By Lemma 2.3.1, in order to find the parent of $\hat{p}$ we must search neighbors of $u_{\hat{p}}$ and neighbors of ancestors of $u_{\hat{p}}$. More specifically, $P(\hat{p})$ is the lowest node in this set such that $|\hat{p}P(\hat{p})| < 2^{j+1}$ where $\hat{p}$ is in some level $j \geq i$ and $P(\hat{p})$ is in level $j+1$. We search for $P(\hat{p})$ from bottom up, first searching all neighbors of $u_{\hat{p}}$ in one level before moving upward to the next level. The first node that fulfills the parent requirement is selected to be $\hat{p}$'s parent. As we move up the hierarchy in search of $P(\hat{p})$, we promote $\hat{p}$ up the hierarchy as well. For each level j that $\hat{p}$ is promoted to, we let $P(\hat{p}^{(j-1)}) = \hat{p}^{(j)}$. If $\hat{p}$ reaches the top of the hierarchy, then we know that $\hat{p}$ has no parent and $\hat{p}$ becomes the parent of the node that was previously at the top.

Step 3: Find all neighbors. In this step we find the neighbors of $\hat{p}$ for every level it resides in. If $\hat{p}$ lies in the highest level of the graph, then the only neighbor $\hat{p}$ has is itself. Otherwise, all neighbors of $\hat{p}$ in a level must also be a cousin (children of neighbors of $P(\hat{p})$) of $\hat{p}$ in the same level. This is trivial to prove. Therefore, we only need to search among the cousins of $\hat{p}$ to determine if they are neighbors. If $|\hat{p}\hat{q}| < c \cdot 2^j$ where $\hat{p}$ and $\hat{q}$ are cousins in level j, then we denote $\hat{p}$ and $\hat{q}$ to be neighbors and create a link between the two nodes. Note that if we have promoted $\hat{p}$, we need to repeat our search in all levels $\hat{p}$ resides in. This search is executed top down in the hierarchy. After all neighbors of $\hat{p}$ are found in all levels, we conclude the insertion process for $\hat{p}$.

When a new level is created below the bottom level j, we have to maintain our structure. To do this, all nodes in level j are copied to the new level $j-1$ and neighbors in level j are searched to find neighbors in level $j-1$.

After all nodes are inserted, it is possible that we need to demote a node below G_0. In this case, we can simply add more levels to the bottom of the hierarchy. It is

also possible that we promote so many nodes such that lower levels are redundant. In this case, we can delete the redundant levels.

The following theorem concludes with the correctness of the algorithm.

Theorem 2.3.2. *This algorithm creates a correct* DefSpanner $\hat{G}$ *of* $\hat{S}$.

Proof: First, we argue that our algorithm terminates. A node may not be demoted by the same node more than once. Therefore the number of demotions is bounded, all nodes are inserted into $\hat{G}$ and the algorithm terminates.

In a DefSpanner, in every level, all points must be sufficiently far apart and every point has a designated parent (except for the topmost node) and links to all neighbors. We argue that, our algorithm, after each point is inserted into $\hat{G}$, constructs a graph that fulfills all of these properties.

The distance between every pair of nodes in any level i must be greater than or equal to 2^i. We prove this by contradiction. Let us assume that two points $\hat{p}$ and $\hat{q}$ are too close in some level i, that is $|\hat{p}\hat{q}| < 2^i$. Without loss of generality, let us assume that $\hat{p}$ is inserted after $\hat{q}$ which also means that $\hat{p}$ has been inserted into a level that $\hat{q}$ resides in. If $|\hat{p}\hat{q}| < 2^i$ then $|\hat{p}\hat{q}| < 2^j$ must be true for any level $j \geq i$ and specifically the level that $\hat{p}$ is inserted into. Here we have a contradiction because in our algorithm, $\hat{p}$ would have been demoted down to a level i where $|\hat{p}\hat{q}| \geq 2^i$.

Now we must show that every node in every level has a designated parent. When a point $\hat{p}$ is inserted into $\hat{G}$, either $c(\hat{p})$ is assigned to be the parent of $\hat{p}$ or neighbors of ancestors of $u_{\hat{p}}$ are searched for $P(\hat{p})$. By Lemma 2.3.1, if the latter is the case, we know that $P(\hat{p})$ must be found or $\hat{p}$ is promoted to the top of the hierarchy. The parent for each node in the lower levels is the same node in the level above. This is assigned when a node is promoted up the hierarchy or when a level is added to the bottom of the hierarchy.

This leaves showing that the neighbors for each node in all levels are found. For any pair of nodes that are neighbors, the connection is made when the second node is inserted into the graph. In the third step, after the parent of the second node is found, the algorithm then finds the first node among the second node's cousins and creates the link between the two. □

2.4 Algorithm Analysis

The cost of preprocessing is the DefSpanner construction cost or $O(n(r+1/\varepsilon)^d \log_2 \alpha)$ [68]. The running time of our algorithm is bounded by the cost of inserting each node, and the cost of adding new levels to the bottom of the hierarchy.

There are three distinct phases during the insertion of a node: checking for demotion, finding the parent, and finding the new neighbors. We will bound the cost of the three phases separately.

Lemma 2.4.1. *The number of comparisons conducted while determining whether a node should be demoted is* $O(n(r + \frac{1}{\varepsilon})^d)$.

Proof: The first step of inserting a node $\hat{p}$ is to check the distance between $\hat{p}$ and all of its previous neighbors in $\dot{G}$ to determine if any two nodes are too close and if so, $\hat{p}$ is to be demoted. According to our algorithm, if a node $\hat{p}$ is demoted, then the next time it is inserted into $\hat{G}$, it is generally inserted before other nodes in that level are inserted. This way, if there is another node that is too close to $\hat{p}$, then the other node is demoted instead of $\hat{p}$. Therefore, for the majority of demoted nodes, they are only demoted once.

The only case where a node is demoted more than once is when two nodes are demoted to the same level and when they are reinserted, they are found to be too close and one of the two nodes is demoted again. In this case, if we let the other node be $\hat{q}$ and the level the two nodes are reinserted to be i, then $|\dot{p}\dot{q}| \leq |\hat{p}\dot{p}| + |\hat{p}\hat{q}| + |\hat{q}\dot{q}| \leq 2r + 2^i \leq c \cdot 2^i \leq c \cdot 2^j$ for some $j > i$ where $\dot{p}, \dot{q} \in \dot{G}_j$ and either $\dot{p} \notin \dot{G}_{j+1}$ or $\dot{q} \notin \dot{G}_{j+1}$. Then $\dot{p}$ and $\dot{q}$ must be neighbors in level j. Since one of the nodes has been demoted before the other was inserted, $\hat{p}$ and $\hat{q}$ have not been compared yet in the construction of $\hat{G}$. Therefore, the total number of comparisons is bounded by n times the number of neighbors a node may have in one level.

Note that previously, we mentioned that the number of neighbors a node has on one level is bounded by $(1 + 2c)^d - 1$. Given our definition of c, this is $O((r + \frac{1}{\varepsilon})^d)$. Thus, the number of comparisons is bounded by $O(n(r + \frac{1}{\varepsilon})^d)$. □

Lemma 2.4.2. *Two nodes can be neighbors in at most* $O(\log(r + \frac{1}{\varepsilon}))$ *levels.*

Proof: Let $\hat{p}$ and $\hat{q}$ be two nodes that are only neighbors in levels j to i, $j \leq i$. Since $\hat{p}$ and $\hat{q}$ are both in level i and are neighbors in level j, $2^i \leq |\hat{p}\hat{q}| \leq c \cdot 2^j$. From this, we can bound the number of levels $\hat{p}$ and $\hat{q}$ are neighbors in, $i - j + 1$, by $\log c + 1$. Therefore, the maximum number of levels that two nodes can be neighbors in is $\log c + 1 = O(\log(r + \frac{1}{\varepsilon}))$.

□

Lemma 2.4.3. *Let* $\hat{p}$ *be a node inserted into* $\hat{G}$ *at level* i *and let* j *be the highest level it resides in. In levels* $i + 1$ *to* $j - 1$, $\hat{p}$ *and all neighbors of ancestors of* $u_{\hat{p}}$ *are cousins.*

Proof: For a level k, $i+1 \leq k \leq j-1$, $\hat{p}$ is in level $k+1$ and $P^{k-i-1}(u_{\hat{p}})$ is the ancestor of $u_{\hat{p}}$ in level k. Let $\hat{q}$ be a node in level k that is also neighbor of $P^{k-i-1}(u_{\hat{p}})$. We need to show that the parents of $\hat{p}$ and $\hat{q}$ are neighbors in level $k+1$.

From Lemma 2.3.1 we know that $|\hat{p}u_{\hat{p}}| \leq (c-2)2^{i+1}$. We can determine the distance between $\hat{p}$ and $\hat{q}$ to be $|\hat{p}\hat{q}| \leq |\hat{p}u_{\hat{p}}| + |u_{\hat{p}}P^{k-i-1}(u_{\hat{p}})| + |\hat{q}P^{k-i-1}(u_{\hat{p}})| \leq (c-2)2^{i+1}+2\cdot 2^k-2^{i+2}+c\cdot 2^k$. Using this information, we can determine the distance between the parents of $\hat{p}$ (which is itself) and $\hat{q}$ to be $|\hat{p}P(\hat{q})| \leq |\hat{p}\hat{q}| + |\hat{q}P(\hat{q})| \leq (c-2)2^{i+1}+2\cdot 2^k-2^{i+2}+c\cdot 2^k+2^{k+1} = (c-4)2^{i+1}+(c+4)2^k \leq (c-4)2^k+(c+4)2^k = c\cdot 2^{k+1}$. Therefore, the parents of $\hat{p}$ and $\hat{q}$ are neighbors in level $k+1$ and $\hat{p}$ and $\hat{q}$ are cousins in level k.

□

Lemma 2.4.4. *The total cost of finding the new neighbors of all nodes during their insertion is* $O(n(r+\frac{1}{\varepsilon})^d \log(r+\frac{1}{\varepsilon}))$.

Proof: For a node $\hat{p}$ inserted into level i and promoted to level j, the search for $\hat{p}$'s neighbors begins after the parent has been found. First, the cousins of $\hat{p}$ in level j are searched for neighbors. Since, a, the maximum number of children a node can have is bounded by 5^d, this takes $O((r+\frac{1}{\varepsilon})^d)$ time. For all nodes, it takes $O(n(r+\frac{1}{\varepsilon})^d)$ time. Then the cousins in levels below j are checked for neighbors. We can bound the number of nodes we check by charging each cousin to the neighbor link between $\hat{p}$ and the cousins' parent. We charge a operations to each link. By Lemma 2.4.2, we know that each edge in $\hat{G}$ represents a neighbor link between two nodes in $O(\log(r+\frac{1}{\varepsilon}))$ levels. When we take into account the time it takes to find neighbors for all nodes, we can bound all operations (that do not include finding neighbors of a node in the highest level it resides in) by the number of edges in $\hat{G}$ multiplied by $O(a\log(r+\frac{1}{\varepsilon}))$. The number of edges in $\hat{G}$ is bounded by $O(n(r+\frac{1}{\varepsilon})^d)$. The total time it takes to find the new neighbors of all nodes is $O(n(r+\frac{1}{\varepsilon})^d) + O(5^d \cdot n(r+\frac{1}{\varepsilon})^d \log(r+\frac{1}{\varepsilon})) = O(n(r+\frac{1}{\varepsilon})^d \log(r+\frac{1}{\varepsilon}))$. □

Lemma 2.4.5. *The total cost of inserting all nodes into* $\hat{G}$ *is* $O(n(r+\frac{1}{\varepsilon})^d \log(r+\frac{1}{\varepsilon}))$.

Proof: We have already proved time bounds for two phases of insertion: checking for demotions ($O(n(r+\frac{1}{\varepsilon})^d)$ by Lemma 2.4.1) and finding new neighbors ($O(n(r+\frac{1}{\varepsilon})^d \log(r+\frac{1}{\varepsilon}))$ by Lemma 2.4.4). All that we need to do is to bound the time spent searching for the parents of the nodes during insertion. By Lemma 2.4.3, we note that for a node $\hat{p}$ where $\hat{p} \in \hat{G}_j$, $\hat{p} \notin \hat{G}_{j+1}$, all nodes that we test as a possible parent in all levels below j are also checked as possible neighbors. All nodes that we test as a possible parent in level $j+1$ for a node $\hat{p}$ can be bound by the number of neighbors

$u_{\hat{p}}$ can have or $O((r+\frac{1}{\varepsilon})^d)$. Therefore, for all nodes, the time it takes to find the parents is $O(n(r+\frac{1}{\varepsilon})^d) + O(n(r+\frac{1}{\varepsilon})^d \log(r+\frac{1}{\varepsilon}))$. When we add up all costs of insertion, the term that dominates is $O(n(r+\frac{1}{\varepsilon})^d \log(r+\frac{1}{\varepsilon}))$. □

Lastly, we need to consider the time it takes to add new levels to the bottom of the hierarchy while the spanner is built.

Lemma 2.4.6. *The cost of adding new levels to the bottom of the hierarchy during spanner construction is* $O(n(r+\frac{1}{\varepsilon})^d \log(r+\frac{1}{\varepsilon}))$.

Proof: Each time a new level is added to the bottom of the hierarchy, all nodes on the bottom level are compared to all of their neighbors. By Lemma 2.4.2, the maximum number of times that two nodes may be compared is the maximum number of levels they are neighbors in or $O(\log(r+\frac{1}{\varepsilon}))$. Since the number of edges in a DefSpanner represents the number of neighbor pairs, the total cost is $O(n(r+\frac{1}{\varepsilon})^d \log(r+\frac{1}{\varepsilon}))$. □

Now we can finalize our analysis.

Theorem 2.4.7. *The entire algorithm after preprocessing takes* $O(n(r+\frac{1}{\varepsilon})^d \log(r+\frac{1}{\varepsilon}))$.

Proof: Summing the costs of inserting all nodes (Lemma 2.4.5) and creating new levels (Lemma 2.4.6), the entire algorithm takes $O(n(r+\frac{1}{\varepsilon})^d \log(r+\frac{1}{\varepsilon}) + n(r+\frac{1}{\varepsilon})^d \log(r+\frac{1}{\varepsilon})) = O(n(r+\frac{1}{\varepsilon})^d \log(r+\frac{1}{\varepsilon}))$. □

2.5 Conclusion

We have presented an $O(n(r+\frac{1}{\varepsilon})^d \log(r+\frac{1}{\varepsilon}))$ time algorithm to construct a $(1+\varepsilon)$-spanner with $O(n\log(\alpha)(r+\frac{1}{\varepsilon})^d)$ preprocessing, when the accurate positions of the points are revealed and each point is at most distance r from its old position. There are many applications of a DefSpanner [68]: well-separated pair decomposition, all near neighbors query, $(1+\varepsilon)$-nearest neighbor, closest pair and collision detection, and k-center. Note that our algorithm for constructing the DefSpanner will immediately imply that given the accurate positions we can immediately get a structure for the above problems in linear running time.

Chapter 3

Distributed Mobile Sensor Coverage

3.1 Introduction

The past few years have witnessed the wide adoption of smartphones. Although smartphones are mainly designed for user communication and social interactions, they also provide an opportunistic platform for communities to sense and contribute sensory information to form a body of knowledge. This has been termed as *participatory sensing* [33] or *crowdsourcing* applications.

There are good reasons to use smartphones for sensing tasks. First, smartphones already come with multiple sensors such as microphones, GPS units, gyroscopes, accelerometers, cameras, digital compasses, and other commodity sensors [102]. Cellular phones are generally carefully maintained and charged. The popularity of smartphones can make sensing at a large scale, both spatial and temporal, possible. Many interesting applications have already emerged, such as using smartphones for map creation and enhancing bikers' experience [56], urban sensing that considers cities and public living environments [35, 50, 101] with monitoring noise pollution [108], road conditions [59] or on-street parking [43] as examples. Commercial software on smartphones are also available (e.g., Waze, Google's OpenSpot). These participatory sensing applications recruit users' cellphones to accomplish environment sensing tasks. Such effort piggybacks on the cellphone platform and benefits substantially from its wide availability, ample computational cycle, battery life, etc. These participatory sensing applications, nevertheless, must consider the relationship of user mobility and the data quality, in terms of coverage and density.

A recent study using anonymous cell phone records discovered that human tra-

jectories show a high degree of temporal and spatial regularity [71]. The study also confirms, not surprisingly, that individuals return to a few highly frequented locations, such as home or work and that there is a lot of overlap in different individuals' trajectories. Therefore it is expected that there will be ample redundant sensors populating certain 'hot spots' and there is an opportunity to selectively turn on such sensors to reserve battery power. We would like to optimally determine where and when to turn on or off sensors so that as much resources are preserved as possible while coverage is maintained.

Problem definition

In this section we study how to efficiently schedule mobile sensors for monitoring the environment. We represent a sensor by a point (which we also refer to as "node") that is moving around in a pre-specified region Σ (e.g., a polygon) in the plane. We assume two different types of sensing requirements. In the *continuous coverage* model, the entire domain Σ needs to be covered at all time. In the *discrete coverage* model, we assume that a discrete set Ł of landmarks inside Σ need to be covered at all time.

Each sensor has a fixed sensing range, and thus we assume its monitoring region is a unit disk (which is a common assumption in the literature). Since the disks may overlap, we consider scheduling sensors to minimize the total energy consumption. Assume that sensor i, kept on, will consume energy at a rate of w_i and it is kept on for a total of t_i time. Then the total energy usage $\sum_i w_i t_i$ is the weighted sum of the time that sensors are turned on. When the weights are uniform (i.e., all nodes consume power at the same rate), the objective becomes the total time that all sensors are kept on.

The research question at hand is to design a clever schedule of the sensors such that the total energy usage is minimized and the coverage quality does not degrade – i.e., what could be covered by the original collection of sensors at any snapshot is still covered. Notice that this problem is related to the problem of removing redundant sensors for static sensor coverage. In our setting the mobility of the sensors makes this problem more challenging – one main difference is the addition of the temporal dimension in the sense that we may selectively turn on and off any sensor to reserve energy.

Our results

We start by introducing a framework to analyze the complexity of mobile coverage. Although the nodes move continuously, their coverage only changes at some discrete

points of time, which we call "critical events". To enable rigorous theoretical analysis on the number of such critical events, we make a number of additional natural assumptions. We take Σ to be the unit square and assume that the critical events are tracked via a standard HELLO beacon broadcast. Also, without loss of generality, we assume that all the sensors, at any snapshot, collectively cover the entire domain Σ or the entire set of landmarks[1].

We then describe our algorithms for finding approximate mobile coverage solution. We first present a static distributed algorithm. This algorithm achieves a constant approximation factor, where the main observation is to use a grid partitioning of the input sensors into squares of side length $r/2$, where r is the sensing range (i.e., the radius of the disks). For all the sensors whose range intersect the same square, we calculate a constant approximate solution for the local version of a geometric set cover problem. The solution is taken as the union of all the local solutions. We show that this results in a constant approximation factor of the global optimum.

Next, we show that the above algorithm for the static case can be extended to the mobile case such that we only need to update the solution for a local problem whenever a critical event occurs. This is achieved due to our assumption that the critical events are tracked via HELLO beacon broadcast.

We also describe the distributed implementation of a polynomial-time approximation scheme (PTAS) for the case where each sensor has only a constant number of neighbors in its communication range. This is by adapting an existing centralized algorithm (the so-called "shifting" technique [82]) with the critical insight that the core structure is similar to our grid based solution—see below.

We also show that the distributed algorithm can be generalized (with only uniform weights though) to handle *k-coverage* with a constant approximation factor, that is, each point of interest is covered by at least k sensors [152, 100]. The k-coverage problem is a natural extension to the standard coverage problem by improving the robustness and accuracy of the coverage solution.

Related Work

Sensor coverage The problem of *sensor coverage* in the static setting has been extensively studied. We only have space to review the most relevant results to our work, and refer the reader to the survey paper [143] for more details. In particular, our work belongs to the category of using the boolean disk coverage model for area

[1]We describe our assumptions and model in more detail in Section 3.2, however, we give a brief description here as well in order to report our results.

coverage (covering the entire region) or point coverage (covering discrete targets).

For area coverage, one of the most well studied problems is to determine the optimal placement pattern for the infinite plane with minimum sensor density, when sensors can be placed anywhere in the plane. It is shown by Kershner that the triangle lattice pattern is the optimal pattern if the sensors have unit disk sensing regions [92]. Bai *et al.* [23] considered both optimal density and network connectivity. They provided a strip-based placement pattern and proved its asymptotic optimality for achieving both complete coverage and 1-connectivity.

Both the area coverage and point coverage problems can be generalized to k-coverage, where each target or each point in the monitored area must be covered by at least k sensors [152, 100].

Our work is also closely related to the problem of sensor activity scheduling, which is to schedule nodes to be activated alternatively such that the network operation time can be prolonged and area coverage requirement is still met. Various optimization objectives have been used for this problem, ranging from ensuring the area coverage ratio, minimizing the number of active sensors, and prolonging network lifetime [138, 84, 147, 49]. A number of local tests have been developed to check for redundant nodes and put them to sleep while still maintaining full coverage(see, e.g., [49, 147]. The list of reference is too long to survey here and we refer the readers to the survey papers [135, 144].

Despite these previous studies, we are not aware of any previous work which considered mobile sensors, and thus our work is seminal in this topic.

Geometric set cover The *geometric set-cover* problem is an abstraction of the sensor coverage problem. In a typical setting, we are given a set $X \subseteq \mathbb{R}^d$ of points (which is either discrete or continuous) and a set $\mathcal{R}$ of simply-shaped regions in $\mathbb{R}^d$ (e.g., halfspaces, balls, simplices, cylinders, etc.). The goal is to compute a smallest set of regions that altogether cover X.

When $X = \mathbb{R}^d$, or when X is a continuous subset in $\mathbb{R}^d$, we refer to this setup of the problem as the *continuous model*, and when X is a finite point set, this setup of the problem is referred to as the *discrete model*.

The geometric set-cover problem is NP-Hard, under both discrete and continuous models, even for very simple geometric settings in $\mathbb{R}^2$, e.g., when $\mathcal{R}$ is a set of unit disks or unit squares [65, 75]. Therefore attention has mostly focused on developing polynomial time approximation algorithms. The well-known greedy algorithm, which always selects the set that maximizes the residual coverage, yields a $O(\log n)$-approximation in polynomial time for the smallest set-cover [141, Chapter 2], and the known lower-bound results suggest that this is the (asymptotically) best

approximation factor one can hope to achieve in polynomial time for arbitrary (that is, not necessarily geometric) settings [63]. However, by exploiting the underlying geometry, one can obtain polynomial-time algorithms with better approximation factors for various geometric set-cover problems. These algorithms employ and adapt a wide range of novel techniques, including variants of the greedy algorithm, dynamic programming, LP-relaxation, and "ε-nets". It is beyond the scope of this report to give a comprehensive review of the known results. We only mention a few results that are directly relevant to our problem, and refer the reader to [30, 37, 61, 82] and the references therein for further details. Specifically, Clarkson and Varadarajan [47] show a constant approximation factor (achievable in expected polynomial time) for arbitrary disks with uniform weights. A recent result by Chan *et al.* [39] is a constant approximation factor (achievable in expected polynomial time) for the *weighted* set-cover problem of points and arbitrary disks in the plane.

3.2 Preliminary Information

One of the main challenges in analyzing mobile coverage problems is to properly factor the continuity of the motion. We next describe how to model the mobile coverage problem in the time-space domain and how to track *critical events*.

The model We consider a domain Σ to be monitored by mobile sensors; Σ is typically of a simple shape, say, a square or a disk. The sensors are represented by a set $\mathcal{D} = \{D_1, \ldots, D_n\}$ of n unit disks in the plane, where each disk D_i is assigned a weight w_i representing its energy consumption at all times, for $i = 1, \ldots, n$. In what follows, the weights are assumed to be arbitrary, unless stated otherwise.

For each sensor, we denote its sensing range by r (i.e., the radius of the disk in our model), and assume that it has a communication range $3r$. In this case, full coverage of Σ implies network connectivity. We note that in practice the sensing ranges are much smaller than the diameter ρ of the domain Σ, we thus assume $r \ll \rho$ (we also note that $\rho \leq r$ implies that Σ can be covered by a small number of disks, which is a scenario of less interest in theory). Applying a common assumption in mobile networks, we assume that the sensors periodically send HELLO beacons to detect and maintain direct communicating neighbors.

Critical Events. Each sensor moves along an arbitrary trajectory. We denote the position c_i of the center of D_i at time t, for $i = 1, \ldots, n$, by $c_i := c_i(t) = (X_i(t), Y_i(t))$, and $D_i := D_i(t)$ denotes D_i at time t. Let $\mathcal{U}(t) := \bigcup_{i=1}^{n} D_i(t)$ be the *union* of the

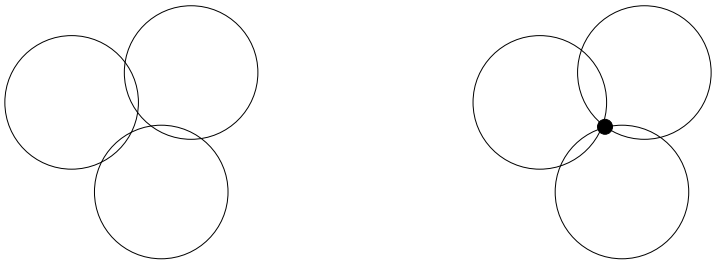


Figure 3.1. (a) A triple of disks with an empty intersection, just before they meet, and (b) their position at the moment they meet in a common point (depicted by the black bullet).

disks at time t (that is, all regions in the plane that are covered by at least one disk at time t). Without loss of generality, we assume that time ranges from 0 to 1 in the mobile coverage problem.

Let us initialize the process with a subset $\mathcal{S}$ of the disks in $\mathcal{D}$ whose union covers the domain Σ at time $t = 0$. Since the union $\mathcal{U}(t)$ of all disks covers the domain Σ at all times $0 \leq t \leq 1$, such a subset cover exists, and let us denote its union in time $t \geq 0$ by $\mathcal{U}'(t) := \bigcup_{D \in \mathcal{S}} D(t)$. We note that the topological structure of the cover $\mathcal{U}'(t)$ changes only at some *critical events*. Each of these events is characterized by a triple of disks that meet at a point p. Such an event may correspond to the appearance of a new connected component in the complement of the union of $\mathcal{U}'(t)$ (that is, a new "hole" in the union appears) or to the disappearance of an existing component (that is, an existing hole is sealed). In the first case we need to add a new disk to $\mathcal{S}$ to restore coverage, and in the latter case we may remove an existing disk from $\mathcal{S}$, to save energy. See Figure 3.1 for an illustration.

Therefore, the optimal solution to the kinetic sensor coverage problem will only change at a critical event, and our assumption on the HELLO beacon broadcast implies that each sensor can locally detect the critical event when its sensing region just starts or stops overlapping with another one's sensing region. Similarly, three sensors can all detect locally when their sensing regions just start or stop having a common intersection.

3.3 Distributed Approximation Algorithms

The General Framework In what follows, we first consider the discrete coverage requirement when a set of landmarks to be covered is given, and is denoted by L. We assume, without loss of generality, that they are covered by the sensors (otherwise, an uncovered point in L can be removed from further consideration). Later on, we

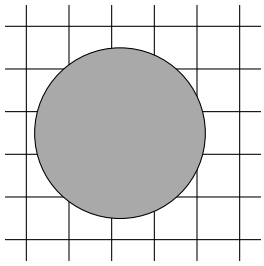


Figure 3.2. A unit disk placed on a grid of side length 1/2.

show how to choose L carefully so that it guarantees a full coverage in the continuous coverage model.

The static problem Our static algorithm is a variant of the set-cover algorithm of Hochbaum and Maass [82], although they refer to the scenario where the disk set is unrestricted (that is, this set is continuous and thus one can choose any arbitrary disk to participate in the coverage), whereas in our scenario this set is *given*.

We place a grid Γ over Σ of side length $r/2$. We say that a cell is *empty* if it does not contain a point of L, otherwise, it is *non-empty*. Our goal is to locally construct a small cover in each non-empty cell of Γ, and then combine all these covers to form the global coverage. Due to our special structure (that is, all disks have radius r) and the existence of a machinery to handle each cell locally, we will be able to show that our coverage is not any larger than the optimal coverage up to a constant factor.

It is easy to verify that each disk $D \in \mathcal{D}$ must fully contain at least one cell of Γ, moreover, it meets at most 25 cells of Γ. See Figure 3.2 for an illustration. Let us now consider a fixed non-empty cell $\tau \in \Gamma$, with a landmark point $p \in L$ inside. By the assumption that all points in L are covered by $\mathcal{D}$, p must have a disk $D \in \mathcal{D}$ with $p \in D$. In particular, D must intersect τ (it either overlaps τ or fully contains it). We now collect all points $p \in L$ that lie inside τ and the disks intersecting τ; let L_τ, $\mathcal{D}_\tau$ be these resulting subsets. We next construct a set cover for L_τ with the disks in $\mathcal{D}_\tau$ using the algorithm of Chan *et al.* [37]. As noted in Section 3.1 this algorithm produces a set cover whose size is only within a constant factor of the smallest set cover. Let $\mathcal{S}_\tau \subseteq \mathcal{D}_\tau$ be the set cover just produced. We then report the set

$$\mathcal{S} := \bigcup_{\tau \in \Gamma, \tau \cap L \neq \emptyset} \mathcal{S}_\tau.$$

We next show:

Lemma 3.3.1. *Let* OPT *be the smallest set cover for* L *and* $\mathcal{D}$. *Then* $|\mathcal{S}| = O(|\mathrm{OPT}|)$.

Proof: Let $\tau \in \Gamma$ be a fixed non-empty cell. We next observe that the optimal set cover OPT^*_τ for L_τ, $\mathcal{D}_\tau$ is at least as good as the entire optimal set cover OPT restricted to τ (as OPT^*_τ is the best coverage for L_τ, $\mathcal{D}_\tau$ whatsoever). That is, $|\mathrm{OPT}^*_\tau| \leq |\mathrm{OPT}_\tau|$, where OPT_τ is the subset of disks in OPT meeting τ.

By the properties of the approximation algorithm of Chan *et al.* [39] it follows that $|\mathcal{S}_\tau| \leq C \cdot |\mathrm{OPT}^*_\tau|$, where $C > 0$ is an absolute constant. On the other hand, we claim that $\sum_{\tau\in\Gamma,\tau\cap L\neq\emptyset} |\mathrm{OPT}_\tau| \leq 25|\mathrm{OPT}|$, since, as observed above, each disk of D can meet at most 25 cells of the grid, and thus the multiplicity of a fixed disk in the optimal solution OPT cannot exceed this constant. Combining the above inequalities we obtain:

$$|\mathcal{S}| \leq \sum_{\tau\in\Gamma,\tau\cap L\neq\emptyset} |\mathcal{S}_\tau| \leq C \sum_{\tau\in\Gamma,\tau\cap L\neq\emptyset} |\mathrm{OPT}^*_\tau| \leq$$

$$C \sum_{\tau\in\Gamma,\tau\cap L\neq\emptyset} |\mathrm{OPT}_\tau| \leq 25C|\mathrm{OPT}|,$$

from which we conclude $|\mathcal{S}| = O(|\mathrm{OPT}|)$, as asserted. □

In order to make the algorithm distributed, we observe that since the communication range of each sensor is $3r$, any two disks that meet the same cell τ can directly communicate with each other. Indeed, all disks (of radius r) that meet (a non-empty cell) τ must have their centers within distance at most r from τ. We now observe that the centers c, c' of two distinct such disks D, D' are located within distance at most $2r + r/\sqrt{2} < 3r$ from each other. This can be verified by placing c, c' within distance r from two opposite corners of τ and the fact that the distance between these two corners is $r/\sqrt{2}$. Having this property at hand, the communication graph refined to the corresponding nodes (whose disks meet τ) forms a clique, and hence all disks meeting τ can "nominate" (say, by selecting the one with smallest ID) one such disk D^* to hold all information about the local setting to our set cover problem, that is, L_τ, $\mathcal{D}_\tau$, and then the computation for the approximated set cover (as described above) can be done within D^*.

The kinetic problem At the initial time t_0 we apply the static algorithm presented above in order to find an approximation for the set cover. Then for each node (that is, a sensor), we keep track of the state of all possible triples with that

node, that is, whether they have an empty or non-empty intersection (recall that our model supports that). Then, when a critical event is detected (and so we also have the triple it involves), the local solution within the grid cell τ on which the critical event occurred will be re-computed using the algorithm for the static case (*only within* τ). We compute τ by locating in the grid Γ the cell containing the common intersection of the disks at the corresponding critical event.

Running time and communication cost For the static problem, computing the approximate set-cover locally within each cell τ is done in expected polynomial time [39]. We recall that each disk D communicates only with other disks whose centers lie in its communication range, in other words, each sensor exchanges messages only with its *neighbors* in the communication network. Recall that we do so independently in each cell $\tau \in \Gamma$ that D meets, and that there are at most 25 such cells, and thus the overall number of exchanged messages that D involves is proportional to its degree in the communication network. Thus the total number of exchanged messages is proportional to the sum of these degrees (over all nodes representing the disks D), which is just $O(|E|)$, where E is the set of links in the network. This bound is no more than the number of messages needed for each node to discover its neighborhood (up to a constant factor), which is a necessary routine for almost all networks.

Concerning the kinetic version, in each round imposed by the periodic beacons, each disk D exchanges messages in its neighborhood as described above for the static problem. Thus the total number of messages exchanged is $O(T|E|)$, where T is the number of rounds. Here too, we piggyback the communication cost on the cost of maintaining neighborhoods, and thus the algorithm does not incur extra cost. We have thus shown:

Theorem 3.3.2. *Given a network of mobile sensors with an* Hello *beacon model as above, where each of the sensors has a unit sensing radius r and a communication radius $3r$, and a set of landmarks L (confined to a unit-square domain Σ) to be monitored, one can compute a coverage for L at each time t, whose overall weight is at most a constant factor from the optimum. The overall number of messages exchanged from each sensor is proportional to its neighborhood size at any time t.*

Sparse networks A natural case that we study is where the communication degree in the network is constant for each node. This scenario arises in the context of participatory sensing, as one cannot condense many participants (e.g., people who carry smartphones) at the same location. We first note that when applying our

previous algorithm in this case, the total number of messages exchanged from and to each node is only $O(1)$ (at each round), and the overall number of messages exchanged in the network is only linear in the number of nodes (at each round). Nevertheless, we show below that for this setting, one can achieve a better approximation factor.

We apply the polynomial approximation scheme (PTAS) of Hochbaum and Maass [82], based on a *shifting technique*. A crucial property in their algorithm is the fact that each point in the plane is covered by at most $O(1)$ disks, which corresponds to our scenario, as for each disk there are only $O(1)$ other disks that it meets. Thus, in particular, a "deep" point (a point that is covered by arbitrarily many disks) implies that there is a large clique in the network, and thus the communication degree (of the nodes in the clique) must be large as well, which contradicts our assumption. Thus all points are "shallow" (i.e., have constant depth).

In this algorithm we fix a small error parameter $\varepsilon > 0$, and then set the side length of the grid Γ to be $\Delta := O(r/\varepsilon)$ (recall that r is the sensing radius of the disks). A key observation of the algorithm in [82] is the fact that due to the constant-depth property, within each cell τ of the grid the *optimal* solution can be computed in polynomial time (where the degree of the polynomial depends on ε). The algorithm in [82] then shows that if we place the origin of this grid at a *random* point in $[0, \Delta]^2$, find the optimal set-cover OPT^*_τ locally in each cell τ, and then collect all these local solutions to form the actual set-cover $\mathcal{S}$, we obtain that $|\mathcal{S}|$ is at most $(1+\varepsilon)|\mathrm{OPT}|$ (on expectation), where OPT is the optimal solution for the whole setting. That is, the main difference between this algorithm and the previous one given for an arbitrary communication network, is the fact that here (i) we randomly shift the grid, and (ii) we compute the actual optimal solution within each grid cell rather than an approximate solution. These two facts eventually leads to the PTAS in [82], which is an improvement over the constant approximation factor we obtained earlier.

Here too we can make the algorithm distributed using a similar approach as in the general framework, however, we observe that the communication graph confined to the nodes, whose corresponding disks meet a fixed cell τ, is not necessarily connected. Nevertheless, we claim that the coverages constructed w.r.t. each connected component must be pairwise disjoint, and then an optimal coverage is just the disjoint union of the respective optimal coverages computed for each connected component. Indeed, two sensors in different connected components must lie within distance at least $3r$ from each other, and thus if we draw a disk of radius r centered in each node of these components, the two resulting structures (each of which is a union of disks that contain an optimal coverage) remain disjoint. Thus in each such connected component we can nominate a disk to compute the optimal set-cover.

In the kinetic problem, we track the critical events and make the local computa-

tion in the grid at each such event, this is done almost verbatim as in the general framework.

Concerning the communication cost, in this solution we need to expand the neighborhood of each node by $O(1/\varepsilon^2)$. Specifically, the fact that each node has only $O(1)$ neighbors in the network, and, on the other hand, each grid cell τ is now of side length $O(r/\varepsilon)$, implies that τ may be intersected by only $O(1/\varepsilon^2)$ disks (we omit the easy computation, which follows from a straightforward packing argument), and thus each disk may exchange messages with that many disks in τ. On the other hand, consider the connected components formed by the disks covering the same grid cell τ. Each connected component has a diameter of at most $O(1/\varepsilon)$. Thus each node searching in its $O(1/\varepsilon)$-hop neighborhood can find all potential nodes in its connected component. Since each disk appears in only $O(1)$ cells (assuming ε is sufficiently small, as otherwise we can resort to the previous algorithm), the total number of exchanged messages involving D is $O(1/\varepsilon^2)$, and this bound is $O(n/\varepsilon^2)$ over all nodes. We have thus shown:

Theorem 3.3.3. *Given a network of mobile sensors of constant communication degree and a set of landmarks L as in Theorem 3.3.2, one can compute a coverage of L at each time t, whose weight is within a factor of $(1+\varepsilon)$ from the optimum, for any $\varepsilon > 0$. The overall number of messages exchanged from each sensor is $O(1/\varepsilon^2)$ at any time t.*

The continuous coverage problem We now aim to cover the entire domain Σ at all times. Consider that one can produce an appropriate point set P by constructing the planar subdivision induced by the disks (also referred to as the "arrangement of the disks"), with the property that P is covered if and only if Σ is covered; such constructions are standard in the theory of computational geometry [51]. This computation involves *in each round* the construction of the planar subdivision induced by the disks.

Since Σ is covered at all times, the underlying communication graph is connected. Moreover, this graph remains connected in any refinement to a subset of nodes, whose corresponding disks meet a fixed cell τ. Thus all disks meeting a cell τ can nominate one disk D^* to locally compute the subdivision within τ. Then the entire subdivision is obtained by gluing together all these "pieces". Having this subdivision at hand, the computation of P is fairly standard (see, e.g., [51]). At time t_0 we compute P with respect to the initial disk locations, and then apply the static algorithm with this set of landmarks. For each critical event, before we update our set cover, we only re-compute the portion of P that intersects with the cell we are updating (this is sufficient as the combinatorial structure of the coverage does not change elsewhere).

Using similar arguments as above, the communication overhead, at any time t, is proportional to the sum of the neighborhood sizes over all nodes.

The k-Coverage Problem The *k-coverage problem* is a special case of the so-called *set multi-cover* problem, and in our scenarios this implies that we require each point to be covered by at least k disks, or, more specifically, each point should be monitored by at least k sensors, where $k > 0$ is an integer parameter (and thus the case $k = 1$ is just the standard set cover problem).

In order to solve this problem efficiently, we use a similar grid construction as the one for the original problem, and instead of applying the set-cover algorithm of Chan *et al.* [39], we apply the set multi-cover algorithm of Chekuri *et al.* [40]. This algorithm can be applied only when the weights are uniform, and computes in expected polynomial time a k-coverage whose size is only within a constant factor from the optimum. Plugging this into our machinery, we obtain a similar bound as in Lemma 3.3.1.

Theorem 3.3.4. *Given the setting of Theorem 3.3.2 (while assuming uniform weights), and an integer parameter $k > 0$, one can compute a k-coverage for L at each time t, whose size is at most a constant factor from the optimum. The overall number of messages exchanged from each sensor, at any time t, is proportional to its neighborhood size.*

3.4 Conclusion

In this section we studied the problem of efficiently constructing coverage by mobile sensors in order to reduce energy consumption, we are not aware of any previous works on this topic. Our approximation algorithms are based on the shifting technique technique of Hochbaum and Maass [82], nevertheless, this machinery has never been subject before to a distributed computing, which, as our work shows, is an important and useful machinery for both theory and practice. We hope that this observation will be useful for existing and future participatory sensing applications.

Chapter 4

The Data Gathering Problem

4.1 Introduction

A number of sensor network designs integrate both static sensor nodes and more powerful mobile nodes, called *data mules*, that serve and help to manage the sensor nodes [129, 87, 86, 132]. The motivation for such designs are twofold. First, there are fundamental limitations with the flat topology of static sensors using short range wireless communication. It is known that such a topology does not scale – the network throughput will diminish if the number of sensors goes to infinity [79], while allowing node mobility will help [74]. Second, a number of fundamental network operations can benefit substantially from mobile nodes. We consider two example scenarios: sensor data collection and battery recharging. In both cases, data mules that tour around the sensors periodically can be used to maintain the normal functionality of the sensors. In addition, data collection by sensors using multihop routing to a fixed base station often suffers from the bottleneck issue near the base station, both in terms of communication and energy usage. Using short range wireless communication with a mobile base station can fundamentally remove such dependency and avoid the single point of failure [142].

Despite the potential benefits of introducing data mules with static sensors, a lot of new challenges emerge at the interface of coordinating the data mules with sensors. One of the most prominent challenges is the scheduling of data mule mobility to serve the sensors in a timely and energy efficient manner. This has been an active research topic for the past few years. However, as surveyed later, most prior work is evaluated by simulations or experiments [8]; algorithms with provable guarantees are scarce. In this section we make contributions in this direction. We formulate data mule scheduling problems with natural objective functions and provide exact

and approximation algorithms.

Our Problem

Suppose there are n sensors and a data mule traveling at a constant speed s to collect data from these sensors. A sensor i generates data at a fixed rate of r_i and has a storage ("bucket") capacity of c_i where $c_i \geq r_i$. When a data mule visits a sensor, all current data stored in the sensor is collected onto the mule. We assume that the mule has unbounded storage capacity. We also assume that data collection at each sensor happens instantaneously, i.e., we ignore the time of data transmission, which is typically much smaller than the time taken by the mule to move between the sensors. If the amount of data generated at a sensor goes beyond its capacity (i.e., its bucket is full), additional data generated is lost. Thus, a natural objective is to schedule data mules to efficiently collect the continuously generated data.

We assume that the data collection and the data mule movement continues indefinitely in time. Therefore, we are mainly concerned about the long-term data gathering efficiency by periodic schedules.

The same problem arises in the case of battery recharging and energy management. In that case, each sensor i uses its battery with capacity c_i at a rate of r_i. When the battery at a sensor is depleted the sensor becomes ineffective. Thus, one would like to minimize the total amount of time of ineffectiveness, over all sensors. We formulate the following three problems.

- **Single Mule Scheduling:** Find a route for a single data mule to collect data from the sensors that maximizes the data collection rate (the average amount of data collected per time unit).
- **k-Mule Scheduling:** Given a budget of k data mules, find routes for them to maximize the rate of data collected from the sensors.
- **No Data Loss Scheduling:** Find the minimum number of data mules, and their schedules, such that all data from all sensors is collected (there is no data loss).

Our Results

We report hardness results, exact algorithms for a few special cases, and approximation algorithms for all three problems. Our algorithmic results are summarized in Table 4.1. When we assume that the capacities of all sensors are the same, we provide results for the different cases where the sensors lie in different metrics. For the case where the capacities of the sensors are different, we provide general results.

With Sensors	Single mule	k-mule	No Data Loss
on a Line	exact	$\frac{1}{3}$	exact
on a Tree	exact pseudo-polynomial	$\frac{1}{3}(1-1/e^{\frac{1}{2+\varepsilon}})$	12
General Metric Space	$1/6-\varepsilon$		
Euclidean Space	$1/3-\varepsilon$	$\frac{1}{3}(1-1/e^{1-\varepsilon})$	
with Different Capacities	$O(\frac{1}{m})$		$O(m)$

Table 4.1. Our approximation algorithm results for different settings. Note that $m \leq \log(\frac{c_{max}}{c_{min}})$ where c_{max} is the largest capacity and c_{min} is the smallest capacity. For the results in the first four rows, we assume that the sensor capacities are all the same. ε is any positive constant.

Without loss of generality, we assume that the minimum data rate is 1 and the mule velocity is 1. In fact, we can further assume that all sensors have a data rate of 1; if a sensor has data rate $r_i > 1$, we can replicate this sensor with r_i copies, each with unit data rate and capacity c_i/r_i. Thus, in the following discussion we focus on the case of all sensors having unit data rates and possibly different capacities. When we consider the case where all sensors have the same capacity, we simplify notation and let the capacity of all sensors be c.

We give the first algorithms for such data mule scheduling problems with provable guarantees. In addition, we provide upper and lower bounds on the optimal solution for both problems, and we evaluate the performance using simulations, for a variety of sensor distributions and densities.

Related Work

Vehicle Routing Problems The problems we study belong to the general family of vehicle routing problems (VRPs) and traveling salesman problems (TSPs) with constraints [20, 103, 119, 150]. But our problem is the first one considering periodically regenerated rewards/prizes and thus is the first of this type.

Related TSP variations stem from the Prize-Collecting Traveling Salesman Problem (PCTSP) [22, 26] which was originally defined by Balas [24] as the problem, given a set of cities with associated prizes and a prize quota to reach, find a path/tour on a subset of the cities such that the quota is met, while minimizing the total distance plus penalties for the cities skipped. (Some recent formulations of this problem do not include penalties for skipped cities.) Archer et. al. [10] provided a $(2-\varepsilon)$-approximation algorithm for this formulation of PCTSP where $\varepsilon \approx 0.007024$.

The Orienteering Problem [70, 140] assigns a prize to each city and, given a constraint on the length of the path, aims to maximize the total prize collected. For

the rooted version in a general metric space, Blum et. al. [28] had proven that the problem is APX-hard and provided a 4-approximation algorithm which was improved to a 3-approximation by Bansal et. al. [25] and finally to a $(2+\varepsilon)$-approximation by Chekuri et. al. [41]. For the rooted version in $\Re^2$, Arkin et. al. [12]give a 2-approximation, which was improved to a $(1+\varepsilon)$-approximation (PTAS) [42] for fixed dimension Euclidean space. For additional information we refer to the review papers [64, 140].

Similar to our problems, the Profitable Tour Problem [19] balances the two competing objectives of maximizing total prize collected and minimizing tour length. In some problems, the profit collected is dependent on the latency [48].

Our problems are also very similar to many multi-vehicle routing problems [11, 57, 60, 94]. Arkin et. al. [14] give constant-factor approximation algorithms for some types of multiple vehicle routing problems including a 3-approximation for the problem of finding a minimum number of tours shorter than a given bound that cover a given graph. Nagarajan and Ravi [115] provide a 2-approximation for tree metrics and a bicriteria approximation algorithm for general metrics. Khani and Salavatipour [93] present a 2.5-approximation algorithm for the problem of finding, for a given graph and bound λ, the minimum number of trees, each of weight at most λ, to cover the graph (improving on a bound given in [14]).

Data Mule Scheduling Increasingly, there has been interest in using mobile data mules to collect data in sensor networks. A common question that has arisen is how to schedule multiple mules effectively and efficiently. Many heuristics have been proposed to schedule multiple mules with various constraints and objective functions (e.g., evenly distributing loads [86], scheduling short path lengths [107, 146], and minimizing energy [6]). Somasundara et. al. [133] address a very similar problem to ours, but with different methods; we obtain provable polynomial-time algorithms, while they employ (worst case exponential-time) integer linear programming and explore heuristics.

4.2 Single Mule Scheduling

Given a single mobile data mule with unit velocity, n sensors with uniform capacity and unit data rate, the goal is to route the mule in effort to maximize its data gathering rate. We explore this problem with sensors on a line, on a tree, and in space.

Exact Algorithms on a Line or a Tree

Line Case We first look at the case when the sensors are on a line. We assume that the input data is integral; specifically, the sensors p_i are located at integer coordinates and the capacities c_i for all i are integers. With this assumption, the optimal schedule can be shown to be periodic.

Lemma 4.2.1. *The optimal schedule that minimizes data loss is periodic, assuming integral input data.*

Proof: If the sensors are located at integral positions, the distances between any two of them are integers as well. Thus, all states of the problem can be encoded by the position of the mule and the current amount of data at each sensor i. All of these values are integers. Thus, the total number of possible states is finite; after a state reappears we realize that the robot must follow the same schedule, making the schedule periodic. □

Theorem 4.2.2. *Let there be n sensors, $p_1, p_2, \ldots, p_n$ on a line. Assume that the capacities and rates of all sensors are the same: $c_i = c$ and $r_i = 1$, for $1 \leq i \leq n$. Then there exists an optimal path that minimizes data loss with the following properties: (1) its leftmost and rightmost points are at sensors, (2) it is a path making U-turns only at the leftmost and rightmost sensors.*

Proof: The first claim is obvious since one can remove the portion of the path beyond the leftmost sensor and shorten the time for one period of the trip, which results in all covered sensors being visited more frequently than before. Now, let there be an optimal path P with leftmost sensor at x_l and rightmost sensor at x_r.

Let us assume that P contains U-turns that do not occur at the extreme points x_l, x_r. A U-turn is called a left U-turn if the mule was traveling from right to left before the turn, and a right U-turn otherwise. Note that one cycle of P can be split into two sub-paths, a path P_l from x_l to x_r and a path P_r from x_r to x_l. These sub-paths may have U-turns at the extreme points x_l, x_r. (In that case, P_l and P_r are not unique.) Without loss of generality, assume that there are U-turns in the sub-path from x_l to x_r. Consider the closest left U-turn to x_l and name the point it occurs at z_l. Since P is periodic, we can assume that in one full period, P starts and ends at x_l.

Now, we will modify the trip P. Notice that for the mule to make a left U-turn at z_l, it must make a right U-turn in the path from x_l to z_l. Let the rightmost such U-turn be z_r. We remove the path that begins at the first time P_l reaches z_l to the last time P_l reaches z_l. See Figure 4.1 for an example.

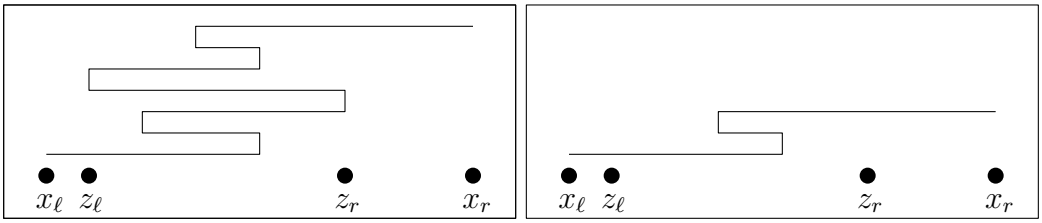


Figure 4.1. Remove a segment of the path to eliminate a U-turn at z_l.

After the surgery, we get a schedule without the round trip beginning at z_l. Suppose the trip eliminated has duration δ and the trip after this surgery, denoted by P', has round-trip time length T. That means, the trip P has duration $T + \delta$.

Since the rate of data accumulation and the bucket size are the same for all sensors, all sensors take the same amount of time to overflow. As a result, when a mule travels from right to left and encounters an overflowing sensor b, then all sensors to the left of b are also overflowing – since the last time they were visited was definitely before the last time that b was visited.

In the shortened path P', the sensors to the left of z_l are reached δ time sooner than in P. In the paths P and P', all sensors to the left of z_l overflow in both paths or there exists a sensor to the left of z_l that does not overflow in either P or P'. In the latter case, lets define y_l to be the leftmost such sensor. We define b_l to be the leftmost of either z_l or y_l. Let n_l denote the number of sensors to the left of b_l. Since all sensors to the left of b_l overflow in both P and P', the difference in the amount of data lost in those sensors between the two paths is δn_l. Also, note that the definition of y_l suggests that there exists a path in either P or P' from y_l to x_r to y_l where none of the sensors from x_r to y_l overflow. The terms b_r, y_r, and n_r are defined similarly in the right direction.

Let m denote the amount of overflow for sensors covered by P' in one full period and let n'' be the number of sensors not covered by the path P' (i.e., those that are either to the left of x_l or the right of x_r). The overflow rate is $m/T + n''$ in path P' and for P it is $\frac{m+\delta(n_l+n_r)+e}{T+\delta} + n''$ where e is any extra overflow in P that is not accounted for such as from sensors between z_l and z_r.

Since P is an optimal path, then $\frac{m+\delta(n_l+n_r)+e}{T+\delta} + n'' \leq m/T + n''$. We simplify this relationship to

$$\frac{m + \delta(n_l + n_r) + e}{T + \delta} + n'' \leq \frac{m}{T} + n''$$

$$\frac{m + \delta(n_l + n_r) + e}{T + \delta} \leq \frac{m + m(\frac{\delta}{T})}{T + \delta}$$

$$n_l + n_r + \frac{e}{\delta} \leq \frac{m}{T}. \tag{4.1}$$

Consider a simple periodic path P'' only making U-turns at b_l and b_r. If b_l or b_r is defined by a non-overflowing sensor, then there is no overflow between b_l and b_r since there exists a longer path between b_l and x_r (or b_r and x_l) in P or P' that does not cause any sensor to overflow. Thus the overflow rate of the path is just $n_l + n_r + n''$. If b_l is z_l and b_r is z_r then the overflow rate of the path is $n_l + n_r + \frac{e'}{\delta} + n''$ where e' is the extra overflow. Note that by definition, $e' \leq e$ since a cycle of P'' is shorter than the path cut out in P. For either case, the overflow rate of the direct periodic path between b_l and b_r is bound by $n_l + n_r + \frac{e'}{\delta} + n''$. By Equation 4.1,

$$n_l + n_r + \frac{e'}{\delta} + n'' \leq \frac{m}{T} + n''$$

$$\frac{(T+\delta)(n_l + n_r + \frac{e'}{\delta})}{T+\delta} \leq \frac{m}{T+\delta} + \frac{\delta(n_l + n_r + (\frac{e}{\delta}))}{T+\delta}$$

$$n_l + n_r + \frac{e'}{\delta} + n'' \leq \frac{m + \delta(n_l + n_r) + e}{T+\delta} + n''$$

Since P is optimal, the direct path must be optimal as well. Therefore, there exists an optimal path that is direct with only U-turns at the extreme points. □

The immediate consequence from Theorem 4.2.2 is that one can find the optimal schedule in $O(n^2)$ time, enumerating all possible pairs of extreme points.

It is important to note that it is sometimes necessary for the mule in the optimal solution to gather data more than once from a given sensor in a period. In Figure 4.2, sensors are split into six groups, where each group has either k or $2k$ sensors. Within each group, each sensor has the same x-coordinate. In the optimal solution, the data mule traverses the entire interval back and forth, picking up data whenever it reaches a sensor. This solution has data gathering rate $\frac{10.5k}{2} = 5.25k$. In comparison, the best solution that gathers data from a sensor at most once per period has rate $4k$.

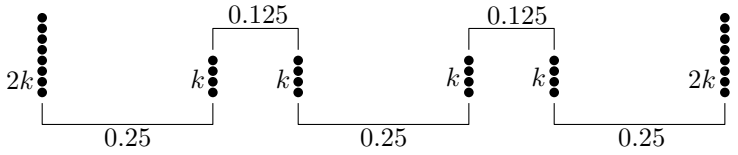


Figure 4.2. The optimal solution repeats sensors

Tree Case We extend our results to a tree topology, with the sensors placed on a tree network embedded in the plane. Then, we show that the structure of an optimal schedule for the mule is to follow (repeatedly) a simple cycle (a doubling of a subtree). Again we assume that all sensors have the same capacity c and the same rate, 1, of data accumulation. We also assume that the input is integral, i.e., c is an integer and the distance between any two sensors on the tree network is an integer.

Theorem 4.2.3. *Let there be n sensors, $p_1, p_2, \ldots, p_n$ on a tree G. For all p_i, $1 \leq i \leq n$, let $c_i = c$ and $r_i = 1$, i.e. let the capacity and rates of all sensors be the same. There exists an optimal path that minimizes data loss with the following properties: (1) it only changes direction at sensors, (2) it is a cycle obtained by doubling a subtree.*

Proof: For the first claim, we argue that any path that contains a U-turn that does not coincide with a sensor can be shortened to the preceding sensor. The shortened path will visit all nodes more frequently and will not lose any more data than the original path.

Our argument for the second claim is similar to the proof for the line case. Let there be an optimal path P that is not a cyclic, depth-first traversal on a subtree G' of G. We denote the set of nodes $X = \{x_1, x_2, \ldots, x_m\}$ to be leaves of G'. Therefore, the path P must contain a subcycle that visits at most one node in X. We also denote P as the set of nodes the path P visits.

Note that any subcycle that visits more than one node in X contains a subcycle that visits exactly one node in X. When a path visits a leaf, the section from the nearest junction to the leaf and back again is such a subcycle.

Consider a subcycle that visits at most one node in X. Let P_i be the shortest section of P that begins and ends with a node in X and completely contains the subcycle. We denote the node z_b to be the closest node in P_i to x_i that is also part of a subcycle.

Since P is periodic, we can assume that in one full period, P always starts at x_i and ends at x_i. Let the path from z_b along P_i and back to z_b be denoted as S.

Now, we will modify the trip P by removing S. Suppose S has duration δ and the path after this subcycle is removed, denoted by P', has duration T. That means, the trip P has duration $T + \delta$. We now calculate the overflow rate of one entire cycle with and without the subcycle S.

Let Y denote the set of sensors that do not lie on S and overflow in both P and P' but are reached δ time sooner in P'. Let $|Y| = n'$ and R be the subtree of G' on the nodes $S \cup O$ where O is the set of nodes that do not overflow in both P and P'. Note that if a node is overflowing and the mule is above the node in the tree, then

all nodes below the node are also overflowing. Therefore, if no nodes in S overflow, then R does not overlap any overflowing nodes.

Let m denote the amount of overflow in P' in one full period not including the overflow of the n'' nodes that P and P' miss altogether. The overflow rate is $m/T+n''$ in path P' and $\frac{m+\delta \cdot n'+\epsilon}{T+\delta} + n''$ in path P where ϵ is any extra overflow in P that is not accounted for. Since P is an optimal path, then $\frac{m+\delta n'+\epsilon}{T+\delta} + n'' \leq m/T + n''$. We simplify this relationship to

$$\frac{m+\delta n'+\epsilon}{T+\delta} + n'' \leq \frac{m}{T} + n''$$

$$\frac{m+\delta n'+\epsilon}{T+\delta} \leq \frac{m+m(\frac{\delta}{T})}{T+\delta}$$

$$n' + \frac{\epsilon}{\delta} \leq \frac{m}{T}. \tag{4.2}$$

Consider a cyclic, depth-first traversal P'' on the subtree R. If $R \setminus S \neq \emptyset$, then there is no overflow within the subtree since there exists a path beginning at x_i that reaches all nodes in R without overflowing. Thus the overflow rate of the path on R is $n' + n''$. If $R = S$ then the overflow rate of the path is $n' + \frac{\epsilon'}{\delta} + n''$ where ϵ' is the extra overflow. Note that by definition, $\epsilon' \leq \epsilon$ since P'' is shorter than or has the same duration as S. For either case, by the overflow rate is bound by $n' + \frac{\epsilon'}{\delta} + n''$. By Equation 4.2,

$$n' + \frac{\epsilon'}{\delta} + n'' \leq \frac{m}{T} + n''$$

$$\frac{T(n'+\frac{\epsilon'}{\delta})}{T+\delta} + \frac{\delta(n'+\frac{\epsilon'}{\delta})}{T+\delta} \leq \frac{m}{T+\delta} + \frac{\delta(n'+\frac{\epsilon}{\delta})}{T+\delta}$$

$$n' + \frac{\epsilon'}{\delta} + n'' \leq \frac{m+\delta n'+\epsilon}{T+\delta} + n''$$

Since P is optimal, the direct path on R must be optimal as well.

Therefore, there exists an optimal path that is a cyclic, depth-first traversal on a subtree of G. □

A consequence of Thereom 4.2.3 is that we can compute an optimal mule route (we can identify an optimal subtree of G) in time that is pseudo-polynomial, using a dynamic programming algorithm.

It is unlikely that there is a strongly polynomial time algorithm for an exact solution, since we show that the problem is weakly NP-hard.

Hardness

We show that single mule scheduling on a tree is weakly NP-hard. Further, we show that the data gathering problem for a single mule and sensors in Euclidean (or any metric) space is NP-hard.

Theorem 4.2.4. *The data gathering problem scheduling a single mule among uniform capacity sensors on a tree is (weakly) NP-hard.*

Proof: Our reduction is from PARTITION (or SUBSET-SUM): given a set $S = \{x_1, \ldots, x_n\}$ of n integers, does there exist a subset, $S' \subset S$, such that $\sum_{x_i \in S'} x_i = M/2$, where $M = \sum_i x_i$? Given an instance of PARTITION, we construct a tree as follows: There is a node v connected to a node u by an edge of length $M/2$. Incident on v are n additional edges, of lengths x_i; the edge of length x_i leads to a node where there are exactly x_i sensors placed. Also, at node u there are M^2 sensors placed. (If one disallows multiple ($x > 1$) sensors to be at a single node w of the tree, we can add x very short (length $\Theta(1/n)$) edges incident to w, each leading to a leaf with a single sensor.) Consider the problem of computing a maximum data-rate tour in this tree, assuming each sensor has capacity $2M$. Then, in order to decide if it is possible to achieve data collection rate of $M^2 + M/2$ we need to decide if it is possible to find a subtree that includes node u (where the large number, M^2, of sensors lie) and a subset of nodes having x_i sensors each, with the sum of these x_i's totaling exactly $M/2$. (If the sum is any less than $M/2$, we fail to collect enough data during the cycle of length $2M$ that is allowed before data overflow; if the sum is any more than $M/2$, we lose data to overflow at u, which cannot be compensated for by additional data collected at the x_i nodes, since M^2 is so large compared to x_i.) □

Theorem 4.2.5. *The data gathering problem scheduling a single mule among uniform capacity sensors in the Euclidean (or any metric) space is NP-hard.*

Proof: We reduce from the Hamiltonian cycle problem in a grid graph where n points are on an integer grid and an edge exists between two points if and only if they are unit distance apart. If we place a sensor at each point with capacity n, it follows that there exists a Hamiltonian cycle in this graph if and only if there exists a data gathering solution with no data loss. □

NP-hardness for this problem also holds for any general metric space.

Approximation Algorithm

Theorem 4.2.6. *For uniform capacity sensors in fixed dimension Euclidean space, there exists a $(1/3 - \varepsilon)$-approximation for maximizing the data gathering rate of a single mule. For general metric spaces, a $(1/6 - \varepsilon)$-approximation exists.*

Proof: In order to achieve this, we approximate the maximum number of distinct sensors a mule can cover in time $c/2$, the amount of time for sensors to fill from empty to half capacity (it can be shown that one half capacity is the optimal choice). The result will be a path, to which we assign one mule to traverse back and forth. The data gathering rate of this solution is equal to the number of distinct sensors covered as a mule on a schedule with period t will collect exactly t units of data from each sensor. We denote R to be the maximum number of distinct sensors that can be covered by a path of length $c/2$. Note that R can be approximated to within a factor of $1 + \varepsilon$ in fixed dimension Euclidean space using the PTAS for orienteering [42]. In general metric spaces, R can be approximated to within a factor of $2 + \varepsilon$ [41]. Let R^* be the data gathering rate of the optimal solution. We now show that $R^* \leq 3R$.

Consider the interval of time $c/2$ in the optimal solution that has the highest data gathering rate. This is an upper bound on R^*. In this time period, we know that the number of distinct sensors visited is at most R. We also know that during this time period at most $\frac{3}{2}c$ units of data can be downloaded from any visited sensor (at most c units of data immediately downloaded and at most $c/2$ units of data downloaded after $c/2$ units of time have passed). Therefore, the total amount of data collected in the optimal solution during this period of time is at most $\frac{3}{2}cR$. Averaging the data collected over the time interval $c/2$, the data gathering rate of the optimal solution is at most $3R$. □

4.3 k-Mule scheduling

Given a budget of k data mules, we now consider the problem of maximizing the total data gathering rate of these mules. We assume the n sensors have uniform capacity, unit data rate, and unit velocity. It is important to note that even with sensors on a line, the optimal solution may not assign mules to private tours; sensors may need to be visited by multiple mules. Consider an input with two mules and sensors uniformly spaced $c/4$ apart from one another. Any time a mule makes a U-turn, it will gather only $c/2$ data from the next sensor it visits. In order to maximize the frequency of full buckets collected, we want to minimize the frequency of U-turns made. This can be done by maintaining separation of length c between the mules and

having the mules zig-zag across (nearly) the entire line (see Figure 4.3). Interestingly, this example also shows that mules can travel arbitrarily far distances.

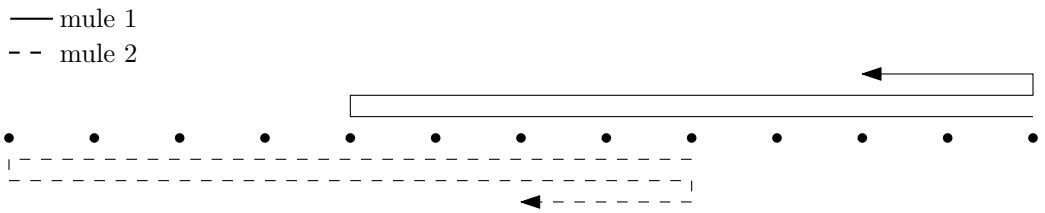


Figure 4.3. With two data mules and sensors uniformly spaced $c/4$ apart, many sensors will be visited by both mules in the optimal solution.

Sensors on a line

Theorem 4.3.1. *Given a budget of k data mules, for uniform capacity sensors on a line, there exists a 1/3-approximation for maximizing the data gathering rate.*

Proof: Similar to the case when $k = 1$, we find the maximum amount of distinct sensors that k mules can cover in time $c/2$ (it can be shown that half capacity is the optimal choice). The result will correspond to a set of disjoint intervals; we assign one mule to each interval. The duration of a cycle for each mule is the length of time a sensor fills up to capacity so no sensor is allowed to overflow. Therefore, the data gathering rate of this solution, call it R, is equal to the number of sensors covered. Note that R, the maximum amount of distinct sensors that can be covered by k disjoint intervals of length at most $c/2$, can be computed exactly in polynomial time using dynamic programming. Let R^* be the data gathering rate of the optimal solution. It follows from the same argument given for the $k = 1$ case (Theorem 4.2.6) that $R^* \leq 3R$.

□

Sensors in a general metric space

Theorem 4.3.2. *Given a budget of k data mules, for uniform capacity sensors in a general metric space, there exists a $\frac{1}{3}(1 - \frac{1}{e^\beta})$-approximation with $\beta = \frac{1}{2+\varepsilon}$ for maximizing the data gathering rate. In fixed dimension Euclidean space there exists a $\frac{1}{3}(1 - \frac{1}{e^\beta})$-approximation with $\beta = 1 - \varepsilon$.*

Proof: The proof is similar to the proof of Theorem 4.2.6. In order to approximate the maximum amount of distinct sensors that k mules can cover in $c/2$ time, we compute an orienteering path with a travel distance budget of $c/2$ on the uncovered sensors. We repeat this operation for a total of k times. In the Maximum Coverage problem, one is given a universe of elements, a collection of subsets, and an integer k. The objective is to maximize the number of elements covered using k subsets. It has been shown by Hochbaum et al.[83] that greedily choosing the set with the largest number of uncovered elements k times yields a $(1 - \frac{1}{e})$-approximation. Interestingly, Hochbaum et al. also show that using a β-approximation for covering the maximum amount of uncovered elements in each of the k rounds yields a $(1-\frac{1}{e^\beta})$-approximation. Computing orienteering k times on only the remaining uncovered sensors, we achieve a $\frac{1}{(2+\varepsilon)}$-approximation each round and therefore a $(1-\frac{1}{e^\beta})$-approximation for $\beta = \frac{1}{2+\varepsilon}$ for covering the maximum amount of sensors with k mules. Using similar arguments as the case where $k = 1$ (Theorem 4.2.6), it is now easy to see that having mules traverse the k orienteering paths back and forth yields a $\frac{1}{3}(1 - \frac{1}{e^\beta})$-approximation. □

4.4 No Data Loss Scheduling

In situations in which it is not possible for a fixed number of data mules to collect all data in the network, it is natural to increase the number of data mules and let them collectively finish the data collection task. In the no data loss scheduling problem, we seek to minimize the number of mules in order to avoid data loss. Throughout this section we assume that all sensors have unit data rate, unit velocity, and uniform capacity.

Exact Algorithm on a Line

When sensors all lie along a line, we show that the problem can be solved in polynomial time. As before, we can assume that the sensors lie at integer coordinates so that, by the same argument as in Lemma 4.2.1, the mules in an optimal solution follow periodic schedules.

Lemma 4.4.1. *For the minimum cardinality data mule problem with no data loss, if the sensors have uniform capacity and lie on a line, there is an optimal schedule in which all mules follow periodic cycles, zigzaging within disjoint intervals, each with length at most $c/2$.*

Proof: The proof is by induction on the number of sensors. If there is only one sensor, then one mule is enough and it remains stationary at the sensor; this is a trivial zigzag schedule.

Suppose we have n sensors and consider the schedule of one optimal solution using k^* mules. The mule i follows along a periodic schedule C_i visiting a set of sensors S_i. Without loss of generality we can assume the mules in this optimal schedule do not visit any point to the left of the leftmost sensor. Consider the leftmost sensor, p_1, and the mules that collect data from it. If p_1 is visited by more than one mule, we will find another optimal schedule in which p_1 is visited by only one mule. To see this, we first fix a time t_1 when p_1 is visited by some mule, denoted as the first mule m_1. Suppose the next visit to p_1 is by a different mule, say m_2, at time t_2. Clearly, at the time when m_1 visits p_1, m_2 is to the right of p_1. Later, at the time when m_2 is visiting p_2, m_1 is to the right of p_1. By continuity of motion, there must be a time in $[t_1, t_2]$ such that m_1 and m_2 meet. At the intersection we can swap the motion plan of m_1 and m_2. So m_1 turns back to visit p_1 and m_2 turns back and follow the original motion plan of m_1. This modification does not change the data rate, so the modified schedule collects all data and remains optimal. Similarly, if p_1 is later visited by another mule m_3, we perform the same swap. This way p_1 is exclusively visited by one mule, m_1.

Since p_1 is only visited by a single mule, m_1, the schedule C_1 does not visit any node that is more than distance $1/2$ away from p_1 to the right (otherwise m_1 does not have enough time to go back to p_1 before its bucket becomes full). Thus we can assume without loss of generality that C_1 is a simple zig-zag tour on an interval I_1 of length $c/2$ with p_1 as the left endpoint. All sensors in this interval can be collected by this mule without data loss. Thus, we can modify the schedule for all other mules such that they do not need to visit any nodes in I_1. Clearly, k^*-1 mules are sufficient to collect data from the remaining sensors with no data loss. Since there are at most $n-1$ sensors to be covered, by the induction hypothesis there is a schedule of k^*-1 mules covering disjoint intervals. Together with the first mule, this is the disjoint schedule whose existence is claimed. □

By the above structural lemma, we can use a simple greedy algorithm to minimize the number of data mules necessary to collect data, without loss, for sensors on a line: Starting at the leftmost sensor, schedule a mule to zigzag within an interval of length $c/2$ whose left endpoint is the leftmost sensor, and then continue to the right, adding further intervals of length $c/2$ until all sensors are covered. This is an $O(n)$ algorithm for n (sorted) sensors.

Hardness

Even when the sensors lie on a tree, the problem of minimum cardinality data mule scheduling with no data loss is already NP-hard.

Theorem 4.4.2. *For uniform capacity sensors on a tree, the problem of minimum cardinality data mule scheduling with no data loss is (strongly) NP-hard.*

Proof: We reduce from 3-PARTITION. Given a multiset, $S = \{x_1, x_2, \cdots, x_{3n}\}$, of $3n$ integers with total sum M, 3-PARTITION asks whether there is a partition of S into n subsets, $S_1, \ldots, S_n$, such that each subset sums to exactly $B = M/n$. It is known that we can assume that the integers x_i satisfy $B/4 < x_i < B/2$, so that each subset S_i must consist of a triple of elements ($|S_i| = 3$). We create an instance of a star having a hub (center node) incident on $3n$ edges ("spokes") to $3n$ sensors, with edge lengths equal to x_i. Each sensor has capacity $2M/n$. Thus if there is a partitioning of S into triples of integers that each sum to M/n, then one (unit-speed) mule can traverse each corresponding 3-spoke subtree in time exactly $2M/n$, resulting in no data loss using n mules. On the other hand, any solution using exactly n data mules and having no data loss determines a valid partition of S into triples S_i. Thus, the 3-PARTITION instance has a solution if and only if n data mules suffice. □

For the general case, with sensors at points of a metric space or in a Euclidean space, the problem of determining the minimum number of data mules necessary to collect all data is also NP-hard. This can be seen by using a similar reduction from Hamiltonian cycle, as in Theorem 4.2.5.

Approximation Algorithm

It is tempting to think that an optimal solution will allocate each mule to cover an exclusive set of sensors S', that are not covered by other mules. We denote such a set of tours as a *private tour set* on S'. However, the following example shows that this is no longer the case when sensors lie in the plane. Consider $n > 2$ sensors placed on a circle, uniformly spaced with adjacent sensors at (Euclidean) distance exactly $c - 1/n$. The convex hull of these sensors is a regular n-gon of perimeter $n(c - 1/n) = nc - 1$. The optimal solution would use $n - 1$ mules, with each mule touring periodically at constant speed (1) along the boundary of this n-gon, with time/distance separation of exactly c between consecutive mules. This ensures that each sensor is visited exactly when its storage (bucket) becomes full. However, any

solution using private tours will have to use n mules, since no mule can use a private tour to cover two or more sensors (since it would have length at least $2c - 2/n > c$).

While it may be that no private tour set is optimal, we now argue that the optimal schedule using only private tours is provably close to optimal (in terms of minimizing the number of data mules). Denote by k^* the minimum number of cycles, each of length at most c, to cover all nodes, which is denoted as a *light cycle cover*. And denote by m^* the minimum number of data mules required to collect all data.

Lemma 4.4.3. $m^* \leq k^* \leq 2m^*$.

Proof: First, note that using k^* mules, each traversing a (private) light cycle, results in all data being collected; thus, $m^* \leq k^*$.

Now consider an optimal schedule of m^* data mules. Mule i moves along a schedule C_i. Consider any particular time t. Each sensor j is visited by at least one mule. We assign it to the mule that visits it first, i.e., at the earliest time after t. We know this time is at most c, since no data is lost at sensor j. Thus, consider mule i, at current position p_i (at time t) and all the sensors along C_i that are assigned to it. They all lie on a path (along C_i) of length at most c. Let s_i be the sensor furthest away from p_i, measuring distance along C_i. Let γ_i be the corresponding path along C_i, from p_i to s_i. Let b_i be the midpoint of this path. Place a clone of mule i at point s_i, and create two private cycles for mule i and his clone: one cycle goes from p_i to b_i along γ_i, then returns to p_i directly (along a shortest path or a straight segment), the other goes from b_i to s_i along γ_i, then returns to b_i directly. Mule i traverses the first cycle; his clone traverses the second cycle. Do this for all mules.

We have doubled (via cloning) the number of mules, but now each mule/clone has a private cycle, of sensors assigned only to it, and these cycles are each of total length at most c. Thus, this is a valid solution to the light cycle cover problem. Thus, the minimum number of light cycles, k^* is no greater than $2m^*$. □

By the above lemma, an α-approximation for the minimum light cycle cover gives a 2α-approximation for the minimum number of data mules. Arkin *et al.* [14] gave a 6-approximation algorithm for the minimum light cycle cover problem; thus, we have a 12-approximation for minimum data mule scheduling. This is summarized in the following theorem.

Theorem 4.4.4. *For uniform capacity sensors within a general metric space or in the Euclidean plane, computing the minimum number of data mules to collect all data is NP-hard. There is a polynomial-time* 12*-approximation algorithm for sensors in a general metric space.*

4.5 Different Capacities

We now consider both the k-mule scheduling problem and the no data loss scheduling problem on n sensors with potentially different sensor storage capacities. Each sensor has unit data rate. The result for the k-mule scheduling problem obviously holds for the single mule problem (i.e. when $k = 1$).

k-Mule Scheduling

Lemma 4.5.1. *With m groups of sensors, each group having the same storage capacity, optimally solving each group independently and taking the solution with the highest data gathering rate yields a $O(1/m)$-approximation to the k-mule scheduling problem.*

Proof: Let $r(\cdot)$ be the data gathering rate of a solution. Let OPT_i be the optimal solution to group i and let OPT be the schedule with highest data rate. $r(OPT) \leq \sum_{i=1}^{m} r(OPT_i) \leq m \cdot \max_i\{r(OPT_i)\}$. The first inequality is from the following observation. Consider the optimal schedule OPT and modify it such that we only visit the nodes in group i. This is obviously a solution for collecting data from group i and thus has data rate no greater than $r(OPT_i)$. □

Let c_{max} and c_{min} be the storage capacities of the largest and smallest sensors respectively. We round the storage capacity of each sensor down to its nearest power of two. Doing so, we create m groups of sensors where m is at most $\log(\frac{c_{max}}{c_{min}})$. Note that m may be significantly smaller than $\log(\frac{c_{max}}{c_{min}})$. In the rounding down process, the storage capacity of each sensor is at most halved, thus the optimal solution on the new sensors has data gathering rate of at least $1/2$ of the same solution before rounding. We approximate the optimal solution to each of the groups within a constant factor and choose the one with highest data gathering rate. By Lemma 4.5.1, we have the following.

Theorem 4.5.2. *By rounding down the sensor capacities into $m \leq \log(\frac{c_{max}}{c_{min}})$ groups, the group with highest data gathering rate has rate at least $O(1/m) \cdot r(OPT)$ where OPT is the optimal solution to the k-mule scheduling problem.*

No Data Loss Scheduling

Theorem 4.5.3. *By rounding down the sensor capacities into $m \leq \log(\frac{c_{max}}{c_{min}})$ groups and solving each group independently, at most $O(m) \cdot |OPT|$ mules are used in total, where $|OPT|$ is the minimum number of mules needed to avoid data loss.*

Proof: Using the same rounding technique as the previous section, we again obtain m groups of sensors with $m \leq \log(\frac{c_{max}}{c_{min}})$. In the rounding down process, the capacity of any sensor is at most halved. Thus, the optimal solution on the rounded down sensors requires at most two times the number of mules as the optimal solution to the original set of sensors. Let $|OPT_i|$ be the minimum number of mules needed for no data loss to occur in group i and let $|OPT|$ be the number of mules in the optimal solution. Since $|OPT| \geq |OPT_i|$ for $1 \leq i \leq m$, we have that $m \cdot |OPT| \geq \sum_{i=1}^{m} |OPT_i|$. Approximating $|OPT_i|$ within a constant factor for all i, we use $O(m) \cdot |OPT|$ mules.

□

4.6 Practical Algorithms

Our discussion on mule scheduling algorithms have focused on getting the best theoretical approximations. Although some of the algorithms (especially the one for the case of a line) are practical and implementable, some of the other algorithms mentioned depend on solving the orienteering problem or k-TSP problem. These algorithms are too complicated for practical implementation. In this section we try to find heuristic algorithms that are easily implementable. We also discuss upper and lower bounds that can be used to estimate the optimal solution and provide comparisons for the heuristic algorithms. These are tested in our simulations.

4.6.1 Single Mule Scheduling

Lower Bounds

For maximizing data rate by a single mule, any algorithm giving a feasible solution will be a lower bound. We run approximation algorithms of TSP on the entire point set, then traverse the TSP in one direction, searching all windows of variable length w in the tour. Each window starts at a distinct point p in the TSP and is pinned to the point that is farthest away from p along the TSP and still within length w. Thus, these windows will always be pinned to two points in the TSP. We can then traverse these interval back and forth and determine the data gathering rate. For example, suppose $w = 1$ and suppose this interval covers r nodes, then cycling through the r nodes gives data rate of at least $r/2$. We will refer to this algorithm as 'shifted window along TSP'.

Upper Bounds

To get an upper bound on the highest data rate by a single mule, we start from the optimal schedule to discuss how to estimate it from above.

Let S be a set of nodes in the plane and let P be the optimal schedule with data rate of R. If the length of P is greater than one, then there must be a unit length interval along P such that the data gathering rate is at least R, by pigeon hole principle. Now we take such a unit length interval and denote it P'. Suppose the number of nodes covered (with repeats) by P' is h and the set of nodes is H. $h \geq |H|$. Note that in a unit length interval, a node can be considered to be visited at most twice because if a node is visited three times, the second visit can always be skipped and no data will be lost. Thus each node in H is visited at most twice in P'. $h \leq 2|H|$. In addition, $R \leq h$ since we collect at most a full bucket of data at each node. Thus we have an upper bound $2|H|$ on the optimal data rate. Since we do not know the optimal schedule, we do not know H either and this upper bound is not immediately useful. In the following we do further relaxation and find an upper bound on $|H|$.

Recall that the set of nodes in H can be connected by a path of length at most 1. Therefore if we can find the maximum number of nodes visited by a path of length at most 1, then we have an upper bound on $|H|$. This is an instance of an orienteering problem and is still hard to solve. But we know that this set H is fully contained in a square of unit side length. The distance between the leftmost and rightmost point in H is at most 1. The same can be said in the vertical direction. Thus, if we lay out a grid over our point set with cell side length of $1/2$, we know that there exists a 3×3 square in the grid that contains any unit square, and therefore we can find such a square that contains H.

We enumerate all 3×3 squares in the grid and on each square we run Kruskal's algorithm until the sum of the edge weights exceeds 1. We choose the 3×3 square that maximizes the number of points picked up. Call this point set K. By the aforementioned argument, we know that $|K| \geq |H|$. Since points in H can be repeated at most twice in the optimal schedule, we conclude that the optimal data gathering rate is at most $2|K|$.

4.6.2 No Data Loss Scheduling

By Lemma 4.4.3, a solution to the minimum light cycle cover is a valid solution to our minimum cardinality mule scheduling problem with no data loss and is a natural upper bound. Further, if we have a lower bound L on the minimum light cycle cover problem, $L \leq k \leq 2k^*$ then $L/2$ is a lower bound on the number of mules needed.

Thus in the following we discuss upper and lower bounds on the minimum light cycle cover problem.

Besides the algorithm by Arkin *et al.* [14], The following algorithm provides a solution (upper bound) on the optimal number of data mules. Take the minimum spanning tree of the set of sensors and remove all edges of length 1 or more. This leaves a number of connected components $S_1, S_2, \cdots, S_k$. For each connected component S_i, we can duplicate each edge to form a Euler tour. Remove duplicate occurrences of any node on this tour and we get a cycle C_i to visit the nodes in S_i. If C_i has length ℓ_i, we use $\lceil \ell_i \rceil$ mules along C_i with uniform timewise separation. This algorithm is called 'chopped MST'.

4.7 Simulations

In this section we show the results of implementations of the upper and lower bounds that were discussed in the previous section

4.7.1 Single Mule Simulations

We generated one hundred different point sets. For each instance we generated 500 uniformly distributed points in a 5×5 square and computed shifted window along TSP with window lengths between 0.5 and 1.0 in increments of 0.05. We compare the results to the grid based upper bound. Figure 4.4 shows the results. With window length 0.7, the worst ratio over all point sets was roughly 0.18353. i.e. the data gathering rate of a mule following our heuristic was 0.18353 times as fast as the rate produced by the grid based upper bound. In order to compare to a sparser point set, we ran the same experiment in a 10×10 square (see Figure 4.5) and noticed improved results.

We also computed shifted window along TSP on the 4,663 cities of Canada [1] (see Figure 4.6) with varying window lengths. The point set was scaled down so that the average distance between any two points is 1.

4.7.2 No Data Loss Simulations

First, we generated one hundred point sets, each set containing 200 points. For each set, we generated 10 uniformly distributed points in a 5×5 square. Centered at each of these points we generated 20 uniformly distributed points in a 1×1 square and discarded the center point. Over all point sets, we compared the number of mules used in a chopped MST solution to the number of mules used in a light cycles

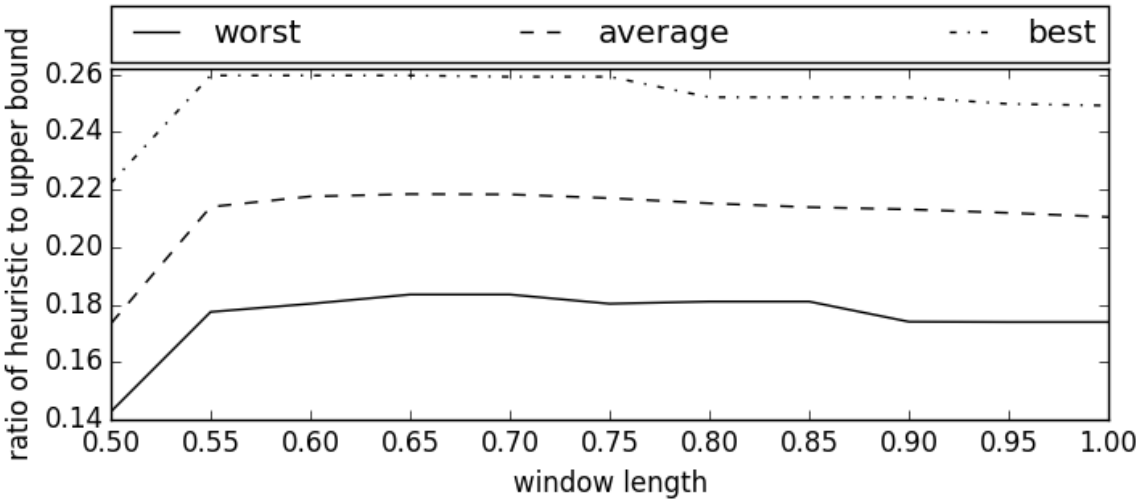


Figure 4.4. Shifted window along TSP of uniformly distributed point sets in a 5 × 5 square

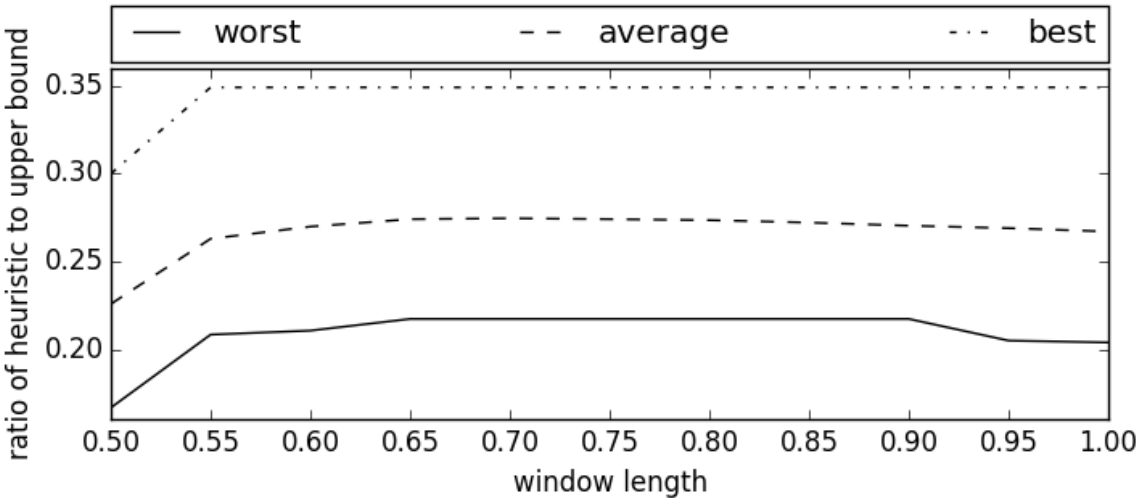


Figure 4.5. Shifted window along TSP of uniformly distributed point sets in a 10 × 10 square

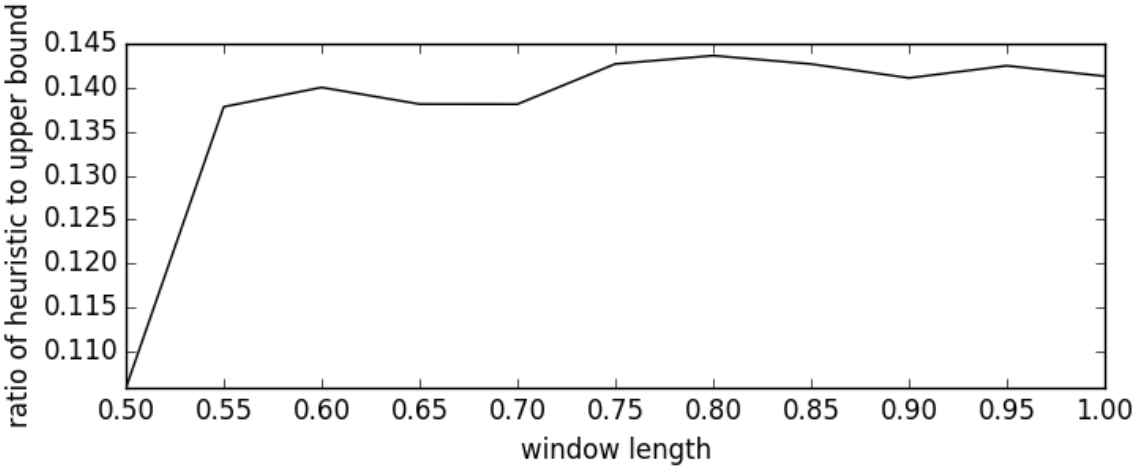


Figure 4.6. Shifted window along TSP of 4,663 cities of Canada

solution. The results can be seen in table 4.2. In addition to the results in this table, we found that the point set with highest ratio of number of mules used in a light cycles solution to number of mules in a chopped MST solution was 1.46154. The lowest ratio was 1.18868. We repeated this experiment for a 10×10 square (see table 4.3). The highest and lowest ratios were 1.45652 and 1.16949 respectively.

	Chopped MST	Light Cycles
Best	39	53
Worst	55	69
Average	47.45	62.75

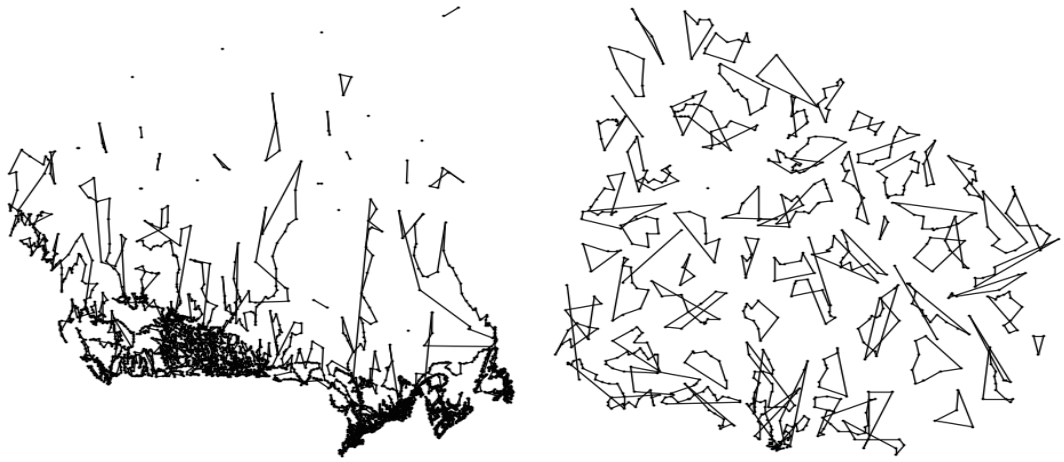
Table 4.2. Number of mules used in chopped MST vs. light cycles in 5×5 square

	Chopped MST	Light Cycles
Best	45	62
Worst	59	75
Average	52.04	67.58

Table 4.3. Number of mules used in chopped MST vs. light cycles in 10×10 square

We computed chopped MST on the cities in Canada (fig 4.7). The points were scaled down so that the average distance between any two points is 10. 602 mules

are enough to cover these points. Note that there are edge crossings in this solution because we used a two approximation to compute TSP.



(i) (ii)

Figure 4.7. (i) Chopped MST on 4,663 cities of Canada. 602 mules used; (ii) Approximation of minimum light cycles on 734 cities of Uruguay. 85 mules used.

We also tested the approximation algorithm to minimum light cycles. This algorithm runs significantly slower than chopped MST. Therefore, we used the 734 cities of Uruguay [1] (fig 4.7). The point set was scaled down so that the average distance between any two points is 1. 85 mules are enough to cover these points. For comparison, chopped MST produces a TSP on the entire point set and uses 50 mules. We also scaled the Uruguay point set so that the average distance between any two points is 10. This time the approximation to minimum light cycles outperformed chopped MST using 436 and 470 mules respectively.

4.8 Conclusion

Our exact and approximation algorithms for single mule scheduling, k-mule scheduling, and no data loss scheduling represent the state of the art results on mule scheduling problems and greatly deepens our understanding of vehicular routing with constraints. In the future, we would like to see if the approximation ratios can be improved, especially with sensors in Euclidean space.

Chapter 5

The Shortest Separating Cycle Problem

5.1 Introduction

In a distributed sensor network, various data is generated at different sensors [121, 127]. One challenge that arises considers how to provide a fully distributed storage and retrieval algorithm to allow mobile users at any location query for such data. Consider an application in which sensors are installed at parking spots to detect if the spot is empty, while mobile users roaming around in the city are in need of such information. The solution of always delivering the query to the data source may suffer from a single point of failure and traffic bottleneck. Therefore a natural solution is to adopt geographical hashing. Each piece of detection data is hashed to a storage sensor. While a piece of data i is delivered from its source p_i (where it is generated) to its storage/hash location q_i using multi-hop routing, it is convenient for all the nodes on the relay path to also cache the data item. For a mobile user seeking data of a particular type, the user can issue a query which only needs to visit a node that has cached the data. However, the routing path connecting p_i and q_i is determined by the specific routing algorithm in use which may be unknown for nodes not on the path. Nevertheless if the user query travels along a tour that separates p_i and q_i, the query will surely hit the cached data. In this retrieval scheme one can easily query for a collection of data that are generated from multiple sensors $p_1, p_2, \cdots p_n$, as long as the query follows a tour that separates each pair of nodes p_i and its corresponding hashed storage node q_i. Finding the shortest separating cycle is natural, as the shortest tour minimizes energy consumption and delay.

Our Problem

Given a set $P = \{(p_i, q_i) | 1 \leq i \leq n\}$ of pairs of points in the plane, we seek a *shortest separating cycle* T, a tour where for every pair of points, one point is inside the tour and the other is outside. Each pair (p_i, q_i) can be represented by a line segment connecting p_i and q_i. Therefore each segment is cut by the tour an odd number of times. Throughout this section, we use whichever interpretation is more intuitive.

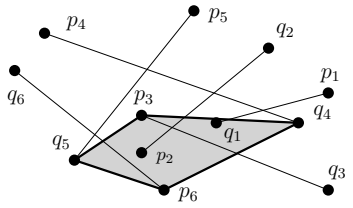


Figure 5.1. The shortest separating cycle problem.

In computational geometry, this problem is related to many traveling salesman problem (TSP) variants, including the red-blue separation problem, TSP with neighborhoods, and one-in-a-set TSP (also known as group TSP), that have been well studied [17, 111]. All of these problems are known to be NP-hard in the Euclidean plane as they all contain the classical TSP as a special case. The shortest separating cycle problem is different from any of these problems. In the red-blue separation problem, given a set of red and blue points in the plane, the aim is to find the shortest tour that separates the blue points from the red points. In our problem, the points in the pairs need to be separated but they are not assigned colors. Thus part of the challenge is to determine which point of each pair is inside the tour and which one is outside.

In TSP with neighborhoods (TSPN), given a set of regions, the goal is to find the shortest tour that visits each region. One may attempt to connect a line segment for each pair in our input and apply an algorithm for TSP with neighborhoods where the neighborhoods are line segments. However, this does not necessarily give a valid solution since the TSP with neighborhoods solution might reflect on the edge and not enclose an endpoint in its cycle such as in Figure 5.2.

For the one-in-a-set TSP, we are given a collection of sets and the problem asks for the shortest tour that visits at least one element in each set. Any one-in-a-set TSP solution to our input can be easily modified to become a separating cycle. However, a separating cycle does not need to visit every point it is including or excluding so the one-in-a-set TSP solution may be excessively long. An example of this can be

seen in Figure 5.3.

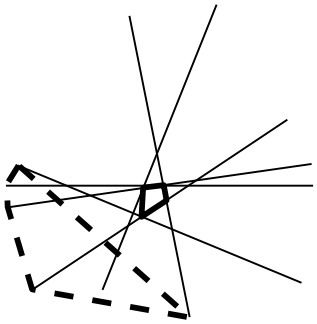


Figure 5.2. The TSP with neighborhoods solution (solid) is a much shorter tour than the shortest separating cycle (dashed). It is also not a valid separating cycle.

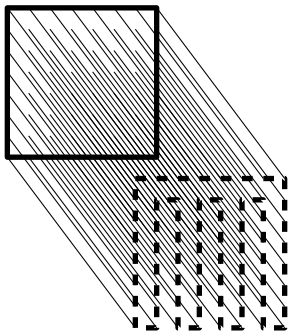


Figure 5.3. The one-in-a-set TSP solution (dashed) is a much longer tour than the shortest separating cycle (solid).

Our Results

In this section we are the first to study the shortest separating cycle problem and provide both hardness and approximation results. In particular, we consider special cases where the orientation of the input pairs, the distance between the input pairs, and the configuration of the domain are restricted. We vary the size of the square board the input points are confined within as well as the range of orientations the input pairs have from strictly horizontal and/or vertical to any orientation. Some cases have additional restrictions such as how far each pair of points are from each

Board Size	Unit Length Horizontal	Unit Length Horizontal &Vertical	Unit Length Arbitrary Orient.	Constant Length Arbitrary Orient. Points on Grid
$2-\varepsilon$	in P	4-approx	NP-hard	NA
$M = O(1)$	$O(1)$-approx	$(M^2+1)/4$-approx	Hard to approx	NA
n	$O(1)$-approx	$O(1)$-approx	Hard to approx	$O(1)$-approx

Table 5.1. Approximation algorithm and hardness results for different settings.

	Unit Length	Any Length
Arbitrary Orientation	APX-hard	2-inapprox

Table 5.2. Additional hardness results.

other and whether or not the input points must lie on a grid. The results are summarized in Tables 5.1 and 5.2.

In general, despite the apparent similarity and connection to many other TSP variants which have easy constant-factor approximation results, the shortest separating cycle problem is a lot harder. Many ideas that were used in typical TSP algorithms are not applicable here. Indeed, we show that the problem is hard to approximate for a small constant factor and is APX-hard even in very restricted scenarios such as unit length segments. We provide a polynomial time algorithm when all pairs are unit length horizontal segments inside a square board of size $2-\varepsilon$. We provide approximation algorithms for unit length horizontal or vertical segments or constant length segments on points laying on a grid. These scenarios are of particular interest to the application setting in a sensor network. Last, for arbitrary pairs we have an $O(\sqrt{n})$-approximation algorithm and an $O(\log n)$-approximation algorithm for the shortest separating planar graph problem, in which the objective is to compute an embedded planar graph of minimum total edge length so that the two endpoints of each pair are in different faces.

Related Work

TSP. The traveling salesman problem is one of the most well known geometric problems in history. It is one of the first problems known to be NP-hard [69, 118]. In a metric setting Christofides provided a 3/2 approximation algorithm [45]. In the Euclidean setting, the problem is known to admit a PTAS, independently shown by Arora [17] and Mitchell [111].

TSP with Red Blue Separation. The red blue separation problem in the plane admits a PTAS (by [111] or by [18]).

TSP with Neighborhoods. TSP with neighborhoods was first studied by Arkin and Hassin [13] in which $O(1)$-approximation algorithms were developed when the neighborhoods are translates of a convex polygon or when the neighborhoods are unit disks. For general (nondisjoint) connected neighborhoods, an $O(\log n)$-approximation algorithm is known where n is the number of neighborhood regions [109, 76], and it is NP-hard to approximate within a $2-\varepsilon$ ratio [123]. For fat regions of bounded depth, there is a PTAS [112] (even in doubling metrics [36]), while for general connected regions of bounded depth, or for convex regions, an $O(1)$-approximation is known in two dimensions [113].

One-of-a-set TSP or Group TSP. The one-of-a-set or group TSP is the TSPN in which the neighborhoods are discrete sets of points (and thus disconnected). Safra and Schwartz [123] show the 2D problem is NP-hard to approximate to within any constant factor; for groups that are sets of k points, they also give approximation lower bounds ($\Omega(\sqrt{k})$). Slavík [131] gives a $(3/2)k$-approximation, based on linear programming methods.

5.2 Hardness

The shortest separating cycle problem is NP-hard by a trivial reduction from the traveling salesman problem (TSP). For any TSP instance with cities at location w_i, we place a pair of points p_i, q_i very close to each w_i. In order to separate the points, the tour will need to visit each city w_i. Thus the shortest separating cycle problem is as hard as TSP. In the following we show stronger results that the problem is hard to approximate even for unit length segments.

5.2.1 APX-hard For Unit Length Segments

Theorem 5.2.1. *Finding the shortest separator is APX-hard for an arbitrary board size if the segments are all unit length but are allowed to have any orientation.*

Proof: Our reduction is from MAX NAE 2-SAT, the problem of maximizing the number of 2-SAT clauses in which exactly one of the literals is true, which is APX-hard [21]. We reduce a MAX NAE 2-SAT problem whose optimal solution satisfies t clauses to a shortest separating cycle problem whose optimal solution is a tour of length $tm^2n^2 + O(mn)$, where n is the number of variables and m is the number of

clauses. The $O(mn)$ term is overhead from gadgets other than the clause gadgets and the clause gadgets are densely packed grids of points that must all be visited individually if the corresponding clause is unsatisfied, but can be grabbed using a (much shorter) loop all at once if the corresponding clause is satisfied.

Let $\phi(x_1, \ldots, x_n) = (y_1 \vee z_1) \wedge (y_2 \vee z_2) \wedge \cdots \wedge (y_m \vee z_m)$ be a MAX NAE 2-SAT instance, where each y or z is either x_i or $\neg x_i$ for some i. Each variable x_i corresponds to a point X_i; if X_i is inside the cycle, we will interpret x_i as TRUE; if not, we'll interpret it as FALSE. Each X_i is the endpoint of some unit segment whose other endpoint $\bar{X}_i$ corresponds to $\neg x_i$.

In our construction, multiple points may correspond to the same variable. We first describe a gadget that ensures that these points are either all inside or all outside of the separating cycle. Given any two points X and Y at distance d and a finite set of points to avoid (possibly from other gadgets) Z_1, ..., Z_k, we can guarantee that X and Y are either both inside or both outside the tour, using at most $d + 10$ other segment endpoints none of which is any of the Z_i, using a gadget as in Figure 5.4.

That is, we first construct a path of an even number of vertices (from X' to Y' in the diagram) such that the displacement between Y' and X' is the same as the displacement between X and Y. Choose a point X' on the unit circle centered at X such that none of the points in the path starting at X' is any of the Z_i; since there are only finitely many pairs of a point on the path and a Z_i and each pair rules out only one X', there does exist such a point on the unit circle. Then X and X' are on opposite sides of the separator, so X and the next point are on the same side of it, and so on, so X and Y are on the same side of the separator, as desired.

Finally, we will construct a clause gadget for a clause $y_i \vee z_i$: choose a point Y and ensure by the above gadget that Y is in the cycle iff y_i is TRUE. Choose $1/\varepsilon$ points at distance 1 from Y, each at distance ε^2 from the previous one along the unit circle centered at Y, such that all of them are within ε of the point $Y + (0, 1)$, and such that none is used elsewhere in the construction. By connecting them to Y, guarantee that they're in the loop iff Y isn't. For each of those points $\bar{Y}$, choose $1/\varepsilon$ points at distance 1 from $\bar{Y}$, each at distance ε^2 from the previous one along the unit circle centered at $\bar{Y}$, such that all of them are within ε of the point $Y + (1, 1)$, and such that none is used elsewhere in the construction. By connecting them to $\bar{Y}$, guarantee that they're in the loop iff Y is. That gives a grid of $1/\varepsilon^2$ points, about equally spaced within a square of side length ε, that are all in the loop iff Y is.

Now, choose a displacement r of length at most ε^3 such that no pair of points yet defined are at distance r. Define $\bar{Z}$ to be $Y + r$, ensure by the above gadget that Y is in the loop iff y_i is, and make a new copy of everything constructed so far in the clause gadget shifted by r (see Figure 5.5).

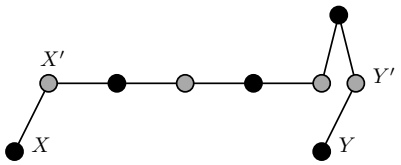


Figure 5.4. A gadget that ensures the parity of X and Y are the same. On the chain each adjacent pair of points is a pair of points in the input P. All points in gray are either all inside the cycle or outside the cycle.

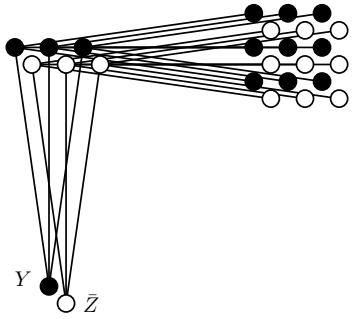


Figure 5.5. A clause gadget.

If Y and Z are both enclosed or both excluded by the separator, then the tour must separate each pair of a point and its displaced version in that gadget, so the tour must pass within ε^3 of each point in that grid of $1/\varepsilon^2$ points in a square of side length ε, requiring a total length of at least $1/\varepsilon - O(1)$ just within that gadget. Conversely, a length of $1/\varepsilon + O(1)$ suffices to separate the points in such a gadget, just by tracing along the unit circles at distance 1 from each point $\bar{Y}$. Also, if a clause is satisfied, then the tour can enclose all the points in the densely packed square at once by traversing its boundary, in length at most ϵ. Also, the total number of points not in the clause gadgets' densely packed squares is $O(nm)$, so if t clauses are satisfied, the total length of the optimal loop is between $r/\varepsilon - O(1)$ and $1/\varepsilon + O(nm)$. Set $1/\varepsilon = \Omega(n^2m^2)$; then any approximation to the length of the optimal loop is an approximation to MAX NAE 2-SAT, as claimed. □

5.2.2 Inapproximability for Any Length Segments

Theorem 5.2.2. *The shortest separating cycle problem with no restrictions on the distance and orientation of input pairs is NP-hard to approximate better than a factor of 1.3606. It is hard to approximate better than a factor of 2 assuming the unique games conjecture.*

Proof: Our reduction is from minimum vertex cover. Given a graph $G = (V, E)$, the goal of the minimum vertex cover problem is to find a minimum cardinality subset of vertices $V^* \subseteq V$ such that every edge in E is incident to at least one vertex in V^*.

To translate this graph to the plane, we place each vertex in V along a circle with a center v' and designate another location v'' at a distance $\Omega(nm)$ from v'. For every edge $e(u, v) \in E$, we place m endpoints in a $\sqrt{m} \times \sqrt{m}$ grid pattern in a $\sqrt{m} - 1$ square around u and their corresponding endpoints overlapping a single point u_e near the grid. Another grid of m endpoints is placed in a similar grid around v with their corresponding points overlapping on a point v_e. Finally, a u_e and v_e are connected by a segment. Therefore, for every gadget that represents an edge, exactly one set of grid points must be inside the cycle.

For every vertex $v \in V$, we place m endpoints overlapping a single point at v' and their corresponding endpoints in a $\sqrt{m} \times \sqrt{m}$ grid pattern around v. Finally, v' and v'' are connected by a long segment. The result is a "wheel" composed of vertices and edges in G with additional "spokes" towards the center, a hub at v', and a large arm from v' to v''.

Note that collecting all endpoints associated with an edge at the hub or any of the vertices while excluding other points involves navigating between different sets of grids. Since points in a grid are unit distance away from each other, a path only

collecting one grid of points has a length of at least m. Collecting the point at v'' would add a length of $O(nm)$ to the cycle. This additional cost is so prohibitive, it must be avoided. So the point at v' is collected and collecting any points at the vertices must navigate between sets of grids. Therefore, if the cycle collects any additional points at the vertices, it has to pay a cost of m for every vertex it visits. The optimal solution visits the minimum number of vertices in V while also separating all pairs of points which is a minimum vertex cover of G. The parity of the segment chains ensure that when one point of a grid is collected, all points in that grid must be collected.

The length of the optimal solution to this separating cycle instance is $O(|OPT|m+ n\sqrt{m})$ where $|OPT|$ is the size of the optimal solution of the corresponding vertex cover problem. If $m \geq n$, then the cost of navigating between grids at the vertices of V dominates the cost of the optimal solution. The rest of the cost, $O(n\sqrt{m})$ is from traveling between vertices and collecting v'.

Now we assume that we are given a δ-approximation to our construction of the shortest separator problem. To convert this solution into a solution for the corresponding vertex cover problem, any grid points collected at a vertex translates to a vertex selected in the vertex cover. This means that any additional vertex above the optimal solution of the vertex cover problem translates to an additional $O(m)$ length in the given approximation of the shortest separator instance. This completes the L-reduction and means that our problem with varying length segments cannot be approximated within ≈ 1.3606 unless $P = NP$. This problem cannot be approximated with a factor better than 2 assuming the unique games conjecture.

□

5.3 Algorithms

We describe exact and approximation algorithms for the shortest separating cycle problem under different scenarios.

5.3.1 Board Size $2 - \varepsilon$, Horizontal Unit Segments

In this section we show a polynomial algorithm for horizontal unit length segments inside a square board of size $2 - \varepsilon$. Most of the challenge is to prove structures of the optimal solution, denoted as the polygon P.

Lemma 5.3.1. *For the optimal polygon P, P and $P' = P + (1, 0)$ (that is, P translated to the right by one unit length) have disjoint interiors.*

Proof: Suppose otherwise, that the intersection of P and P' is not empty. We take R to be one connected component of $P \cap P'$. For any point p in R, $p \in P$ and $p + (-1, 0)$ are both inside P. See Figure 5.6 for an example. Take any segment $p_i q_i$ (with the convention that p_i is to the left of q_i). p_i and q_i cannot both be in the interior of P – for P to be a valid separating cycle. Thus for any q_i inside R, it must stays on the boundary of R. Further, the boundary of R is then formed by pieces of the boundaries of P and of P', because any point in the interior of both of them is in the interior of their intersection.

If there is any convex vertex x on the boundary of R such that the boundary segments on either side of it (to points w and y) come from the same one of P and P' (without loss of generality, P), we can replace the boundary segments wx and xy of P with the segment wy. This shortens the perimeter of P by the triangle inequality. And we argue that after the shortcut, the cycle still separates all pairs. To see that, recall that all input points of $\{p_i, q_i\}$ inside R must be on the boundary. Thus the only point removed would be x and x in this case is a right endpoint. Since $x + (-1, 0)$ is inside P, x is on the boundary of P and is not considered to be inside P. Thus after the shortcutting x and $x + (1, 0)$ are still separated. This will lead to a contradiction.

By the argument above, for every convex vertex on the boundary of R, the boundary of R comes from P on one side and P' on the other side.

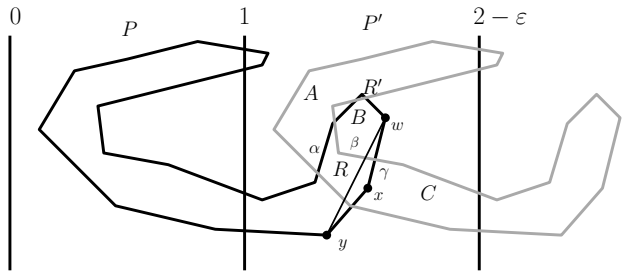


Figure 5.6. P and $P' = P + (1, 0)$ have disjoint interiors.

R has positive area, so it has at least three convex vertices. Thus there are at least two disjoint segments of the boundary of R that come from P and at least two from P'. One of the latter segments separates R from $R + (-1, 0)$ in P, leaving at least three other consecutive segments of the boundary of R. Denote these segments as α, β, and γ, separated by convex vertices, such that α is part of the boundary of P, β is part of the boundary of P' and doesn't separate R from $R + (-1, 0)$ in P, and γ is part of the boundary of P. Let A be the part of P' that is separated from

R by α, B be the part of P that is separated from R by β, and C be the part of P' that is separated from R by γ. See Figure 5.6.

If either A or C (say, A) has no points with x-coordinate greater than $2-\varepsilon$, then we'll add A to P and remove $A-(1,0)$ from P. This doesn't increase the total length of the boundary of P, but it makes a convex vertex of R (say, the one separating α from β) having its boundaries on both sides come from P, which is a contradiction to the conclusion of the previous paragraph. Thus both A and C must contain points with x-coordinate greater than $2-\varepsilon$.

Similarly, if B has no point with x-coordinate less than 1, we can remove B from P and add $B+(-1,0)$. So B has a point with x-coordinate less than 1 and A and C both have points with x-coordinate greater than $2-\varepsilon$.

We now claim that B intersects at least one of A and C. If not, it's contained in a region bounded by A, R, C, and the line $x = 2-\varepsilon$ — both A and C intersect $x = 2-\varepsilon$. And this region is entirely contained in the region $x \geq 1$, contradicting the claim that B had a point with x-coordinate less than 1.

Therefore, there are two regions R and R' in the intersection of P and P', and the region B' between them is contained in the region $x \geq 1$. Then we can remove B' from P and add $B'-(1,0)$ to P, which satisfies all the same constraints but has a perimeter shorter by at least the length of β, which is a contradiction.

So, P and $P+(1,0)$ have disjoint interiors, as claimed. □

Lemma 5.3.2. *The optimal polygon P is convex.*

Proof: Suppose for contradiction that it has a nonconvex vertex x. Not both $x-(1,0)$ and $x+(1,0)$ are inside the board, since its side length is less than 2, so without loss of generality (by reflecting the board through its vertical center line if necessary), let $x-(1,0)$ not be inside the board. If $x+(1,0)$ is in the interior of P, then there's some neighborhood of $x+(1,0)$ in the interior of P, so some neighborhood of x must not be in the interior of P, so x isn't actually a boundary point. Otherwise, $x+(1,0)$ isn't in the interior of P.

We can choose points w and y on either side of x on the boundary of P, close enough to x that the triangle wxy contains no constraint points except possibly x, and replace the boundary segments wx and xy by the single segment wy. By the triangle inequality, that shortens the length of the boundary; for no constraint point except possibly x does it change whether that point is inside the polygon, and it includes x in the polygon, which is allowed. So, whether $x+(1,0)$ is on the boundary of P or strictly outside, it can be counted as outside for purposes of separation, so x is allowed to be strictly inside P. Hence P is convex, as claimed. □

Now we argue that the optimal solution is actually the shortest convex cycle that contains at least one point from each pair, which can be found by dynamic programming, following the method of [15]; details for the algorithm are omitted in this extended abstract. For the proof of correctness that this cycle is indeed what we are looking for, we just need an argument that the shortest convex cycle that contains at least one point from each pair contains *exactly* one point of each pair inside. This proof is essentially the same as Lemma 5.3.1 and is thus omitted. Further, by Lemma 5.3.2 adding the extra constraint of convexity on the cycle to be find is of no problem.

5.3.2 Board Size $2 - \varepsilon$, Horizontal and Vertical Unit Segments

In this scenario, all n input segments are inside a square board of size $2 - \varepsilon$ for some $\varepsilon > 0$ and are restricted to have horizontal or vertical orientations. Without loss of generality we assume all the endpoints of different input segments do not share a common x-coordinate or y-coordinate. This can be done by perturbing the input slightly.

First, all unit length boxes which strictly contains *at least* one endpoint of each segment are found. This can be executed in polynomial time by checking all possible combinatorial configurations of unit length boxes. The total number of combinatorial types of such squares is $O(n^2)$ since we can assume without loss of generality that the square always has two input endpoints (from two different segments) on its boundary. Aside from the points on the boundary, each such box actually contains *exactly* one endpoint of each input segment. We handle boundary points by enumerating all combinations of boundary points in a box. The convex hull of all endpoints inside the box is a candidate separating cycle. We can enumerate all such boxes to find the shortest separating cycle.

If a unit length box that strictly contains one endpoint of every segment cannot be found, then the board is divided into four squares each of size $1 - \varepsilon/2$ and are colored in a checker board pattern. The square are named S_1, S_2, S_3, S_4 in a counter clockwise manner. Consider two squares along the diagonal (named S_1 and S_3, see Figure 5.8). We create a tour that walks along the perimeter of their union. To accommodate corner cases, we consider the top and left border of S_3 to be open edges. This generates a curve T' of length $8 - 4\varepsilon$.

Theorem 5.3.3. *For the shortest separating cycle problem with a square $2 - \varepsilon$ domain where input pairs are exactly one apart and have either horizontal or vertical*

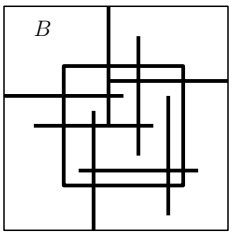


Figure 5.7. Case (i): Exactly one endpoint of each input pair can be enclosed in a unit square.

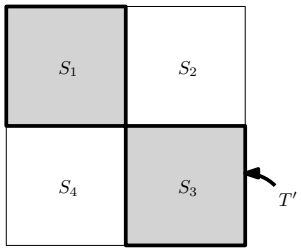


Figure 5.8. Case (ii): The curve T' traverses around S_1 and S_3.

orientation, our algorithm outputs a cycle that is a 4-approximation to the optimal solution.

Proof: There are two cases in the algorithm which outputs two different types of cycles. In case (i), the optimal solution fits inside a unit length box B. Every segment has one endpoint inside B and B will be discovered in the first phase of the algorithm. The convex hull of the points inside B is the shortest separating tour that contains all points in B.

For case (ii) we first argue that T' is a valid separating cycle. Since each segment has unit length, the two endpoints of each segment cannot fit inside any single square of size $1 - \varepsilon/2$ and thus cannot both lie inside $S_1 \cup S_3$ nor inside $S_2 \cup S_4$. Therefore each segment must have exactly one endpoint inside $S_1 \cup S_3$. Therefore, T is a valid separating cycle. Since T' has length $8 - 4\varepsilon$ and the optimal tour has length at least 2, T' is a 4-approximation of the optimal solution. □

5.3.3 Constant Board Size

We can extend the checkerboard strategy for any constant board size M, where M is an integer. The only modification is that in the second case, a larger checkerboard

of $M \times M$ unit squares is used. The squares are colored in a checkerboard manner, and partitioned into white squares and dark squares. To make the tiling a perfect partition of the plane, we consider each square pixel to include its top edge, except for the NE corner, and to include its left edge, except for the SW corner. Again any unit length vertical or horizontal segment has two endpoints in different colored squares. Thus a tour T' that separates the white squares from the dark squares would be a valid separating cycle. Such a tour can always be found by taking a cycle along the boundary of the outermost ring of the dark squares, and iterating towards the center. All the tours can be joined into a single tour of the same length. See Figure 5.9 for an illustration.

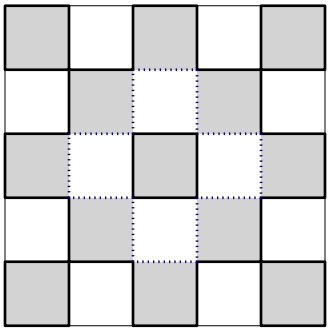


Figure 5.9. Constant sized square board with cycles along the perimeter of the dark squares.

Theorem 5.3.4. *We consider the case of the shortest separating cycle problem with an $M \times M$ square domain and where the input pairs are restricted to be exactly one apart with either horizontal or vertical orientation. Our algorithm including the checkerboard strategy is an $(M^2+1)/4$-approximation.*

Proof: Our proof is similar to the proof of Theorem 5.3.3. Either we can find a unit length box containing at least one endpoint of each segment (in which case we find the optimal solution), or the optimal tour has length at least 2. In the second case we will take the tour T' along the perimeter of the union of the dark squares. T' has length at most $M^2/2$ if M is even, and at most $(M^2+1)/2$ if M is odd. Thus the approximation factor is at most $(M^2+1)/4$. □

5.3.4 Any Board Size, Horizontal and Vertical Unit Segments

If the board size M is a constant, we can apply the same checkerboard idea as in the previous section. But when M is large, we have to use a different idea to get a constant approximation.

First, we overlay a grid of unit squares over the domain partitioning the domain into light and dark squares in a checkerboard pattern. For each grid cell, we consider the top edge, excluding the NE corner, and the left edge, excluding the SW corner, as closed edges.

We refer to dark squares that have a point (from a pair) in them as "occupied". Let S be the occupied squares. Let S' be the 3-by-3 squares centered on the squares of S. In the following we assume without loss of generality that $|S| \geq 5$. The case $|S| \leq 4$ can have an arbitrarily small optimal value; but this constant-size case can be easily handled.

Now we consider the shortest TSP with Neighborhoods (TSPN) tour on the set S' of enlarged squares and name the length as $TSPN(S')$. This tour connects the regions in S' but we must also separate the pairs of points in each region. We further apply the constant factor approximation to each region in S'. Our algorithm is simply this: Run a TSPN algorithm on S' (for which there is a PTAS [54]), and augment the tour with our approximation algorithm for square regions of constant size.

Theorem 5.3.5. *Our algorithm for the shortest separating cycle problem for input pairs restricted to a separation of exactly 1 and only horizontal or vertical orientation, is a constant approximation.*

Proof: We have two cases regarding the size of S, $|S| \geq 5$ and $|S| \leq 4$. In the first case, $|S| \geq 5$, the length of the output is at most $(1+\varepsilon)TSPN(S') + O(|S|)$. Since the optimal solution must visit every enlarged square in S', then $OPT = \Omega(TSPN(S'))$. Assuming no single point stabs all squares of S' (i.e., assuming $|S| \geq 5$), the standard packing argument shows that TSPN on a set of nearly disjoint (i.e., constant depth of overlap), equal-sized squares requires length proportional to the number of squares times the side length of the squares. This leads us to claim that $OPT = \Omega(|S|)$. Therefore, $(1+\varepsilon)TSPN(S') + O(|S|)$ is $O(OPT)$ and our tour is a constant approximation.

For the case where $|S| \leq 4$, our strategy only changes if a single point is contained in all squares of S'. In this case, our entire input can be contained in a 5×5 square and we refer to our algorithm for constant sized domains. □

5.3.5 Any Orientation, Bounded Aspect Ratio

We now assume that the distance between any two points (not restricted to designated pairs) in S are greater than or equal to 1. The segments defined by the pairs of points may have any orientation and their distances are bounded by a constant. Let $r = cL$ for some constant value of $c \geq 1$ where L is the length of the longest segment. The aim is to find a subset I of pairs of points where the shortest distance between any two segments is greater than r and the shortest distance between any input segment and it's closest segment in I is less than r. Then we find the TSPN path on line segments in I. Next, we divide the region into neighborhoods by assigning the remaining segments to their nearest segment in I. The TSPN tour of the segments, $TSPN(I)$, is augmented with detours that separate all of the segments in each neighborhood. The resulting tour is a constant approximation of the optimal separating tour.

To find such an independent set, we randomly select neighborhoods of size $O(r)$ until all segments have been selected. A segment s is randomly chosen and removed from the set of input segments along with all remaining segments within distance (shortest distance) r of s. The segment s is placed in the set I. This procedure is repeated until all segments are removed. The shortest distance between any two segments in I is greater than r. The shortest distance between any segment and it's closest segment in I is less than or equal to r. Each segment is assigned to it's closest segment in I. The set of segments assigned to a segment s in I is denoted as $N(s)$.

A tour is constructed by first finding a TSPN tour on I. Then the path is augmented by a shorter tour within each neighborhood of each segment in I. When a tour reaches a segment s in I, then it makes a separating detour that separates all of the segments in $N(s)$. The length of a tour is bounded by $O(L^2)$ since all of the segments in $N(s)$ is within a neighborhood of radius $2r$ of s and by a packing argument, there are $O(L^2)$ possible points in such a neighborhood. In the worst case, the separating tour visits every point in the neighborhood and includes or excludes each point as required. Note that the detour must exclude segments that are not in $N(s)$.

Theorem 5.3.6. *The path our algorithm produces is a $O(L^2)$ approximation of the shortest separator.*

Proof: Let T be the length of the path our algorithm produces, let OPT be the length of the optimal solution and let $TSPN(I)$ be the length of the TSPN path on I. Our path is a separating tour because every segment is separated by the tour within its neighborhood and excluded by the tour everywhere else. We claim the length of

such a tour is bounded above by $TSPN(I) + O(|I|L^2)$. Since the TSPN path of I must enter and exit the neighborhood of every segment in I, $TSPN(I) = \Omega(|I|)$. Therefore $T = O(L^2 \cdot TSPN(I))$. Since $OPT = \Omega(TSPN(I))$ and $T \geq OPT$, then $T = O(L^2 \cdot OPT)$. □

5.3.6 The General Case

For general pairs of points in the plane, we observe that an $O(\sqrt{n})$-approximation follows from known results on the Euclidean TSP in the plane. Specifically, we first compute a minimum-size square, Q, that contains at least one point of each pair. (This is easily computed, since the n point pairs determine only $O(n^3)$ combinatorially distinct squares.) Now, within Q, we compute an approximate TSP tour T (using any constant-factor approximation method for TSP) on the points that are inside (or on the boundary of) Q, making sure the approximate tour is a simple polygon. We obtain a valid separating cycle for the input pairs as follows: Consider traversing T, starting from an arbitrary point. Each time we reach a point along this traversal, we either make a slight detour to include it (if it is the first time we have encountered a point from this pair), or make a slight detour to exclude it (if it is the second encounter with a pair). In this way, we obtain a valid separating cycle just slightly longer than T. By classic results on the Euclidean TSP (see, e.g., Karloff [89]), we know that the length of T is at most $O(|Q|\sqrt{n})$, where $|Q|$ is the side length of the square Q. Since we know that $\Omega(|Q|)$ is a lower bound on the length of an optimal separating cycle, we have shown that in polynomial time one can obtain an $O(\sqrt{n})$-approximation for the general case of our problem.

5.3.7 Separating Subdivision Problem

We consider now a different version of the separating cycle problem – the *separating subdivision problem*, in which the goal is to compute an embedded planar graph of minimum length such that every input pair has its points in different faces of the subdivision. We give an $O(\log n)$-approximation algorithm. We outline the approach, deferring details to the full paper. We argue that an optimal subdivision, S, can be converted to a special (recursive) "guillotine" structure, increasing its length by a factor $O(\log n)$; then, we show that an optimal solution among guillotine structures can be computed in polynomial time, using dynamic programming. The conversion goes as follows. First, increasing the total edge length of S by at most a constant factor, we can convert its faces to all be rectilinear: we enclose S with its bounding box, and replace each face of S with a rectilinear polygon, with axis-parallel edges

that lie on the grid induced by the input point pairs, while keeping all points within their respective faces. Then, we partition each simple rectilinear face into rectangles, adding axis-parallel chords that lie on the grid; this causes the total edge length to go up by a factor $O(\log n)$; see [109]. Then, using the charging scheme of [109], we know that we can convert the resulting *rectangular subdivision* to a *guillotine rectangular subdivision*, in which one can recursively partition the subdivision using axis-parallel "guillotine" cuts that do not enter the interior of rectangular faces. Optimizing the length of a guillotine rectangular subdivision is done with dynamic programming, in which subproblems are axis-aligned rectangles all of whose boundary is (by definition) included in the edge set of the subdivision. This implies that any input pair of points that "straddles" the boundary of a subproblem, with one point inside, one point outside, is already satisfied automatically with respect to pair separation (the points lie in different faces/rectangles). This means that the subproblem is only responsible for the separation of the point pairs both of whose points lie within the defining rectangle of the subproblem. The algorithm computes a minimum-length guillotine rectangular subdivision, separating all point pairs. Since an optimal solution can be converted to the class of guillotine rectangular subdivisions at a lengthening factor $O(\log n)$, we obtain the claimed approximation.

5.4 Conclusion

The shortest separating cycle is a new variant of the TSP family that has not been studied before. This section provides the first set of hardness bounds and a number of approximation algorithms under different settings. The gap for the approximation ratios and hardness results is still big and narrowing or closing the gap should be prioritized in the future.

Chapter 6

r-gather Clustering

6.1 Introduction

With the ubiquitous use of GPS receivers on mobile devices, it is now common practice to record and collect the locations of these mobile devices. This raises privacy concerns as location information is sensitive and can be used to identify the users of the devices. One common practice in the treatment of this location data is to adopt the k-anonymity criterion [137], which says that the locations are grouped into clusters, each of at least k points. The cluster is used to replace individual locations such that any one user cannot be differentiated from $k-1$ others. Thus minimizing the diameter of the clusters can lead to location data with the best accuracy with the constraints of not intruding user privacy.

Our Problem

Given a set of n points $P = \{p_1, p_2, \ldots, p_n\}$ in Euclidean space and a value r, the aim of the r-gather problem is to cluster the points into groups of r such that the largest diameter of the clusters is minimized. We consider this problem with static and mobile points. For the static setting, we have two definitions of the diameter of a cluster: the distance between the furthest pair of points and the diameter of the smallest disk that covers all points. For the dynamic setting, with mobile points, we have three different variations depending on the number of times the clustering of the points are allowed to change (either never, k, or unlimited). In this setting, the definition of a the diameter of a cluster is the diameter of the smallest covering disk.

Our Results

For the application of protecting location privacy, the data points are actually in Euclidean spaces. Thus, we ask whether the hardness of approximation still holds in the Euclidean space. In the following, we assume that the input points are in the Euclidean plane.

In the static setting, for the case where the diameter of a cluster is the diameter of the smallest covering disk, we show it is NP-hard to approximate better than $\sqrt{13}/2 \approx 1.802$ when $r = 3$ and $\frac{\sqrt{35}+\sqrt{3}}{4} \approx 1.912$ when $r \geq 4$. For the case where the diameter of a cluster is the distance between the furthest pair of points, then it is NP-hard to approximate better than $\sqrt{2+\sqrt{3}} \approx 1.931$ when $r \in \{3, 4\}$ and 2 when $r \geq 5$.

We also show that the lower bound for the static setting translates to all versions of the dynamic setting. When we allow no reclusterings, we are able to strengthen the lower bound by showing that its is NP-hard to approximate better than 2. In addition, we provide 2-approximation algorithms when we allow no or k reclusterings. For the dynamic variation where an unlimited number of reclusterings are allowed, we present an example where clusters change $O(n^3)$ times. Finally, we describe a distributed algorithm that guarantees a solution that is a 4-approximation of the optimal solution.

Related Work

The r-gather problem is shown to be NP-hard to approximate at a ratio better than 2 when $r > 6$ and the points are in a general metric by Aggarwal et al.[5]. They also provide a 2-approximation algorithm. The approximation algorithm first guesses the optimal diameter and greedily selects clusters with twice the diameter. Then, a flow algorithm is constructed to assign at least r points to each cluster. This procedure is repeated until a good guess is found. Note that this solution only selects input points as cluster centers.

Armon [16] extended the result of Aggarwal et al. by proving it is NP-hard to approximate at a ratio better than 2 for the general metric case when $r > 2$. He also specifies a generalization of the r-gather clustering problem named the r-gathering problem which also considers a set of potential cluster centers (referred to as potential facility locations in Armon's paper) and their opening costs in the final optimization function. They provide a 3-approximation to the min-max r-gathering problem and prove it is NP-hard to have a better approximation factor. They also provide various approximation algorithms for the min-max r-gathering problem with the proximity requirement; a requirement for all points to be assigned to their nearest

cluster center.

For the case where $r = 2$, both [9] and [130] provide polynomial time exact algorithms. Shalita and Zwick's [130] algorithm runs in $O(mn)$ time, for a graph with n nodes and m edges. A similar facility location problem has also been studied in [77, 88, 136].

6.2 Static r-Gather

Theorem 6.2.1. *The r-gather problem for the case where the diameter of a cluster is measured by the furthest distance between two points is NP-hard to approximate better than a factor of* 2 *when* $r \geq 5$.

Proof: Our reduction is from the NP-hard problem, planar 3SAT. Given a formula in 3CNF composed of variables $x_i, i = 1, \ldots, n$ and their complements $\overline{x_i}$, we construct an instance of r-gather on the plane. Figure 6.1 illustrates a clause gadget of the clause $C = x_i \vee x_j \vee x_k$ and part of a variable gadget for x_i. In the figure, each point represents multiple points in the same location, the number of which is noted in parenthesis. All distances between groups of points connected by a line are distance 1 apart. Note that all clusters shown in the figure have a diameter of 1. If all clusters have a diameter of 1, then we can signify the parity of a variable by whether solid or dashed clusters are chosen. Here the solid clusters signify a positive value for x_i that satisfies the clause since the center point of the clause gadget is successfully assigned to a cluster. Note that the variable gadget in Figure 6.1 swaps the parity of the signal sent away from the gadget. We also include a negation gadget shown in Figure 6.2 that swaps the parity of the signal and can be used when connecting parts of the variable gadget together. If an optimal solution to this r-gather construction can be found, the diameter of all clusters is 1.

The center point of the clause gadget must be assigned to a cluster that contains all r points of one of the variable clusters or else a cluster of diameter 2 is forced. WLOG, let the center point be clustered with the r points of the x_i gadget. What results is the solid clusters in figure 6.1 are selected above the triangle splitter and the dashed clusters are selected below the splitter. The group of points at the top of the triangle splitter is unassigned to a cluster. It must merge with one of the neighboring clusters which results in a cluster of diameter 2. Therefore, it is NP-hard to approximate r-gather below a factor of 2 for $r \geq 5$. □

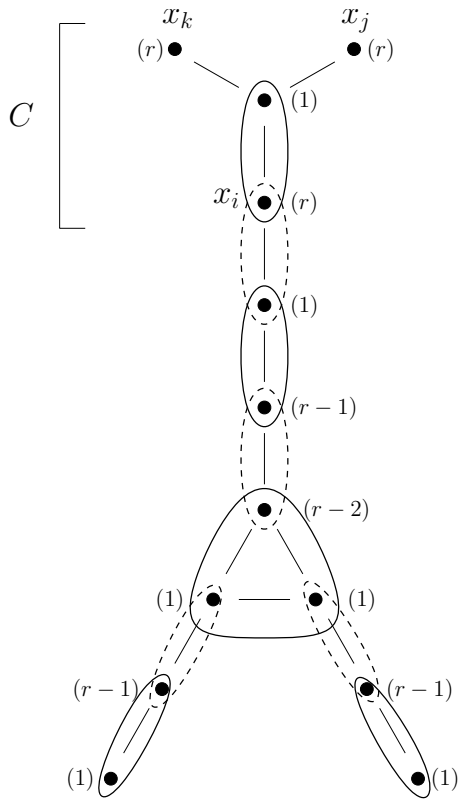


Figure 6.1. Clause and splitter gadget

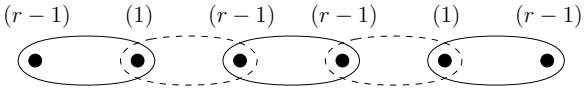


Figure 6.2. Signal negation gadget

Theorem 6.2.2. *The r-gather problem for the case where the diameter of a cluster is measured by the diameter of the smallest covering disk is NP-hard to approximate better than a factor of* $\frac{\sqrt{35}+\sqrt{3}}{4} \approx 1.912$ *when* $r \geq 4$.

Proof: The reduction is very similar to the proof of Theorem 6.2.1. The only difference is the splitter which is illustrated in Figure 6.3. □

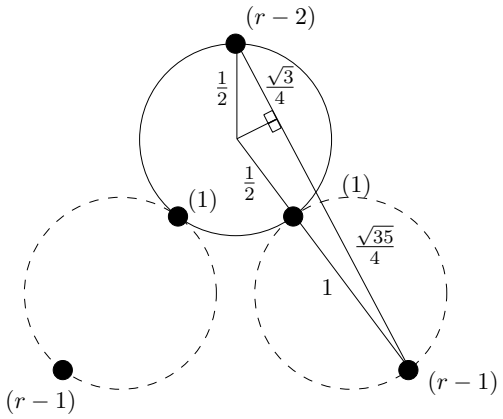


Figure 6.3. Close up of the splitter gadget

Theorem 6.2.3. *The r-gather problem for the case where the diameter of a cluster is measured by the diameter of the smallest covering disk is NP-hard to approximate better than a factor of* $\sqrt{13}/2 \approx 1.802$ *when* $r = 3$.

Proof: We reduce from the NP-hard problem planar circuit SAT. We are given a planar boolean circuit with a single output. Similar to the previous proofs, a wire gadget consists of a line of points that alternate between a single point and a group of $r-1$ points at the same location. The parity of the clusters chosen signify a true signal or a false signal. When the clusters combine a group of $r-1$ points followed by a single point, the signal of the wire is true. It is simple to enforce the output to be a true signal by ending the output wire with a single point. The beginning of the input wires have a group of r points so that the inputs can be either true or false. Figure 6.4 illustrates the NAND gadget, a universal gate. The solid clusters illustrate two true inputs into the gate and a false output. If either or both of the inputs is false, then two groups of points in the triangle (or all three) will become a cluster and the output will be true. Figure 6.5 illustrates the splitter circuit where

the solid clusters indicate a true signal and the dashed clusters indicate a false signal. As before, if the optimal solution to the r-gather construction can be found, then cluster diameter will be 1. Otherwise, three groups will form a cluster, two from the triangle and one adjacent to the triangle. The diameter of such a cluster is $\sqrt{13}/2 \approx 1.802$ when $r = 3$. Finally, note that in order to connect the wires, they must be able to turn somehow. We can bend the wire such that no three groups of points can form a cluster that has diameter smaller than $\sqrt{13}/2$. Thus concludes our proof. □

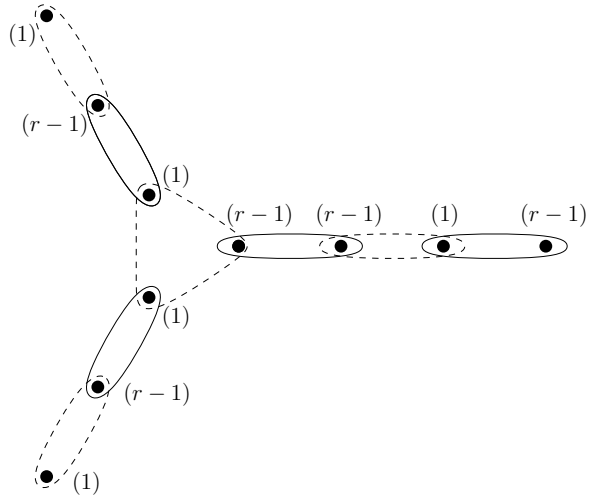


Figure 6.4. NAND gadget

Theorem 6.2.4. *The r-gather problem for the case where the diameter of a cluster is the distance between the furthest pair of points is NP-hard to approximate better than* $\sqrt{2+\sqrt{3}} \approx 1.931$ *when* $r = 3$ *or* 4.

Proof: The proof is similar to the proof of Theorem 6.2.3 except that with the different definition of diameter, the triangles in the gadgets are slightly larger. This makes a cluster with two groups from a triangle and one adjacent to the triangle have a diameter of $\sqrt{2+\sqrt{3}} \approx 1.931$. □

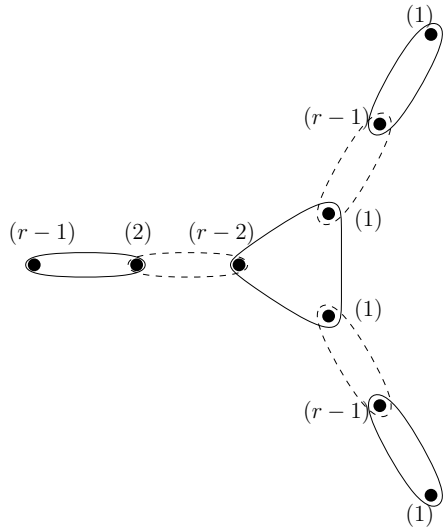


Figure 6.5. Splitter gadget

6.3 Dynamic r-Gather

The natural progression of investigating the r-gather problem is to consider clustering when the points are mobile. The conversion of the r-gather problem to a dynamic setting may appear in many forms. In this section, we detail several versions of the mobile r-gather problem. In each version, we assume that the trajectories of the points are piecewise linear.

In the simplest formulation of r-gather in a mobile setting, we are given a set of trajectories over a time period T and we want to cluster the trajectories such that each cluster has at least r trajectories and the largest diameter of each cluster over the entire time period is minimized. Here we designate the diameter of a cluster at a single point in time to be the distance between the furthest pair of points. Points are assigned to a single cluster for the entire length of T and do not switch clusters. We claim that the 2-approximation strategy for static r-gather can also be applied to this problem. We also claim that 2-inapproximation lower bound from Aggarwal et. al. applies as well [5]. We use the distance metric defined by Lemma 6.3.1.

Lemma 6.3.1. *The distance function $d_t(p, q)$ between two trajectories p and q over a time period T is defined as the distance between p and q at time $t \in T$. Then $d(p, q) = \max_{t \in T} d_t(p, q)$. The function $d(p, q)$ is a metric.*

Proof: The function by definition is symmetric, follows the identity condition, and

is always non-negative. To show that the metric follows the triangle equality, we first assume that there is a pair of trajectories x and z where $d(x,z) > d(x,y)+d(y,z)$ for some y. There is some time $t \in T$, where $d_t(x,z) = d(x,z)$. By triangle inequality,

$$d_t(x,z) \leq d_t(x,y) + d_t(y,z).$$

This contradicts our assumption and concludes our proof. □

Theorem 6.3.2. *For the definition of dynamic r-gather where all trajectories are clustered once, it is NP-hard to approximate better than a factor of 2 for $r > 6$.*

Proof: We construct a reduction from the problem 3-SAT where each variable is restricted to 3 clauses. Our proof is similar to the lower bound proof in [5]. Given a boolean formula in 3-CNF form with m clauses C_j, $1 \leq j \leq m$, composed of n variables x_i, $1 \leq i \leq n$, where i and j are integers, we construct a set of trajectories over a time period T. For every variable x_i and its complement $\bar{x_i}$, we have two points v_i^T and v_i^F and an additional $(r-2)$ points u_i^k for integer k, $1 \leq k \leq r-2$. We will construct trajectories where $d(v_i^T, v_i^F) = d(v_i^T, u_i^k) = d(v_i^F, u_i^k) = 1$, for $1 \leq i \leq n$ and $1 \leq k \leq r-2$. In addition, for every cluster C_j, we construct another point w_j. We construct trajectories such that the distance between w_j and the points that represent the literals in the clause C_j is 1. All other distances are 2.

Constructing the trajectories that obey these distance constraints is simple. Place all points at the origin on a number line. For every point p, we have one timestep that establishes the distance between it and all other points. In that timestep, p moves to 2 on the number line. At the same time, all other points that must have a distance of 1 to p move to 1 on the number line. This procedure is repeated for every point.

We claim, that if there is an r-gather clustering of maximum radius 1, then the 3-SAT formula can be satisfied. Here, for every variable, we have two possibilities for the center of a cluster of radius 1, v_i^T and v_i^F. There are not enough points close to v_i^T and v_i^F to have both as the center of two clusters. Any cluster with u_i^k as the center will have fewer than r points within a distance of 1 and therefore cannot be a cluster center with radius 1. Finally, every point w_j has only three other points within a distance of 1. Therefore, in any r-gather clustering with maximum radius 1, there is one cluster for every variable with a center of either v_i^T or v_i^F. This clustering must be done in such a way that for every point w_j, at least one of the points that represents a literal in C_j must be chosen as a cluster center. Thus if an optimal r-gather clustering of our trajectories can be found, then we can determine if the corresponding 3-SAT formula is satisfiable. □

The next step in expanding dynamic r-gather is by allowing the clusters to change throughout T. We amend our problem formulation to allow k regroupings. Each regrouping allows all clusters to be modified or changed completely. The lower bounds for the earlier version of r-gather applies here too for the same reasons. We claim that with the assumption that the trajectories are piecewise linear, we can construct a 2-approximation solution using dynamic programming.

Let $|T|$ be the number of timesteps in the time period T. Each trajectory is a piecewise linear function that only changes directions at a timestep in T. Let C_{ij} denote the max diameter of the 2-approximation clustering at time i over the time period $[i, j]$, $i < j$. We can create a $|T| \times |T|$ table $\mathcal{T}$ where entry $\mathcal{T}(i, j) = C_{ij}$. One clustering takes $O(\frac{1}{\epsilon}kn^2)$ and there are $|T|$ clusterings in total. However, for each clustering, the max diameter is recalculated for each timestep. The cost of recalculating the max diameter of a clustering is $O(n/r)$. The total number of times a clustering is recalculated is $O(n|T|/r)$. The total time it takes to compute the table $\mathcal{T}$ is $O(n|T|^2/r + \frac{1}{\epsilon}k|T|n^2)$.

We formulate a subproblem $S(t, i)$, where $0 \leq t \leq |T|$ and $i \leq k$, for our dynamic program to find the optimal clustering of the points in the time period $[0, t]$ where there are exactly i reclusterings. Let $l(t, i)$ denote the last timestep a reclustering occurred for the optimal solution of $S(t, i)$.

The subproblem of our dynamic program is:

$$S(t, i) = \min(\max_{j<t}(S(j, i), C_{l(t,i)t}), \max_{j<t}(S(j, i-1), C_{tt}))$$

The entry $S(t, i)$ checks $2t$ previous entries and $2t$ entries in the table $\mathcal{T}$. The entire table takes $k|T|^2$ to execute with the additional preprocessing of the table $\mathcal{T}$.

Our lower bound proofs for static r-gather apply here as well. The points arranged in any of the lower bound proofs can be static points for the duration of T or may move in a fashion where the distances between points do not increase. Then the arguments for static r-gather translate to this simple version of dynamic r-gather directly.

Theorem 6.3.3. *The lower bound results for static r-gather apply to any definition of dynamic r-gather. Further, we can approximate mobile r-gather, when k clustergings are allowed, within a factor of 2.*

Another variation allows unlimited regroupings in a continuous dynamic setting. We know that in this setting, the optimal clustering may change many times, as much as $O(n^3)$ times. Consider this example: $n/2$ points lie on a line where the points are spaced apart by 1 and 3 points are overlapping on the ends. In this example, $r = 3$.

The optimal clustering of the points on the line is to have three points in a row be in one cluster with a diameter of 2. There are three different such clusterings which differ in the parity of the clusterings. In each clustering, there are $O(n)$ clusters. If another point travels along the line, when it is within the boundaries of a cluster, it will just join that cluster. However, when it reaches the boundary of a cluster and exits it, the optimal clustering would be to shift the parity of the clustering. This results in a change in all of the clusters along the line. The clustering change every time the point travels a distance of 2. Therefore, as the point travels along the line, the number of times the entire clustering changes is $O(n)$ which results in a total of $O(n^2)$ changes to individual clusters. Since there are $O(n)$ points that travel along the line, the total number of clusters that change is $O(n^3)$.

6.4 Decentralized Algorithm

In this section, we consider the r-gather as a distributed computation problem. This approach is particularly relevant in the context of locations, where data is naturally spread over a spatial region, and we can use local computations at access points and local devices for anonymization and cloaking. This approach also provides better security and privacy, since it is harder for an attacker to compromise many devices spread over a large region.

6.4.1 Maximal Independent Neighborhoods

We assume there is a set of n mobile nodes $1, 2, \ldots, n$, and use p_i to denote the location of node i. The set P is the set of locations (p_is) of the nodes. For any point p_i, we let $p_i^{(r)}$ denote its r^{th} nearest neighbor in P, and we let $d_r(p_i) = |p_i - p_i^{(r)}|$ denote the corresponding distance. We let $N_r(p_i)$ denote the set of r nodes nearest to point p_i, and let $N(P) = \{N(p_i) : p_i \in P\}$ denote the set of such *r-neighborhoods*. If $N_r(p_i) \cap N_r(p_j) = \emptyset$, we say that the r-neighborhoods of p_i and p_j are *independent*.

We let $\mathcal{G}$ denote the set of clusters, at any stage of our algorithm; $\mathcal{G}$ is initialized to $\emptyset$. We let c_G denote the *center* of cluster $G \in \mathcal{G}$; the center c_G may be either a node (in P) or another location.

The basic algorithm executes the following steps to construct the set $\mathcal{G}$ of clusters:

M1 At each point $p_i \in P$, compute $p_i^{(r)}$, $d_r(p_i)$ and $N_r(p_i)$.

M2 Find a maximal independent subset of neighborhoods from the set $N_r(P)$, add each as a cluster in $\mathcal{G}$, and *mark* the nodes (as "clustered") in these sets.

M3 For any unmarked node $p_i \in P$, assign p_i to the cluster $G \in \mathcal{G}$ whose center, c_G, is closest to p_i.

The nodes that belong to r-neighborhoods of cluster centers and added to clusters in step M2 are called *canonical* members of the cluster, while nodes that are added in step M3, are called the *outer* members.

Next, we argue that this simple algorithm approximates an optimal clustering. Let $d_r^{\max} = \max_i d_r(p_i)$ be the largest distance from a node to its r^{th} nearest neighbor. Let D_{OPT} be the diameter of the largest cluster in an optimal clustering. We then observe

Observation 6.4.1. *$D_{OPT} \geq d_r^{\max}$.*

Proof: Suppose $p_i \in P$ is a point achieving $d_r^{\max}$: $d_r(p_i) = d_r^{\max}$. Since any disk of radius less than d_r^{max} centered at p_i does not contain r nodes, a disk that contains p_i as well as (at least) $r-1$ other nodes must have radius at least $d_r^{max}/2$. Thus the diameter D_{OPT} must be at least $d_r^{\max}$. □

Lemma 6.4.2. *For any cluster $G \in \mathcal{G}$ and any node $p_i \in G$, $|p_i - c_G| \leq d_r(p_i) + d_r^{\max}$.*

Proof: Consider a cluster $G \in \mathcal{G}$, centered at c_G, and $p_i \in G$.

If p_i was assigned to cluster G in step M2, then we know that $p_i \in N_r(c_G)$, implying that $|p_i - c_G| \leq d_r(c_G) \leq d_r(p_i) + d_r(c_G)$.

If p_i was not assigned to cluster G in step M2, but was instead assigned to G in step M3, then we know, by maximality of the independent set, that the r-neighborhood $N_r(p_i)$ intersects some other r-neighborhood, say $N_r(p_j)$, that was a cluster in the maximal independent set in step M2. (It may or may not be the case that $G = N_r(p_j)$.) Thus, there is a node $p_y \in N_r(p_i) \cap N_r(p_j)$, implying that $|p_i - p_y| \leq d_r(p_i)$ and that $|p_y - p_j| \leq d_r(p_j)$. The triangle inequality implies then that $|p_i - p_j| \leq |p_i - p_y| + |p_y - p_j| \leq d_r(p_i) + d_r(p_j) \leq d_r(p_i) + d_r^{\max}$. Since p_i is closer to c_G than to the alternative center p_j, we get the claimed inequality, $|p_i - c_G| \leq |p_i - p_j| \leq d_r(p_i) + d_r^{\max}$. □

From the above lemma it follows that the algorithm produces a 4-approximation of the diameter:

Corollary 6.4.3. *The diameter of any $G \in \mathcal{G}$ is at most $4D_{OPT}$.*

Proof: Consider any $p_i, p_j \in G$. By Lemma 6.4.2, $|p_i - c_G| \leq d_r(p_i) + d_r^{\max} \leq 2d_r^{\max}$ and $|p_j - c_G| \leq d_r(p_j) + d_r^{\max} \leq 2d_r^{\max}$. Thus, by the triangle inequality, $|p_i - p_j| \leq 2d_r^{\max} + 2d_r^{\max} = 4d_r^{\max} \leq 4D_{OPT}$. □

The maximal independent subsets in step M2 can be computed rapidly, in time $O(\log n)$, using the randomized parallel algorithm of Alon et al. [7], applied to compute a maximal independent set in the intersection graph of the neighborhoods $N(P)$ (i.e., in the graph whose nodes are the r-neighborhoods $N(p_i)$ and whose edges link two r-neighborhoods that have a nonempty intersection).

6.5 Conclusion

In this section we build upon the work of others and further explore the r-gather problem. We provide further insight in to the Euclidean version of r-gather in the static, dynamic, and distributed setting. Still, little is known about the dynamic setting and it should be the focus of future work.

Chapter 7

Geographical Routing in 3D

7.1 Introduction

With the vision of the Internet of Things, sensors are ubiquitously installed on physical objects and at an accelerating speed penetrate into our everyday life. In this section we examine a classical yet important problem in wireless sensor networks – how to find an efficient and practical routing method for sensors that stay in three dimensional space.

While numerous work have been done in this direction, we take the approach of geographical routing and try to make it *practical* and *for 3D*. Geographical routing uses geographical locations of the sensor nodes for routing decisions. It is the most natural choice in the setting when the data that a sensor gets is more important than the sensor node itself. Geocast, i.e., routing to a sensor whose location is at or near the specific geographical target, is the most relevant type of routing requests than routing to any specific sensor ID. The second motivation is the efficiency and compactness of geographical greedy routing – on a network of constant node degree the routing scheme uses constant state per node.

Unfortunately geographical greedy routing alone does not always work and existing recovery schemes face serious challenges in practice, namely, the complex propagation models of wireless communication, and the challenges when nodes are generally in a 3D space. Requiring that the nodes to always stay on 2D plane starts to become a major limitation as in many applications sensors are deployed in 3D space, with different levels of elevations or even airborne.

We will first review the history of geographical routing in the literature and then present our new ideas.

History

In geographical routing, the physical locations of the sensor nodes are used to find the path a packet takes in the network. Geographical routing was originally proposed by Bose *et al.* [29], and independently by Karp and Kung [90] for routing in resource-constrained and dynamic networks such as ad hoc mobile networks or sensor networks. Further improvements on worst-case efficiency were provided by the GOAFR+ family of protocols [99, 98]. A nice summary of geographical routing schemes with a focus on some subtle issues is presented in [66]. Here we quickly review the basics.

Geographical routing has two modes: the *greedy mode* and the *recovery mode*. In the greedy mode, the node currently holding the packet "advances" it towards the destination, based only on the location of itself, its immediate neighbors and the destination. The advance rule may be defined in many ways. The most popular way of defining the advance is to examine the Euclidean distance to the destination and select the next hop as the node closest to the destination. The greedy routing often suffices to deliver a packet in a dense network, but may fail in a sparse network. For example, the packet may reach a *local minimum* whose neighbors are all further away from the destination than itself.

When the greedy mode fails to advance a packet at some node, the routing process switches to the recovery mode. The recovery mode defines how to forward the packet at a local minimum, in order to guarantee eventual delivery to the destination. Some examples of the methods to get out of local minima are simple flooding [134], terminode routing [27], breadth-first search or depth-first search [85], the face routing or perimeter routing [29, 90].

When the sensor network is dense, there has also been a lot of work that generates virtual coordinates with which greedy routing becomes easier. For example, by using Ricci flow the network can be transformed to a circular domain such that greedy routing alone guarantees delivery [126]. Other ideas that use polar coordinates system [117] or hyperbolic coordinates [97] are much more flexible with density, but tend to create load imbalance.

Challenges in Practice

Many of the proposed ideas work nicely on paper. When applied to real world networks they face a number of critical challenges that may greatly degrades their performance. The most notable challenges are the following.

Geographical routing in 3D None of the proposed geographical mechanism is for nodes that generally stay in 3D. Many of them explicitly make use of the two dimensional structure of the network geometry and break down when the dimension is elevated. For example, in perimeter routing, a planar subgraph is extracted from the communication graph and a message when stuck at a local minimum will be routed along the boundary of a face towards the destination. In 3D we can try to mimic this and build a volume decomposition, but the boundary of each 'solid piece' is a surface in general, and routing on a surface is nontrivial.

This challenge has also been demonstrated by worst case theoretical analysis. In theory, geographical routing protocols using face or perimeter routing use constant state information in the packet and guarantees delivery, Durocher *et al.* [55] showed that it is impossible to find a constant state routing algorithm in 3D unit ball networks.

Unit disk graph assumption In many proposed mechanisms it was important that the communication model follows the unit disk graph model – i.e., two nodes have a link if their distance is within 1 and not otherwise. For example, the correct construction of a planar subgraph critically depends on such an assumption.

On the other hand, experimental evaluations of the communication model on sensor nodes reveal various spatial and temporal radio irregularities [67, 151]. Close-by nodes may not be able to communicate directly while some long high-quality links may exist. Links might also be asymmetric/directional and over time link quality varies. In practice, the planar subgraph extracted may become disconnected or still contain crossing edges, as shown in [128, 96]. This subsequently causes the delivery rate to drop to about 68% on a real testbed [96].

A number of repair mechanisms propose to remove crossing links by probing [96, 95, 73]. These mechanisms improve the delivery rate at the cost of higher processing overhead. But again this will only work for nodes that embed in 2D. In 3D the notion of crossing does not exist.

Node density Many of the algorithms that use global transformation to build virtual coordinates for geographical routing assume that the network is dense such that one can talk about the geometric shape of the network. This is true for large scale dense network topologies but in smaller, sparser networks the assumption is less appropriate.

New Ideas

The new ideas we present in this section is to supplement geographical greedy routing by additional *virtual edges*. With the augmented edges geographical routing can successful deliver the message towards the destination. And we want to add a minimum of such edges. These virtual edges are created either by extending communication range when possible, or implemented by routes stored in the network. In the latter case, each virtual edge (u, v) actually corresponds to a path from u to v which is stored locally at the nodes along the path. The node u remembers the location of v and can use v as a neighbor in geographical greedy routing. When v is identified as the next hop the routing table entries will be used to guide the message towards v.

Mathematically, consider a graph on vertices embedded in the Euclidean space, we would like to add a minimum number of virtual edges such that greedy routing using the node coordinates succeeds. That is, for any source node s and any destination t, there is an edge from s to w such that w's Euclidean distance to t is strictly smaller than the Euclidean distance from s and t, i.e., $|wt| < |st|$. This way a message can always reach the destination via a distance decreasing path by purely local operations.

This problem has two versions, depending on whether the edges in the network are directed or not. In wireless communication it is possible that a link is directional in the sense that u can send a message to v but v cannot send it directly back to u. Correspondingly, the virtual edges can be directed or undirected. It is actually a little surprising that the problem in the directed case, i.e., finding the minimum number of directed virtual edges to enable greedy routing, is easier to solve and the result for directed case provides a 2-approximation for the undirected setting. In this section we show that finding the minimum number of directed virtual edges is actually polynomially solvable. And the maximum number of virtual edges is no greater than 5 for 2D and at most 11 for 3D. This result is for *general directed graphs*. Thus the result would apply for real world network when the links do not necessarily follow unit disk graph model and can be asymmetric and dynamic. In our simulations the number of virtual edges is actually a lot less than the proved upper bound. The total routing table state size is also at a modest level compared to the worst case bound.

Related Work

The problem of augmenting a graph to enable greedy routing is related to the question which types of graphs naturally support greedy routing. The one mostly known is the Delaunay triangulation. Given a set of nodes in the plane, the Delaunay triangulation

is a planar graph in which each node has at least a neighbor whose Euclidean distance is closer to the destination. Therefore, if the communication graph already contains the Delaunay triangulation on the same set of vertices, greedy routing automatically works. If not, it is obvious that adding these missing Delaunay edges will suffice. This idea works fine in 2D, as the Delaunay triangulation is a planar graph and thus has at most a linear number of edges. However, this is not precisely what our algorithm will do. Just to point out that the in a Delaunay triangulation a node may have degree $\Theta(n)$ and Delaunay triangulation for nodes in 3D may have a quadratic number of edges but in our algorithm we always add the minimum number possible and will not add more than a linear number of edges in total.

We would also want to point out that the edges we construct is different from the Yao graph [148] or Θ-graph [46, 91]. In both setting, a cone partition with angle $\pi/3$ is placed at each node p in the network and p connects to all the nodes that are nearest inside each cone (in Yao graph), or with the closest projection on the bisector of each cone (in the case of the Θ-graph). In our setting, we are not always adding 6 edges for each node but rather make use of existing edges as much as possible. In the simulations, the total number of edges augmented is much fewer than the worst case asymptotics.

7.2 Graph Augmentation

In this section we focus on the mathematical proofs and centralized algorithms for finding a minimum number of directed virtual edges to augment a given graph such that greedy routing works. We handle the two cases of directed edges and undirected edges separately.

7.2.1 Directed Graphs

Given a set of n wireless nodes in $\mathbb{R}^d$, $d = 2, 3$, the graph G contains the communication links between them. Notice that G is assumed to be generally directional and edges may not be asymmetric. Thus we will call G a digraph. A digraph G with coordinates of the vertices in $\mathbb{R}^d$ is a digraph drawing. We will add additional virtual edges, which are assumed to be directional as well. That is, the edge uv is directed from u to v. In the network the node u considers v as a neighbor and remembers v's geographical coordinates. We show below how to find the minimum number of additional directional edges to enable greedy routing between all pairs in polynomial time.

This algorithm is based on two observations. The first observation is that we must add edges for each node u to ensure that u has a neighbor closer to any possible destination. Since the edges are directional, the edges added from u to v can only help u but not v. Thus we can consider the edges added to each node separately. In other words, greedy routing in a digraph is a local property:

Lemma 1. *A digraph drawing is greedily routable if and only if, for every pair of vertices s and t, there is a vertex u such that (s, u) is an edge and $d(u, t) < d(s, t)$.*

Proof: If a digraph drawing is greedily routable, then for any s and t, the first vertex u after s on a greedy routing from s to t satisfies the desired conditions. Conversely, if for every pair of vertices s and t there's a vertex $u(s, t)$ such that (s, u) is an edge and $d(u, t) < d(s, t)$, then the digraph drawing is greedily routable: the sequence s, $u(s, t)$, $u(u(s, t), t)$, ... is a path of vertices with strictly decreasing distance to t, and there are finitely many vertices in the digraph, so the path terminates at t. □

The next observation is a geometric lemma. Consider a node u and all neighbors $N(u)$. We draw a bisector between u and each neighbor $v \in N(u)$. Such a bisector defines a halfplane containing v, denoted by $H_u(v)$. A message with a destination that falls inside $H_u(v)$ can not get stuck — it can at least be delivered to v which is closer to the destination than u is.

Therefore we will consider only the region that is outside all the halfplanes $H_u(v)$, for all $v \in N(u)$. Denote by this region $C(u)$. We take the node w in $C(u)$ closest to u and add an edge from u to w. Then add the neighbor w to $N(u)$ and iterate. We argue that this procedure will repeat at most 6 times until $C(u)$ is empty of points. See Figure 7.1 for an illustration which also shows that not all Delaunay edges are needed.

The number 6 is not an arbitrary number but rather the kissing number in 2D – the number of non-overlapping unit spheres that can be arranged such that they each touch another given unit sphere. The result can be generalized to higher dimensions. For 3D it is 12. We have the following theorem.

Theorem 2. *Every digraph drawing in d dimensions has a greedily routable superdigraph formed by adding at most $k(d)$ edges from each vertex, where $k(d)$ is the kissing number in d dimensions.*

There is a digraph drawing that needs to add $k(d)$ edges from some vertex, simply by putting one vertex with no edges from it at distance 1 from each of $k(d)$ other vertices, with no other vertices in the graph. Similarly, there are digraph drawings that need to add as many edges as the d-dimensional *lattice* kissing number at almost

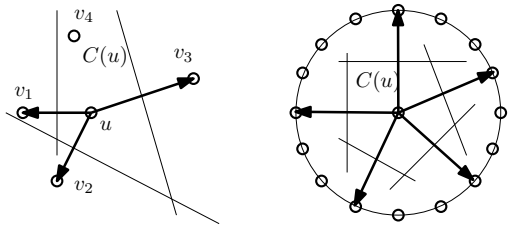


Figure 7.1. Left: the bisectors of u and each neighbor in $N(u)$ are shown. v_4 is not inside $C(u)$ and thus is added as a virtual neighbor; Right: in the case when there are many nodes on a circle centered at u, in the Delaunay triangulation all the edges from u to each node on the circle is a Delaunay edge but to enable greedy routing one does not need to keep all such Delaunay edges – five edges in this example suffice.

every vertex, by arranging many vertices in a section of a lattice attaining that kissing number. We can now prove Theorem 2:

Proof: Given a vertex $s = s_0$, define a sequence inductively by letting s_i be a closest vertex to s_0 (other than s_0 itself) such that $d(s_i, s_j) \geq d(s_i, s_0)$ for all $j \in [1, i-1]$. If no such vertex exists, the sequence ends.

We claim that if edges are added from s to every other vertex s_i in that sequence, then we can route greedily from s. Indeed, let t be any vertex. If t is one of the s_i, we can greedily route straight to it. Otherwise, if i is the least index such that $d(s, s_i) > d(s, t)$, then, since t was not picked as s_i, there's some vertex s_j such that $d(t, s_j) < d(t, s)$, but there's an edge from s to s_j that we can follow to get closer to t, as desired.

We also claim that for any two vertices s_i and s_j with $i > j > 0$, $\angle s_i s_0 s_j \geq \frac{\tau}{6}$. Indeed, we have $d(s_i, s_j) \geq d(s_i, s_0)$ (by the inductive choice of i) and $d(s_i, s_0) \geq d(s_j, s_0)$ (since s_j was picked before s_i, and is therefore closer to s_0), so in triangle $s_0 s_i s_j$, $s_i s_j$ is the (nonstrictly) longest side, so the angle opposite it is the triangle's greatest, and is therefore at least $\frac{\tau}{6}$.

Therefore, if we center a unit sphere at distance 2 from s_0 in the direction of each s_i, they form a no overlapping sphere packing around the unit sphere S centered at s_0, so the d-dimensional kissing number is at least the number of terms in the sequence, which is at least the minimum number of edges that must be added from s for greedy routability, as desired. □

What still remains is a means to efficiently find the nearest neighbor of a vertex u within the region $C(u)$. This can be done with a combination of simplex range queries and Voronoi diagrams. Using methods by Matousek [110] and later, Chan [38],

quick nearest neighbor queries can be executed in $O(n^{1-1/d} \log n)$ time with some preprocessing.

Lemma 7.2.1. *Given a set of n points in $\mathbb{R}^2$, a structure can be constructed in $O(n \log n)$ time with $O(n)$ space that can find the nearest neighbor to an input point u within a given polygon $C(u)$ in $O(\sqrt{n} \log n)$ time w.h.p.. The number of sides that $C(u)$ has is bound by the kissing number. When the points are in $\mathbb{R}^d$, the preprocessing time is $O(n^2)$, the space is $O(n^{\lceil \frac{1}{2} d \rceil})$, and the query time is $O(n^{1-1/d} \log n)$ w.h.p.*

Proof: We use the techniques by Chan [38] for simplex range queries. He describes a construction of a partition tree on a set of points that can execute simplex range queries in $O(n^{1-1/d})$ time with $O(n \log n)$ preprocessing time and linear storage space w.h.p. This tree has constant depth.

We can amend this tree by constructing a voronoi diagram in each cell in each node of the tree. Since each level of the tree is a partition of the input points it takes $O(n \log n)$ time to construct voronoi diagrams for all points in one level. There are a constant number of levels so the preprocessing time for our structure remains $O(n \log n)$. The size of the structure does not change as well. In higher dimensions, each level takes $O(n^2)$ time to construct and takes $O(n^{\lceil \frac{1}{2} d \rceil})$ space so the preprocessing time and space is larger.

When executing nearest neighbor queries, each node in the tree queried also queries it's corresponding voronoi diagram. This adds a $O(\log n)$ factor to the query time increasing it to $O(n^{1-1/d} \log n)$. □

Corollary 7.2.2. *Finding the greedy routing number of a digraph drawing G is d dimension is in FPT (fixed parameter tractable); that is, it can be done in time $O(k(d)n^{2-1/d} \log n)$.*

Proof: By lemma 1, determining the greedy routing number of a digraph is solvable locally: for each vertex s, we need to add exactly as many edges from s as needed to greedily route from s, independent of what edges are added from other vertices.

By Theorem 2, there is a solution that adds at most $k(d)$ edges from every vertex. Each vertex can be found using a nearest neighbor query with u and $C(u)$. With Lemma 7.2.1, we can find all virtual edges in time $O(k(d)n^{2-1/d} \log n)$ with $O(n \log n)$ preprocessing in 2D and $O(n^2)$ preprocessing time in higher dimensions w.h.p. □

In applications where the aspect ratio of the sensors is bounded by a constant, a practical solution is to use a quadtree with voronoi diagrams to execute queries in $O(|C(u)| \log n)$ time per query where $|C(u)|$ is the length of the perimeter of $C(u)$ (or size of the surface area of the polytope $C(u)$ when $d \geq 3$).

Lemma 7.2.3. *Let a set of n points in $\mathbb{R}^2$ have aspect ratio α, the ratio between the closest and furthest points, be bounded by a constant. A quadtree structure can be computed on these points in $O(n \log n)$ time with $O(n)$ space that supports nearest neighbor queries in $O(|C(u)| \log n)$ time of an input point u within an input convex polygon $C(u)$. When the points are in $\mathbb{R}^3$, this structure can be constructed in $O(n^2 \log n)$ time with $O(n^2)$ space.*

Proof: Given a set of n points, construct a quadtree on the input. For each cell, on each level, a voronoi diagram is constructed on the points inside. Using existing methods, all of the voronoi diagrams for one level of the quadtree can be constructed in $O(n \log n)$ time with $O(n)$ space in 2D and $O(n^2)$ time with $O(n^{\lceil \frac{1}{2} d \rceil})$ space for $d \geq 3$. Due to the assumptions on α, the depth of the tree is constant and the quadtree can be constructed in linear time. Therefore, the entire structure can be constructed in $O(n \log n)$ time in 2D with $O(n)$ space and $O(n^2)$ time in higher dimensions with $O(n^{\lceil \frac{1}{2} d \rceil})$ space.

To query the quadtree for the nearest neighbor of an input point u within a convex polytope $C(u)$, we traverse through the quadtree to find the cells that intersect $C(u)$. During this traversal, we also find the canonical cells within $C(u)$, the set of cells that are fully contained within $C(u)$ whose parent cells are not fully contained in $C(u)$. On a grid with cell size 2^l where l is bounded by α, a constant, the number of cells that the boundary of $C(u)$ intersects is $O(|C(u)|)$. Note that $|C(u)| = O(\alpha)$, though at times it may be much smaller.

Now, for any canonical cell, its parent cell must intersect the boundary of $C(u)$ which implies that for any l, the number of canonical cells of size 2^l is $O(|C(u)|)$. Since there are a constant number of levels in the quadtree, the total number of canonical cells and the total number of cells touched by our search is $O(|C(u)|)$.

To compute the nearest neighbor of our query point u, for every canonical cell, its corresponding voronoi diagram is queried. Each voronoi query is executed in $O(\log n)$ time. Each noncanonical leaf cell that intersects the boundary of the query polygon contains at most one point. The result is a total of $O(|C(u)|)$ candidate points from canonical and noncanonical cells. The closest candidate point to u is the solution. Finding all canonical and noncanonical cells takes $O(|C(u)|)$ time. Finding candidate points takes $O(|C(u)| \log n)$ time from canonical cells and $O(|C(u)|)$ time from noncanonical cells. Therefore, the total query time of a nearest neighbor search is $O(|C(u)| \log n)$. □

Undirected Graphs

When all edges are undirected, the virtual edges can also be treated as undirected edges. Basically, if we identify uv as a virtual edge, and the path $P(u, v)$ is used to enable this virtual edge – i.e., for all nodes on the path they remember the coordinates of both upstream and downstream neighbors. Thus this virtual edge uv can be used in both directions.

We can thus formulate the same problem in the undirected case: find a minimum number of virtual edges to enable greedy routing. This problem turns out to be more challenging than the directed case, as the selection of virtual edges is no longer a local property. An edge uv that works for u may also help node v. Nevertheless, we can show that the solution computed by the algorithm for the directed case in the previous subsection gives a 2-approximation for the undirected case.

Theorem 7.2.4. *The solution provided by the algorithm in the directed case is a 2-approximation to the graph augmentation problem in the undirected case.*

Proof: Suppose E is the set of edges produced by the graph augmentation algorithm in the directed case and E^* is the optimal solution for the undirected case. It is obvious that $|E^*| \leq |E|$. It is also clear that if we treat each edge in E^* as two directed edges (one in each direction), the solution suffices for enabling greedy routing in the directed case. Thus $|E| \leq 2|E^*|$. □

7.3 Experiments

Implementation

For our experiments, we created a network simulation with sizes ranging from 250 nodes to 2000 nodes. These nodes were randomly generated locations in three dimensions. Links between nodes are determined by the wireless radius of the nodes in the given scenario. Using our algorithm, additional directed virtual links are added to the network. The virtual links are represented in the routing tables of the source node as the shortest path between the two nodes, simply found using breadth-first search. When a packet is routed using the greedy algorithm, if a virtual edge is chosen for the next step of the path, then the packet is routed along the path in the routing table before the greedy method is applied again.

In addition, we also simulated a network using details of the INDRIYA testbed [53]. We used the relative locations of the nodes and their topology in some of our experiments. All of our simulations were programmed in python using the NumPy package.

Experimental Results

We ran experiments in static and dynamic scenarios on various networks. This includes random networks in sizes ranging from 250 to 2000 nodes and a simulated network inspired by the INDRIYA testbed. For experiments run on a single network, we use a randomly generated network of 1000 nodes.

In the static scenarios, we measured properties of the network and the greedy paths between pairs of nodes in relation to the density and size of the network. The density of the network is manipulated by the communication radius of the nodes. If the communication radius of the nodes is larger, then a node is more likely to have more neighbors and the network graph becomes more dense. When experiments were run on different network sizes, we wanted to see the effects of just having more nodes in the network. We attempted to negate the influence of the density of the network so the size of the domain was increased accordingly so that the average number of neighbors a node would have would be roughly the same in all network sizes. For these two factors, we measured the number of added edges per node when our practical algorithm is run and the average length of different paths in the network. We consider the length of a path to be the number of hops a message travels. The results of our experiments in the static case are illustrated in Figures 7.2 through 7.5.

As indicated in Figures 7.2 and 7.3, our algorithm adds few additional edges. The average number of added edges per node is very small, much less than the theoretical upper bound of 6.

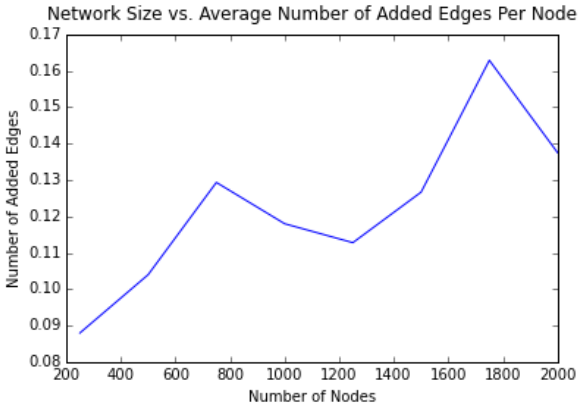


Figure 7.2. The average number of added edges per node after adding virtual links to aid greedy routing. (n=250-2000)

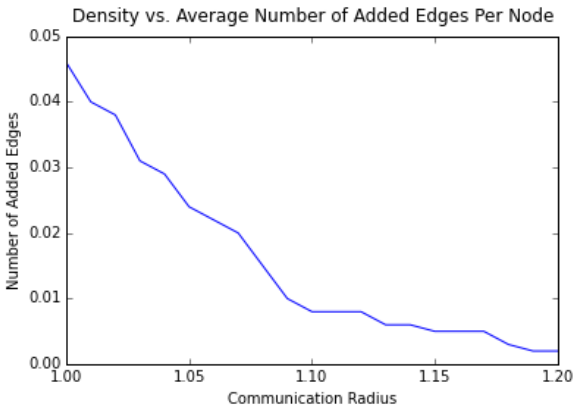


Figure 7.3. The average number of added edges per node after adding virtual links to aid greedy routing. (n=1000)

As expected, the length of paths increase as the network size increases and as the density decreases. Figures 7.4 and 7.5 show the average number of hops of several paths between 100 randomly generated source and destination pairs. The shortest path on a network with virtual links is determined by the path with the fewest number of physical or virtual links. However, in Figures 7.4 and 7.5, they are measured on the y axis by the number of physical hops in the path. In Figure 7.4, all paths increase as the network size increases as is expected but additionally the gap between the shortest path with and without virtual links increases as well. In Figure 7.5, when the communication radius is low, the shortest path with virtual links suggests that it is more likely that such a path would include a virtual link which would increase the physical hop count. As the communication radius increases, the chances that a shortest path with virtual links would actually include a virtual link is lower which corresponds to the lower number of virtual links added to the network as the communication radius increases and the network becomes more closely connected. Note that there are a very narrow set of densities where the greedy paths on the network has a stretch that is greater than 1 and the network is still connected. This suggests that as long as the network is connected, our method scales well and produces a path with very low stretch.

We also examined the robustness of the modified network in the static and dynamic setting. On a network of 1000 nodes, we explore the effect of a certain probability of link failure on the paths of 1000 random source and destination pairs. Figure 7.6 shows the effect of different probabilities of link failure on the length of

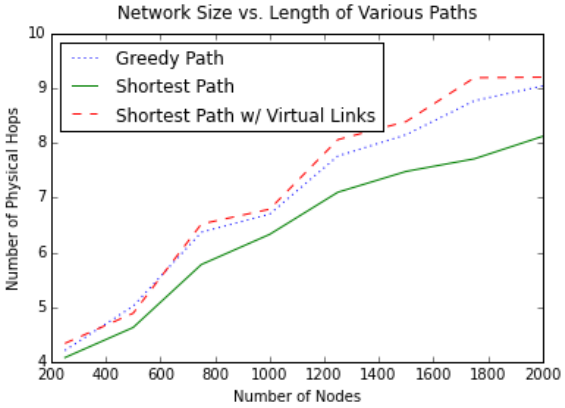


Figure 7.4. The length of the greedy path, the shortest path, and the shortest path when a virtual link is considered as a single hop of 100 random node pairs. (n=250-2000)

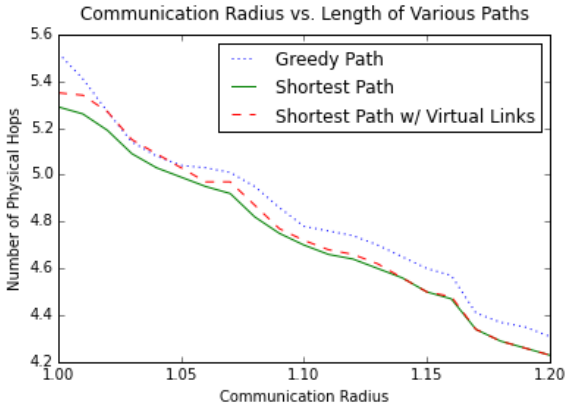


Figure 7.5. The length of the greedy path, the shortest path, and the shortest path when a virtual link is considered as a single hop of 100 random node pairs. On a network with 1000 nodes with different communication ranges.

the repaired path in the static setting. That is, we only consider each link to fail once with the probability designated. We compare the length of the greedy path after link failure to the length of shortest path after link failure and the greedy path before link failure. The gap between the greedy path after link failure and the other two paths increases as the probability of link failure increases. Figure 7.7 shows the average number of additional entries in a routing table of a node. Predictably, the average number of additional entries in a routing table of a node increases as the probability of link failure increases.

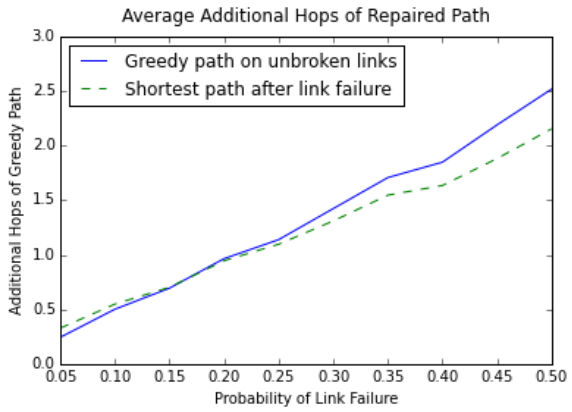


Figure 7.6. The average number of additional hops the repaired greedy path contains compared to the greedy path on unbroken links and the shortest path after link failure. (n=1000)

In the dynamic setting, when links fail, we assume that they are not repaired and the broken links compound as time increases. Figures 7.8 to 7.10 show the effect of compounding link failure on a network of 1000 nodes. Of course, if the links in the network are allowed to fail with no possibility of repair, eventually the entire network is disconnected. This is illustrated in Figure 7.9 which shows the proportion of 100 randomized node pairs that become disconnected. In Figure 7.10 the routing tables of the nodes increase as time goes on and plateaus when all test pairs become disconnected. Figure 7.9 shows the average number of additional hops of the repaired paths of 100 random node pairs. In that figure, the number of additional hops steadily increases until at a certain point, the chart dips and spikes. At some point, certain pairs become disconnected and are removed from calculation which causes the dips in the chart since the disconnected paths are likely to have been longer paths beforehand. Note that the 100 random pairs of nodes are randomly chosen in the beginning and remain the same for all time steps.

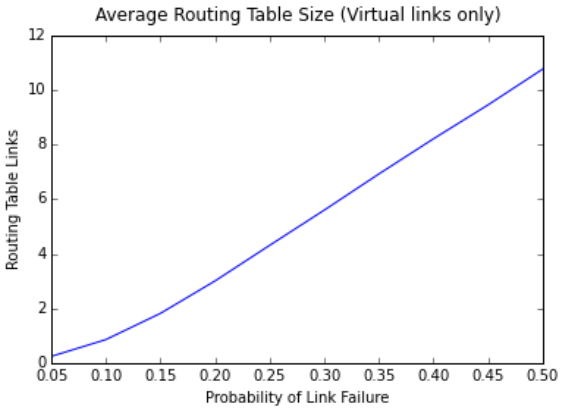


Figure 7.7. The average number of additional routing table entries for different probabilities of link failure. (n=1000)

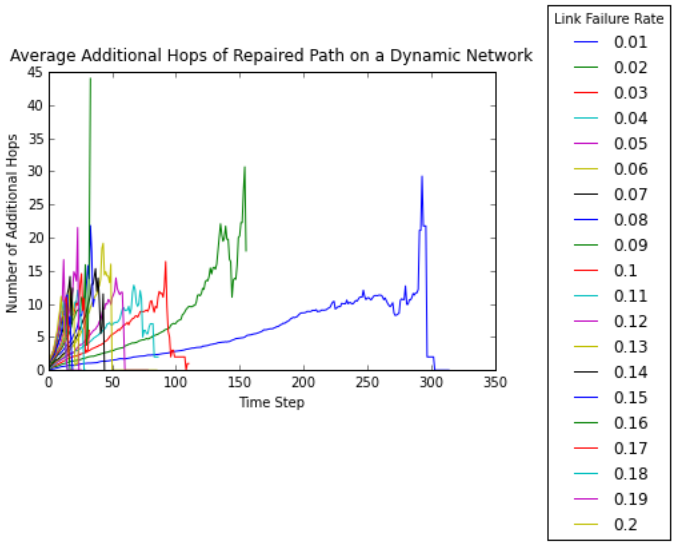


Figure 7.8. The average number of additional hops of the paths between 100 randomized start/finish pairs over time. (n=1000)

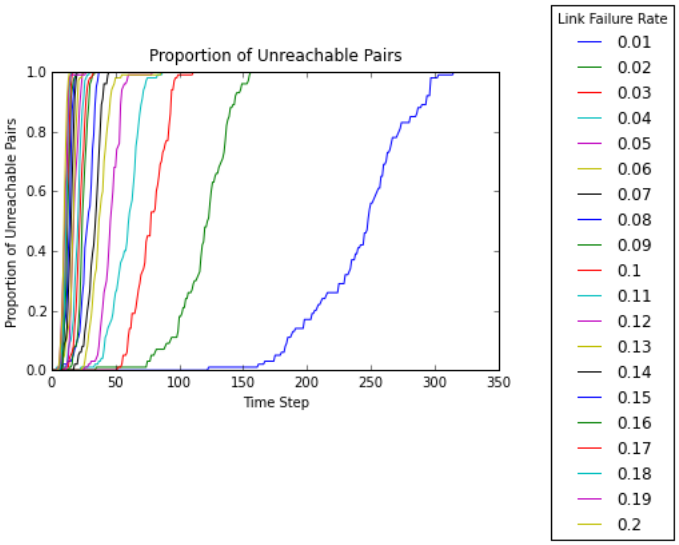


Figure 7.9. The proportion of 100 randomized start/finish pairs that cannot reach each other in the network as time goes on. (n=1000)

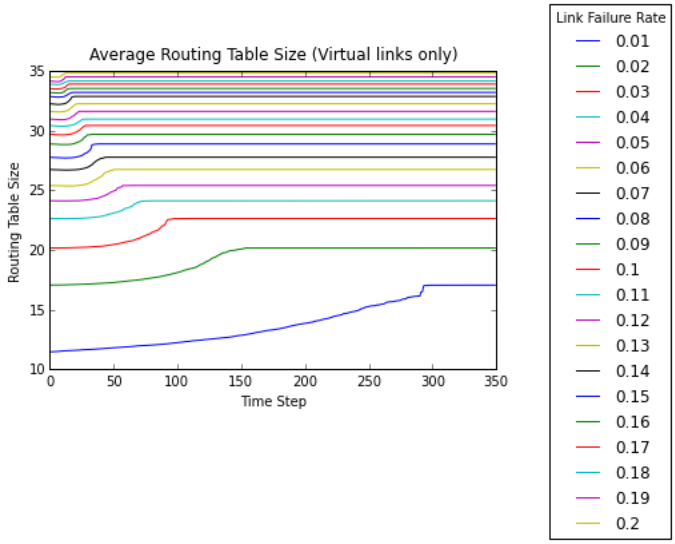


Figure 7.10. The average number of additional entries in a routing table created while 100 messages are routed with randomized origins and destinations. The routing tables are monitored over 350 time steps. (n=1000)

Finally, we run the same link failure experiments on a network simulated to reflect the real world INDRIYA testbed. The physical coordinates of roughly 100 sensors and their connectivity topology are simulated in python. Figures 7.11 to 7.14 show that this networks behaves very similarly to our randomly generated network.

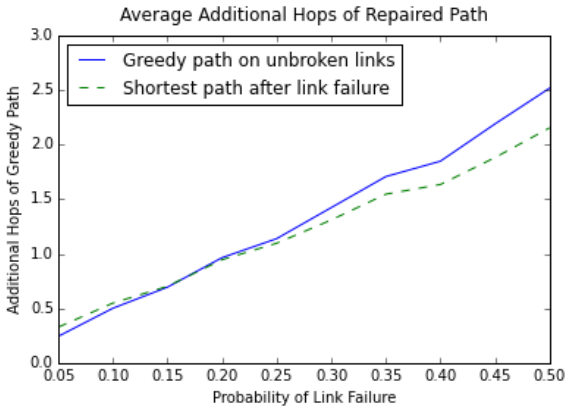


Figure 7.11. The average number of additional hops in a greedy path on the mended network over all pairs of nodes. On a real world network.

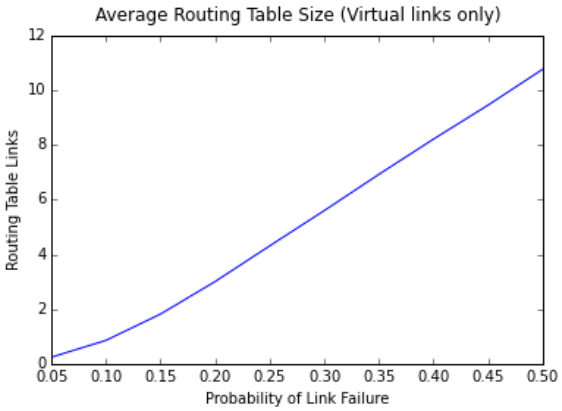


Figure 7.12. The average number of additional entries in a routing table after link failure and messages have been sent between all pairs of nodes. On a real world network.

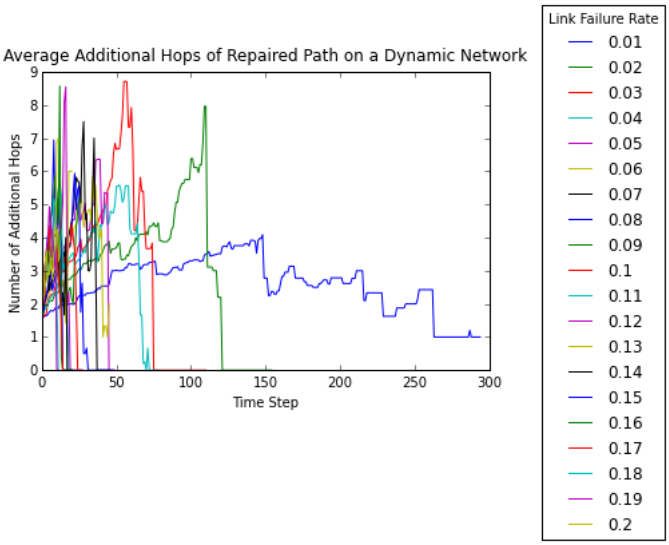


Figure 7.13. The average number of additional hops of greedy paths as time increases. On a real world network, averaged over 100 randomized start/finish pairs.

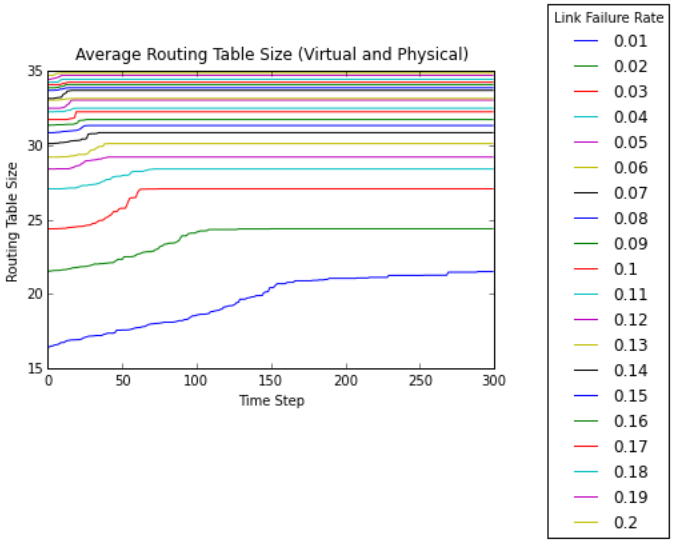


Figure 7.14. The average number of additional entries in a routing table as messages are sent between 100 randomized node pairs. On a real world network.

7.4 Conclusion

In this section we have found ways to amend a network in higher dimensions to guarantee the effectiveness of the simple and intuitive greedy algorithm. Through experiments, we have shown the effectiveness of our procedure and that its overhead is lower than our already low theoretical bounds. We emphasize the practicality of our algorithm among a field of routing algorithms that may be effective in 2D but are not workable in 3D.

Chapter 8

Future Work

There are many further avenues we would like to investigate in several projects. We detail the questions here:

The Data Gathering Problem We would like to further close the gap for the variations of The Data Gathering Problem we have explored. While we have many approximation algorithms, for our lower bounds, we have only proven NP-hardness for most of the problems. We suspect that these lower bounds can be raised.

The Shortest Separator Problem While we have narrowed the gap in theoretical bounds for The Shortest Separator Problem, it is still quite large and we would like to narrow them further.

***r*-gather Clustering** While we have narrowed the theoretical bounds for both variation of r-gather clustering, there are other avenues we would like to explore.

The 2-approximation for r-gather is for the variation where the definition of the diameter of a cluster is the diameter of the smallest enclosing circle of the cluster. Do these approximation algorithms apply for the variation where the diameter is the distance between the furthest pair of points? Also, can we fine tune our understanding of the problem? Is r-gather in FPT?

We have begun to explore the dynamic version of r-gather. However, we still know little about the most general definition of this problem. Also, can we find a faster or more stable algorithm that has an advantage over just recomputing a new clustering every step?

Bibliography

[1] National traveling salesman problems. `http://www.math.uwaterloo.ca/tsp/world/countries.html`. Accessed: 2014-12-15.

[2] M. A. Abam, P. Carmi, M. Farshi, and M. Smid. On the power of the semi-separated pair decomposition. In *Proceedings of the 11th International Symposium on Algorithms and Data Structures*, WADS '09, pages 1–12, Berlin, Heidelberg, 2009. Springer-Verlag.

[3] M. A. Abam and M. de Berg. Kinetic spanners in $\mathbb{R}^d$. *Discrete & Computational Geometry*, 45(4):723–736, June 2011.

[4] I. Abraham, D. Dolev, and D. Malkhi. LLS: A locality aware location service for mobile ad hoc networks. In *Proceedings of the 2004 Joint Workshop on Foundations of Mobile Computing*, pages 75–84, 2004.

[5] G. Aggarwal, R. Panigrahy, T. Feder, D. Thomas, K. Kenthapadi, S. Khuller, and A. Zhu. Achieving anonymity via clustering. *ACM Transactions on Algorithms*, 6(3):49:1–49:19, July 2010.

[6] K. Almi'ani, A. Viglas, and L. Libman. Tour and path planning methods for efficient data gathering using mobile elements. *International Journal of Ad Hoc and Ubiquitous Computing*, 2014.

[7] N. Alon, L. Babai, and A. Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of Algorithms*, 7(4):567–583, 1986.

[8] G. Anastasi, M. Conti, and M. Di Francesco. Data collection in sensor networks with data mules: An integrated simulation analysis. In *IEEE Symposium on Computers and Communications*, pages 1096–1102, July 2008.

[9] E. Anshelevich and A. Karagiozova. Terminal backup, 3D matching, and covering cubic graphs. *SIAM Journal on Computing*, 40(3):678–708, 2011.

[10] A. Archer, M. Bateni, M. Hajiaghayi, and H. Karloff. Improved approximation algorithms for prize-collecting steiner tree and tsp. *SIAM Journal on Computing*, (2):309–332, Mar.

[11] C. Archetti, M. Speranza, and D. Vigo. Vehicle routing problems with profits. Technical report, Tech. Report WPDEM2013/3, University of Brescia, 2013.

[12] E. Arkin, J. S. B. Mitchell, and G. Narasimhan. Resource-constrained geometric network optimization. In *Proceedings of the 14th Annual ACM Symposium on Computational Geometry*, pages 307–316, 1998.

[13] E. M. Arkin and R. Hassin. Approximation algorithms for the geometric covering salesman problem. *Discrete Applied Mathematics*, 55(3):197–218, Dec. 1994.

[14] E. M. Arkin, R. Hassin, and A. Levin. Approximations for minimum and min-max vehicle routing problems. *Journal of Algorithms*, (1):1 – 18.

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

[16] A. Armon. On min–max r-gatherings. *Theoretical Computer Science*, 412(7):573–582, 2011.

[17] S. Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *Journal of the ACM*, 45(5):753–782, sep 1998.

[18] S. Arora and K. Chang. Approximation schemes for degree-restricted MST and redblue separation problems. *Algorithmica*, 40(3):189–210, 2004.

[19] G. Ausiello, V. Bonifaci, and L. Laura. The online prize-collecting traveling salesman problem. *Information Processing Letters*, (6):199 – 204.

[20] G. Ausiello, S. Leonardi, and A. Marchetti-Spaccamela. In *Algorithms and Complexity*.

[21] G. Ausiello, M. Protasi, A. Marchetti-Spaccamela, G. Gambosi, P. Crescenzi, and V. Kann. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1st edition, 1999.

[22] B. Awerbuch, Y. Azar, A. Blum, and S. Vempala. Improved approximation guarantees for minimum-weight k-trees and prize-collecting salesmen. In *SIAM Journal on Computing*, pages 277–283, 1995.

[23] X. Bai, S. Kumar, D. Xuan, Z. Yun, and T. H. Lai. Deploying wireless sensors to achieve both coverage and connectivity. In *Proceedings of the 7th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pages 131–142, 2006.

[24] E. Balas. The prize collecting traveling salesman problem. *Networks*, 19(6):621–636, 1989.

[25] N. Bansal, A. Blum, S. Chawla, and A. Meyerson. Approximation algorithms for deadline-tsp and vehicle routing with time-windows. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, STOC '04, pages 166–174, New York, NY, USA. ACM.

[26] D. Bienstock, M. X. Goemans, D. Simchi-Levi, and D. Williamson. A note on the prize collecting traveling salesman problem. *Mathematical Programming*, (3):413–420, May.

[27] L. Blazević, L. Buttyán, S. Capkun, S. Giordano, H. Hubaux, and J. L. Boudec. Self-organization in mobile ad hoc networks: the approach of terminodes. *IEEE Communications Magazine*, pages 166–175, 2001.

[28] A. Blum, S. Chawla, D. Karger, T. Lane, A. Meyerson, and M. Minkoff. Approximation algorithms for orienteering and discounted-reward tsp. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, pages 46–55, Oct 2003.

[29] 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.

[30] H. Brönnimann and M. T. Goodrich. Almost optimal set covers in finite VC dimensions. *Discrete & Computational Geometry*, 14:463–479, 1995.

[31] K. Buchin, M. Löffler, P. Morin, and W. Mulzer. Preprocessing imprecise points for delaunay triangulation: Simplified and extended. *Algorithmica*, 61(3):674–693, 2011.

[32] K. Buchin and W. Mulzer. Delaunay triangulations in $o(sort(n))$ time and more. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '09, pages 139–148, Washington, DC, USA, 2009. IEEE Computer Society.

[33] J. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, S. Reddy, and N. Srivanstava. Participatory sensing. In *Proceedings of the World Sensor Web Workshop*, 2006.

[34] P. B. Callahan and S. R. Kosaraju. Faster algorithms for some geometric graph problems in higher dimensions. In *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '93, pages 291–300, Philadelphia, PA, USA, 1993. Society for Industrial and Applied Mathematics.

[35] A. T. Campbell, S. B. Eisenman, N. D. Lane, E. Miluzzo, and R. A. Peterson. People-centric urban sensing. In *Proceedings of the 2nd Annual International Workshop on Wireless Internet*, New York, NY, USA ACM, 2006.

[36] T. H. Chan and S. H. Jiang. Reducing curse of dimensionality: Improved PTAS for TSP (with neighborhoods) in doubling metrics. In *Proceedings of the 27rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 754–765, 2016.

[37] T. M. Chan. Polynomial-time approximation schemes for packing and piercing fat objects. *Journal of Algorithms*, 46:178–189, 2003.

[38] T. M. Chan. Optimal partition trees. *Discrete & Computational Geometry*, 47(4):661–690, 2012.

[39] T. M. Chan, E. Grant, J. Könemann, and M. Sharpe. Weighted capacitated, priority, and geometric set cover via improved quasi-uniform sampling. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 1576–1585, 2012.

[40] C. Chekuri, K. L. Clarkson, and S. Har-Peled. On the multi-cover problem in geometric settings. In *Proceedings of the 25th Annual ACM Symposium on Computational Geometry*, pages 341–350, 2009.

[41] C. Chekuri, N. Korula, and M. Pál. Improved algorithms for orienteering and related problems. *ACM Transactions on Algorithms*, (3):23:1–23:27, July.

[42] K. Chen and S. Har-Peled. The euclidean orienteering problem revisited. *SIAM Journal on Computing*, 38(1):385–397, 2008.

[43] X. Chen, E. Santos-Neto, and M. Ripeanu. Crowdsourcing for on-street smart parking. In *Proceedings of the Second ACM International Symposium on Design and Analysis of Intelligent Vehicular Networks and Applications*, pages 1–8, 2012.

[44] O. Chipara, C. Lu, T. C. Bailey, and G.-C. Roman. Reliable clinical monitoring using wireless sensor networks: Experiences in a step-down hospital unit. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, pages 155–168, New York, NY, USA, 2010. ACM.

[45] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie Mellon University, 1976.

[46] K. Clarkson. Approximation algorithms for shortest path motion planning. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, STOC '87, pages 56–65, New York, NY, USA, 1987. ACM.

[47] K. L. Clarkson and K. Varadarajan. Improved approximation algorithms for geometric set cover. *Discrete & Computational Geometry*, 37:43–58, 2007.

[48] S. Coene and F. C. R. Spieksma. Profit-based latency problems on the line. *Operations Research Letters*, (3):333–337, May.

[49] B. Cărbunar, A. Grama, J. Vitek, and O. Cărbunar. Redundancy and coverage detection in sensor networks. *ACM Transactions on Sensor Networks*, 2(1):94–128, February 2006.

[50] D. Cuff, M. Hansen, and J. Kang. Urban sensing: out of the woods. *Communications of the ACM*, 51(3):24–33, March 2008.

[51] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 3rd edition, 2008.

[52] O. Devillers. Delaunay triangulation of imprecise points, preprocess and actually get a fast query time. *Journal of Computational Geometry*, pages 30–45, 2011.

[53] M. Doddavenkatappa, M. Chan, and A. Ananda. Indriya: A low-cost, 3d wireless sensor network testbed. In T. Korakis, H. Li, P. Tran-Gia, and H.-S. Park, editors, *Testbeds and Research Infrastructure. Development of Networks and Communities*, volume 90 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 302–316. Springer Berlin Heidelberg, 2012.

[54] A. Dumitrescu and J. S. Mitchell. Approximation algorithms for TSP with neighborhoods in the plane. *Journal of Algorithms*, 48(1):135 – 159, 2003. Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms.

[55] S. Durocher, D. Kirkpatrick, and L. Narayanan. On routing with guaranteed delivery in three-dimensional ad hoc wireless networks. In *Proceedings of the 9th International Conference on Distributed Computing and Networking*, ICDCN'08, pages 546–557, Berlin, Heidelberg, 2008. Springer-Verlag.

[56] S. B. Eisenman, E. Miluzzo, N. D. Lane, R. A. Peterson, G.-S. Ahn, and A. T. Campbell. Bikenet: A mobile sensing system for cyclist experience mapping. *ACM Transactions on Sensor Networks*, 6(1):6:1–6:39, January 2010.

[57] B. Eksioglu, A. V. Vural, and A. Reisman. Survey: The vehicle routing problem: A taxonomic review. *Computers & Industrial Engineering*, (4):1472–1483, Nov.

[58] 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.

[59] J. Eriksson, L. Girod, B. Hull, R. Newton, S. Madden, and H. Balakrishnan. The pothole patrol: using a mobile sensor network for road surface monitoring. In *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services*, pages 29–39, 2008.

[60] G. Even, N. Garg, J. Knemann, R. Ravi, and A. Sinha. Covering graphs using trees and stars. In *Approximation, Randomization, and Combinatorial Optimization.. Algorithms and Techniques*, volume 2764 of *Lecture Notes in Computer Science*, pages 24–35. 2003.

[61] G. Even, D. Rawitz, and S. Shahar. Hitting sets when the VC-dimension is small. *Information Processing Letters*, 95:358–362, 2005.

[62] E. Ezra and W. Mulzer. Convex hull of imprecise points in $o(n \log n)$ time after preprocessing. In *Proceedings of the 27th Annual ACM Symposium on Computational Geometry*, SCG '11, pages 11–20, New York, NY, USA, 2011. ACM.

[63] U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45:634–652, 1998.

[64] D. Feillet, P. Dejax, and M. Gendreau. Traveling salesman problems with profits. *Transportation Science*, (2):188–205, May.

[65] R. Fowler, M. Paterson, and S. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Information Processing Letters*, 12:133–137, 1981.

[66] H. Frey and I. Stojmenovic. On delivery guarantees of face and combined greedy-face routing in ad hoc and sensor networks. In *Proceedings of the 12th Annual International Conference on Mobile Computing and Networking*, MobiCom '06, pages 390–401, New York, NY, USA, 2006. ACM.

[67] 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.

[68] J. Gao, L. J. Guibas, and A. Nguyen. Deformable spanners and applications. In *Proceedings of the 20th Annual ACM Symposium on Computational Geometry*, pages 179–199, 2004.

[69] M. R. Garey, R. L. Graham, and D. S. Johnson. Some NP-complete geometric problems. In *Proceedings of the 8th Annual ACM Symposium on Theory of Computing*, STOC '76, pages 10–22, New York, NY, USA, 1976. ACM.

[70] B. L. Golden, L. Levy, and R. Vohra. The orienteering problem. *Naval Research Logistics*, 34:307–318, 1987.

[71] M. C. Gonzalez, C. A. Hidalgo, and A.-L. Barabasi. Understanding individual human mobility patterns. *Nature*, 453(7196):779–782, June 2008.

[72] L.-A. Gottlieb and L. Roditty. An optimal dynamic spanner for doubling metric spaces. In *Proceedings of the 16th Annual European Symposium on Algorithms*, ESA '08, pages 478–489, Berlin, Heidelberg, 2008. Springer-Verlag.

[73] Y.-J. K. R. Govindan, B. Karp, and S. Shenker. Lazy cross-link removal for geographic routing. In *Proceedings of the 4th ACM Conference on Embedded Networked Sensor Systems*, pages 112–124, 2006.

[74] M. Grossglauser and D. Tse. Mobility increases the capacity of ad hoc wireless networks. *IEEE/ACM Transactions on Networking*, 10(4):477–486, August 2002.

[75] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2. of Algorithms and Combinatorics, Springer, 1988.

[76] J. Gudmundsson and C. Levcopoulos. Hardness result for TSP with neighborhoods. Technical report, Technical Report LU-CS-TR:2000-216, Department of Computer Science, Lund University, Sweden, 2000.

[77] S. Guha, A. Meyerson, and K. Munagala. Hierarchical placement and network design problems. Technical report, Stanford, CA, USA, 2000.

[78] L. Guibas, D. Salesin, and J. Stolfi. Constructing strongly convex approximate hulls with inaccurate primitives. In *Proceedings of the International Symposium on Algorithms*, SIGAL '90, pages 261–270, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[79] P. Gupta and P. R. Kumar. The capacity of wireless networks. *IEEE Transactions on Information Theory*, 46(2):388–404, 2000.

[80] M. Hefeeda and M. Bagheri. Wireless sensor networks for early detection of forest fires. In *Proceedings of the 2007 IEEE International Conference on Mobile Adhoc and Sensor Systems*, pages 1–6, Oct 2007.

[81] M. Held and J. S. B. Mitchell. Triangulating input-constrained planar point sets. *Information Processing Letters*, 109(1):54–56, Dec. 2008.

[82] D. S. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *Journal of the ACM*, 32:130–136, 1985.

[83] D. S. Hochbaum and A. Pathria. Analysis of the greedy approach in problems of maximum k-coverage. *Naval Research Logistics*, 45(6):615–627, 1998.

[84] C.-F. Huang, Y.-C. Tseng, and H.-L. Wu. Distributed protocols for ensuring both coverage and connectivity of a wireless sensor network. *ACM Transactions on Sensor Networks*, 3(1), March 2007.

[85] R. Jain, A. Puri, and R. Sengupta. Geographical routing using partial information for wireless ad hoc networks. *IEEE Personal Communications*, 8(1):48–57, Feb. 2001.

[86] D. Jea, A. Somasundara, and M. Srivastava. Multiple controlled mobile elements (data mules) for data collection in sensor networks. In *Proceedings of the First IEEE International Conference on Distributed Computing in Sensor Systems*, DCOSS '05, pages 244–257, 2005.

[87] A. Kansal, M. Rahimi, W. J. Kaiser, M. B. Srivastava, G. J. Pottie, and D. Estrin. Controlled mobility for sustainable wireless networks. In *IEEE Sensor and Ad Hoc Communications and Networks*.

[88] D. R. Karget and M. Minkoff. Building steiner trees with incomplete global knowledge. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science*, FOCS '00, pages 613–, Washington, DC, USA, 2000. IEEE Computer Society.

[89] H. J. Karloff. How long can a Euclidean traveling salesman tour be? *SIAM Journal on Discrete Mathematics*, 2(1):91–99, 1989.

[90] B. Karp and H. Kung. GPSR: Greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, MobiCom '00, pages 243–254, 2000.

[91] J. M. Keil. Approximating the complete euclidean graph. In *Proceedings of the 1st Scandinavian Workshop on Algorithm Theory*, pages 208–213, London, UK, UK, 1988. Springer-Verlag.

[92] R. Kershner. The number of circles covering a set. *American Journal of Mathematics*, 61(3):665–671, 1939.

[93] M. R. Khani and M. R. Salavatipour. Improved approximation algorithms for the min-max tree cover and bounded tree cover problems. *Algorithmica*, 69(2):443–460, 2014.

[94] D. Kim, R. Uma, B. Abay, W. Wu, W. Wang, and A. Tokuta. Minimum latency multiple data mule trajectory planning in wireless sensor networks. *IEEE Transactions on Mobile Computing*, 13(4):838–851, April 2014.

[95] Y.-J. Kim, R. Govindan, B. Karp, and S. Shenker. Geographic routing made practical. In *Proceedings of the Second USENIX/ACM Symposium on Networked System Design and Implementation*, May 2005.

[96] Y.-J. Kim, R. Govindan, B. Karp, and S. Shenker. On the pitfalls of geographic face routing. In *Proceedings of the 2005 Joint Workshop on Foundations of Mobile Computing*, pages 34–43, 2005.

[97] R. Kleinberg. Geographic routing using hyperbolic space. In *Proceedings of the 26th Conference of the IEEE Communications Society*, pages 1902–1909, 2007.

[98] F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger. Geometric ad-hoc routing: Of theory and practice. In *Proceedings of the 22nd ACM International Symposium on the Principles of Distributed Computing*, pages 63–72, 2003.

[99] F. Kuhn, R. Wattenhofer, and A. Zollinger. Asymptotically optimal geometric mobile ad-hoc routing. In *Proceedings of the 6th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pages 24–33, 2002.

[100] S. Kumar, T. H. Lai, and J. Balogh. On k-coverage in a mostly sleeping sensor network. In *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking*, MobiCom '04, pages 144–158, 2004.

[101] S. Kuznetsov and E. Paulos. Participatory sensing in public spaces: activating urban surfaces with sensor probes. In *Proceedings of the 8th ACM Conference on Designing Interactive Systems*, pages 21–30, 2010.

[102] N. D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. T. Campbell. A survey of mobile phone sensing. *IEEE Communications Magazine*, 48(9):140–150, September 2010.

[103] G. Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, (3):345 – 358.

[104] U. Lee, B. Zhou, M. Gerla, E. Magistretti, P. Bellavista, and A. Corradi. Mobeyes: smart mobs for urban monitoring with a vehicular sensor network. *IEEE Wireless Communications*, 13(5):52–57, October 2006.

[105] M. Löffler and J. M. Phillips. Shape fitting on point sets with probability distributions. *Computing Research Repository*, abs/0812.2967, 2008.

[106] M. Löffler and J. Snoeyink. Delaunay triangulation of imprecise points in linear time after preprocessing. *Computational Geometry: Theory and Applications*, 43(3):234–242, Apr. 2010.

[107] M. Ma and Y. Yang. Data gathering in wireless sensor networks with mobile collectors. In *IEEE International Symposium on Parallel and Distributed Processing*, pages 1–9. IEEE, 2008.

[108] N. Maisonneuve, M. Stevens, M. E. Niessen, P. Hanappe, and L. Steels. Citizen noise pollution monitoring. In *Proceedings of the 10th Annual International Conference on Digital Government Research: Social Networks: Making Connections between Citizens, Data and Government*, pages 96–103, 2009.

[109] C. S. Mata and J. S. B. Mitchell. Approximation algorithms for geometric tour and network design problems. In *Proceedings of the 11th Annual ACM Symposium on Computational Geometry*, SCG '95, pages 360–369, New York, NY, USA, 1995. ACM.

[110] J. Matousek. Range searching with efficient hierarchical cuttings. In *Proceedings of the 8th Annual ACM Symposium on Computational Geometry*, pages 276–285, 1992.

[111] J. S. B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric tsp, k-mst, and related problems. *SIAM Journal on Computing*, 28(4):1298–1309, 1999.

[112] J. S. B. Mitchell. A PTAS for TSP with neighborhoods among fat regions in the plane. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 11–18, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[113] J. S. B. Mitchell. A constant-factor approximation algorithm for TSP with pairwise-disjoint connected neighborhoods in the plane. In *Proceedings of the 26th Annual ACM Symposium on Computational Geometry*, pages 183–191, 2010.

[114] T. Nagai, S. Yasutome, and N. Tokura. Convex hull problem with imprecise input. In *Revised Papers from the Japanese Conference on Discrete and Computational Geometry*, JCDCG '98, pages 207–219, London, UK, UK, 2000. Springer-Verlag.

[115] V. Nagarajan and R. Ravi. Approximation algorithms for distance constrained vehicle routing problems. *Networks*, 59(2):209–214, 2012.

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

[117] J. Newsome and D. Song. GEM: graph embedding for routing and data-centric storage in sensor networks without geographic information. In *Proceedings of the 1st ACM Conference on Embedded Networked Sensor Systems*, pages 76–88, 2003.

[118] C. H. Papadimitriou. The Euclidean travelling salesman problem is NP-complete. *Theoretical Computer Science*, 4(3):237 – 244, 1977.

[119] V. Pillac, M. Gendreau, C. Guéret, and A. L. Medaglia. A review of dynamic vehicle routing problems. *European Journal of Operational Research*, (1):1 – 11.

[120] J. M. Rabaey, M. J. Ammer, J. L. da Silva, D. Patel, and S. Roundy. Picoradio supports ad hoc ultra-low power wireless networking. *Computer*, 33(7):42–48, Jul 2000.

[121] 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 *Proceedings of the 1st ACM Workshop on Wireless Sensor Networks and Applications*, pages 78–87, 2002.

[122] L. Roditty. Fully dynamic geometric spanners. In *Proceedings of the 23rd Annual ACM Symposium on Computational Geometry*, SCG '07, pages 373–380, New York, NY, USA, 2007. ACM.

[123] S. Safra and O. Schwartz. On the complexity of approximating TSP with neighborhoods and related problems. In *Proc. 11th Annu. European Sympos. Algorithms*, volume 2832 of *Lecture Notes Comput. Sci.*, pages 446–458. Springer-Verlag, 2003.

[124] D. Salesin, J. Stolfi, and L. Guibas. Epsilon geometry: Building robust algorithms from imprecise computations. In *Proceedings of the 5th Annual ACM Symposium on Computational Geometry*, SCG '89, pages 208–217, New York, NY, USA, 1989. ACM.

[125] R. Sarkar and J. Gao. Differential forms for target tracking and aggregate queries in distributed networks. In *Proceedings of the 16th Annual International Conference on Mobile Computing and Networking*.

[126] R. Sarkar, X. Yin, J. Gao, F. Luo, and X. D. Gu. Greedy routing with guaranteed delivery using ricci flows. In *Proceedings of the 8th International Symposium on Information Processing in Sensor Networks*, pages 97–108, April 2009.

[127] R. Sarkar, X. Zhu, and J. Gao. Double rulings for information brokerage in sensor networks. *IEEE/ACM Transactions on Networking*, 17(6):1902–1915, Dec. 2009.

[128] K. Seada, A. Helmy, and R. Govindan. On the effect of localization errors on geographic face routing in sensor networks. In *Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks*, pages 71–80, 2004.

[129] R. C. Shah, S. Roy, S. Jain, and W. Brunette. Data mules: Modeling a three-tier architecture for sparse sensor networks. In *Proceedings of the 1st IEEE International Workshop on Sensor Network Protocols and Applications*, pages 30–41, 2003.

[130] A. Shalita and U. Zwick. Efficient algorithms for the 2-gathering problem. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '09, pages 96–105, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.

[131] P. Slavík. The errand scheduling problem. Technical report 97-2, Department of Computer Science, SUNY, Buffalo, 1997.

[132] A. Somasundara, A. Kansal, D. Jea, D. Estrin, and M. Srivastava. Controllably mobile infrastructure for low energy embedded networks. *IEEE Transactions on Mobile Computing*, 5(8):958–973, Aug 2006.

[133] A. A. Somasundara, A. Ramamoorthy, and M. B. Srivastava. Mobile element scheduling with dynamic deadlines. *IEEE Transactions on Mobile Computing*, 6(4):395–410, 2007.

[134] I. Stojmenovic and X. Lin. Loop-free hybrid single-path/flooding routing algorithms with guaranteed delivery for wireless networks. *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1023–1032, 2001.

[135] I. Stojmenovic and J. Wu. Broadcasting and activity scheduling in ad hoc networks. *Mobile Ad Hoc Networking*, pages 205–229, 2004.

[136] Z. Svitkina. Lower-bounded facility location. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '08, pages 1154–1163, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.

[137] L. Sweeney. k-anonymity: a model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10:557–570, Oct. 2002.

[138] D. Tian and N. D. Georganas. *Wireless Communications and Mobile Computing*, (2):271–290.

[139] M. van Kreveld, M. Löffler, and J. S. B. Mitchell. Preprocessing imprecise points and splitting triangulations. *SIAM Journal on Computing*, 39(7):2990–3000, June 2010.

[140] P. Vansteenwegen, W. Souffriau, and D. V. Oudheusden. The orienteering problem: A survey. *European Journal of Operational Research*, (1):1 – 10.

[141] V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, New York, 2001.

[142] Z. Vincze and R. Vida. Multi-hop wireless sensor networks with mobile sink. In *Proceedings of the 2005 ACM Conference on Emerging Network Experiments and Technologies*, CoNEXT '05, pages 302–303, New York, NY, USA, 2005. ACM Press.

[143] B. Wang. Coverage problems in sensor networks: A survey. *ACM Computing Surveys*, 43(4):32:1–32:53, October 2011.

[144] B. Wang, K. C. Chua, V. Srinivasan, and W. Wang. Scheduling sensor activity for point information coverage in wireless sensor networks. In *Proceedings of the 4th International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, pages 1–8, April 2006.

[145] T. Wark, P. Corke, P. Sikka, L. Klingbeil, Y. Guo, C. Crossman, P. Valencia, D. Swain, and G. Bishop-Hurley. Transforming agriculture through pervasive wireless sensor networks. *IEEE Pervasive Computing*, 6(2):50–57, April 2007.

[146] F.-J. Wu and Y.-C. Tseng. Energy-conserving data gathering by mobile mules in a spatially separated wireless sensor network. *Wireless Communications and Mobile Computing*, 13(15).

[147] G. Xing, X. Wang, Y. Zhang, C. Lu, R. Pless, and C. Gill. Integrated coverage and connectivity configuration for energy conservation in sensor networks. *ACM Transactions on Sensor Networks*, 1(1):36–72, Aug. 2005.

[148] A. C.-C. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM Journal on Computing*, 11(4):721–736, 1982.

[149] M. L. Yiu, C. S. Jensen, X. Huang, and H. Lu. Spacetwist: Managing the trade-offs among location privacy, query performance, and query accuracy in mobile services. In *Proceedings of the 24th International Conference on Data Engineering*, pages 366–375, 2008.

[150] W. Yu and Z. Liu. Vehicle routing problems with regular objective functions on a path. *Naval Research Logistics*, (1):34–43.

[151] G. Zhou, T. He, S. Krishnamurthy, and J. A. Stankovic. Impact of radio irregularity on wireless sensor networks. In *Proceedings of the 2nd International Conference on Mobile Systems, Applications, and Services*, pages 125–138, 2004.

[152] Z. Zhou, S. Das, and H. Gupta. Connected k-coverage problem in sensor networks. In *Proceedings of the International Conference on Computer Communications and Networks*, pages 373–378, 2004.