Fall 2010

Approximation algorithms for distributed systems

Saurav Pandit
University of Iowa

APPROXIMATION ALGORITHMS

FOR DISTRIBUTED SYSTEMS

by

Saurav Pandit

An Abstract

Of a thesis submitted in partial fulfillment of the
requirements for the Doctor of Philosophy
degree in Computer Science
in the Graduate College of
The University of Iowa

December 2010

Thesis Supervisor: Associate Professor Sriram Pemmaraju

ABSTRACT

Distributed Approximation is a new and rapidly developing discipline that lies at the crossroads of various well-established areas of Computer Science - Distributed Computing, Approximation Algorithms, Graph Theory and often, Computational Geometry. This thesis focuses on the design and analysis of distributed algorithms to solve optimization problems that usually arise in large-scale, heavily dynamic, resource constrained networks, e.g. wireless ad-hoc and sensor networks, P2P systems, mobile networks etc. These problems can often be abstracted by variations of well-known combinatorial optimization problems, such as topology control, clustering etc. Many of these problems are known to be hard (NP-complete). But we need fast and light-weight distributed algorithms for these problems, that yield near-optimal solutions.

The results presented in this thesis can be broadly divided in two parts. The first part contains a set of results that obtain improved solutions to the classic problem of computing a sparse "backbone" for Wireless Sensor Networks (WSNs). In graph-theoretic terms, the goal is to compute a spanning subgraph of the input graph, that is sparse, lightweight and has low stretch. The term "low stretch" indicates that in spite of dropping many edges, the distance between any two nodes in the graph is not increased by much. We model WSNs as geometric graphs - unit ball graphs, quasi-unit ball graphs etc. in Euclidean spaces, as well as in more general metric spaces of low doubling dimension. We identify and exploit a variety of geometric features of

those models to obtain our results.

In the second part of the thesis we focus on distributed algorithms for clustering problems. We present several distributed approximation algorithms for clustering problems (e.g., minimum dominating set, facility location problems) that improve on best known results so far. The main contribution here is the design of distributed algorithms where the running time is a "tunable" parameter. The advent of distributed systems of unprecedented scale and complexity motivates the question of whether it is possible to design algorithms that can provide non-trivial approximation guarantees even after very few rounds of computation and message exchanges. We call these algorithms "k-round algorithms". We design k-round algorithms for various clustering problems that yield non-trivial approximation factors even if k is a constant. Additionally, if k assumes poly-logarithmic values, our algorithms match or improve on the best-known approximation factors for these problems.

Abstract Approved: _____

Thesis Supervisor

_____

Title and Department

_____

Date

APPROXIMATION ALGORITHMS

FOR DISTRIBUTED SYSTEMS

by

Saurav Pandit

A thesis submitted in partial fulfillment of the
requirements for the Doctor of Philosophy
degree in Computer Science
in the Graduate College of
The University of Iowa

December 2010

Thesis Supervisor: Associate Professor Sriram Pemmaraju

Graduate College
The University of Iowa
Iowa City, Iowa

CERTIFICATE OF APPROVAL

PH.D. THESIS

This is to certify that the Ph.D. thesis of

Saurav Pandit

has been approved by the Examining Committee for the thesis requirement for the Doctor of Philosophy degree in Computer Science at the December 2010 graduation.

Thesis Committee: _____
Sriram Pemmaraju, Thesis Supervisor

_____
Kasturi Varadarajan

_____
Sukumar Ghosh

_____
James Cremer

_____
Eunjin Jung

_____
Jeffrey Ohlmann

ACKNOWLEDGEMENTS

I would like to express my gratitude to my advisor Sriram Pemmaraju for investing so much time in me and guiding me through my thesis. Not only that he helped me understand new concepts, investigate problems and obtain solutions, but he made me acquainted with the customs of academic life and research. For the academic year 2009-2010, he has supported my studies with a research assistantship (NSF Grant CCF 0915543). It has been a tremendous pleasure working so closely with him and I am proud to be able to say that I am his student.

I would also like to thank Sukumar Ghosh; it was his Distributed Systems and Algorithms class that first sparked my interest in the subject. He has always provided valuable guidance, both in and outside of academics. My thanks to Mirela Damian for working with us on our topology control results. I also thank Kasturi Varadarajan for his general guidance and introducing me to several geometric techniques I found valuable in my research. I want to thank my past and present colleagues, Rajiv Raman, Kevin Lillis, Imran Pirwani, Benton McCune, Erik Krohn, Matt Gibson, Gaurav Kanade and members of the Algorithms Reading Group, for many stimulating exchange of ideas. My sincere thanks to EJ Jung, Jim Cremer and Jeff Ohlmann for agreeing to be in my defense committee.

I thank the Department of Computer Science for supporting me with several teaching assistantships. It was a privilege being part of this department. My heartiest thanks to its faculty, staff and students.

Finally, many thanks to my parents, my family and Swagata, a special someone, for all the support and encouragement. And a big shout out to all my friends in Iowa City who has helped make this town my home for the last six years. This experience would not have been complete without you.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 Background

Distributed approximation is a new and emerging area that bridges two well-studied areas of computer science: *approximation algorithms* for discrete optimization problems and *distributed systems*. A distributed system is generally a network of *nodes* without a centralized decision making mechanism. These nodes could be connected by physical wires or they could be wireless devices communicating via radio communication. The proliferation of large-scale, complex, dynamic and highly decentralized networks (e.g. the Internet, P2P systems, sensor networks etc.) is giving rise to a large variety of challenges. Although, many of these challenges can be abstracted by traditional and well-studied algorithmic problems, the lack of centralized control and the limitation to *local* knowledge renders most of the traditional network algorithms unusable in these environments. A common example, one that we will study later in detail, is *minimum weight spanner construction*. It is often necessary in a distributed system to keep only a small number of communication links functional at a time, yet not to disconnect any node from the network at any point of time. This problem roughly reduces to the problem of constructing a low weight spanning subgraph. However, most traditional sequential algorithms for this problem are greedy and require a strict ordering of edges (according to their weight). Such global ordering cannot be implemented efficiently in a distributed environment. More

generally, many algorithms that provide guaranteed approximation factors are greedy and perform tasks in an inherently sequential manner. The fundamental challenge for distributed approximation algorithms is to relax the need for a global ordering on tasks and yet provide a guarantee on the quality of the solution.

1.1.0.1 Challenges in Distributed Computing

In most distributed systems, gathering information about the whole network topology is either too resource consuming or simply impossible due to mobility, dynamics, or churn. Hence, no node is typically able to collect or maintain a global state about the network and ideally each node should base its decision on local information only. This is the first and foremost challenge in distributed computing and provokes the question of what can and cannot be computed locally [66, 50, 55]. Furthermore, the integration of handheld *mini* devices into the Internet and advent of wireless ad hoc and sensor networks pose many additional and fundamentally new challenges. Firstly, most of these devices/nodes are battery powered. Hence energy conservation is a key issue. One approach of dealing with energy conservation is to ensure that only the necessary nodes are *awake* at any time and the remaining nodes are in some type of a *sleep* mode. This gives rise to various clustering and scheduling problems. Additionally many of the real life ad hoc and sensor networks are made of radio sensors that are built using *off-the-shelf* components. Not only do they have very limited bandwidth and working memory, often their communication capabilities are unreliable due to hardware issues, as well as due to interference of radio signals.

Due to these difficulties and due to the dynamic nature of such networks, we need algorithms that are not only *local*, but also fast so that they can be run repeatedly. Technically speaking, the (worst-case) number of rounds of communication is the most commonly used measure of efficiency of distributed algorithms. This measure is often also regarded as the *time complexity* of such algorithms.

1.1.0.2 Need for Approximation Algorithms

The study of approximation algorithms for combinatorial optimization problems (e.g. set cover, facility location etc.) is an extremely active area of research in mainstream theoretical computer science. A seminal $O(\log n)$-approximation for set cover [11] in 1979, a $O(1)$-approximation for metric facility location [36] in 2001, a $O(1)$-approximation for weighted minimum dominating set in unit disk graphs [2] in 2006 - these are just few of the many examples that demonstrate the attention researchers have been paying to approximation algorithms for decades. In the quest for better approximation algorithms researchers developed a great variety of elegant and sophisticated mathematical techniques, such as randomized rounding, the primal-dual method, semi-definite programming, metric embeddings etc.

Since most combinatorial optimization problems are NP-complete, usually there is very little hope for achieving optimal results. Hence, approximation algorithms seem to be a natural fit for this scenario. With the emergence of more and more complex distributed environments every day, *distributed approximation* has emerged as a rapidly developing field. For example, let us look at *minimum domi-*

nating set[1] (MDS), which has been studied for decades. It is an NP-complete problem. However it has been shown that it can be approximated well by a natural greedy algorithm which repeatedly adds the node that covers the most number of uncovered nodes. The greedy algorithm achieves an $\Theta(\log \Delta)$-approximation where Δ equals the maximum degree of a node plus one [38, 62]. In the distributed setting, Jia et al. [37] then relax this greedy algorithm that, in each step, picks several *good* nodes instead of picking the single best choice. This speeds up the process, but hurts the quality of the solution by selecting more nodes than necessary. To compensate for that they use a randomized *pruning* process to guarantee a $O(\log n)$-approximation.

1.1.1 Overview of Our Approach

In spite of all the aforementioned challenges, there are some aspects of distributed systems and computing that allow us to design fast and near-optimal algorithms. The first set of results in my thesis utilize the geometric nature of some distributed systems, especially those that arise in wireless radio networks. Another trademark of some of the problems we explore in this thesis seems to be an inherent "local" nature of the problem. This issue is usually not important in a sequential setting, but is critical in a distributed setting and we try to exploit it to obtain our second set of results. Next, we elaborate on these aspects.

[1] A dominating set of a graph $G = (V, E)$ is a subset of nodes $D \subseteq V$, such that each node in V are either in D or has an edge to some node in D. The MDS problem looks for a dominating set with minimum cardinality.

1.1.1.1 Underlying Geometry

Unlike the Internet, P2P systems etc., the nodes in a wireless network reside in Euclidean space. Also in such networks, each node can only communicate with a restricted set of other nodes. Given a node u, let $\Gamma(u)$ (also known as the *neighborhood* of u) be the set of nodes it is capable of exchanging messages with. In wireless networks, $\Gamma(u)$ depends on u's communication range. These communication ranges are generally shaped likes disks, polygons or more generally, geometric *fat objects*. Researchers have utilized the geometric structure of the network graphs to come up with better approximation algorithms than in general graphs.

If we start with the assumption that each node has a transmission range that is disk-like, we obtain several different models for wireless networks. The simplest of these is known as the *unit disk graph* [64] or UDG[2]. A natural 3-dimensional extension is a *unit ball graph* (UBG). These models are useful due to their simplicity, but one could argue they are not a very realistic representation of wireless networks. Some of the criticisms of the UDG (or UBG) model can be deflected by a further improved model called α-quasi unit ball graph [15, 40] or α-qUBG[3]. The α-UBG model goes beyond the unrealistic "flat world" assumption of UDGs and also takes into account transmission errors, fading signal strength, and physical obstructions. We also study

[2]In an UDG, the nodes lie on an Euclidean plane. Two nodes have an edge between them if they are at most unit distance away.

[3]In an α-qUBG ($\alpha < 1$), a node is guaranteed to have an edge between other nodes within distance α, and not to have an edge beyond unit distance. No guarantees in the annulus in between.

these problems without the assumption of an Euclidean metric. A more general assumption is metric spaces with constant doubling dimension[4](commonly known as *doubling metric*). Doubling metrics are robust under distortions in distance measures and its bounded growth property seem to be the characteristics of large-scale wireless networks.

Many combinatorial optimization problems are much more approximable in UDGs than in general graphs. The problems considered include maximum independent set, minimum vertex cover, minimum coloring, minimum dominating set etc. Marathe et al. [64] present a series of heuristics for these problems is UDGs and use the geometric properties to establish good approximation factors. Most topology control problems become easier in UDGs (or other geometric graphs) as well.

Beyond the modeling of the network, we identify various assumptions (geometric or otherwise) whose presence or absence have significant implications on the complexity and hardness of the problems arising in these networks. Some of these assumptions are: whether nodes know the coordinates of their neighbors, or just the distance information; whether the nodes know the total number of nodes in the network, or at least have an estimate.

1.1.1.2 Approximability vs Efficiency: A Trade-off

Our next set of results can be credited to a the inherent "local" nature of some of the problems we consider. For example, `MDS`, facility location, coloring problems

[4]The *doubling dimension* of a metric space is the smallest ρ such that any ball in this metric space can be covered by 2^{ρ} balls of half the radius.

etc. are local in nature, i.e. near-optimal distributed algorithms could be constructed for the problem where nodes do not need information about the whole network. In a distributed setting, we are able to obtain an optimal solution in poly-logarithmic rounds for most of these problems. But the more interesting fact is that the approximation factor degrades gracefully with decreasing number of rounds.

These problems exhibit a delicate "trade-off" between available resources (e.g. number of communication rounds, message size etc.) and the quality of the algorithm (i.e. the approximation factor). Such trade-offs are not very common in sequential algorithms. In the sequential setting, often the best known or best possible approximation factors is achieved via fairly sophisticated and efficient algorithms. For example, Jain and Vazirani present a 3-approximation [36] for metric, uncapacitated facility location. But it is not possible to run this algorithm for more number of rounds, exchange more information and obtain a better solution. But, in the distributed setting, there seems to be a more visible (negative) correlation between the approximability and the number of rounds or message size. Researchers have provided approximation guarantees for various problems with algorithms that run in polynomial, or poly-logarithmic, or even $O(\log n)$ rounds. But we seek algorithms that provide non-trivial approximation guarantees for even smaller number of rounds, e.g. what is the best we can do in k rounds, even if k is a constant? We utilize this inherent trade-off between the approximability and running time in seeking answers to such questions.

A resource that play key role in the above mentioned trade-off is the amount

of information available and the locality of that information. One common measure for the amount of available information is the *message size*. We describe our message passing models based on a classification by David Peleg [71]. In the $\mathcal{LOCAL}$ model, each node can send one message per round to its neighbors. But there is no bound on the message size. This is a crucial point in distributed computing as, with unlimited message size, a node can send out the whole neighborhood information in one round. By abstracting away the nuances of information exchange, the $\mathcal{LOCAL}$ provides an ideal abstraction layer for analyzing the effects of locality on distributed computation. Even if, all the neighborhood data can simply be made available with one round of message transactions, some problems (e.g. k-median, minimum spanning tree) still remain hard. This indicates that to solve those problem we need more than just local information. Naturally, these problems are referred to as *non-local*. On the other hand, the algorithm by Jia et al. [37] demonstrates a $O(\log n)$-approximation for `MDS` needs the nodes to know information only from their $O(\log n)$-neighborhood. The importance of the $\mathcal{LOCAL}$ model also stems from the fact that it provides a one-to-one correspondence between the notion of *time complexity* of distributed algorithms and the graph theoretic notion of *neighborhood information*. In particular, having a distributed algorithm perform k communication rounds is equivalent to a scenario in which distributed decision makers at the nodes of a graph must base their decision on (complete) knowledge about their k-hop neighborhood. To see this, note that because messages are unbounded, every node can collect the identifiers and interconnections of all nodes in its k-hop neighborhood in k communication rounds. Collecting the

complete k-neighborhood can be achieved if all nodes send their complete states to all their neighbors in every round. After round i, all nodes know their i-neighborhood. Learning the i-neighborhoods of all neighbors in round $(i+1)$ suffices to know the $(i+1)$-neighborhood. On the other hand, a node cannot obtain any information from a node at distance $(k+1)$ or more, because sending information over $(k+1)$ hops requires at least as many communication rounds. Therefore, in the $\mathcal{LOCAL}$ model, knowing ones k-hop neighborhood is exactly as powerful as employing a distributed algorithm with running time k.

However, it is impractical for most distributed systems to allow large messages. Such systems are modeled by the $\mathcal{CONGEST}$ model, where each node can send one message per round to each neighbor, where message size in bounded by $O(\log n)$ bits. The significance of $O(\log n)$ bits is that we need only $\log n$ bits to represent n unique node IDs. Also, the input constants are assumed to be representable using $\log n$ bits. The network diameter itself is not of much concern in this model, but the amount of information in the neighborhood itself could be high. For example, consider the facility location problem on a complete bipartite graph with m facilities and n clients. It is only a diameter 2 graph. But there are mn pieces of information just to define all the connection costs. A successful algorithm in the $\mathcal{CONGEST}$ model will have to make a smart and careful decision about which of these pieces of information need to be exchanged. In summary, The understanding of the concept of locality has helped us successfully utilize the trade-off and design better approximation algorithms, even when message size is bounded and only small number of rounds are available.

1.1.1.3 The Primal-Dual Scheme

At the core of several of our results, especially those in the later half of this dissertation, is the *primal-dual* technique. This is known to be a powerful technique in obtaining good approximation factors for combinatorial optimization problems, e.g. vertex cover, facility location etc. Say, we are given a minimization problem. In a primal-dual scheme, we start with an integer program (IP) formulation of the given problem. We call this the *primal*. Then we consider its LP-relaxation (LP). However, we do not bother solving the LP, instead we obtain the *dual* of the LP-relaxation. The goal is to obtain an integral, feasible solution (say L) of the LP and a feasible solution D of the DP and show that $cost(L)$ is within a factor c of $cost(D)$. By the *Weak Duality Theorem*, this implies a c approximation to the original minimization problem. In a sequential setting, we start with a small but feasible dual solution and continuously increase the dual variables until their corresponding dual constraints become tight. Every time that happens we (integrally) add to the primal solution and freeze the dual variables involved. We found the primal-dual technique is an attractive choice in distributed computing. In this subsection we elaborate on that.

Firstly, for many problems the tightness of the dual constraints can be checked locally in $O(1)$ rounds of message transactions, even in the $\mathcal{CONGEST}$ model.

Secondly, this technique is easily customizable for k-round distributed algorithms. In a distributed setting, we need a fast and discrete increase of the dual variables. First we find an upper bound (say, D^*) on the value of the dual objective function. Then we need to find an initial dual feasible solution (say, D) that is not

IP:

$$\text{minimize} \sum_{i \in F, j \in C} c(i,j) \cdot x_{ij} + \sum_{i \in F} f(i) \cdot y_i$$

$$\text{subject to} \sum_{i \in F} x_{ij} \geq 1, j \in C$$

$$y_i - x_{ij} \geq 0, i \in F, j \in C$$

$$x_{ij} \in \{0,1\}, i \in F, j \in C$$

$$y_i \in \{0,1\}, i \in F$$

DP:

$$\text{maximize} \sum_{j \in C} \alpha_j$$

$$\text{subject to } \alpha_j - \beta_{ij} \leq c(i,j), i \in F, j \in C$$

$$\sum_{j \in C} \beta_{ij} \leq f(i), i \in F$$

$$\alpha_j \geq 0, j \in C$$

$$\beta_{ij} \geq 0, i \in F, j \in C$$

Figure 1.1: Integer Program for `FacLoc` and the dual of its LP-relaxation

too small. Starting from D we increase the dual variables by a certain factor in every round. For example, in our algorithm for metric `FacLoc` (see Chapter 5), the gap between D^* and D is n^2 (n being the number of clients). It is easy to see that if we increase the dual variables geometrically (say, by a constant b), we can finish the algorithm in $\log_b^2 n$ rounds. We also show that this geometric increase only introduces an factor b to the approximation factor. In case of the k-round algorithm, our increase in $n^{\frac{1}{r}}$ in the rth round. Note that, this technique underlines the trade-off between the running time and approximation factor. For example, if we want to finish the algorithm in smaller number of rounds, our increase in dual variables per round will have to be *coarser*, resulting in a higher approximation factor. On the other hand, a finer increase in dual variables will result in a better approximation factor at the cost of increased running time.

However, this discrete increase can be seen as a *relaxation* and makes a *pruning* procedure an integral part of any distributed primal-dual algorithm. We have been able to successfully use randomized sparsification techniques to satisfy our pruning

requirements, even for small number of rounds.

1.2 Results

Now, let us give a more technical overview of our results. In my thesis, we study two types of problems. The first set of problems fall under the category of *topology control* problems. These problems arise in wireless networks where each pair of nodes may or may not have an open communication link between them. Maintaining many communication links in wireless networks can be expensive, as the event of *message passing* itself is considered to have a high overhead in terms of resources (e.g. battery power). Hence, given a network graph, we try to find a spanning subgraph (or *spanner*) by selecting only a few of the given communication links, represented here by edges. However, dropping too many edges can increase (*stretch*) the shortest-path-distance between two nodes, and in turn increase communication overhead, e.g. more dependence on routing. We need to choose a sparse set of edges that spans the graph, while keeping the stretch factor low.

The second set of problems are examples of *clustering problems*, where we select certain nodes in the network to be *leaders* and every other node is assigned to a leader. The goal is to find a *backbone* of the network, so that at any point of time, the nodes in the backbone can effectively represent the whole network. The `MDS` problem, as well as its weighted version, `WMDS`) are standard examples of clustering problems. However, we pay close attention to the communication costs as well, i.e. we intend not to select too many leaders, but if we select too few the other nodes

may have to incur high costs trying to communicate with their cluster leader. This problem can be abstracted by variations of the classic *facility location* problem. All these problems are studied within the geometric or the resource trade-off context (or both).

Each of the following chapters represents a result (or a set of results) motivated by one of the two (or both) factors mentioned earlier, i.e., utilizing the geometric nature of the network to obtain better approximation guarantees and/or studying the trade-off between various resources and their effect on the quality of the output. Also, each problem encountered in these chapters can fall under the category of either topology control or clustering problems.

1.2.1 Topology Control

The next two chapters contain algorithms and analysis from the domain of topology control in wireless networks. For an overview of topology control, see the survey by Rajaraman [73]. Since an ad-hoc network does not come with fixed infrastructure, there is no topology to start with and informally speaking, the topology control problem is one of selecting neighbors for each node so that the resulting topology has a number of useful properties such as sparseness, small weight, or maximum vertex degree bounded above by a constant. Next we describe these properties in technical terms.

Let $G = (V, E)$ be the underlying network graph with edge weights $w : E \rightarrow \mathbb{R}^+$. Naturally V represents the nodes and E represents the set of edges, i.e. the set

of communication links. An edge weight is a measure for the cost of unit message transaction using that particular link. For $t \geq 1$, a t-spanner of G is a spanning subgraph G' of G such that for all pairs of vertices $u, v \in V$, the length of a shortest uv-path in G' is at most t times the length of a shortest uv-path in G. The problem of constructing a sparse t-spanner, for small t, of a given graph G has been extensively studied by researchers in distributed computing and computational geometry and more recently by researchers in ad-hoc wireless networks. Let us briefly discuss the results in these chapters.

Chapter 2 presents a distributed algorithm for wireless networks that runs in poly-logarithmic number of rounds in the size of the network and constructs a lightweight, linear size, $(1+\varepsilon)$-spanner for any given $\varepsilon > 0$. The wireless network is modeled by a d-dimensional α-UBG. The main result in this chapter is the following: for any fixed $\varepsilon > 0$, $0 < \alpha \leq 1$, and $d \geq 2$ there is a distributed algorithm running in $O(\log n \cdot \log^* n)$ communication rounds on an n-node, d-dimensional α-UBG G that computes a $(1+\varepsilon)$-spanner G' of G with maximum degree $\Delta(G') = O(1)$ and total weight $w(G') = O(w(MST(G))$. The technical contributions in this chapter include a new, sequential, greedy algorithm with relaxed edge ordering and lazy updating, and clustering techniques for filtering out unnecessary edges. This result, as mentioned earlier, is motivated by the topology control problem in wireless ad-hoc networks and improves on existing topology control algorithms along several dimensions. The results in this chapter were published [15] in the 25th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (*PODC 2006*).

Chapter 3 essentially improves the results in Chapter 2. More precisely, this chapter subsumes the results in Chapter 2 that apply to Euclidean metric spaces, and extends these results to doubling metrics. Given a quasi unit ball graph G residing in a metric space of constant doubling dimension, our algorithm constructs, for any $\varepsilon > 0$, a $(1+\varepsilon)$-spanner H of G with maximum degree bounded above by a constant. In addition, we show that H is "lightweight", in the following sense. Let Δ denote the aspect ratio of G, that is, the ratio of the length of a longest edge in G to the length of a shortest edge in G. The total weight of H is bounded above by $O(\log \Delta) \cdot wt(MST)$, where MST denotes a minimum spanning tree of the metric space. Finally, we show that H satisfies the so called *leapfrog property*, an immediate implication being that, for the special case of Euclidean metric spaces with fixed dimension, the weight of H is bounded above by $O(wt(MST))$. We employ the idea of *net trees* to obtain a hierarchical decomposition of the metric space. This decomposition plays a crucial role in determining which edges to keep and which to drop. As mentioned before, UBGs in doubling metric spaces are robust models of wireless networks and this paper is motivated by topology control for wireless networks. These results were published [14] in the 10th International Conference On Principles Of Distributed Systems (*OPODIS 2006*).

1.2.2 Clustering Problems

Our second set of results deal with efficiently partitioning the network in small clusters, so that one node from each cluster can collectively represent the whole

network. Clustering can play a critical role in increasing the performance and lifetime of wireless networks. Each cluster will have a cluster leader and we need to pay a cost for opening a node as the cluster leader. The cluster leaders stay awake at one time and the non-leader nodes can go into sleep mode and stay in touch with their respective cluster leaders time to time. As mentioned before, these scenario can be abstracted by various classic problems, e.g. *minimum dominating set problem* (MDS), *domatic partition problem*, *facility location problem* etc. Most of our results in this area deal with the facility location problems. Unlike many other clustering problems, the facility location problem pay attention to the connection costs and this lead to better clustering. Formally, the *facility location problem* takes as input a graph $G = (V, E)$, *opening costs* $f : V \to \mathbb{R}^+$ associated with nodes, and *connection costs* $c : E \to \mathbb{R}^+$ associated with the edges. The problem is to find a subset $I \subseteq V$ of nodes to open (as "facilities") and a function $\phi : V \to I$ that assigns every node ("client") to an open facility in such a way that the total cost of opening the facilities and connecting clients to open facilities is minimized.

Like the previous two chapters, the result in Chapter 4 is also motivated by the geometry of wireless networks. In this chapter, we present the first constant-factor approximation algorithm for the facility location problem on UDGs. In this version of the problem, connection costs are not *metric*, i.e., they do not satisfy the triangle inequality, because the cost of connecting to any non-neighbor can be seen as ∞. In non-metric settings the best approximation algorithms guarantee an $O(\log n)$-factor approximation, but we are able to use structural properties of UDGs

to obtain a constant-factor approximation. Our approach combines ideas from the primal-dual algorithm for facility location due to Jain and Vazirani [36] with recent results on the *weighted* MDS problem for UDGs [2, 34]. We then show that the facility location problem on UDGs is inherently *local* and one can solve local subproblems independently and combine the solutions in a simple way to obtain a good solution to the overall problem. This leads to a distributed version of our algorithm in the $\mathcal{LOCAL}$ model that runs in constant rounds and still yields a constant-factor approximation. Even if the UDG is specified without geometry, we are able to combine recent results on *maximal independent sets* and *clique partitioning* of UDGs, to obtain an $O(\log n)$-approximation that runs in $O(\log^* n)$ rounds. This paper [68] won the Best Paper award in the 10th International Conference on Distributed Computing and Networking (*ICDCN 2009*).

In Chapter 5, we present our first result motivated by the trade-off of running time and approximation factor. We present fast, distributed approximation algorithms for the *metric facility location* problem in the $\mathcal{CONGEST}$ model, where message sizes are bounded by $O(\log N)$ bits, N being the network size. We first show how to obtain a 7-approximation in $O(\log m + \log n)$ rounds via the primal-dual method; here m is the number of facilities and n is the number of clients. Subsequently, we generalize this to a k-round algorithm, that for every constant k, yields an approximation factor of $O(m^{2/\sqrt{k}} \cdot n^{2/\sqrt{k}})$. These results answer a question posed by Moscibroda and Wattenhofer (*PODC 2005*). Our techniques are based on the same primal-dual algorithm due to Jain and Vazirani [36] and a recent rapid random-

ized sparsification technique due to Gfeller and Vicari [24]. These results complement the results of Moscibroda and Wattenhofer [65] for *non-metric* facility location and extend the results of Gehweiler et al. [23] for uniform metric facility location. This paper was published [69] in the 28th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (*PODC 2009*).

In Chapter 6 we present our most recent work. This chapter provides further insight into how a rapid pruning mechanism can be fit into our primal-dual framework to design k-round algorithms. This chapter also explores the equivalence of primal-dual and greedy approaches. We start by defining a *pruning process* involving sellers on one side and buyers on the other. The goal is to quickly select a subset of the sellers so that the products that these sellers bring to the market has small *cost ratio*, i.e., the ratio of the total cost of the selected sellers' products to amount that interested buyers are willing to pay. As modeled here, the pruning process can be used to speed up distributed implementations of greedy algorithms (e.g., for MDS, facility location, etc). We present a randomized instance of the pruning process that, for any positive k, runs in $O(k)$ communication rounds in the $\mathcal{CONGEST}$ model, yielding a cost ratio of $O(N^{c/k})$. Here N is the product of the number of sellers and number of buyers and c is a small constant. Using this $O(k)$-round pruning algorithm as the basis, we derive several simple, greedy, $O(k)$-round distributed approximation algorithms for MDS and facility location (both metric and non-metric versions). Our algorithms achieve optimal approximation ratios in polylogarithmic rounds and in some cases shave a "logarithmic factor" off the best, known, approximation factor, typically

achieved using LP-rounding techniques. This paper has recently been accepted [70] in the 29th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (*PODC 2010*).

Finally, in Chapter 7, we discuss our plans to extend our current results, as well as to approach new problems that we hope can be solved with smart applications of the primal-dual/pruning mechanism.

CHAPTER 2
LOCAL APPROXIMATION SCHEMES FOR TOPOLOGY CONTROL

2.1 Introduction

Let $G = (V, E)$ be a graph with edge weights $w : E \to \mathcal{R}^+$. For $t \geq 1$, a t-spanner of G is a spanning subgraph G' of G such that for all pairs of vertices $u, v \in V$, the length of a shortest uv-path in G' is at most t times the length of a shortest uv-path in G. The problem of constructing a sparse t-spanner, for small t, of a given graph G has been extensively studied by researchers in distributed computing and computational geometry and more recently by researchers in ad-hoc wireless networks. In this chapter we present a fast distributed algorithm for constructing a linear size, lightweight t-spanner of bounded degree for any given $t > 1$, on wireless networks. Below, we describe our result more precisely.

2.1.1 Network model

We model wireless networks using d-dimensional quasi unit ball graphs. For any fixed α, $0 < \alpha \leq 1$ and integer $d \geq 2$, a *d-dimensional α-quasi unit ball graph* (*α-UBG*, in short) is a graph $G = (V, E)$ whose vertex set V can be placed in one-one correspondence with a set of points in the d-dimensional Euclidean space and whose edge set E satisfies the constraint: if $|uv| \leq \alpha$ then $\{u, v\} \in E$ and if $|uv| > 1$ then $\{u, v\} \notin E$. Here we use $|uv|$ to denote the Euclidean distance between the points corresponding to vertices u and v. The α-UBG model does not prescribe whether a pair of vertices whose distance is in the range $(\alpha, 1]$ are to be connected by an edge or

not. Specifically, if $|uv| \in (\alpha, 1]$, then it is assumed that an adversary determines if $\{u, v\} \in E$ or not. This is an attempt to take into account transmission errors, fading signal strength, and physical obstructions. Our algorithm does not need to know the locations of nodes of the α-UBG in d-dimensional Euclidean space; just the pairwise Euclidean distances.

The α-UBG model is a higher dimensional generalization of the somewhat simplistic unit disk graph (UDG) model of wireless networks that is popular in literature. Specifically, when $\alpha = 1$ and $d = 2$, a d-dimensional α-UBG is just a UDG. UDGs are attractive due to their mathematical simplicity, but have been deservedly criticized for being unrealistic models of wireless networks [45]. In our view, d-dimensional α-UBGs are a significant step towards a more realistic model of wireless networks. Two-dimensional α-UBGs were proposed in [4] as a model of wireless ad-hoc networks with unstable transmission ranges and the difficulty of doing geometric routing in such networks was shown.

Our communication model is the standard synchronous message passing model that does not account for channel access and collision issues. Time is divided into rounds and in each round, each node can send a different message to each of its neighbors, receive different messages from all neighbors and perform arbitrary (polynomial) local computation. The length of messages exchanged between nodes is logarithmic in the number of nodes. We measure the cost of our algorithm in terms of the number of communication rounds. Although this model is somewhat idealized, it is nevertheless interesting because it demonstrates the locality of computations.

2.1.2 Our result

For any edge weighted graph J, we use $w(J)$ to denote the sum of the weights of all the edges in J and $MST(J)$ to denote a minimum weight spanning tree of J. For any fixed $\varepsilon > 0$, $0 < \alpha \leq 1$, and $d \geq 2$ our algorithm runs in $O(\log n \cdot \log^* n)$ communication rounds on an n-node, d-dimensional α-UBG and computes a $(1+\varepsilon)$-spanner G' of G whose maximum degree $\Delta(G') = O(1)$ and whose total weight $w(G') = O(w(MST(G))$. Since any spanner of G has weight bounded below by $w(MST(G))$, the weight of the output produced by the algorithm is within a constant times the optimal weight. As far as we know, our result significantly improves all known results of a similar kind along several dimensions. More on this further below.

2.1.3 Topology control

Our result is motivated by the *topology control* problem in wireless ad-hoc networks. In Chapter 1 we have already provided an intuition for the topology control problems. Let us look into further technical details. Let V be a set of nodes that can communicate via wireless radios and for each $v \in V$, let $N(v)$ denote the set of all nodes that v can reach when transmitting at maximum power. The induced digraph $G = (V, E)$, where $E = \{(u, v) \mid v \in N(u)\}$, represents the network in which every node has chosen to transmit at maximum power and has designated every node it can reach as its neighbor. The topology control problem is the problem of devising an efficient and local protocol P for selecting a set of neighbors $N_P(v) \subseteq N(v)$ for each node $v \in V$. The induced digraph $G_P = (V, E_P)$, where $E_P = \{(u, v) \mid v \in N_P(u)\}$

is typically required the satisfy properties such as symmetry (if $v \in N_P(u)$ then $u \in N_P(v)$), sparseness ($|E_P| = O(|V|)$) or bounded degree ($|N_P(v)| \leq c$ for all nodes v and some constant c), and the spanner property. Sometimes stronger versions of connectivity such as k-vertex connectivity or k-edge connectivity (for $k > 1$) are desired, both for providing fault-tolerance and for improving throughput [28, 29]. If the input graph consists of nodes in the plane, it is quite common to require that the output graph be planar [57, 58, 59, 79, 81]. This requirement is motivated by the existence of simple, memory-less, geometric routing algorithms that guarantee message delivery only when the underlying graph is planar [41].

Though the topology control problem is recent, there is already an extensive body of literature on the problem to which the above sample of citations do not do justice. However, many of the topology control protocols that provide worst case guarantees on the quality of the topology, assume that the network is modeled by a UDG. A recent example [59] presents a distributed algorithm that requires a linear number of communication rounds in the worst case to compute a planar t-spanner of a given UDG with $t \approx 6.2$ and in which each node has degree at most 25. These two constants can be slightly tuned – t can be brought down to about 3.8 with a significant increase in the degree bound. We improve on the result in [59] along several dimensions. As is generally known among practitioners in ad-hoc wireless networks, the "flat world" assumption and the identical transmission range assumption of UDGs are unrealistic [45]. By using an α-UBG we generalize our model of wireless networks, hopefully moving much closer to reality. For any $\varepsilon > 0$, our algorithm returns a $(1+\varepsilon)$-

spanner; as far as we know, this is the first distributed algorithm that produces an *arbitrarily good* spanner for an α-UBG model of wireless networks. We also guarantee that the total weight of the output is within constant times optimal – a guarantee that is not provided in [59]. Finally, using algorithmic techniques and distributed data structures that might be of independent interest, we ensure that our protocol runs in $O(\log n \cdot \log^* n)$ communication rounds. We are not aware of any topology control algorithm that runs in poly-logarithmic number of rounds and provides anywhere close to the guarantees provided by our algorithm.

2.1.4 Spanners in computational geometry

Starting in the early 1990's, researchers in computational geometry have attempted to find sparse, lightweight spanners for complete Euclidean graphs. Given a set P of n points in $\mathcal{R}^d$, the tuple (P, E), where E is the set of line segments $\{\{p, p'\} \mid p, p' \in P\}$, is called the *complete Euclidean graph* on P. For any subset $E' \subseteq E$, (P, E') is called a *Euclidean graph* on P. The specific problem that researchers in computational geometry have considered, is this. Given a set P of n points in $\mathcal{R}^d$ and $t > 1$, compute a Euclidean graph on P that is a t-spanner of the complete Euclidean graph on P, whose maximum degree is bounded by $O(1)$ and whose weight is bounded by the weight of a minimum spanning tree on P. For an early example, see [56] in which the authors show that there are "planar graphs almost as good as the complete graphs and almost as cheap as minimum spanning trees." This was followed by a series of improvements [13, 16, 17, 25], with the most

recent paper [13] presenting algorithms for constructing Euclidean subgraphs that provide the additional property of k-fault tolerance. Most of the papers mentioned above start with the following simple, greedy algorithm.

---

width 4.7in

Algorithm Overview `SEQ-GREEDY` $(G = (V, E), t)$

1. Order the edges in E in non-decreasing order of length.

2. $E' \leftarrow \phi$, $G' \leftarrow (V, E')$

3. For each edge $e = \{u, v\} \in E$ if there is no uv-path in G' of length at most $t \cdot |uv|$

   (a) $E' \leftarrow E' \cup \{e\}$

   (b) $G' \leftarrow (V, E')$

Output G'.

It is well-known [17] that if the input graph $G = (V, E)$ is the complete Euclidean graph, then the output graph $G' = (V, E')$ produced by `SEQ-GREEDY` has the following useful properties: (i) G' is a t-spanner of G, (ii) $\Delta(G') = O(1)$, and (iii) $w(G') = O(w(MST(G)))$. A naive implementation of `SEQ-GREEDY` takes $O(n^3 \log n)$ time because a quadratic number of shortest path queries need to be answered on a dynamic graph with $O(n)$ edges. Consequentially, papers in this area [17, 25] focus on trying to implement `SEQ-GREEDY` efficiently. For example, Das and Narasimhan [17] show how to use certain kind of graph clustering to answer shortest path queries efficiently, thereby reducing the running time of `SEQ-GREEDY` to $O(n \log^2 n)$. One of our main contributions presented in this chapter is to show how a variant of the Das-

Narasimhan clustering scheme can be implemented and maintained efficiently, in a distributed setting.

2.1.5 Summary of our contributions

In obtaining the main result, our technical contributions are the following:

1. We first show that sparse, lightweight t-spanners for arbitrarily small $t > 1$, not only exist for d-dimensional α-UBGs, but can be computed using `SEQ-GREEDY`. Note that sparse t-spanners for arbitrarily small values of $t \geq 1$ do not exist for general graphs. For example, there is a classical graph-theoretic result that shows that for any $t \geq 1$, there exist (infinitely many) unweighted n-vertex graphs for which every t-spanner needs $\Omega(n^{1+1/(t+2)})$ edges (see Page 179 in [71]).

2. We then consider a version of `SEQ-GREEDY` in which the requirement that edges be considered in increasing order of length is relaxed. More precisely, the edges are distributed into $O(\log n)$ bins $B_0, B_1, B_2, \ldots$ such that edges in B_i are all shorter than edges in B_{i+1}. It is then shown that *any* ordering of the edges in which edges in B_0 come first, followed by edges in B_1, followed by the edges in B_2, etc., is good enough for the correctness of `SEQ-GREEDY`, even for d-dimensional α-UBGs. More importantly, we show that the update step in `SEQ-GREEDY` (Step 3(a)) need not be performed after each edge is queried. Instead, a more lazy update may be performed, after each bin is completely processed. Being able to perform a lazy update is critical for a distributed im-

plementation; roughly speaking, we want the nodes to query all edges in a bin in parallel and not to have to wait on answers to queries on other edges in a bin.

3. We also use a clustering technique as a way to reduce the number of edges to be queried per node. Reducing the number of query edges per node, is critical to being able to guarantee that the output of our distributed version of `SEQ-GREEDY` does not have too many edges incident on a node.

4. We then show that this relaxed version of `SEQ-GREEDY` can be implemented in a distributed setting in $O(\log n)$ phases — one phase corresponding to each bin — such that each phase requires $O(\log^* n)$ rounds. Each phase requires the computation of maximal independent sets (MIS) on some derived graphs. We show that the derived graphs are unit ball graphs of *constant doubling dimension* [51] and use the $O(\log^* n)$-round MIS algorithm of Kuhn et al. [51].

2.1.6 Extensions to our main result

Here we briefly report on extensions to our main result that we have obtained.

1. Let $G = (V, E)$ be an edge-weighted graph. For any $t > 1$ and positive integer k, a *k-vertex fault-tolerant t-spanner* of G is a spanning subgraph G' if for each subset S of vertices of size at most k, $G'[V \setminus S]$ is a t-spanner of $G[V \setminus S]$. A *k-edge fault-tolerant* t-spanner is defined in a similar manner. Using ideas from [13] we can extend our algorithm to produce a k-vertex (or a k-edge) fault-tolerant t-spanner in polylogarithmic number of communication rounds.

2. So far we have used Euclidean distances as weights for the edges of the input graph G. However, if the metric $c \cdot |uv|^\gamma$, for positive constant c and $\gamma \geq 1$, is used in place of Euclidean distances $|uv|$, we can show that our algorithm still produces a spanner with all three desired properties. Relatives of Euclidean distances, such as the function mentioned above, may be used to produce *energy spanners*.

3. Let $G = (V, E)$ be an edge-weighted graph. The *power cost* of a vertex $u \in V$ is $power(u) = \max\{w(u, v) \mid v \text{ is a neighbor of } u\}$. In other words, the power cost of a vertex u is proportional to the cost of u transmitting to a farthest neighbor. The *power cost* of G is $\sum_{u \in V} power(u)$ [30]. We can show that the output of our algorithm is not only lightweight with respect to the usual weight measure (sum of the weights of all edges) but also with respect to the power cost measure.

2.2 Sequential Relaxed Greedy Algorithm

Now we show that a *relaxed version* of `SEQ-GREEDY` produces an output G' with all three desired properties, even when the input is not a complete Euclidean graph, but is a d-dimensional, α-UBG for fixed d and α. Relaxing the requirement in `SEQ-GREEDY` that the edges be totally ordered by length and allowing for the output to be updated lazily are critical to obtaining a distributed algorithm that runs in polylogarithmic number of rounds.

Let $r > 1$ be a constant to be fixed later and let $W_i = r^i \alpha / n$ for each $i =$

$0, 1, 2, \ldots$. Let $I_0 = (0, \alpha/n]$ and for each $i = 1, 2, \ldots$ let $I_i = (W_{i-1}, W_i]$. Let $m = \lceil \log_r \frac{n}{\alpha} \rceil$. Then, since no edge has length greater than 1, the length of any edge in E lies in one of the intervals $I_0, I_1, \ldots, I_m$. Let $E_i = \{\{u, v\} \in E : |uv| \in I_i\}$.

We now eliminate the restriction that edges within a set E_i be processed in increasing order by length. We run `SEQ-GREEDY` in $m + 1$ phases: in phase i, the algorithm processes edges in E_i in arbitrary order and adds a subset of edges in E_i to the spanner. For $0 \leq i \leq m$, we use G_i to denote the spanning subgraph of G consisting of edges $E_0 \cup E_1 \cup \cdots \cup E_i$. Thus G_i is the portion of the input graph that the algorithm has processed in phase i and earlier. We use G'_i to denote the output of the algorithm at the end of phase i. In other words, G'_i is the spanning subgraph of G consisting of edges of G that the algorithm has decided to retain in phases $0, 1, \ldots, i$. The final output of the algorithm is $G' = G'_m$.

The way E_0 is processed is different from the way E_i, $i > 0$ is processed. We now separately describe these two parts.

2.2.1 Processing Edges in E_0

We start by stating a property of G_0 that follows easily from the fact that all edges in G_0 are small.

Lemma 1. *Every connected component of G_0 induces a clique in G.*

The algorithm `PROCESS-SHORT-EDGES` for processing edges in E_0 consists of three steps (i) determine the connected components of G_0, (ii) use `SEQ-GREEDY` to compute a t-spanner for each connected component (that is, a clique), and (iii) let G'_0

be the union of the t-spanners computed in Step (ii) and output G'_0. The following theorem states the correctness of the `PROCESS-SHORT-EDGES` algorithm. Its proof follows easily from the correctness of `SEQ-GREEDY`.

Theorem 2. *G'_0 satisfies the following properties. (i) For every edge $\{u, v\} \in E_0$, G'_0 contains a uv-path of length at most $t \cdot |uv|$, (ii) $\Delta(G'_0) = O(1)$, and (iii) $w(G'_0) = O(w(MST(G)))$.*

2.2.2 Processing Long Edges

We now describe how edges in E_i are processed, for $i > 0$. The algorithm `PROCESS-LONG-EDGES` has five steps: (i) computing a cluster cover for G'_{i-1}, (ii) selecting query edges in E_i, (iii) computing a cluster graph H_{i-1} for G'_{i-1}, (iv) answering shortest path queries for the query edges selected in Step (ii), and (v) adding edges to G'_{i-1} to obtain G'_i and then removing redundant edges from G'_i. These steps are described in the next five subsections.

For any graph J, let $V(J)$ denote the vertex set for J. For any pair of vertices $u, v \in V(J)$ let $\mathtt{sp}_J(u, v)$ denote the length of a shortest uv-path in J. Define a *cluster* of J with *center* $u \in V(J)$ and *radius* r to be a set of vertices $C_u \subseteq V(J)$ such that, for each $v \in C_u$, $\mathtt{sp}_J(u, v) \leq r$. A set of clusters $\{C_{u_1}, C_{u_2}, \ldots\}$ of J is a *cluster cover of J of radius r* if every cluster in the set has radius r, every vertex in $V(J)$ belongs to at least one cluster, and for any pair of cluster centers u_i and u_j, $\mathtt{sp}_J(u_i, u_j) > r$.

2.2.2.1 Computing a Cluster Cover for G'_{i-1}

At the beginning of phase i we compute a cluster cover of radius δW_{i-1}, where $\delta < 1$ is a constant that will be fixed later. We start with an arbitrary vertex $u \in V$ and run Dijkstra's shortest path algorithm with source u on G'_{i-1}, in order to identify nodes $v \in V$ with the property that $\mathtt{sp}_{G'_{i-1}}(u, v) \leq \delta W_{i-1}$; each such node v gets included in the cluster C_u. Once C_u has been identified, recurse on $V \setminus C_u$ until all nodes belong to some cluster and we have a cluster cover of G'_{i-1} of radius δW_{i-1}.

2.2.2.2 Selecting Query Edges in E_i

As defined earlier, edges in E_i have weights in the interval $I_i = (W_{i-1}, W_i]$, while the cluster cover for G'_{i-1} has radius δW_{i-1}, with $\delta < 1$. This implies that each edge in E_i has endpoints in different clusters. Our goal is to select a unique query edge per pair of clusters. This will guarantee that there are a constant number of query edges incident on any node (see Lemma 4) and this fact will be critically used by the distributed version of our algorithm to guarantee the degree bound on the spanner that is constructed.

Let θ be a quantity that satisfies $0 < \theta < \frac{\pi}{4}$ and $t \geq 1/(\cos\theta - \sin\theta)$. Note that for any value $t > 1$, no matter how small, there always exists a θ that satisfies these restrictions and as $t \to 1$, we have that $\theta \to 0$. Define an edge $e = \{u, v\} \in E_i$ to be a *covered edge* if there is a $z \in V$ such that (i) $\{u, z\} \in G'_{i-1}$, $|vz| \leq \alpha$ and $\angle vuz \leq \theta$ or (ii) $\{v, z\} \in G'_{i-1}$, $|uz| \leq \alpha$ and $\angle uvz \leq \theta$. Any edge in E_i that is not covered is a *candidate* query edge. The motivation for these definitions is the

following geometric lemma, due to Czumaj and Zhao [13].

Lemma 3 (Czumaj and Zhao [13])**.** *Let* $0 < \theta < \frac{\pi}{4}$ *and* $t \geq \frac{1}{\cos\theta - \sin\theta}$*. Let* u, v, z *be three points in* $\mathcal{R}^d$ *with* $\angle vuz \leq \theta$*. Suppose further that* $|uz| \leq |uv|$*. Then the edge* $\{u, z\}$ *followed by a* t*-spanner path from* z *to* v *is a* t*-spanner path from* u *to* v *(see Figure 2.1).*

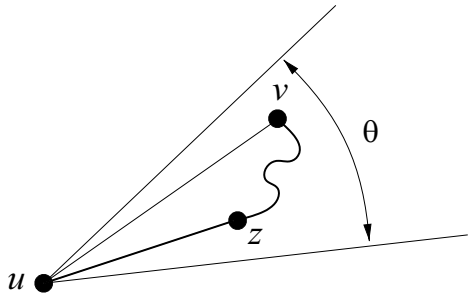


Figure 2.1: u, v, z are three points in $\mathcal{R}^d$ with $\angle vuz \leq \theta$(a) Edge $\{u, v\}$ is *covered*: $\{u, z\}$ followed by a t-spanner zv-path is a t-spanner uv-path.

Now note that for each covered edge $\{u, v\} \in E_i$, there exists z that satisfies the preconditions of Lemma 3 (by definition), and using this lemma we can show that G'_{i-1} already contains a uv-path of length at most $t \cdot |uv|$. This suggests that covered edges need not be queried and therefore we can start with the complement of the set of covered edges as candidate query edges.

Now we show that the set of candidate query edges can be further pared down. For each pair of clusters C_a and C_b, let $E_i[C_a, C_b]$ denote the subset of candidate query edges in E_i with one endpoint in C_a and the other endpoint in C_b. Our algorithm

selects a unique query edge $\{x, y\}$ from each nonempty subset $E_i[C_a, C_b]$. Assuming that $x \in C_a$ and $y \in C_b$, the edge $\{x, y\}$ is selected so as to minimize

$$t \cdot |xy| - \mathtt{sp}_{G'_{i-1}}(a, x) - \mathtt{sp}_{G'_{i-1}}(b, y) \tag{2.1}$$

The quantity in (2.1) is carefully chosen to guarantee that, if a t-spanner path between the endpoints of an edge $\{x, y\}$ that minimizes (2.1) exists in G'_i, then t-spanner paths between the endpoints of *all* edges in $E_i[C_a, C_b]$ exist in G'_i (this property will later be shown in the proof of Theorem 10). This implies that, for each pair of clusters C_a and C_b, it is sufficient to query just the edge $\{x, y\}$ in $E_i[C_a, C_b]$ that minimizes (2.1).

The following lemma shows that selecting query edges as described above filters all but a constant number of edges per cluster. The proof follows from two observations: (i) if a pair of cluster centers are connected by an edge in E_i, then the clusters are not too far from each other in Euclidean space (in particular, no farther than $(4\delta+r)W_{i-1}$), and (ii) the Euclidean distance between any pair of cluster centers is bounded from below by $\delta W_{i-1}/t$, because they would otherwise be part of the same cluster.

Lemma 4. *The number of query edges in E_i that are incident on any cluster is a constant ($O(t^d(\frac{4\delta+r}{\delta})^d)$, at most).*

2.2.2.3 Computing a Cluster Graph

For each selected query edge $\{x, y\} \in E_i$, we need to know if G'_{i-1} contains an xy-path of length at most $t \cdot |xy|$. In general, the number of hops in a shortest xy-path in G'_{i-1} can be quite large and having to traverse such a path would mean that

the shortest path query corresponding to edge $\{x, y\}$ could not be answered quickly enough. To get around this problem, we use an idea from [17] in which the authors construct an approximation to G'_{i-1}, called a *cluster graph*, and show that for any edge $\{x, y\} \in E_i$, the shortest path query for $\{x, y\}$ can be answered approximately on H_{i-1} in a constant number of steps. The goal of Das and Narasimhan [17] was to improve the running time of `SEQ-GREEDY` on complete Euclidean graphs, but we show that the Das-Narasimhan data structure can be constructed and maintained in a distributed fashion for efficiently answering shortest path queries for edges belonging to a α-UBG. In the following, we describe a sequential algorithm that starts with a cluster cover of G'_{i-1} of radius δW_{i-1}, and builds a *cluster graph* H_{i-1} of G'_{i-1}. This algorithm is identical to the one in Das and Narasimhan [17] and is included mainly for completeness.

The vertex set of H_{i-1} is V and the edge set of H_{i-1} contains two types of edges: *intra-cluster* edges and *inter-cluster* edges. An edge $\{a, x\}$ is an intra-cluster edge if a is a cluster center and x is node in C_a. Inter-cluster edges are between cluster centers. An edge $\{a, b\}$ is an inter-cluster edge if a and b are cluster centers, and at least one of the following two conditions holds: (i) $\mathtt{sp}_{G'_{i-1}}(a, b) \leq W_{i-1}$, or (ii) there is an edge in G'_{i-1} with one endpoint in C_a and the other endpoint in C_b. See Figure 2.2.

Regardless of the type of a cluster edge $e = \{a, b\}$ (inter- or intra-), the weight of e is the value of $sp_{G'_{i-1}}(a, b)$. The following lemma follows easily from the definition of inter-cluster edges.

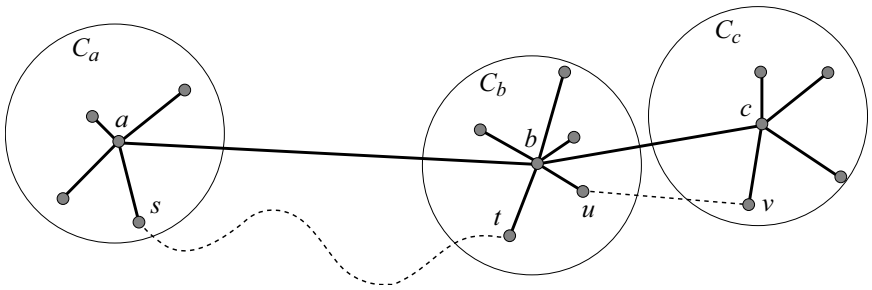


Figure 2.2: Edges interior to disks are *intra-cluster* edges. Edge $\{a, b\}$ is an *inter-cluster* edge because $\mathtt{sp}_{G'_{i-1}}(a, b) \leq W_{i-1}$, and $\{b, c\}$ is an *inter-cluster* edge because $\{u, v\}$ is in G'_{i-1}. An st-path in G'_{i-1}, shown by the dashed curve may be approximated by the path s, a, b, t in H_{i-1}.

Lemma 5. *For any inter-cluster edge $\{a, b\}$ in H_{i-1}, we have that $\mathtt{sp}_{G'_{i-1}}(a, b) \leq (2\delta + 1)W_{i-1}$.*

The above upper bound also implies that $|ab| \leq (2\delta + 1)W_{i-1}$. Using this and arguments similar to those used for Lemma 4, we can show that the number of inter-cluster edges incident to a cluster center is $O((5 + 1/\delta)^d)$, so we have the following lemma.

Lemma 6. *The number of inter-cluster edges in H_{i-1} incident to a cluster center is $O((5 + 1/\delta)^d)$, a constant.*

The main reason for constructing the cluster graph H_{i-1} is that lengths of paths in H_{i-1} are close to lengths of corresponding paths in G'_{i-1} and shortest path queries for edges in E_i can be answered quickly in H_{i-1}. The following lemma (whose proof appears in Das and Narasimhan [17]) shows that we can construct H_{i-1} such that path lengths in H_{i-1} approximate path lengths in G'_{i-1} to any desired extent, depending on the choice of δ.

Lemma 7. *For any edge $\{x, y\} \in E_i$, if there is a path between x and y in G'_{i-1} of length L_1, then there is a path between x and y in H_{i-1} of length L_2 such that $L_1 \leq L_2 \leq \frac{1+6\delta}{1-2\delta} L_1$.*

2.2.2.4 Answering Shortest Path Queries

For query edges $\{x, y\} \in E_i$, we are interested in knowing whether G'_{i-1} has an xy-path of length at most $t \cdot |xy|$. We ask this question on the cluster graph H_{i-1}. If H_{i-1} contains an xy-path of length at most $t \cdot |xy|$, we do not add $\{x, y\}$ to G'_i; otherwise we do. If H_{i-1} contains an xy-path of length at most $t \cdot |xy|$, then so does G'_{i-1} (by Lemma 7, since $L_1 \leq L_2$). Therefore, not adding $\{x, y\}$ to the spanner is not a dangerous choice. On the other hand, even if H_{i-1} does not contain an xy-path of length at most $t \cdot |xy|$, G'_{i-1} might contain such a path and in this case adding edge $\{x, y\}$ is unnecessary. Adding extra edges is of course not problematic for the t-spanner property. It will turn out that this is not a problem even for the requirement that the spanner should have bounded degree and small weight, given that paths in H_{i-1} can approximate paths in G'_{i-1} to an arbitrary degree.

Given the structure of the cluster graph, all but at most 2 edges in any simple xy-path are inter-cluster edges. Since the radius of each cluster is δW_{i-1}, each inter-cluster edge has weight greater than δW_{i-1}. We are looking for a path of length at most $t \cdot |xy|$. Since $|xy| \in (W_{i-1}, W_i]$, we are looking for a path of length at most $t \cdot W_i = t \cdot r \cdot W_{i-1}$. Any simple path in H_{i-1} of length at most $t \cdot r \cdot W_{i-1}$ has at most $2 + \lceil tr/\delta \rceil$ hops, which is a constant. This yields the following lemma.

Lemma 8. *For any edge $\{x, y\} \in E_i$, if $\mathtt{sp}_{H_{i-1}}(x, y) \leq t \cdot |xy|$, then H_{i-1} contains a shortest xy-path with $O(1)$ hops (no more than $2 + \lceil tr/\delta \rceil$).*

One issue we need to deal with, especially when attempting to construct and answer queries in H_{i-1} in a distributed setting, is that edges in H_{i-1} need not be present in the underlying network G. Specifically, for an intra-cluster edge $\{u, a\}$, where C_a is a cluster and $u \in C_a$, it may be the case that $|ua| > \alpha$ and $\{u, a\}$ may be absent from G. Similarly, an inter-cluster edge $\{a, b\}$ in H_{i-1} may be absent in G. However, for any edge $\{x, y\}$ in H_{i-1} (intra- or inter-cluster edge), we have the bound $\mathtt{sp}_{G'_{i-1}}(x, y) \leq (2\delta + 1)W_{i-1}$. This follows from Lemma 5 and the fact that the radius of each cluster is δW_{i-1}. Thus a shortest xy-path in G'_{i-1} lies entirely in a ball of radius $(2\delta + 1)W_{i-1}$ centered at x. Since G'_{i-1} is a spanning subgraph of G, this implies that there is a shortest xy-path P in G that lies entirely in the d-dimensional ball of radius $(2\delta + 1)W_{i-1}$ centered at x. Since any two vertices in P that are two hops away from each other are at least α apart (in the d-dimensional Euclidean space), P contains at most $\lceil 2(2\delta+1)W_{i-1}/\alpha \rceil < \lceil 2(2\delta+1)/\alpha \rceil$ hops. This argument yields the following theorem.

Theorem 9. *For any edge $\{x, y\} \in E_i$, if $\mathtt{sp}_{H_{i-1}}(x, y) \leq t \cdot |xy|$, then G contains a shortest xy-path with $O(1)$ hops (no more than $\lceil 2(2\delta + 1)/\alpha \rceil$).*

This theorem implies that brute force search initiated from one of the endpoints, say x, will be able to answer the shortest path query on edge $\{x, y\}$ in $O(1)$ rounds in a distributed setting.

2.2.2.5 Removing Redundant Edges

Recall that shortest path queries for edges in E_i are answered on H_{i-1}, and so updates to G'_i in phase i do not influence subsequent shortest path queries in phase i. Thus it is possible that in phase i two edges $\{u, v\}$ and $\{u', v'\}$ get added to G'_{i-1}, yet both of the following hold:

(i) $\mathtt{sp}_{H_{i-1}}(v, u') + |u'v'| + \mathtt{sp}_{H_{i-1}}(v', u) \leq t \cdot |uv|$

(ii) $\mathtt{sp}_{H_{i-1}}(v', u) + |uv| + \mathtt{sp}_{H_{i-1}}(v, u') \leq t \cdot |u'v'|$

Note that, since $\mathtt{sp}_{G'_{i-1}}(x, y) \leq \mathtt{sp}_{H_{i-1}}(x, y)$ holds for any pair of nodes x and y, conditions (i) and (ii) above imply that $G'_i - \{u, v\}$ contains a t-spanner path from u to v and $G'_i - \{u', v'\}$ contains a t-spanner path from u' to v'. We call two edges $\{u, v\}$ and $\{u', v'\}$ satisfying conditions (i) and (ii) above *mutually redundant*: one of them could potentially be eliminated from G_i, without compromising the t-spanner property of G_i. In fact, such mutually redundant pairs of edges need to be eliminated from G'_i because our proof that G' has small weight (Theorem 13) depends on the absence of such pairs of edges.

To do this, we build a graph J that has a node for each edge in a mutually redundant pair and an edge between every pair of nodes that correspond to a mutually redundant pair of edges in G'_i. We construct an MIS I of J and eliminate from G'_i all edges associated with nodes in J that do not appear in I.

2.2.3 The Three Desired Properties

Recall that $G' = G'_m$ is the spanner at the end of phase m. We now prove that G' satisfies the three properties that the output of `SEQ-GREEDY` was guaranteed to have. The proofs of these theorems form the technical core of this chapter and are presented next in this section.

Theorem 10. *For any $t > 1$, $0 < \delta \leq \frac{t-1}{4}$, the output G' is a t-spanner.*

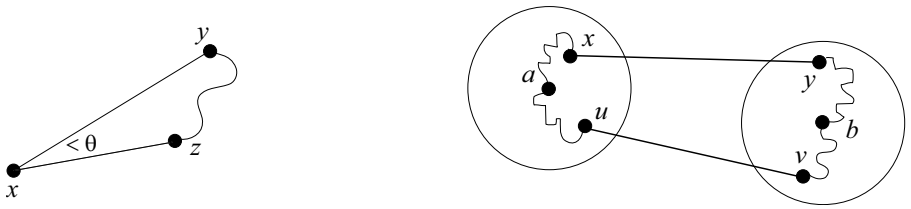
Proof. We first prove that the theorem holds for all query edges in E, then we extend the argument to non-query edges as well. Let $\{x, y\}$ be an arbitrary query edge and let $i \geq 1$ be such that $\{x, y\} \in E_i$. Then either (i) $\{x, y\}$ is added to the spanner in phase i, or (ii) $\mathtt{sp}_{H_{i-1}}(x, y) \leq t \cdot |xy|$. If the former is true and $\{x, y\}$ is not a redundant edge, then the theorem holds. If $\{x, y\}$ is a redundant edge but does not get removed from G'_i, then again the theorem holds. If $\{x, y\}$ is a redundant edge that gets removed from G_i, then at least one mutually redundant counterpart edge must remain in G'_i (since removed edges form a maximal independent set), ensuring a t-spanner xy-path in G_i. If (ii) is true, then from Lemma 7, $\mathtt{sp}_{G'_{i-1}}(x, y) \leq \mathtt{sp}_{H_{i-1}}(x, y)$ (first part of the inequality) and therefore $\mathtt{sp}_{G'_{i-1}}(x, y) \leq t \cdot |xy|$.

For non-query edges, the proof is by induction on the length of edges in G. The base case corresponds to edges in E_0, for which `SEQ-GREEDY` ensures that the theorem holds.

Assume that the theorem is true for any edge in E of length no greater than some value q, and consider a smallest non-query edge $\{x, y\}$ in G of length greater than q. We prove that $\mathtt{sp}_{G'}(x, y) \leq t \cdot |xy|$. Let i be such that $\{x, y\} \in E_i$. We now

consider two cases, depending on whether $\{x, y\}$ is a *candidate* query edge in phase i or not.

If $\{x, y\}$ is not a candidate query edge, then it is a covered edge. That is, there exists an edge $\{x, z\}$ in G'_{i-1} such that $|yz| \leq \alpha$ and $\angle yxz \leq \theta$, or an edge $\{y, z\}$ in G'_{i-1} such that $|xz| \leq \alpha$ and $\angle xyz \leq \theta$. The two cases are symmetric and so without loss of generality, assume that the former is true. Here θ satisfies the hypothesis of the Czumaj-Zhao lemma (Lemma 3), that is, $0 < \theta < \frac{\pi}{4}$ and $t \geq \frac{1}{\cos\theta - \sin\theta}$. Since $|yz| \leq \alpha$ and G is an α-UBG, this implies that $\{y, z\}$ is an edge is E. Furthermore, since $0 < \theta < \frac{\pi}{4}$, we have $|yz| < |xy|$. Refer to Figure 2.3(a). If $\{y, z\}$ is a query edge, then by the argument above we have that G' contains a t-spanner yz-path p. Otherwise, if $\{y, z\}$ is not a query edge, since its length is less than the length of $\{x, y\}$, by the inductive hypothesis we get that there is a t-spanner yz-path p. In either case, Lemma 3 tells us that $\{x, z\}$ followed by p is a t-spanner path from x to y, completing this case.



(a) (b)

Figure 2.3: If $\{y, z\}$ is a query edge, then by the argument above we have that G' contains a t-spanner yz-path p. Otherwise, if $\{y, z\}$ is not a query edge, since its length is less than the length of $\{x, y\}$, by the inductive hypothesis we get that there is a t-spanner yz-path p. (a) $\{x, y\}$ is a covered edge (b) $\{u, v\}$ is a query edge: if G_i contains a t-spanner uv-path, then G_i contains a t-spanner xy-path.

We now consider the case when $\{x, y\}$ is a candidate query edge in phase i, but not a query edge. Let a and b be such that $x \in C_a$ and $y \in C_b$, and let $\{u, v\}$ be the query edge selected in phase i, with $u \in C_a$ and $v \in C_b$. Refer to Figure 2.3b. Due to the criteria for selecting $\{u, v\}$, we have

$$\begin{aligned} t \cdot |uv| - \mathtt{sp}_{G'_{i-1}}(a, u) - \mathtt{sp}_{G'_{i-1}}(b, v) \leq \\ t \cdot |xy| - \mathtt{sp}_{G'_{i-1}}(a, x) - \mathtt{sp}_{G'_{i-1}}(b, y). \end{aligned} \tag{2.2}$$

Recall that G'_i is the partial spanner at the end of phase i. We show that $\mathtt{sp}_{G'_i}(x, y) \leq t \cdot |xy|$. We discuss two cases, depending on whether $\{u, v\}$ was added to G'_i or not.

Assume first that $\{u, v\}$ was not added to G'_i. This means that $\mathtt{sp}_{H_{i-1}}(u, v) \leq t \cdot |uv|$. Note however that

$$\begin{aligned} \mathtt{sp}_{H_{i-1}}(u, v) &= \mathtt{sp}_{G'_{i-1}}(u, a) + \mathtt{sp}_{H_{i-1}}(a, b) + \mathtt{sp}_{G'_{i-1}}(b, v) \\ &\leq t \cdot |uv|. \end{aligned} \tag{2.3}$$

We now evaluate

$$\begin{aligned} \mathtt{sp}_{G'_{i-1}}(x, y) &\leq \mathtt{sp}_{G'_{i-1}}(x, a) + \mathtt{sp}_{G'_{i-1}}(a, b) + \mathtt{sp}_{G'_{i-1}}(b, y) \\ &\leq \mathtt{sp}_{G'_{i-1}}(x, a) + \mathtt{sp}_{H_{i-1}}(a, b) + \mathtt{sp}_{G'_{i-1}}(b, y) \\ &\leq t \cdot |xy|. \end{aligned}$$

This latter inequality involves simple substitutions that use inequalities (2.2) and (2.3), and completes this case.

Now assume that $\{u, v\}$ was added to G'_i. Since $u \in C_a$ and C_a has radius δW_{i-1}, we have that $\mathtt{sp}_{G'_{i-1}}(a, u) \leq \delta W_{i-1}$. Similarly, $\mathtt{sp}_{G'_{i-1}}(b, v) \leq \delta W_{i-1}$. These together with (2.2) yield

$$t \cdot |uv| - 2\delta W_{i-1} \leq t \cdot |xy| - \mathtt{sp}_{G'_{i-1}}(a, x) - \mathtt{sp}_{G'_{i-1}}(b, y). \tag{2.4}$$

The existence of $\{u, v\}$ in G'_i enables us to construct in G'_i a path from a to b of weight

$$\begin{aligned} \mathtt{sp}_{G'_i}(a, b) &\leq \mathtt{sp}_{G'_i}(a, u) + |uv| + \mathtt{sp}_{G'_i}(v, b) \\ &\leq 2\delta W_{i-1} + |uv|, \end{aligned} \tag{2.5}$$

since $\mathtt{sp}_{G'_i}(a, u) \leq \mathtt{sp}_{G'_{i-1}}(a, u) \leq \delta W_{i-1}$, and same for $\mathtt{sp}_{G'_i}(v, b)$. We can now construct a path in G'_i from x to y of weight

$$\begin{aligned} \mathtt{sp}_{G'_i}(x, y) &\leq \mathtt{sp}_{G'_i}(a, x) + \mathtt{sp}_{G'_i}(b, y) + \mathtt{sp}_{G'_i}(a, b) \\ &\leq t \cdot |xy| + 2\delta W_{i-1} - t \cdot |uv| + \mathtt{sp}_{G'_i}(a, b) \\ &\leq t \cdot |xy| + 4\delta W_{i-1} - (t-1) \cdot |uv| \\ &< t \cdot |xy| + 4\delta W_{i-1} - (t-1) W_{i-1} \end{aligned}$$

In deriving this chain of inequalities, we have used (2.4), (2.5) and the fact that $|uv| > W_{i-1}$. Note that for any $\delta \leq \frac{t-1}{4}$, the quantity $4\delta W_{i-1} - (t-1) \cdot W_{i-1}$ above is negative, yielding $\mathtt{sp}_{G_i}(x, y) < t \cdot |xy|$. This completes the proof. □

Theorem 11. *G' has $O(1)$ degree.*

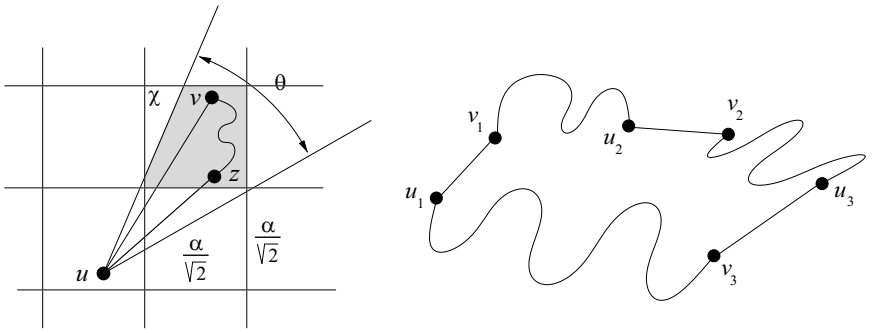


Figure 2.4: Leapfrog Property. (a) Region χ contains two neighbors v and z of u. (b) Definition of the t-leapfrog property with $S = \{\{u_1, v_1\}, \{u_2, v_2\}, \{u_3, v_3\}\}$.

Proof. Let θ be a quantity satisfying the conditions of Lemma 3. Fix a vertex u and consider the d-dimensional unit radius ball centered at u. For some T that depends only on θ and d, this ball can be partitioned into T cones, each with apex u, such that for any x, y in a cone, $\angle xuy \leq \theta$. Yao [83] shows how to construct such a partition with $T = O(d^{3/2} \cdot \sin^{-d}(\theta/2) \cdot \log(d \sin^{-1}(\theta/2)))$ cones. Place an infinite axis-parallel grid of d-dimensional cubes, each of dimension $\frac{\alpha}{\sqrt{d}} \times \frac{\alpha}{\sqrt{d}} \times \cdots \times \frac{\alpha}{\sqrt{d}}$, on the plane. See Figure 2.4(c) for a 2-dimensional version of this picture. There are $O(1/\alpha^d)$ cells that intersect the unit ball centered at u, and therefore there are $O(1/\alpha^d)$ cells that intersect each cone in the cone partition of this unit ball. Thus the cones and the square cells together partition the unit ball centered at u into $O(T/\alpha^d)$ regions. We show that in G', u has $O(\frac{t^d(4\delta+r)^d}{\delta^d})$ neighbors in each region, which is a constant.

Let $v_1, v_2, \ldots, v_k$ be neighbors of u in G' that lie in a region χ. Without loss of generality, assume that $|uv_1| \geq |uv_j|$, for $j = 2, \ldots, k$, and let i be such that $\{u, v_1\} \in E_i$. Since $|uv_j| \leq |uv_1|$, we have that for all $j = 2, \ldots, k$, $\{u, v_j\} \in E_\ell$, with $\ell \leq i$.

We now prove that $\{u, v_j\}$ is in fact in E_i for all j. To derive a contradiction, assume that there is a $j > 1$ such that $\{u, v_j\} \in E_\ell$, with $\ell < i$. This means that just before edge $\{u, v_1\}$ is processed, G' contains edge $\{u, v_j\}$. Also note that since v_1 and v_j lie in the same region, $|v_1 v_j| \leq \alpha$. But, this means that $\{u, v_1\}$ is a covered edge in phase i and will not be queried. This contradicts the presence of edge $\{u, v_1\}$ in G'.

We have shown that $\{u, v_j\} \in E_i$ for all j. Recall that our algorithm picks a unique query edge per pair of clusters. This along with Lemma 4 proves that k is constant. □

In the next theorem, we show that the spanner produced by the algorithm has small weight. The proof relies on the line segments in the spanner satisfying a property known as the *leapfrog property* [13, 25]. For any $t \geq t' > 1$, a set of line segments, denoted F, has the (t', t)-*leapfrog property* if for every subset $S = \{\{u_1, v_1\}, \{u_2, v_2\}, \ldots, \{u_s, v_s\}\}$ of F

$$t' \cdot |u_1 v_1| < \sum_{i=2}^{s} |u_i v_i| + t \cdot \Big(\sum_{i=1}^{s-1} |v_i u_{i+1}| + |v_s u_1| \Big). \tag{2.6}$$

Informally, this definition says that if there exists an edge between u_1 and v_1, then any path not including $\{u_1, v_1\}$ must have length greater than $t'|u_1 v_1|$ (see Figure 2.4(c) for an illustration of this definition). The following implication of the (t', t)-leapfrog property was shown by Das and Narasimhan [17].

Lemma 12. *Let $t \geq t' > 1$. If the line segments F in d-dimensional space satisfy the (t', t)-leapfrog property, then $wt(F) = O(wt(MST))$, where MST is a minimum spanning tree connecting the endpoints of line segments in F. The constant in the*

asymptotic notation depends on t, t' and d.

Theorem 13. *Let $0 < \delta < (t-1)/(6+2t)$. Let t_δ denote $t \cdot (1-2\delta)/(1+6\delta)$. Let $1 < r < (t_\delta + 1)/2$. When the relaxed greedy algorithm is run with these values of δ and r, the output G' satisfies $w(G') = O(wt(MST(G)))$.*

Proof. Let $\beta > 1$ be a constant picked as follows. When $t\alpha < 1$, pick β satisfying $1 < \beta < \min\{2, 1/(1-t\alpha)\}$. Otherwise, pick β satisfying $1 < \beta < 2$. Partition the edges of G' into subsets $F_0, F_1, \ldots$ such that $F_0 = \{\{u,v\} \in G' \mid |uv| \leq \alpha\}$ and for each $j > 0$, $F_j = \{\{u,v\} \in G' \mid \alpha\beta^{j-1} < |uv| \leq \alpha\beta^j\}$. Let $\ell = \lceil \log_\beta \frac{1}{\alpha} \rceil$. Then every edge in G' is in some subset F_j, $0 \leq j \leq \ell$. We will now show that each F_j satisfies the (t', t)-leapfrog property, for any t' satisfying:

$$1 \leq t' < \min\{\frac{t_\delta + 1}{r} - 1, \frac{2}{r}, \frac{t}{r}, \frac{2}{\beta}, t\alpha + \frac{1}{\beta}\}. \tag{2.7}$$

It is easy to check that our choice for δ, r, and β guarantee that each quantity inside the min operator is strictly greater than 1. Showing the (t', t)-leapfrog property for F_j would imply that $w(F_j) = O(w(MST(G)))$, and since the edges of G' are partitioned into a constant number of subsets F_j, $w(G') = O(w(MST(G)))$.

Consider an arbitrary subset $S = \{\{u_1, v_1\}, \{u_2, v_2\}, \ldots, \{u_s, v_s\}\} \subseteq F_0$. To prove inequality (2.6) for S, it suffices to consider the case when $\{u_1, v_1\}$ is a longest edge in S. We consider F_0 separately from F_j, $j > 0$.

The F_0 case. If for any $1 \leq k < s$, $|v_k u_{k+1}| > |u_1 v_1|$ or $|v_s u_1| > |u_1 v_1|$, then the leapfrog property holds. So we assume that for all $1 \leq k < s$, $|v_k u_{k+1}| \leq |u_1 v_1|$ and $|v_s u_1| \leq |u_1 v_1|$. Let i be the phase in which $\{u_1, v_1\}$ gets processed, i.e., $\{u_1, v_1\} \in E_i$.

Since $|u_1v_1| \leq \alpha$, it is the case that for all $1 \leq k < s$, $|v_k u_{k+1}| \leq \alpha$ and $|v_s u_1| \leq \alpha$. Hence, $\{\{v_s, u_1\}\} \cup \{\{v_k, u_{k+1}\} \mid 1 \leq k < s\}$ is a subset of edges of G and each edge in this set gets processed in phase i or earlier.

Assume first that at least one edge in the set $\{\{v_s, u_1\}\} \cup \{\{v_k, u_{k+1}\} \mid 1 \leq k < s\}$ gets processed in phase i. Then the right hand side of inequality (2.6) is at least tW_{i-1}, since edges in E_i have weights in the interval $I_i = (W_{i-1}, rW_{i-1}]$. Also since $t'|u_1v_1| \leq t'rW_{i-1}$, and since the inequality $t'rW_{i-1} < tW_{i-1}$ is guaranteed by the values of r and t' in (2.7), the leapfrog property holds for this case.

Assume now that all edges in $\{\{v_s, u_1\}\} \cup \{\{v_k, u_{k+1}\} \mid 1 \leq k < s\}$ have been processed in phase $i-1$ or earlier, meaning that t-spanner paths between their endpoints exist in G'_{i-1} at the time $\{u_1, v_1\}$ gets processed. For $1 \leq k < s$, let P_k be a shortest $v_k u_{k+1}$-path in G'_{i-1}, and let P_s be a shortest $v_s u_1$-path in G'_{i-1}. Let P be the following u_1v_1-path in G'_i: $P = P_1 \oplus \{u_2, v_2\} \oplus P_2 \oplus \{u_3, v_3\} \oplus \cdots \oplus P_s$. Here, we use $\oplus$ to denote concatenation. We distinguish three cases, depending on the size of the subset $S \cap E_i$.

(i) $|S \cap E_i| > 2$. Then, $w(P) \geq 2W_{i-1}$. We also have that $|u_1v_1| \leq rW_{i-1}$, since $\{u_1, v_1\} \in E_i$. It follows that $w(P) > t'|u_1, v_1|$ for any $t' < \frac{2}{r}$. Furthermore, $w(P)$ is no greater than the right hand side of the (t', t)-leapfrog inequality (2.6), so lemma holds for this case as well.

(ii) $|S \cap E_i| = 2$. In addition to $\{u_1, v_1\}$, assume that $\{u_k, v_k\} \in E_i$ for some k, $1 < k \leq s$. It the (t', t)-leapfrog inequality (2.6) holds, we are done and so let

us assume the opposite of that:

$$t' \cdot |u_1 v_1| \geq \sum_{i=2}^{s} |u_i v_i| + t \cdot \Big(\sum_{i=1}^{s-1} |v_i u_{i+1}| + |v_s u_1| \Big). \tag{2.8}$$

Since all edges $\{u_j, v_j\}$, $1 \leq j \leq s$, except for $\{u_1, v_1\}$ and $\{u_k, v_k\}$ are in G'_{i-1}, and since G'_{i-1} contains t-spanner $v_j u_{j+1}$-paths for all j, $1 \leq j < s$, and a t-spannner $v_s u_1$-path, the above inequality yields

$$t' \cdot |u_1 v_1| \geq \mathtt{sp}_{G'_{i-1}}(v_1, u_k) + |u_k v_k| + \mathtt{sp}_{G'_{i-1}}(v_k, u_1).$$

Multiplying both sides by $(1 + 6\delta)/(1 - 2\delta)$ and using $t' < t_\delta$ (which is implied by our choice of t') and Lemma 7, we get

$$t \cdot |u_1 v_1| \geq \mathtt{sp}_{H_{i-1}}(v_1, u_k) + |u_k v_k| + \mathtt{sp}_{H_{i-1}}(v_k, u_1). \tag{2.9}$$

Let $\Delta = \sum_{i=1}^{s-1} |v_i u_{i+1}| + |v_s u_1|$. We now observe that

$$t_\delta \cdot |u_k v_k| < \sum_{i=1}^{k-1} |u_i v_i| + \sum_{i=k+1}^{s} |u_i v_i| + t \cdot \Delta \tag{2.10}$$

implies the (t', t)-leapfrog property. To see this use the fact that both $\{u_1, v_1\}$ and $\{u_k, v_k\}$ belong to E_i and therefore $|u_1 v_1| < r \cdot |u_k v_k|$, which substituted in (2.10) yields:

$$t_\delta \cdot |u_k v_k| - (r-1) \cdot |u_k v_k| < \sum_{i=2}^{s} |u_i v_i| + t \cdot \Delta.$$

We get the lower bound $t' \cdot |u_1 v_1|$ on the left hand side of the above inequality by using $|u_k v_k| > |u_1 v_1|/r$ again and our choice of $t' < (t_\delta + 1)/r - 1$. This yields the (t', t)-leapfrog property. So we assume that inequality (2.10) does not hold,

that is,

$$t_\delta \cdot |u_k v_k| \geq \sum_{i=1}^{k-1} |u_i v_i| + \sum_{i=k+1}^{s} |u_i v_i| + t \cdot \Delta.$$

Since all edges $\{u_j, v_j\}$, $1 \leq j \leq s$, except for $\{u_1, v_1\}$ and $\{u_k, v_k\}$ are in G'_{i-1}, and since G'_{i-1} contains t-spanner $v_j u_{j+1}$-paths for all j, $1 \leq j < s$, and a t-spannner $v_s u_1$-path, the above inequality yields

$$t_\delta \cdot |u_k v_k| \geq \mathtt{sp}_{G'_{i-1}}(v_1, u_k) + |u_1 v_1| + \mathtt{sp}_{G'_{i-1}}(v_k, u_1).$$

Multiplying both sides by $(1 + 6\delta)/(1 - 2\delta)$ and using Lemma 7, we get

$$t \cdot |u_k v_k| \geq \mathtt{sp}_{H_{i-1}}(v_1, u_k) + |u_1 v_1| + \mathtt{sp}_{H_{i-1}}(v_k, u_1). \tag{2.11}$$

Inequalities (2.9) and (2.11) imply that edges $\{u_1, v_1\}$ and $\{u_2, v_2\}$ are mutually redundant and therefore cannot both exist in the spanner — a contradiction.

(iii) $|S \cap E_i| = 1$. This means that P exists in G'_{i-1} at the time $\{u_1, v_1\}$ is processed. Furthermore, $w(P) > t \cdot |u_1 v_1| > t' \cdot |u_1 v_1|$, otherwise $\{u_1, v_1\}$ would not have been added to the spanner, a contradiction.

The F_j case, $j > 0$. In this case, $|u_k v_k| > |u_1 v_1|/\beta$ for all $k = 2, 3, \ldots, s$. If $|S| \geq 3$, then the right hand side of the (t', t)-leapfrog inequality (2.6) is at least $2 \cdot |u_1 v_1|/\beta$ and therefore the (t', t)-leapfrog inequality goes through for any $1 < t' < 2/\beta$. Otherwise, if $|S| = 2$, then we need to show that $t' \cdot |u_1 v_1| < |u_2 v_2| + t \cdot (|u_1 v_2| + |u_2 v_1|)$. If each of $|u_1 v_2|$ and $|u_2 v_1|$ is at most α, then using the same argument as in the F_0-case with $|S \cap E_i| = 2$, we can show that $\{u_1, v_1\}$ and $\{u_2, v_2\}$ are mutually redundant and will not both exist in the spanner. Otherwise, if one of $|u_1 v_2|$ or $|u_2 v_1|$ is greater

than α, then the right hand side of the (t', t)-leapfrog inequality (2.6) is greater than $|u_1v_1|/\beta + t\alpha$. To ensure that the inequality goes through, we require that $t' \cdot |u_1v_1| \leq \frac{|u_1v_1|}{\beta} + t\alpha$. Since $|u_1v_1| \leq 1$, the above inequality is satisfied for any $1 < t' \leq t\alpha + \frac{1}{\beta}$, which holds true cf. (2.7). □

2.3 Distributed Relaxed Greedy Algorithm

We now describe a distributed version of the relaxed greedy algorithm from Section 2.2. Like the sequential relaxed greedy algorithm, this algorithm also runs in $O(\log n)$ phases — with edges in E_i being processed in phase i. We will show that edges in E_0 can be processed in $O(1)$ rounds. Recall that each subsequent phase consists of the following five steps: (i) computing a cluster cover of G'_{i-1}, (ii) selecting query edges in E_i, (iii) computing a cluster graph H_{i-1} of G'_{i-1}, (iv) answering shortest path queries for selected query edges, and (v) deleting some redundant edges. We will show that Steps (ii), (iii), and (iv) can be completed in $O(1)$ rounds and Steps (i) and (v) take $O(\log^* n)$ rounds. Step (i) and Step (v) will each involve computing an MIS in a certain derived graph and in both cases, we will show that the derived graph is a UBG that resides in a metric space of constant doubling dimension. Putting this all together, we will show that the algorithm runs in $O(\log n \cdot \log^* n)$ communication rounds.

2.3.1 Distributed Processing of Short Edges

Lemma 1 implies that vertices in the same component of $G_0 = G[E_0]$ induce a clique and therefore can communicate in one hop with each other. In the distributed

version of the algorithm, each vertex u obtains the topology of its closed neighborhood along with pairwise distances between neighbors in one hop. Using this information, u determines the connected component C of G_0 that it belongs to. Then u simply runs `SEQ-GREEDY` on C and computes a t-spanner of C. Finally, u identifies the edges of the t-spanner incident on itself and informs all its neighbors of this.

Theorem 14. *The edges in E_0 can be processed in $O(1)$ rounds of communication.*

2.3.2 Distributed Processing of Long Edges

In this section, we show how long edges, that is, edges in E_i, $i > 0$, can be processed in a distributed setting. The first step of this process is the computation of a cluster cover for the spanner G'_{i-1} updated at the end of the previous phase.

2.3.2.1 Distributed Cluster Cover for G'_{i-1}

Recall that in this step our goal is to compute a cluster cover $\{C_{u_1}, C_{u_2}, \ldots\}$ of G'_{i-1} of radius δW_{i-1}. To do this, each node u first identifies all nodes v in G satisfying $\mathtt{sp}_{G'_{i-1}}(u, v) \leq \delta W_{i-1}$. Using arguments similar to those in Section 2.2.2.4, we can show that any node v satisfying $\mathtt{sp}_{G'_{i-1}}(u, v) \leq \delta W_{i-1}$ must be at most $2\delta W_{i-1}/\alpha$ hops from u. So each node u constructs the subgraph of G'_{i-1} induced by nodes that are at most $2\delta W_{i-1}/\alpha$ hops away from it in G. Node u then runs a (sequential) single source shortest path algorithm with source u on the local view of G'_{i-1} it has obtained and identifies all nodes v satisfying $\mathtt{sp}_{G'_i}(u, v) \leq \delta W_{i-1}$.

At the end of the above process, every node u in the network is a cluster center. We now force some nodes to cease being cluster centers, so that all pairs of cluster

centers are far enough from each other. Let J be the graph with vertex set V and whose edges $\{x, y\}$ are such that $x \in C_y$ (and by symmetry, $y \in C_x$). If $\{x, y\}$ is an edge in J, it is the case that $\mathtt{sp}_{G'_{i-1}}(x, y) \leq \delta W_{i-1}$. Now assign to every pair of nodes $\{x, y\}$ in V a weight $w(x, y) = \mathtt{sp}_{G'_{i-1}}(x, y)$. The weights w form a metric simply because shortest path distances in any graph form a metric. Thus J is a graph whose nodes reside in a metric space and whose edges connect pairs of nodes separated by distance of at most δW_{i-1} (in the metric space). By scaling the quantity δW_{i-1} up to one, we see that J is a UBG in the underlying metric space defined by the weights w. Recall from [51] that the *doubling dimension* of a metric space is the smallest ρ such that every ball can be covered by at most 2^ρ balls of half the radius. To see that the metric space induced by the weights w has constant doubling dimension, start with a ball of radius R centered at an arbitrary vertex u. Every vertex v in this ball satisfies $\mathtt{sp}_J(u, v) \leq R$. Now cover the vertices in this ball using balls of radius $R/2$ as follows: repeatedly pick an uncovered vertex v in the radius-R ball and grow a radius $R/2$ ball centered at v. It is easy to see that the number of radius $R/2$ balls is bounded because any pair of centers of these balls are far apart.

Lemma 15. *J is a UBG that resides in a metric space of constant doubling dimension.*

Let I be an MIS of J constructed using the MIS algorithm in [51]. This algorithm runs in $O(\log^* n)$ communication rounds on a UBG that resides in a metric space of constant doubling dimension. Then each node in $V \setminus I$ has one or more neighbors in I. Each node $u \in I$ is declared a cluster center, and each node $v \in V \setminus I$

attaches itself to the neighbor in I with the highest identifier. This gives us the desired cluster cover of radius δW_{i-1}.

Theorem 16. *A cluster cover of G'_{i-1} of radius δW_{i-1} can be computed in $O(log^* n)$ rounds of communication.*

2.3.2.2 Distributed Query Edge Selection

Only nodes that are cluster heads need to participate in the process of selecting query edges. Each cluster head a seeks to gather information on all edges in E_i between the cluster C_a and any other cluster C_b. Using the argument in Section 2.2.2.4, we know that every node in C_a is at most $2\delta W_{i-1}/\alpha$ hops away from a in G. Therefore, if there is an edge $\{u, v\} \in E_i$, $u \in C_a$ and $v \in C_b$, then v is at most $1 + 2\delta W_{i-1}/\alpha$ hops away from a. So a gets information from nodes that are at most $1 + 2\delta W_{i-1}/\alpha$ hops away from it and it identifies all edges in $E_i[C_a, C_b]$. Recall that this is the set of edges in E_i which connect a node in C_a and a node in C_b. Node a then discards all covered edges from $E_i[C_a, C_b]$, leaving only candidate query edges in E_i between C_a and C_b. Finally, from among the candidate query edges, node a selects an edge $\{u, v\}$ that minimizes $t \cdot |uv| - \mathtt{sp}_{G'_{i-1}}(a, u) - \mathtt{sp}_{G'_{i-1}}(b, v)$.

Theorem 17. *Query edges from E_i can be selected in $O(1)$ rounds of communication.*

2.3.2.3 Distributed Construction of the Cluster Graph

As in the query edge selection step, only the cluster heads need to perform actions to compute the cluster graph. Any member u of a cluster C_a lies at most

$2\delta W_{i-1}/\alpha$ hops away from a in G. Thus a can identify intra-cluster edges incident on it by gathering information from at most $2\delta W_{i-1}/\alpha$ hops away. If C_b is a cluster with $\mathtt{sp}_{G'_{i-1}}(a,b) \leq W_{i-1}$, then node a can identify the inter-cluster edge $\{a,b\}$ by gathering information from at most $2W_{i-1}/\alpha$ hops away. If C_b is a cluster such that there is an edge $\{u,v\}$ in G'_{i-1} with $u \in C_a$ and $v \in C_b$, then node a can identify the inter-cluster edge $\{a,b\}$ by gathering information from at most $2(2\delta+1)W_{i-1}/\alpha$ hops away. Note that the information that a gathers contains a local view of G'_{i-1} along with all pairwise distances. Using this information, node a is able to run a single source shortest path algorithm with source a and determine the weights of all inter-cluster and intra-cluster edges incident on a.

Theorem 18. *Computing the cluster graph H_{i-1} of G'_{i-1} takes $O(1)$ communication rounds.*

2.3.2.4 Answering Shortest Path Queries

Each node u knows all the query edges incident on it. As proved in Section 2.2.2.4, node u only needs to gather information from nodes that are at most a constant number of hops away, to be able to determine locally, for all incident query edges $\{u,v\} \in E_i$, whether $\mathtt{sp}_{H_{i-1}}(u,v) \leq t \cdot |uv|$. Thus, after constant number of communication rounds, u knows the subset of incident query edges $\{u,v\}$ for which $\mathtt{sp}_{H_{i-1}}(u,v) > t \cdot |uv|$ and u identifies these as the incident edges to be added to G'_i.

Theorem 19. *Answering shortest path queries takes $O(1)$ communication rounds.*

2.3.2.5 Distributed Removal of Redundant Edges

Two edges $\{u, v\}$ and $\{u', v'\}$ in G'_i are mutually redundant if (i) $\mathtt{sp}_{H_{i-1}}(v, u') + |u'v'| + \mathtt{sp}_{H_{i-1}}(v', u) \leq t \cdot |uv|$ and (ii) $\mathtt{sp}_{H_{i-1}}(v', u) + |uv| + \mathtt{sp}_{H_{i-1}}(v, u') \leq t \cdot |u'v'|$. Each node u takes charge of all edges $\{u, v\}$ added to G_i in phase i and for which the identifier of u is higher than the identifier of v. For each such edge $\{u, v\}$ that u is in charge of, u determines all edges $\{u', v'\}$ such that $\{u, v\}$ and $\{u', v'\}$ form a mutually redundant pair. Note that the nodes u and v' are a constant number of hops away from each other in G, and similarly for nodes v and u'. Node u then contributes to the construction of the graph J by adding to $V(J)$ a vertex for each redundant edge u is in charge of, and to $E(J)$ an edge connecting nodes in $V(J)$ that correspond to mutually redundant edges in G_i. Using an argument similar to the one used in Lemma 15, we can show the following:

Lemma 20. *J is a UBG that resides in a metric space of constant doubling dimension.*

Let I be an MIS of J constructed using the MIS algorithm in [51] that takes $O(\log^* n)$ communication rounds on a UBG that resides in a metric space of constant doubling dimension. Each node u then removes from G_i all incident edges in $V(J) \setminus I$.

Theorem 21. *Removing redundant edges takes $O(\log^* n)$ communication rounds.*

2.4 Conclusion

The results presented in this chapter apply to α-UDGs embedded in constant-dimension Euclidean spaces, and do not directly generalize to doubling metric spaces.

However, the techniques presented in this paper use a key property (the leapfrog property) that does not seem to generalize to metrics of low doubling dimension. Hence we looked for new techniques, with the goal of constructing an $O(\log^* n)$-round distributed algorithm that, for low dimensional doubling metric spaces, produces a $(1+\varepsilon)$-spanner with constant maximum degree. Our findings are described in the next chapter.

CHAPTER 3
DISTRIBUTED SPANNER CONSTRUCTION IN DOUBLING METRIC SPACES

3.1 Introduction

Before we start, let us revisit the concept of doubling metric spaces. The *doubling dimension* of a metric space is the smallest ρ such that any ball in this metric space can be covered by 2^ρ balls of half the radius. It is easy to verify that the d-dimensional Euclidean space, equipped with any of the L_p norms, has doubling dimension $\Theta(d)$. If ρ is a fixed constant, then we call the metric a *doubling metric*. For convenience, let us call an UBG on a doubling metric a *doubling UBG*. In this chapter we present a distributed algorithm for constructing a low-weight $(1+\varepsilon)$-spanner of bounded degree for doubling UBGs.

Precisely stated, our result is this: for any fixed $\varepsilon > 0$, our algorithm runs in $O(\log^* n)$ communication rounds on an n-node UBG G that resides in a doubling metric space, to construct a $(1+\varepsilon)$-spanner H of G with maximum degree bounded above by a constant. This constant depends on ε and ρ, the doubling dimension of the metric space in which G resides. Recall that $\log^* n = \min\{t \mid \log^{(t)} n \leq 2\}$, where $\log^{(0)} n = n$ and $\log^{(i)} n = \log(\log^{(i-1)} n)$ for any positive integer i. In addition, we show that H is "lightweight," in the following sense. Let Δ denote the aspect ratio of G, that is, the ratio of the length of a longest edge in G to the length of a shortest edge in G. We show that the total weight of H is bounded above by $O(\log \Delta) \cdot wt(MST)$, where MST denotes a minimum spanning tree of G (Section 3.2). Thus we obtain

a spanner that provides an $O(\log \Delta)$-approximation to a spanner of G of minimum weight. We also show that H satisfies the so called *leapfrog property* [17] described in the previous chapter, which informally says that any uv-path in H (not including $\{u, v\}$) must have length greater than $\{u, v\}$ by a constant factor. An immediate implication of this property is that, for the special case of Euclidean metric spaces with fixed dimension, the weight of H is bounded above by $O(wt(MST))$ [16]. Thus, our current result subsumes the results in [15] that apply to Euclidean metric spaces, and extends these results to metric spaces with constant doubling dimension. Finally, we show that the result of this chapter extends to the more general qUBG network model.

3.1.1 Topology Control In Doubling Metric

Like the results in Chapter 2, the results in this chapter are also motivated by the *topology control* problem in wireless ad-hoc networks. Most topology control protocols that provide worst case guarantees on the quality of the topology assume that the network is modeled by a unit disk graph (UDG) (see [59] for a recent example). The results in this chapter apply to the more general model of doubling unit ball graphs (UBG). Doubling metric spaces have received a great deal of attention recently [6, 46, 47, 51, 77], partly because they are thought to capture real-world phenomena such as latencies in peer-to-peer networks and in the Internet. Also, doubling metrics are robust in the sense that the doubling dimension is roughly preserved under distortion (see Proposition 3 in [77]). Thus distorted versions of low dimen-

sional Euclidean space also have small doubling dimension. Consequently, doubling UBGs can model wireless networks in which nodes have non-uniform transmission ranges or have erroneous perception of distances to other nodes. Finally, doubling metrics imply the following "bounded growth" phenomenon that seems to be characteristic of large scale wireless ad-hoc and sensor networks: the number of nodes that are far away from each other and yet are all in the vicinity of a particular node, is small. In other words, no node can have an arbitrarily large independent set in its neighborhood.

3.1.2 Net Trees

Let (V, d) be a metric space with $|V| = n$ and doubling dimension ρ. In a recent paper, Chan, Gupta, Maggs, and Zhou [7] show how to construct, via a sequential, polynomial-time algorithm, a $(1 + \varepsilon)$-spanner of (V, d) with maximum degree bounded above by $\left(\frac{1}{\varepsilon}\right)^{O(\rho)}$. We will refer to this algorithm as the CGMZ algorithm. The problem of constructing a spanner for a metric space can be thought of as a special case of our problem, in which the given UBG is a complete graph. Underlying the result in [7] is the notion of *net trees*, independently proposed by Har-Peled and Mendel [31]. Let $B(u, r)$ denote the ball of radius r centered at point u. A subset $U \subseteq V$ is an *r-net* of V if it satisfies two properties:

r-packing: For every u and v in U, $d(u, v) > r$.

r-covering: The union $\cup_{u \in U} B(u, r)$ covers V.

Such nets always exist for any $r > 0$, and can be easily computed using a greedy algorithm. Assume without loss of generality that the largest pairwise distance in V

is exactly 1 (this can be achieved by appropriate scaling). Pick constant α such that

$$\sqrt[3]{1+\varepsilon} \leq \alpha < \sqrt{1+\varepsilon} \tag{3.1}$$

These constraints on α are necessary to ensure that our spanner satisfies various properties and will become clear later. Let $\gamma = \frac{2\alpha}{\alpha-1}\left(1+\frac{4\alpha}{\varepsilon}\right)$. Let h be the smallest positive integer such every pairwise distance is greater than $\frac{1}{\alpha^h}$. Let $r_0 = \frac{1}{\alpha^h}$ and let $r_i = \alpha \cdot r_{i-1}$, for $i > 0$. A *net tree* is a sequence of subsets $\langle V_0, V_1, V_2, \ldots, V_h\rangle$, such that $V_0 = V$ and V_i is an r_i-net of V_{i-1}, for $i > 0$. Note that every V_i, including V_0, is an r_i-packing. Also note that V_h, which is a 1-net of V_{h-1}, is a singleton, since the maximum separation between any pair of points is 1. To view the sequence $\langle V_0, V_1, V_2, \ldots, V_h\rangle$ as a tree, let $i(v) = \max\{i \mid v \in V_i\}$ for each $v \in V$. Then, for each $v \in V$, $i(v)+1$ copies of v appear as nodes in the tree. These are denoted $(0,v),(1,v),\ldots,(i(v),v)$, where (i,v) represents the occurrence of v in V_i. For each $0 \leq i < i(v)$, the parent of node (i,v) is $(i+1,v)$. Node $(i(v),v)$ has no parent and is the root of the net tree, if $i(v) = h$; otherwise, vertex $v \notin V_{i(v)+1}$ and there is some vertex $u \in V_{i(v)+1}$ such that $B(u, r_{i(v)+1})$ contains v. Arbitrarily pick one such u and let $(i(v)+1,u)$ be the parent of $(i(v),v)$. Informally speaking, higher levels in the net tree (leaves are at level 0) represent the structure of V at lower resolution. Figure 3.1 shows an example of a net tree with 6 levels. Below we present the CGMZ algorithm [7]. For any two points $u, v \in V$, we use $d(u,v)$ to denote the distance between u and v in the underlying metric space.

The CGMZ Algorithm.

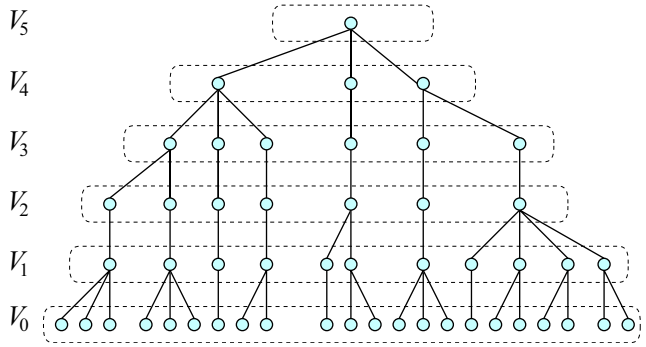


Figure 3.1: A net tree with six levels.

1. Build a net tree $\langle V_0, V_1, \ldots, V_h \rangle$ of V.

2. Let $\gamma = \frac{2\alpha}{\alpha - 1}\left(1 + \frac{4\alpha}{\varepsilon}\right)$. Construct the edge sets

$$E_0 = \{\{u, v\} \in V_0 \times V_0 \mid \ d(u, v) \leq \gamma \cdot r_0\},$$

   and

$$E_i = \{\{u, v\} \in V_i \times V_i \mid \ \gamma \cdot r_{i-1} < d(u, v) \leq \gamma \cdot r_i\},$$

   for each $i = 1, \ldots, h$ and let $\widehat{E} = \cup_i E_i$.

3. Replace some edges in $\widehat{E}$ by other edges to obtain a new edge set $\widetilde{E}$.

Chan and coauthors [7] work with the version of the algorithm for $\alpha = 2$. They show that the graph $H = (V, \widehat{E})$ obtained after Step (2) is a $(1 + \varepsilon)$-spanner of the metric space and has linear number of edges, but may not satisfy the bounded degree requirement. Short paths in H can be obtained from the net tree in a natural manner. A uv-path in H whose length is at most $(1 + \varepsilon) \cdot d(u, v)$ can be obtained by traveling up the net tree from the leaf u and from the leaf v until some level i is reached, such that the ancestors of u and v at level i are connected by an edge in

H. In Step (3), a subset of the edges in $\widehat{E}$ is considered and each edge in this subset is replaced by at most one new edge. This step, which will be described in detail in Section 3.2.2, redistributes the edges so that all vertex-degrees are bounded above by a constant. The techniques used by Chan and coauthors for bounding vertex degrees play a critical role in our results as well. In [15] (refer to Chapter 2) we also describe an algorithm for constructing a bounded-degree $(1+\varepsilon)$-spanner for Euclidean UBGs, but our results rely on purely geometric arguments to bound the vertex degree of the constructed spanner. Chan and coauthors [7] obtain the following theorem.

Theorem 22. [Chan, Gupta, Maggs, Zhou] *Let (V, d) be a finite metric with doubling dimension bounded by ρ. For any $\varepsilon > 0$, there is a $(1+\varepsilon)$-spanner for (V, d), with maximum degree bounded above by $\left(\frac{1}{\varepsilon}\right)^{O(\rho)}$.*

Our algorithm is a modification of the CGMZ algorithm [7] that takes into account the fact that pairs of points separated by a distance greater than 1 are not connected by an edge and therefore such edges cannot be used in the spanner. A high-level view of our algorithm is as follows. Using a slightly modified version of the CGMZ algorithm, we construct a graph H that may contain some *virtual edges*, that is, edges of length more than 1. H has all the desired properties with respect to the input UBG G. Subsequently, we show how to replace each virtual edge in H by at most one *real edge*, that is, an edge of length at most 1. The resulting graph is a $(1+\varepsilon)$-spanner of G with degree bounded above by a constant.

To obtain a distributed implementation of the above idea in $O(\log^* n)$ rounds, we use an algorithm due to Kuhn, Moscibroda, and Wattenhofer [51]. For a given

n-node UBG G in a doubling metric space, the algorithm in [51] deterministically computes a $(1, O(1))$-network decomposition, that is, a partition of G into clusters such that each cluster has diameter 1 and the resulting cluster graph has chromatic number $O(1)$. We use the same algorithm to compute a net tree. After computing the net tree, we require a constant number of additional rounds to construct the spanner.

3.2 Spanners for Doubling UBGs

Let (V, d) be a metric space with doubling dimension ρ. Let $G = (V, E)$ be the UBG induced by this metric space. Thus, for all $u, v \in V$, $u \neq v$, $\{u, v\} \in E$ if and only if $d(u, v) \leq 1$. For a fixed $\varepsilon > 0$, let the quantities h, r_i, α and γ be defined as in Section 3.1.2. Run Steps (1) and (2) of the CGMZ Algorithm to construct a set of edges $\widehat{E}$. Let $H = (V, \widehat{E})$. Note that V_h may not be a singleton since V may contain points whose pairwise distance is more than 1. So the sequence $\langle V_0, V_1, \ldots, V_h \rangle$ should be viewed as a forest of net trees, rooted at points in V_h. Recall that $\widehat{E} = \cup_{i=0}^{h} E_i$ and further recall that for $i > 0$, E_i consists of edges connecting all pairs of points $u, v \in V$ such that $d(u, v) \in (\gamma \cdot r_{i-1}, \gamma \cdot r_i]$. Note that there are values of i for which the right endpoint of the interval $(\gamma \cdot r_{i-1}, \gamma \cdot r_i]$ may be greater than 1 and for such values of i, E_i may contain edges that are not in E. Thus H is not necessarily a subgraph of G. Let $\delta = \lceil \log_\alpha \gamma \rceil$. It is easy to verify that for $0 \leq i \leq h - \delta$, $E_i \subseteq E$; for $i = h - \delta + 1$, the edge-set E_i may contain some edges in E and some edges not in E; and for $i > h - \delta + 1$, all edges in E_i are outside E. We call edges in H that also belong to E, *real edges*. Any edge in H that is not real is a *virtual edge*. Clearly,

a spanner for G may not contain virtual edges, however virtual edges in H do carry important proximity information that will provide clues on how to replace them with real edges.

3.2.1 Properties of H

We will now prove some important properties of H. Let d_H be the distance metric induced by shortest paths in H. Specifically, we will show that H satisfies the following three properties:

1. **Spanner Property.** For every $\{u, v\} \in E$, $d_H(u, v) \leq (1+\varepsilon) \cdot d(u, v)$ (Lemma 27).
2. **Degree Property.** Edges of H can be oriented in such a way that the out-degree of H is bounded by $\left(\frac{1}{\varepsilon}\right)^{O(\rho)}$ (Lemma 28).
3. **Weight Property.** The weight of H is $wt(H) = O(\log \Delta) \cdot \left(\frac{1}{\varepsilon}\right)^{O(\rho)} \cdot wt(MST)$ (Lemma 29).

Property 1 implies that H is connected, since G is assumed to be connected. Property 2 implies that H has a linear number of edges, though it does not imply that H has bounded maximum degree. In Section 3.2.2 we describe a method to alter H so as to bound the in-degree of H as well, while maintaining all the properties listed above. The proofs of these properties are based on some intermediate results, that we now establish. Proofs of Lemma 27 and Lemma 28 are similar to those in [8]. The next observation follows immediately from the definition of the doubling dimension of a metric space.

Proposition 23. *If (X, d) is a metric with doubling dimension ρ and $Y \subseteq X$ is a*

subset of points with aspect ratio Δ*, then* $|Y| \leq 2^{\rho \cdot \lceil \log_2 \Delta \rceil}$.

For any point $u \in V_i$, let $N_i(u) = \{v \in V_i \mid \{u, v\} \in E_i\}$ denote the set of points connected to u by edges in E_i. We now show an upper bound on the size of $N_i(u)$.

Lemma 24. *For each* $u \in V_i$, $|N_i(u)| \leq \left(\frac{1}{\varepsilon}\right)^{O(\rho)}$.

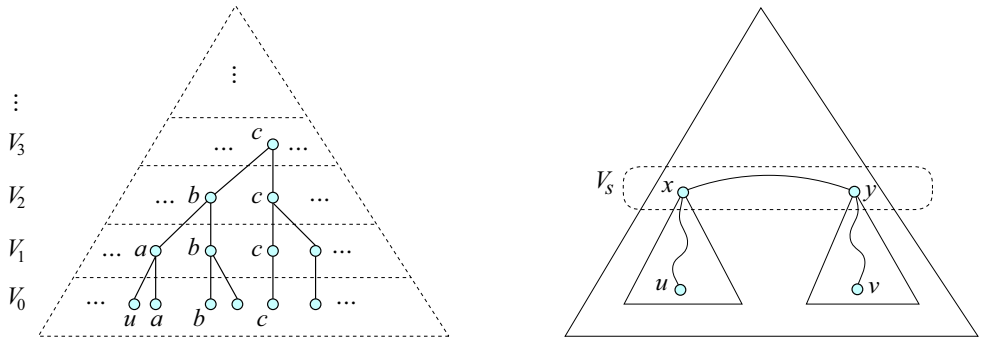
Proof. That the aspect ratio of $N_i(u)$ is bounded by 2γ follows from two observations: (1) any two points in $N_i(u)$ are more than distance r_i apart, and (2) any point in $N_i(u)$ is at distance at most $\gamma \cdot r_i$ from u and therefore, by using the triangle inequality, any two points in $N_i(u)$ are at most $2\gamma \cdot r_i$ apart. Then Proposition 23 implies the lemma. □

Lemma 25. *Suppose* $u, v \in V_i$ *and* $d(u, v) \leq \gamma \cdot r_i$*. Then* $\{u, v\} \in E_j \subset \widehat{E}$*, for some* $j \leq i$.

Proof. If $i > 0$ and $\gamma \cdot r_{i-1} < d(u, v) \leq \gamma \cdot r_i$, then by definition of E_i, $\{u, v\} \in E_i$. Otherwise, (a) $d(u, v) \leq \gamma \cdot r_0$ or (b) for some $j < i$, $\gamma \cdot r_{j-1} < d(u, v) \leq \gamma \cdot r_j$. Since $V_i \subseteq V_j$ for all $0 \leq j \leq i$, in case (a), $\{u, v\} \in E_0$ and in case (b), $\{u, v\} \in E_j$. □

Lemma 26. *For each* $u \in V$ *and for each* i*, there exists* $v \in V_i$ *such that* $d_H(u, v) \leq \frac{\alpha}{\alpha - 1} \cdot r_i$.

Proof. The proof is by induction on i. For $i = 0$, $u \in V_0 = V$ and $d_H(u, u) = 0 < \frac{\alpha}{\alpha-1} \cdot r_0$, proving this case true. For $i > 0$, apply the inductive hypothesis to infer that there exists $w \in V_{i-1}$ such that $d_H(u, w) \leq \frac{\alpha}{\alpha-1} \cdot r_{i-1}$. Furthermore, since V_i is an



(a) (b)

Figure 3.2: Illustrated proofs of Lemmas. (a) Proof of Lemma 26: in V_0, $d_H(u,u) = 0 < \frac{\alpha}{\alpha-1} \cdot r_0$; in V_1, $d_H(u,a) \leq r_1 < \frac{\alpha}{\alpha-1} \cdot r_1$; in V_2, $d_H(u,b) \leq d_H(u,a) + d_H(a,b) \leq \frac{\alpha}{\alpha-1} \cdot r_2$; and in V_3, $d_H(u,c) \leq d_H(u,b) + d_H(b,c) \leq \frac{\alpha}{\alpha-1} \cdot r_3$ (b) Proof of Lemma 27: The uv-path via x and y.

r_i-net of V_{i-1}, there exists $v \in V_i \subseteq V_{i-1}$ such that $d(w,v) \leq r_i \leq \gamma \cdot r_{i-1}$. This along with Lemma 25 shows that $\{w,v\} \in \widehat{E}$ and therefore $d_H(w,v) = d(w,v) \leq r_i$. By the triangle inequality we have that $d_H(u,v) \leq d_H(u,w) + d_H(w,v) \leq \frac{\alpha}{\alpha-1} \cdot r_{i-1} + r_i = \frac{\alpha}{\alpha-1} \cdot r_i$. See Figure 3.2a for an example. □

In addition to proving the existence of a vertex v at each level i, Lemma 26 implies a certain path from vertex u to $v \in V_i$. Start from node $(0,u)$ in the tree (that is, the copy of u corresponding to a leaf) and follow the path through a sequence of parents, until a level-i node (i,v) is reached. Lemma 26 shows that the distance in H along this path is at most $\frac{\alpha}{\alpha-1} \cdot r_i$.

Lemma 27. [Spanner Property] *For each edge $\{u,v\} \in E$, $d_H(u,v) \leq (1+\varepsilon) \cdot d(u,v)$.*

Proof. For ease of presentation, let $\lambda = \frac{\alpha}{\alpha-1}$. Let q be the smallest integer such that

$\frac{4\lambda}{\alpha^q} \leq \varepsilon < \frac{8\lambda}{\alpha^q}$. Thus $q = \lceil \log_\alpha \frac{4\lambda}{\varepsilon} \rceil$. Let k be such that $r_k \leq d(u,v) < r_{k+1}$, and assume first that $k \leq q-1$. Then $d(u,v) < \alpha^q \cdot r_0 \leq \frac{8\lambda}{\varepsilon} \cdot r_0 \leq \gamma r_0$, since $\gamma = 2\lambda\left(1 + \frac{4\alpha}{\varepsilon}\right) > \frac{8\lambda}{\varepsilon}$. Also since both u and v belong to V_0, by Lemma 25, we have that $\{u,v\} \in \widehat{E}$. This implies that $d_H(u,v) = d(u,v)$, proving the lemma true for this case. Assume now that $k \geq q$ and let $s = k - q \geq 0$. Note that $r_k = \alpha^q \cdot r_s$. By Lemma 26, there exist $x,\ y \in V_s$ such that $d_H(u,x) \leq \lambda \cdot r_s$ and $d_H(v,y) \leq \lambda \cdot r_s$. By the triangle inequality,

$$
\begin{aligned}
d(x,y) &\leq d(x,u) + d(u,v) + d(v,y) \\
&\leq \lambda \cdot r_s + d(u,v) + \lambda \cdot r_s && (d(x,u) \leq d_H(x,u),\ d(v,y) \leq d_H(v,y)) \\
&< \lambda \cdot r_s + \alpha \cdot r_k + \lambda \cdot r_s && (\text{since } d(u,v) < r_{k+1}) \\
&= r_s(2\lambda + \alpha \cdot \alpha^q) && (\text{since } r_k = \alpha^q \cdot r_s) \\
&\leq r_s(2\lambda + \alpha\tfrac{8\lambda}{\varepsilon}) \\
&= \gamma \cdot r_s
\end{aligned}
$$

Hence, by Lemma 25, $\{x,y\} \in \widehat{E}$ and therefore $d_H(x,y) = d(x,y)$. Using the triangle inequality again, we get

$$
\begin{aligned}
d_H(u,v) &\leq d_H(u,x) + d_H(x,y) + d_H(y,v) \\
&\leq 2\lambda \cdot r_s + d(x,y) \\
&\leq 4\lambda \cdot r_s + d(u,v) && (\text{from the upper bound derivation of } d(x,y)) \\
&\leq (1 + \tfrac{4\lambda}{\alpha^q}) \cdot d(u,v) && (\text{since } r_i = \alpha^q \cdot r_s \leq d(u,v)) \\
&\leq (1+\varepsilon) \cdot d(u,v)
\end{aligned}
$$

This completes the proof. □

Lemma 27 also identifies a uv-path in H of length at most $(1+\varepsilon)\cdot d(u,v)$. Simply follow the sequence of parents, starting at the node $(0,u)$ in the tree and similarly, starting at the node $(0,v)$. At a certain level (denoted s in the proof), the ancestor x of u and the ancestor y of v at that level are connected by an edge in H (see Figure 3.2b.

We now prove that H has degree bounded above by a constant. Recall the notation: for each point u, $i(v) = \max\{i \mid v \in V_i\}$. For each edge $\{u,v\} \in \widehat{E}$, direct $\{u,v\}$ from u to v, if $i(u) < i(v)$. If $i(u) = i(v)$, pick an arbitrary orientation. This edge orientation is identical to the one used in [7]. Call the resulting digraph $\overrightarrow{H}$.

Lemma 28. [Degree Property] *The out-degree of $\overrightarrow{H}$ is bounded above by $(\frac{1}{\varepsilon})^{O(\rho)}$.*

Proof. Let $\{u,v\} \in \widehat{E}$ be an arbitrary edge directed from u to v, and let i be such that $\{u,v\} \in E_i$. Then $d(u,v) \le \gamma \cdot r_i$. Now note that $r_{i+\delta} = \alpha^\delta \cdot r_i \ge \gamma \cdot r_i$ (recall that $\delta = \lceil \log_\alpha \gamma \rceil$). This, along with the fact that $V_{i+\delta}$ is an $r_{i+\delta}$-net, implies that it is not possible for both u and v to exist in $V_{i+\delta}$. Since $i(u) \le i(v)$ (by our assumption), it follows that $i(u) \le i + \delta$. On the other hand, $u \in V_i$ and so $i(u) \ge i$.

Summarizing, we have that $i(u) - \delta \le i \le i(u)$. This tells us that there are at most $\delta + 1 = O(\log_\alpha \gamma)$ values of i for which E_i may contain an edge outgoing from u. For each such i, by Lemma 24 there are at most $|N_i(u)| \le \left(\frac{1}{\varepsilon}\right)^{O(\rho)}$ edges in E_i outgoing from u. It follows that the total number of edges in $\widehat{E}$ outgoing from u is $\left(\frac{1}{\varepsilon}\right)^{O(\rho)} \cdot O(\log_\alpha \gamma) = \left(\frac{1}{\varepsilon}\right)^{O(\rho)}$. □

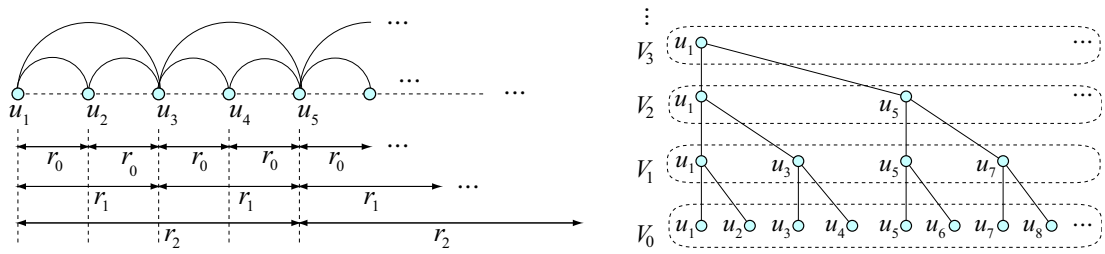
We now show that H has bounded weight.

Lemma 29. [Weight Property] *The total weight of H is $wt(H) = O(\log \Delta) \cdot \left(\frac{1}{\varepsilon}\right)^{O(\rho)} \cdot wt(MST)$, where MST is a minimum spanning tree of V, and Δ is the aspect ratio of G.*

Proof. We show that, for each i, $wt(E_i) = \left(\frac{1}{\varepsilon}\right)^{O(\rho)} \cdot wt(MST)$. This along with the fact that there are $h + 1 = \log_\alpha \frac{1}{r_0} + 1 = O(\log_\alpha \Delta)$ levels i, proves the claim of the lemma.

Let $U_i \subseteq V_i$ be the points in V_i incident to edges in E_i, and let $t = |U_i|$. Recall that any edge $\{u, v\} \in E_i$ satisfies $r_i < d(u, v) \leq \gamma \cdot r_i$. Thus, any spanning tree of a set of points containing U_i has weight at least $(t-1) \cdot r_i$, implying that $wt(MST) \geq (t-1) \cdot r_i$. Also note that the weight of E_i is bounded by $\Sigma_{u \in U_i} |N_i(u)| \cdot \gamma \cdot r_i \leq \left(\frac{1}{\varepsilon}\right)^{O(\rho)} \cdot t \cdot \gamma \cdot r_i$, using the upper bound on $|N_i(u)|$ given by Lemma 24. Using the lower bound on $wt(MST)$, we see that the weight of E_i is bounded above by $\left(\frac{1}{\varepsilon}\right)^{O(\rho)} \cdot \gamma \cdot (wt(MST) + r_i)$. Summing this expression over all E_i, yields the upper bound claimed in the lemma. □

The graph example from Figure 3.3 shows that the bound of Lemma 29 is tight. Vertices of the graph are placed at equal distance slightly larger than r_0 along a line segment (recall that the Euclidean space is a doubling metric space). The value of α in this example is 2, and satisfies the lower bound from inequality (3.1); the upper bound from (3.1) is only used in Theorem 38, which applies to an altered version of H and needs not hold for our example. Since $d(u_i, u_{i+1}) \approx r_0 < \gamma r_0$, all edges $\{u_i, u_{i+1}\}$, for $i = 1, 2, \ldots$, are in H (cf. Lemma 25). Note that these are



(a) (b)

Figure 3.3: An example to demonstrate the tightness result. (a) Graph H with total weight $wt(H) = \Omega(\log \Delta) \cdot wt(MST)$ (b) Net tree for the vertex set V of H; $V_0 = V_1 = V$, and $V_k = \{u_{1+i\cdot 2^k}, \quad i = 0, 1, 2, \ldots\}$, for $k \geq 2$.

precisely the edges that constitute a MST. Consider now all pairs of points in V_k, for some $k \geq 2$, at distance $\approx r_k$. Since $r_k < \gamma r_k$, edges connecting such pairs of points are all in H (cf. Lemma 25). For a fixed k, these edges span the entire line segment (see Figure 3.3b), therefore the total weight of these edges is equal to the length of the segment, which is precisely $wt(MST)$. Since there are $\Omega(\log \Delta)$ levels k, we have that $wt(H) = \Omega(\log \Delta) \cdot wt(MST)$, proving the bound of Lemma 29 tight.

3.2.2 Altering H for Bounded Degree

In this section we show how to modify H so as to bound the degree of each vertex by a constant. Lemma 28 shows that an oriented version of H, namely $\overrightarrow{H}$, has bounded out-degree. Next we describe a method that carefully replaces some directed edges in $\overrightarrow{H}$ by others so as to guarantee constant bound on the in-degree as well, without increasing the out-degree. The replacement procedure is similar to the one used in [7], slightly adjusted to work with UBGs. Assume without loss of generality that $\varepsilon \leq \frac{1}{2}$; otherwise, if $\varepsilon > \frac{1}{2}$, we proceed with $\varepsilon = \frac{1}{2}$. We use the fact

that $\varepsilon \leq \frac{1}{2}$ in the proof of Lemma 32. Let ℓ be the smallest positive integer such that $\frac{1}{\alpha^{\ell-1}} \leq \varepsilon$. Thus $\ell = O(\log_\alpha \frac{1}{\varepsilon})$.

Edge Replacement Procedure. Let u be an arbitrary point in V and let $M(u,i)$ be the set of all vertices $v \in V_i$ such that $\{v,u\}$ is an edge in E_i directed from v to u in $\overrightarrow{H}$. Let $I(u) = \langle i_1, i_2, \ldots \rangle$ be the increasing sequence of all indices i_k for which $M(u,i_k)$ is nonempty. For $1 \leq k \leq \ell$, we do not disturb any of the edges from points in $M(u,i_k)$ to u. For each $k > \ell$ such that $i_k \leq h - \delta - 3$, real edges $\{v,u\}$ connecting $v \in M(u,i_k)$ to u are replaced by other edges. Specifically, an edge $\{v,u\}$, with $v \in M(u,i_k)$, is replaced by an edge $\{v,w\}$, where w is an arbitrary vertex in $M(u,i_{k-\ell})$. The replacement can be equivalently viewed as happening in either H or its oriented version $\overrightarrow{H}$. In $\overrightarrow{H}$, we replace the directed edge (v,u) by the directed edge (v,w). In the next two lemmas, our arguments will use $\overrightarrow{H}$ or H, as convenient.

Let $\widetilde{E}$ be the resulting set of edges. By our construction, $|\widetilde{E}| \leq |\widehat{E}|$. An important observation here is that the replacement procedure above is carried out only for real edges in E_i, with $i \leq h - \delta - 3$ (that is, only edges of length no greater than $1/\alpha^3$). This is to ensure that only real edges get replaced and no virtual edges get added, a guarantee that is shown in the following lemma.

Lemma 30. *$\widetilde{E} \setminus \widehat{E}$ contains no virtual edges.*

Proof. Let $\{v,u\}$ be an edge that gets replaced by $\{v,w\}$, with $v \in M(u,i_k)$ and $w \in M(u,i_{k-\ell})$. Recall that $k > \ell$ and $i_k \leq h-\delta-3$. Using the definitions of E_{i_k} and $E_{i_{k-\ell}}$ and the fact that $\frac{1}{\alpha^{\ell-1}} \leq \varepsilon$, it follows that $d(w,u) \leq \varepsilon \cdot d(v,u)$. By the triangle

inequality, $d(v,w) \leq d(v,u) + d(w,u) \leq (1+\varepsilon)d(v,u)$. Now note that $d(v,u) \leq 1/\alpha^3$. This is because edges in E_{i_k} have length no greater than $\gamma \cdot r_{i_k} \leq 1/\alpha^3$, for any $i_k \leq h - \delta - 3$. Therefore $d(v,w) \leq (1+\varepsilon)/\alpha^3 \leq 1$, for any $\alpha^3 \geq (1+\varepsilon)$. □

Let $J = (V, \widetilde{E})$. First we show that J indeed has bounded degree (Lemma 31). Second we show that the metric distance d_J induced by shortest paths in J is a good approximation of d_H (Lemma 32). A consequence of this is that J remains connected, and maintains spanner paths between endpoints of real edges.

Lemma 31. *Every vertex in $J = (V, \widetilde{E})$ has degree bounded by $(\frac{1}{\varepsilon})^{O(\rho)}$.*

Proof. Let A be the maximum out-degree of a vertex of $\overrightarrow{H}$. By Lemma 28, $A \leq (\frac{1}{\varepsilon})^{O(\rho)}$. Let B be the largest of $|N_i(u)|$, for all i and all u. By Lemma 24, $B \leq (\frac{1}{\varepsilon})^{O(\rho)}$. The edge-replacement procedure replaces a directed edge (v,u) by a directed edge (v,w). So the out-degrees of vertices remain unchanged by the edge-replacement procedure, and continue to be bounded above by $(\frac{1}{\varepsilon})^{O(\rho)}$. Thus, we can simply focus on the in-degrees of vertices. We bound these by accounting for the in-degree of an arbitrary vertex x with respect to old edges (in $\widetilde{E} \cap \widehat{E}$) and with respect to new edges (in $\widetilde{E} \setminus \widehat{E}$); we show that both in-degrees are bounded above by $(\frac{1}{\varepsilon})^{O(\rho)}$.

In-degree of x with respect to $\widetilde{E} \cap \widehat{E}$. Out of the edges in $\overrightarrow{H}$ that come into x, at most $B(\ell + \delta + 3)$ remain in $\widetilde{E}$. More specifically, at most B edges at each of the first ℓ levels $i_1, i_2, \ldots, i_\ell$ in $I(x)$, plus at most B edges in each of E_i, $i = h-\delta-2, h-\delta-1, \ldots, h$, remain in $\widetilde{E}$. Any other real edge directed into x gets replaced by an edge not incident to x. We end this case by noting that $B(\ell+\delta+3) = (\frac{1}{\varepsilon})^{O(\rho)}$.

In-degree of x with respect to $\widetilde{E} \setminus \widehat{E}$. Vertex x has a new in-coming edge whenever it plays the role of w in the edge-replacement procedure. Recall that in the edge-replacement procedure, w and v are both in-neighbors of u. For each edge (w, u), there are at most B edges (v, u) directed into u that may get replaced by (v, w). Furthermore, there are A edges (w, u) outgoing from w. This gives an upper bound of $AB = (\frac{1}{\varepsilon})^{O(\rho)}$ on the in-degree of x. □

It remains to show that d_J is a good approximation of d_H. Intuition for this is provided by the proof of Lemma 30. In that proof, it is shown that when $\{v, w\}$ replaces $\{v, u\}$, $d(w, u) \leq \varepsilon \cdot d(v, u)$ and $d(v, w) \leq (1 + \varepsilon) \cdot d(v, u)$. Thus, if the path $\langle v, w, u\rangle$ existed in $\widetilde{E}$, this path would have length at most $(1 + 2\varepsilon) \cdot d(v, u)$. However, edge $\{w, u\}$ may not exist in $\widetilde{E}$, since it may itself have been replaced. Thus a shortest path from w to u in $\widetilde{E}$ may be longer than $d(w, u)$. However, since $d(w, u) \leq \varepsilon \cdot d(v, u)$, the extra cost of replacing $\{w, u\}$ is marginal and the eventual sum of all of these lengths is still bounded above by $(1 + 2\varepsilon) \cdot d(v, u)$. Thus we have the following lemma:

Lemma 32. $d_J \leq (1 + 2\varepsilon)d_H$.

Proof. It suffices to show that, for each edge $\{v, u\} \in \widehat{E}$ that gets replaced, $d_J(v, u) \leq (1 + 2\varepsilon) \cdot d_H(v, u)$. Assume without loss of generality that edge $\{v, u\}$ directs into u, and let k be such that $v \in M(u, i_k)$. Then it must be that $k > \ell$ and $i_k \leq h - \delta - 3$, otherwise $\{v, u\}$ would not get replaced.

Let $w_0 = v$, and assume that $\{w_0, u\}$ gets replaced by $\{w_0, w_1\}$. By construc-

tion, $w_1 \in M(u, i_{k-\ell})$. We now show that $d(w_1, u) \leq \varepsilon \cdot d(w_0, u)$ and $d(w_0, w_1) \leq (1+\varepsilon) \cdot d(w_0, u)$. This claim follows from the following observations:

1. $i_{k-\ell} \leq i_k - \ell$ (since increasing indices in $I(u)$ are not necessarily incremental). This implies that $r_{i_{k-\ell}} \leq r_{i_k - \ell}$, which in turn implies that $d(w_1, u) \leq \gamma \cdot r_{i_k-\ell} = \gamma \cdot r_{i_k}/\alpha^{\ell}$.

2. $d(w_0, u) \geq \gamma \cdot r_{i_k - 1} = \gamma \cdot r_{i_k}/\alpha$ (by definition). This along with the first observation implies that $d(w_1, u) \leq d(w_0, u)/\alpha^{\ell-1} = \varepsilon \cdot d(w_0, u)$.

3. By the triangle inequality, $d(w_0, w_1) \leq d(w_1, u) + d(w_0, u) \leq (1+\varepsilon) \cdot d(w_0, u)$.

So if $\{w_1, u\} \in \widetilde{E}$, then the claim of the lemma follows immediately from the observations above and the triangle inequality: $d_J(w_0, u) \leq d_J(w_0, w_1) + d_J(w_1, u) = d(w_0, w_1) + d(w_1, u) \leq (1+2\varepsilon) \cdot d(w_0, u)$. Otherwise, $\{w_1, u\} \in \widehat{E}$ in turn gets replaced by $\{w_1, w_2\} \in \widetilde{E}$, and the process repeats itself. Let $w_0, w_1, \ldots, w_r$ be a shortest path in J that leads to an edge $\{w_r, u\} \in \widetilde{E} \cap \widehat{E}$. The replacement procedure ensures that such a path always exists. This means that $\{w_0, w_1\}, \{w_1, w_2\}, \ldots, \{w_{r-1}, w_r\}$ are all new edges in $\widetilde{E} \cap \widehat{E}$. The three observations above translated to lower levels yield, for each $j = 1, 2, \ldots, r$, the following two inequalities: (i) $d(w_j, u) \leq \varepsilon \cdot d(w_{j-1}, u)$, and (ii) $d(w_{j-1}, w_j) \leq (1+\varepsilon) \cdot d(w_{j-1}, u)$. Repeated application of the first inequality yields $d(w_j, u) \leq \varepsilon^j \cdot d(w_0, u)$. Finally, we have:

$$\begin{aligned} d_J(v,u) &\leq \sum_{j=1}^{r} d(w_{j-1},w_j) + d(w_r,u) \\ &\leq (1+\varepsilon)\sum_{j=1}^{r} \varepsilon^{j-1} d(w_0,u) + \varepsilon^r d(w_0,u) \\ &\leq d(w_0,u)\cdot(1+\varepsilon)/(1-\varepsilon) \\ &\leq (1+2\varepsilon)\cdot d(v,u) \end{aligned}$$

This latter inequality follows from the fact that, for $0 < \varepsilon < 1/2$, $(1+\varepsilon)(1-\varepsilon) \leq 1+2\varepsilon$. □

3.2.3 Eliminating Virtual Edges

The only impediment in having $J = (V, \widetilde{E})$ serve as a spanner for the input UBG G is the presence of virtual edges in J. Recall that these are edges of length greater than 1 and clearly do not exist in G. In this section we show that there exist real edges that can take over the role of virtual edges in J, without violating the properties J is expected to have.

Let $\{u,v\} \in E$ be an arbitrary (real) edge and let k be such that $r_k \leq d(u,v) < r_{k+1}$. Let q be as in the proof of Lemma 27: the smallest integer such that $\frac{\alpha}{\alpha-1}\cdot\frac{4}{\alpha^q} \leq \varepsilon < \frac{\alpha}{\alpha-1}\cdot\frac{8}{\alpha^q}$. As mentioned in Section 3.2.1, the proof of Lemma 27 implies a certain uv-path of length at most $(1+\varepsilon)\cdot d(u,v)$ in $H = (V, \widehat{E})$. If $k \leq q-1$, this path is just the edge $\{u,v\}$, because $\{u,v\}$ is guaranteed to exist in $\widehat{E}$ (see proof of Lemma 27). The Edge Replacement Procedure (Section 3.2.2) ensures that only real edges are replaced, and each real edge is replaced by a path consisting only of real edges. This along with Lemma 32 ensures that even in $\widetilde{E}$ there is a uv-path of length at most $(1+2\varepsilon)\cdot d(u,v)$, consisting of real edges only. If $k \geq q$, the uv-path in H implied

by Lemma 27 may have more than one edge. Let $s = k - q$ and (s, u^*) (respectively, (s, v^*)) be the level-s ancestor of the leaf $(0, u)$ (respectively, the leaf $(0, v)$) in the net tree $\langle V_0, V_1, \ldots, V_h \rangle$. Then the edge $\{u^*, v^*\}$ is guaranteed to be present in $\widehat{E}$ and the uv-path implied by Lemma 27 starts at $(0, u)$, goes up the net tree via parents to (s, u^*), then to (s, v^*), and then follows the unique path down the tree from (s, v^*) to $(0, v)$. It is easy to check that of all the edges in this path, only $\{u^*, v^*\}$ may be virtual – specifically, when the edge $\{u, v\}$ is long enough to guarantee that $k \geq h - \delta + 1 + q$, then $s = k - q \geq h - \delta + 1$ and the edge $\{u^*, v^*\}$ may belong to E_s. Recall that for $i \geq h - \delta + 1$, edges in E_i may not be real and in particular $\{u^*, v^*\}$ may be a virtual edge. Since the uv-path implied by Lemma 27 passes through edge $\{u^*, v^*\}$, one has to be careful in replacing $\{u^*, v^*\}$ by a real edge. Our virtual edge replacement procedure is given below.

For any node (i, v) in the net tree, let $T(i, v)$ denote the set of all vertices $u \in V$, such that the subtree of the net tree rooted at (i, v) contains a copy of u. In other words, $T(i, v) = \{u \in V \mid (i, v)$ is an ancestor of (j, u) for some $j \leq i\}$.

Virtual Edge Replacement Procedure. For a virtual edge $\{u, v\} \in E_i$, if there is a real edge $\{x, y\}$ already in the spanner H, with $x \in T(i, u)$ and $y \in T(i, v)$, then simply delete $\{u, v\}$. Similarly, if there is no such real edge $\{x, y\}$ in the input graph G with $x \in T(i, u)$ and $y \in T(i, v)$ then simply delete $\{u, v\}$. Otherwise, find a real edge $\{x, y\} \in E$, $x \in T(i, u)$ and $y \in T(i, v)$, and replace $\{u, v\}$ by $\{x, y\}$.

The reason why this replacement procedure works can be intuitively explained

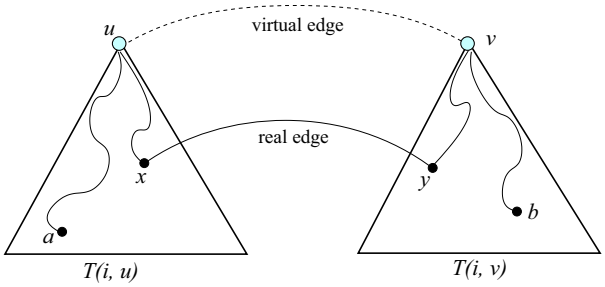


Figure 3.4: A short ab-path passes through virtual edge $\{u, v\}$. After replacing virtual edge $\{u, v\}$ by real edge $\{x, y\}$, there is a short ab-path through $\{x, y\}$.

as follows. A virtual edge $\{u, v\} \in E_i$ is important for pairs of vertices $\{a, b\}$, with $a \in T(i, u)$ and $b \in T(i, v)$, for which all ab-paths of length at most $(1+\varepsilon) \cdot d(a, b)$ pass through $\{u, v\}$. Replacing $\{u, v\}$ by $\{x, y\}$ provides the following alternate ab-path that is short enough: starting at the leaf a, go up the tree rooted at (i, u) via parents until an ancestor common to a and x is reached, then come down to x, take edge $\{x, y\}$, go up the tree rooted at (i, v) until an ancestor common to b and y is reached, and finally go down to b. Figure 3.4 illustrates this alternate path. Note that this entire path consists only of real edges.

We finally state our main result. Let G' be the graph obtained from J by replacing virtual edges using the Virtual Edge Replacement Procedure.

Theorem 33. *$G' = (V, E')$ is a $(1 + \varepsilon)$-spanner of G with degree bounded above by $(\frac{1}{\varepsilon})^{O(\rho)}$ and weight bounded above by $O(\log \Delta) \cdot (\frac{1}{\varepsilon})^{O(\rho)} \cdot wt(MST)$.*

A proof similar to that of Lemma 27 can be used to show the spanner property of G'. The fact that G' is lightweight simply follows from the fact that a virtual edge of length greater than 1 in J, either gets eliminated, or gets replaced by at most one

real edge of length at most 1 in G'. The constant degree bound follows from the observation that, for a vertex x to acquire a new incident edge, there is an ancestor of x in the net tree at level $h - \delta + 1$ or higher, that loses an incident edge at that level. There are a constant number of such ancestors and from Lemma 24, we know that any vertex has a constant number of incident edges at any particular level.

We conclude this section with a summary of our algorithm.

Algorithm Overview SPANNER$((V, d),\ \varepsilon)$

Let $\sqrt[3]{1+\varepsilon} < \alpha < \sqrt{1+\varepsilon}$ be a constant, $\gamma = \frac{2\alpha}{\alpha-1}\left(1 + \frac{4\alpha}{\varepsilon}\right)$, and $\delta = \lceil \log_\alpha \gamma \rceil$.
Let h be the smallest such that $\frac{1}{\alpha^h}$ is smaller than the minimum inter-point distance.
Let $r_0 = \frac{1}{\alpha^h}$ and let $r_i = \alpha \cdot r_{i-1}$, for all $i > 0$.

Constructing a linear size $(1+\varepsilon)$-spanner $H = (V, \widehat{E})$.

1. Construct the net tree $\langle V_0, V_1, \ldots, V_h \rangle$.
   [Let $i(u) = \max\{i \mid u \in V_i\}$.]
2. Construct the sets
   $$E_0 = \{\{u, v\} \in V_0 \times V_0 \mid\ d(u, v) \le \gamma \cdot r_0\},$$
   $$E_i = \{\{u, v\} \in V_i \times V_i \mid \gamma \cdot r_{i-1} <\ d(u, v) \le \gamma \cdot r_i\}, \text{ for } 1 \le i \le h.$$
   [Let $\widehat{E} = \cup_i E_i$ and $H = (V, \widehat{E})$.]

Replacing edges to obtain a constant degree bound.

3. Orient each edge $\{u, v\} \in \widehat{E}$ from u to v if $i(u) \le i(v)$, breaking ties arbitrarily.
   [Let $M(u, i)$ denote the set of vertices $v \in V_i$, with $\{v, u\} \in \widehat{E}$.
4. For each $u \in V$, construct the increasing sequence $I(u) = \langle i_1, i_2, \ldots, \rangle$ of all i_k with $M(u, i_k) \neq \emptyset$. [Let ℓ be the smallest integer with $\frac{1}{\alpha^{\ell-1}} \le \varepsilon$.]
5. For each $u \in V$ and each $i_k \in I(u)$, with $k > \ell$ and $i_k \le h - \delta - 3$, do
6. Replace directed edge (v, u), $v \in M(u, i_k)$ by edge (v, w),
   for arbitrary $w \in M(u, i_{k-\ell})$.
   [Let $J = (V, \widetilde{E})$ be the resulting graph, with distance metric d_J.]

Replacing virtual edges by real ones.

   [Let $T(i, v) = \{x \in V \mid (i, v)$ is an ancestor of (j, x) for some $j \le i\}$.]
7. For each $i \ge h - \delta + 1$ and each virtual edge $\{u, v\} \in E_i$ do
8. If there is a real edge $\{x, y\} \in \widetilde{E}$, $x \in T(i, u)$ and $y \in T(i, v)$, then do nothing.
9. Otherwise, if there is a real edge $\{x, y\} \in E$, with $x \in T(i, u)$ and $y \in T(i, v)$,
   replace $\{u, v\}$ by $\{x, y\}$.
   [Let E' be the set of resulting edges. Output is $G' = (V, E')$.]

3.3 Leapfrog Property

In Lemma 29, we showed that $H = (V, \widehat{E})$ has total weight bounded above by $O(\log \Delta) \cdot \left(\frac{1}{\varepsilon}\right)^{O(\rho)} \cdot wt(MST)$, where Δ is the aspect ratio of G. Thus, for fixed ε and constant doubling dimension ρ, the upper bound is within $O(\log \Delta)$ times the optimal value. In an attempt to show a bound that is within $O(1)$ times the optimal value, we use a tool that is widely used in the computational geometry literature [17, 13, 26]. In the context of building lightweight $(1+\varepsilon)$-spanners for Euclidean spaces, Das and Narasimhan [17] have shown that, if the set of edges in the spanner satisfy a property known as the *leapfrog property*, then the total weight of the spanner is bounded above by $O(wt(MST))$. Refer to Chapter 2 (Equation 2.6) for a detailed definition of the leapfrog property.

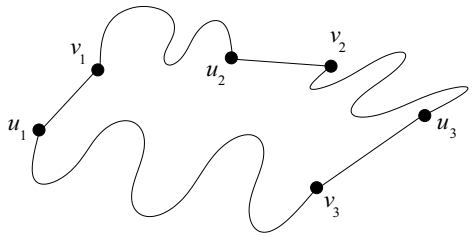


Figure 3.5: Definition of the t-leapfrog property with $S = \{\{u_1, v_1\}, \{u_2, v_2\}, \{u_3, v_3\}\}$.

Das and Narasimhan [17] show the following connection between the leapfrog property and the weight of the spanner.

Lemma 34. *Let $t \geq t' > 1$. If the line segments F in d-dimensional space satisfy the (t', t)-leapfrog property, then $wt(F) = O(wt(MST))$, where MST is a minimum*

spanning tree connecting the endpoints of line segments in F. The constant in the asymptotic notation depends on t, t' and d.

It is well known that, if a spanner is built "greedily", then the set of edges in the spanner satisfies the leapfrog property [17, 13, 26]. In [15] we showed that even a "relaxed" version of the greedy algorithm would ensure that the spanner edges have the leapfrog property. This was critical to showing that the spanner constructed in a distributed manner for UBGs in Euclidean spaces [15] had total weight bounded above by $O(wt(MST))$. Here we ask if it is possible to do the same for UBGs in metric spaces with constant doubling dimension. In an attempt to answer this question we show that, using a variant of the SPANNER algorithm (outlined at the end of Section 3.2), we can build, for a given UBG G in a doubling metric space, a $(1+\varepsilon)$-spanner with degree bounded above by a constant and with the (t, t')-leapfrog property, for some constants $t \geq t' > 1$. Note that this does not give us the desired $O(wt(MST))$ bound on the weight of the constructed spanner because we do not know if the equivalent of Lemma 34 holds for non-Euclidean metric spaces. The proof of Lemma 34 in [17] is quite geometric and does not suggest an approach to its generalization to metric spaces of constant doubling dimension.

To guarantee that the output spanner satisfies the (t', t)-leapfrog property, we need to make two modifications to the SPANNER algorithm as follows. First, in step 2 of the algorithm, we add an edge $\{u, v\}$ to E_i only if the partial spanner H_{i-1} contains no uv-path of length at most $(1+\varepsilon) \cdot d(u, v)$. Second, we eliminate from H (the graph induced by $E_0 \cup E_1 \cup \ldots \cup E_h$) edges that are "redundant". For each i,

call two edges $\{u_1, v_1\}$ and $\{u_2, v_2\}$ in E_i *mutually redundant* if both of the following conditions hold:

(a) $d_H(v_1, u_2) + d(u_2, v_2) + d_H(v_2, u_1) \leq (1 + \varepsilon) \cdot d(u_1, v_1)$

(b) $d_H(v_2, u_1) + d(u_2, v_2) + d_H(v_1, u_2) \leq (1 + \varepsilon) \cdot d(u_2, v_2)$

These two conditions imply that (i) $H \setminus \{u_1, v_1\}$ contains a u_1v_1-path of length at most $(1 + \varepsilon) \cdot d(u_1, v_1)$, and (ii) $H \setminus \{u_2, v_2\}$ contains a u_2v_2-path of length at most $(1 + \varepsilon) \cdot d(u_2, v_2)$. Thus, one of the edges $\{u_1, v_1\}$ and $\{u_2, v_2\}$ can potentially be eliminated from H, without compromising the $(1 + \varepsilon)$-spanner property of H. In fact, it is necessary to eliminate such pairs of edges in order to ensure the leapfrog property for H (and ultimately for the output spanner). To this end, we construct a *redundancy* graph Γ such that nodes in Γ correspond to edges in H, and edges in Γ correspond to mutually redundant edges in H. Note that Γ contains at least h connected components, one for each i (since mutually redundant edges belong to a same set E_i by definition). We determine a Maximal Independent Set (MIS) I of Γ and eliminate from H all edges associated with nodes in Γ that are not in I. The modified algorithm, called Leapfrog-Spanner, is outlined below.

Algorithm Overview Leapfrog-Spanner$((V, d),\ \varepsilon)$

Let α, γ, h and r_i be defined as in the Spanner algorithm.

Constructing a linear size $(1 + \varepsilon)$-spanner $H = (V, \widehat{E})$.

1. Construct the net tree $\langle V_0, V_1, \ldots, V_h \rangle$.
2. Construct

$$E_0 = \{\{u, v\} \in V_0 \times V_0 \mid\ d(u, v) \leq \gamma \cdot r_0\} \text{ and } H_0 = (V, E_0).$$

$$E_i = \{\{u, v\} \in V_i \times V_i \mid \gamma \cdot r_{i-1} <\ d(u, v) \leq \gamma \cdot r_i, \text{ and there is no } uv\text{-path of length at most } (1 + \varepsilon) \cdot d(u, v) \text{ in } H_{i-1}\}, \text{ and } H_i = (V, E_i)$$

for $1 \leq i \leq h$.

[Let $H = (V, E_H)$ be the resulted graph.]

3. Eliminate Redundant Edges:
   3.1 Construct the redundancy graph Γ of H.
   3.2 Determine a Maximal Independent Set I of Γ (over all connected components of Γ).
   3.3 Eliminate from E_H all edges associates with nodes not in I.
   [Let $H = (V, \widehat{E})$ be the resulted graph.]

Replacing edges to obtain a constant degree bound.
As in the SPANNER algorithm.

Replacing virtual edges by real ones.
As in the SPANNER algorithm.
[Let E^ℓ be the set of resulting edges.]
Output is $G^\ell = (V, E^\ell)$.]

In the rest of this section we show that output of the LEAPFROG-SPANNER algorithm is indeed a spanner that satisfies the leapfrog property. Let $H = (V, \widehat{E})$ be the graph obtained after Step (3) of the LEAPFROG-SPANNER algorithm, in which all redundant edges have been removed.

Lemma 35. *Let $u, v \in V_i$ such that $d(u, v) \leq \gamma \cdot r_i$. Then $d_{H_i}(u, v) \leq (1 + \varepsilon) \cdot d(u, v)$ and $d_H(u, v) \leq (1 + \varepsilon) \cdot d(u, v)$.*

Proof. This is the analogous of Lemma 25. We first show that the lemma is true for $i = 0$. Since $\{u, v\}$ is added to E_0 in Step (2) of the LEAPFROG-SPANNER algorithm, we have that $d_{H_0}(u, v) = d(u, v)$. If $\{u, v\}$ is not eliminated from Γ in Step (3) of the algorithm, then $d_H(u, v) = d(u, v)$. Otherwise, there exists in Γ an edge $\{x, y\}$ mutually redundant with respect to $\{u, v\}$; such an edge would correspond to a node in I (determined in Step (3.2) of the algorithm). The redundancy condition ensures that H contains a uv-path passing through $\{x, y\}$ of length $d_H(u, v) \leq (1+\varepsilon) \cdot d(u, v)$.

Assume now that $i > 0$ and let $j \leq i$ be such that $\gamma \cdot r_{j-1} < d(u,v) \leq \gamma \cdot r_j$. Since $V_i \subseteq V_j$, we have that $u, v \in V_j$ and therefore $\{u,v\}$ is added to E_j in Step (2) of the algorithm, unless H_{j-1} already contains a uv-path of length no greater than $(1+\varepsilon) \cdot d(u,v)$. Arguments similar to the ones used for the case $i = 0$ proves the lemma true for this case as well. □

Lemma 36. [Spanner Property] *For each edge $\{u,v\} \in E$, $d_H(u,v) \leq (1+3\varepsilon) \cdot d(u,v)$.*

Proof. This is the analogous of Lemma 27. Let λ, k, q and s be as in the proof of Lemma 27: $\lambda = \frac{\alpha}{\alpha-1}$; $q = \lceil \log_\alpha \frac{4\lambda}{\varepsilon} \rceil$; k is such that $r_k \leq d(u,v) < r_{k+1}$; and $s = \min(0, k-q)$. As shown in that proof, there exist $x, y \in V_s$ such that $d(x,y) \leq \gamma \cdot r_s$. By Lemma 35 we have that $d_H(x,y) \leq (1+\varepsilon) \cdot d(x,y)$. Following the same proof idea as in Lemma 27, we get

$$\begin{aligned}
d_H(u,v) \leq\ & d_H(u,x) + d_H(x,y) + d_H(y,v) \\
\leq\ & 2\lambda \cdot r_s + d_H(x,y) \\
\leq\ & 2\lambda \cdot r_s + (1+\varepsilon) d(x,y) \\
\leq\ & [\tfrac{2\lambda}{\alpha^q}(2+\varepsilon) + (1+\varepsilon)] \cdot d(u,v) && \text{(since } d(x,y) \leq 2\lambda \cdot r_s + d(u,v) \text{ and } r_s \leq d(u,v)/\alpha^q\text{)} \\
<\ & (1+3\varepsilon) \cdot d(u,v) && \text{(since } \tfrac{4\lambda}{\alpha^q} \leq \varepsilon\text{)}
\end{aligned}$$

This completes the proof. □

The following lemma shows a similar result, but this time restricted to partial spanner H_i, for each $i \geq 1$. This result will be used in proving the leapfrog property of H in Theorem 38.

Lemma 37. *Let $i > 0$ and let u, v be such that $d(u,v) \leq \gamma \cdot r_{i-1}$. Then $d_{H_i}(u,v) \leq (1+3\varepsilon) \cdot d(u,v)$.*

Proof. Let λ, q, and k be as in the proof of Lemma 27: : $\lambda = \frac{\alpha}{\alpha-1}$; $q = \lceil \log_\alpha \frac{4\lambda}{\varepsilon} \rceil$; and k is such that $r_k \leq d(u,v) < r_{k+1}$. Since we restrict our attention to H_i only, we choose $s = \min(i, k-q)$. We show that there exist $x, y \in V_s$ such that $d_{H_i}(x,y) \leq (1+\varepsilon) \cdot d(x,y)$. This enables us to use the proof of Lemma 36 to show that $d_{H_i}(u,v) \leq (1+3\varepsilon) \cdot d(u,v)$. We discuss two cases, depending on the value of s.

1. $s = k - q \leq i$. Then $r_k = \alpha^q \cdot r_s$, and we can use the proof of Lemma 27 to show that there exist $x, y \in V_s$ such that $d(x,y) \leq \gamma \cdot r_s$. Cf. Lemma 37 we have that $d_{H_s}(x,y) \leq (1+\varepsilon) \cdot d(x,y)$ and since H_s is a subgraph of H_i, it follows that $d_{H_i}(x,y) \leq (1+\varepsilon) \cdot d(x,y)$.

2. $s = i > k - q$. In this case $r_k > \alpha^q \cdot r_s$ and therefore we cannot use the proof of Lemma 27 to show a similar result. Note however that, cf. Lemma 26, there exist $x, y \in V_i$ such that $d_{H_i}(u,x) \leq \lambda \cdot r_i$ and $d_{H_i}(v,y) \leq \lambda \cdot r_i$. Using the triangle inequality,

$$
\begin{array}{lcll}
d(x,y) & \leq & d(x,u) + d(u,v) + d(v,y) & \\
& \leq & \lambda \cdot r_i + d(u,v) + \lambda \cdot r_i & \text{since } d(x,u) \leq d_{H_i}(x,u), d(v,y) \leq d_{H_i}(v,y)) \\
& \leq & 2\lambda \cdot r_i + \gamma r_{i-1} & (\text{since } d(u,v) \leq \gamma r_{i-1}) \\
& = & (2\lambda + \frac{\gamma}{\alpha}) \cdot r_i & (\text{since } r_i = \alpha \cdot r_{i-1}) \\
& \leq & \gamma r_i & \text{for any } \alpha \geq \sqrt[3]{1+\varepsilon} > \frac{1+\sqrt{1+\varepsilon}}{2}
\end{array}
$$

Thus we can apply the result of Lemma 35 to show that $d_{H_i}(x,y) \leq (1+\varepsilon) \cdot d(x,y)$.

This completes the proof. □

Theorem 38. [Leapfrog Property] *The edge set $\widehat{E}$ constructed in Step (2) of the* Leapfrog-Spanner *algorithm satisfies the (t',t)-leapfrog property, for any $1 < t' < \frac{t}{\alpha^2}$ and any $\sqrt[3]{1+\varepsilon} < \alpha < \sqrt{1+3\varepsilon}$. Here $t = 1+3\varepsilon$.*

Proof. Consider an arbitrary subset $S = \{\{u_1,v_1\},\{u_2,v_2\},\ldots,\{u_m,v_m\}\}$ of $\widehat{E}$. To prove inequality (2.6) for S, it suffices to consider the case when $\{u_1,v_1\}$ is a longest edge in S. First observe that, if either $d(v_m,u_1) > d(u_1,v_1)$ or $d(v_k,u_{k+1}) > d(u_1,v_1)$ for any k, $1 \leq k < s$, then inequality (2.6) holds. Thus it suffices to discuss the case when $d(v_m,u_1) \leq d(u_1,v_1)$ and $d(v_k,u_{k+1}) \leq d(u_1,v_1)$ for each k, $1 \leq k < s$.

Let i be such that $\{u_1,v_1\} \in E_i$. Consider first the case in which at least one of the edges in the set $S' = \{\{v_m,u_1\}\} \cup \{\{v_k,u_{k+1}\} \mid 1 \leq k < s\}$ has length greater than γr_{i-2}. Then the right hand side of the inequality (2.6) is at least $t \cdot \gamma \cdot r_{i-2}$. We also have that $t' \cdot d(u_1,v_1) \leq t' \cdot \gamma \cdot r_i = t' \cdot \gamma \cdot \alpha^2 \cdot r_{i-2} < t \cdot \gamma \cdot r_{i-2}$ for any $1 < t' < t/\alpha^2$, and so the leapfrog property holds for this case, for any $\alpha < \sqrt{t} = \sqrt{1+3\varepsilon}$.

Consider now the case in which each edge in the set S' has length no greater than γr_{i-2}. Cf. Lemma 35, the partial spanner H_{i-1} induced by the edge set $E_0 \cup E_1 \cup \ldots \cup E_{i-1}$ contains spanner paths between the endpoints of each edge in S'. More precisely, for each edge $\{v,u\} \in S'$, $d_{H_{i-1}}(v,u) \leq (1+3\varepsilon) \cdot d(v,u) = t \cdot d(v,u)$.

For $1 \leq k < s$, let P_k be a shortest $v_k u_{k+1}$-path in H_{i-1}, and let P_m be a shortest $v_m u_1$-path. From the discussion above it follows that $wt(P_k) \leq t \cdot d(v_k, u_{k+1})$ and $wt(P_m) \leq t \cdot d(v_m, u_1)$. We discuss three cases, depending on the number of edges in E_i that belong to S.

Case 1: $|S \cap E_i| = 1$. In other words, $S \cap E_i = \{u_1, v_1\}$. In this case $P = P_1 \oplus \{u_2, v_2\} \oplus P_2 \oplus \{u_3, v_3\} \oplus \ldots \oplus P_m$ is a path from u_1 to v_1 in H_{i-1}, and $wt(P)$ is no greater than the right hand side of the leapfrog inequality (2.6). Furthermore, $wt(P) > t \cdot d(u_1, v_1) > t' \cdot d(u_1, v_1)$, otherwise the edge $\{u_1, v_1\}$ would not have been added to E_i in Step (2.1) of the LEAPFROG-SPANNER algorithm. This shows that the leapfrog property holds for this case.

Case 2: $|S \cap E_i| > 2$. We use the fact that $d(u, v) > \frac{d(u_1, v_1)}{\alpha}$ for each edge $\{u, v\} \in |S \cap E_i|$, to show that the right hand side of the leapfrog inequality (2.6) is greater than $\frac{2 \cdot d(u_1, v_1)}{\alpha}$. Thus the leapfrog property holds for any $t' < \frac{2}{\alpha}$.

Case 3: $|S \cap E_i| = 2$. Let $k > 1$ be such that $\{u_k, v_k\} \in E_i \cap S$. Thus $S \cap E_i = \{\{u_1, v_1\}, \{u_k, v_k\}\}$. Our proof that the leapfrog inequality holds for this case is by contradiction. Assume to the contrary that the leapfrog inequality (2.6) does not hold:

$$t' \cdot d(u_1, v_1) \geq \sum_{i=2}^{m} d(u_i, v_i) + t \cdot \Big(\sum_{i=1}^{m-1} d(v_i, u_{i+1}) + d(v_m, u_1) \Big). \tag{3.2}$$

This along with the fact that H_{i-1} contains t-spanner paths $P_1, P_2, \ldots, P_m$, yields

$$t \cdot d(u_1, v_1) \geq t' \cdot d(u_1, v_1) \geq d_{H_{i-1}}(v_1, u_k) + d(u_k, v_k) + d_{H_{i-1}}(v_k, u_1). \tag{3.3}$$

Next we consider the path from u_k to v_k induced by edges in $P \cup \{u_1, v_1\}$. Suppose first that

$$t \cdot d(u_k, v_k) \geq \sum_{i=1}^{k-1} d(u_i, v_i) + \sum_{i=k+1}^{m} d(u_i, v_i) + t \cdot \Big(\sum_{i=1}^{m-1} d(v_i, u_{i+1}) + d(v_m, u_1) \Big).$$

This implies that

$$t \cdot d(u_k, v_k) \geq d_{H_{i-1}}(v_1, u_k) + d(u_1, v_1) + d_{H_{i-1}}(v_k, u_1) \tag{3.4}$$

However, inequalities (3.3) and (3.4) tell us that the edges $\{u_1, v_1\}, \{u_k, v_k\} \in E_i$ are mutually redundant and therefore they cannot coexist in the spanner after Step (3.3) of the Leapfrog-Spanner algorithm. Thus we have reached a contradiction.

So it must be that

$$t \cdot d(u_k, v_k) < \sum_{i=1}^{k-1} d(u_i, v_i) + \sum_{i=k+1}^{m} d(u_i, v_i) + t \cdot \Big(\sum_{i=1}^{m-1} d(v_i, u_{i+1}) + d(v_m, u_1) \Big).$$

Adding $d(u_k, v_k)$ and subtracting $d(u_1, v_1)$ from both sides of this inequality yields

$$t \cdot d(u_k, v_k) + d(u_k, v_k) - d(u_1, v_1) \leq \sum_{i=2}^{m} d(u_i, v_i) + t \cdot \Big(\sum_{i=1}^{m-1} d(v_i, u_{i+1}) + d(v_m, u_1) \Big). \tag{3.5}$$

Now note that, since $d(u_k, v_k) > \frac{d(u_1, v_1)}{\alpha}$, the quantity $(\frac{t+1}{\alpha} - 1) \cdot d(u_1, v_1)$ is no greater than the left side of the inequality (3.5). Thus we have that

$$t' \cdot d(u_1, v_1) \leq \sum_{i=2}^{m} d(u_i, v_i) + t \cdot \Big(\sum_{i=1}^{m-1} d(v_i, u_{i+1}) + d(v_m, u_1) \Big).$$

for any $1 < t' < \frac{t}{\alpha^2} < \frac{t+1}{\alpha} - 1$. Thus the leapfrog property holds for this case as well. □

An immediate consequence of Theorem 38 is that, if the input graph G resides in an Euclidean metric space of fixed dimension, then $wt(H) = O(wt(MST(G)))$ (cf.

Lemma 34). Arguments similar to the ones used in proving Theorem 33 show that $wt(G^\ell) = O(wt(H))$. Furthermore, it can be verified that the other properties listed in Theorem 33 hold for G^ℓ as well. Thus we have the following result.

Theorem 39. *The* Leapfrog-Spanner *algorithm produces a* $(1+\varepsilon)$*-spanner* G^ℓ *of* G *of degree bounded above by* $(\frac{1}{\varepsilon})^{O(\rho)}$ *and weight bounded above by* $O(\log \Delta) \cdot (\frac{1}{\varepsilon})^{O(\rho)} \cdot wt(MST)$. *If* G *resides in an Euclidean metric space of fixed dimension, then* $wt(G^\ell) = O(wt(MST(G))$.

3.4 Distributed Implementation

In this section, we show that both the Spanner algorithm (Section 3.2) and the Leapfrog-Spanner algorithm (Section 3.3) have distributed implementations that run in $O(\log^* n)$ rounds of communication.

Spanner **Algorithm.** It turns out that Step (1) of this algorithm takes $O(\log^* n)$ rounds, whereas the remaining steps take $O(1)$ additional rounds. We first examine Steps (2)-(9) of the algorithm.

It is easy to verify that in Steps (2)-(9), a node u needs to communicate only with other nodes that are either neighbors of u in G, or to which u is connected by a virtual edge. The main difficulty here is that the endpoints of a virtual edge $\{u, v\}$ may not be neighbors in the network. Consider a virtual edge $\{u, v\} \in E_i$. By definition of E_i, $d(u, v) \leq \gamma \cdot r_i \leq \gamma$. Even though the distance between u and v in the underlying metric space is bounded above by a constant, it is not necessary that the hop distance between u and v in G be similarly bounded above. Let us call

a virtual edge $\{u, v\} \in E_i$, *useful*, if there exist $x \in T(i, u)$ and $y \in T(i, v)$ such that $\{x, y\} \in E$. Notice that only useful virtual edges need to be considered by our algorithm. If a virtual edge $\{u, v\}$ is not useful, then even though it is added in Step (2), it is eliminated in Steps (7)-(9). In the following lemma we show that the hop distance between endpoints of useful virtual edges is small.

Lemma 40. *The hop distance in G between the endpoints of any useful virtual edge $\{u, v\} \in E_i$ is at most $2(\frac{2\alpha}{\alpha-1} + 1)$.*

Proof. By definition of a useful virtual edge, there are points $x \in T(i, u)$ and $y \in T(i, v)$ such that $\{x, y\}$ is an edge in G. Thus a path in G between u and v is the following: start at node (i, u) in the net tree and travel down to a copy of x, follow the edge $\{x, y\}$, and then travel up to node (i, v). Note that the edge $\{x, y\} \in E$, but it may not belong to $\widetilde{E}$. The length of this path is at most $2(1 + \frac{1}{\alpha} + \frac{1}{\alpha^2} + \ldots) + 1$, implying that $d_G(u, v) \leq \frac{2\alpha}{\alpha-1} + 1$. Now consider a shortest uv-path in G, say $\langle w_0 = u, w_1, \ldots, w_{k+1} = v\rangle$. Because G is a UBG and due to the triangle inequality, $d(w_i, w_{i+2}) > 1$ for all $0 \leq i \leq k - 1$ (otherwise w_{i+1} could be eliminated from the uv-path to obtain an even shorter uv-path in G). This yields a lower bound of $k/2$ on $d(u, v)$, and since $d_G(u, v) \geq d(u, v)$, we have that $d_G(u, v) \geq k/2$. Combining this with the upper bound of $\frac{2\alpha}{\alpha-1} + 1$, we obtain that $k \leq 2(\frac{2\alpha}{\alpha-1} + 1)$. □

Thus, in Steps (2)-(9) of the SPANNER algorithm, a node only needs to communicate with nodes that are at most $O(\frac{1}{\alpha-1})$ hops away. This suggests a simple way of implementing Steps (2)-(9): after Step (1) is completed, each node u gathers neigh-

borhood information and the values of $i(v)$ from all nodes v that are $O(\frac{1}{\alpha-1})$ hops away. After this, node u can do all of its computation with no further communication.

The fact that Step (1) can be implemented in $O(\log^* n)$ rounds of communication follows from a clever argument in [51]. Suppose that we have computed the set V_{i-1}. The computation of the set V_i, which is an r_i-net of V_{i-1}, reduces to a maximal independent set (MIS) computation on a degree-bounded graph. To see this, create a graph, say G_i, whose vertex set is V_{i-1} and whose edges connect any pair of vertices $u, v \in V_{i-1}$, if $d(u, v) \leq r_i$. Then it is easy to see that an MIS in G_i is an r_i-net of V_{i-1}. Furthermore, the fact that G_i has bounded degree follows from the fact that the underlying metric space has bounded doubling dimension. There is a well-known deterministic algorithm due to Linial [61] for computing an MIS, that runs in $O(\log^* n)$ communication rounds on graphs with bounded degree. Using this algorithm, one can compute the r_i-net V_i of V_{i-1} in $O(\log^* n)$ rounds. Since there are $h + 1 = O(\log \Delta)$ such sets to compute, it seems like this approach will take $O(\log \Delta \cdot \log^* n)$ rounds. However, in [51] it is shown that in this algorithm, each node uses information only from nodes that are at most $O(\log^* n)$ hops away in G. Therefore, this algorithm has a $O(\log^* n)$-round implementation in which each node u first gathers information from nodes that are at most $O(\log^* n)$ hops away and then performs all steps of the Spanner algorithm locally, using the collected information.

Leapfrog-Spanner **Algorithm.** It is only Steps (2) and (3) of this algorithm that may incur communication cost in addition to the communication cost of the

SPANNER algorithm. We show that the construction of each set E_i in Step (2) of the algorithm takes a constant number of communication rounds. We also show that Step (3) of the algorithm can be implemented in $O(\log^* n)$ rounds of communication. As mentioned before, by collecting information from nodes that are $O(\log^* n)$ hops away, each node can run Step (3) of the LEAPFROG-SPANNER algorithm locally.

In Step (2) of the LEAPFROG-SPANNER algorithm, a node u considers a subset of (real or useful virtual) edges and determines, for each such edge $\{u, v\}$, whether H_{i-1} contains a path from u to v of length no greater than $(1 + \varepsilon) \cdot d(u, v)$. Cf. Lemma 40, node u needs to gather neighborhood information that is $O(2(1+\varepsilon)(\frac{2\alpha}{\alpha-1} + 1))$ hops away and compute this information locally.

In Step (3) of the LEAPFROG-SPANNER algorithm, each node must contribute to the construction of the redundancy graph Γ. To this end, each node u takes charge of all edges $\{u, v\}$ for which the identifier of u is higher than the identifier of v. For each such edge $\{u, v\} \in E_i$, u determines its mutually redundant correspondents $\{u', v'\} \in E_i$, and adds this information to Γ. Arguments similar to the ones above show that the nodes u, u', v and v' are a constant number of hops away from one another in G. We now show the following:

Lemma 41. Γ *is a quasi-UBG that resides in a metric space of constant doubling dimension.*

Proof. Let a and b be vertices in Γ corresponding to edges $\{u_a, v_a\}$ and $\{u_b, v_b\}$ in H. Assign to the vertex pair (a, b) a weight equal to

$$d_\Gamma(a, b) = \min(d_H(u_a, u_b) + d_H(v_a, v_b),\ d_H(u_a, v_b) + d_H(v_a, u_b)).$$

First we show that the weights defined by d_Γ form a metric. Clearly $d_\Gamma(a,a) = 0$ and $d_\Gamma(a,b) = d_\Gamma(b,a)$. To prove the triangle inequality, consider three vertices a, b and c in Γ. Assume w.l.o.g. that

$$\begin{aligned} d_\Gamma(a,b) &= d_H(u_a,u_b) + d_H(v_a,v_b) \\ d_\Gamma(b,c) &= d_H(u_b,u_c) + d_H(v_b,v_c) \end{aligned}$$

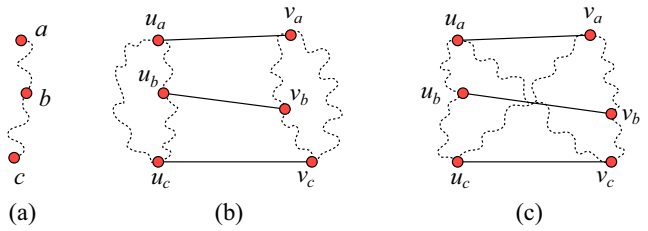


Figure 3.6: d_Γ satisfies the triangle inequality: (a) Nodes a, b, c in Γ corresponding to edges $\{u_a, u_b\}$, $\{u_b, u_c\}$, $\{u_a, u_c\}$ in H. (b) $d_\Gamma(a,c) = d_H(u_a,u_c) + d_H(v_a,v_c)$ (c) $d_\Gamma(a,c) = d_H(u_a,v_c) + d_H(v_a,u_c)$.

We identify two possible scenarios:

(1) $d_\Gamma(a,c) = d_H(u_a,u_c) + d_H(v_a,v_c)$ (see Fig. 3.6b). Since d_H is itself a metric, it follows immediately that $d_\Gamma(a,c) \leq d_\Gamma(a,b) + d_\Gamma(b,c)$.

(2) $d_\Gamma(a,c) = d_H(u_a,v_c) + d_H(v_a,u_c)$ (see Fig. 3.6c). Then it must be that $d_\Gamma(a,c) \leq d_H(u_a,u_c) + d_H(v_a,v_c) \leq d_\Gamma(a,b) + d_\Gamma(b,c)$, cf. scenario (1).

We have shown that d_Γ defines a metric space. We now show that Γ is a quasi-UBG residing in the metric space defined by d_Γ. For each edge $\{a,b\}$ in Γ, the following redundancy conditions hold:

(a) $d_H(u_a, u_b) + d_H(v_b, v_a) \leq (1+\varepsilon) \cdot d(u_a, v_a) - d(u_b, v_b)$

(b) $d_H(u_b, u_a) + d_H(v_a, v_b) \leq (1+\varepsilon) \cdot d(u_b, v_b) - d(u_a, v_a)$

Recall that $\{u_a, v_a\}$ and $\{u_b, v_b\}$ are edges that belong to a same set E_i, for some $i \geq 0$. This implies that their lengths differ by a factor of α at the most: $\gamma \cdot r_{i-1} < d(u_a, v_a) \leq \gamma \cdot \alpha \cdot r_{i-1}$ and $\gamma \cdot r_{i-1} < d(u_b, v_b) \leq \gamma \cdot \alpha \cdot r_{i-1}$. Thus the right hand side of the inequalities (a) and (b) above is a quantity that lies in the interval $((1+\varepsilon)-\alpha)r_{i-1}, ((1+\varepsilon)\alpha - 1)r_{i-1})$. By scaling $((1+\varepsilon)\alpha - 1)r_{i-1}$ to unit distance we can say that Γ is an $\frac{(1+\varepsilon)-\alpha}{(1+\varepsilon)\alpha-1}$ - qUBG in the underlying metric space defined by d_Γ.

It remains to show that the metric space defined by d_Γ has constant doubling dimension. Throughout the rest of the proof we use B_Γ (B_H) to denote a ball in the metric space defined by d_Γ (d_H).

Consider a ball $B_\Gamma(x, R)$ of radius R centered at an arbitrary vertex $x \in \Gamma$. To cover all vertices in $B_\Gamma(x, R)$, do the following repeatedly: (i) pick an uncovered vertex $a \in \Gamma$, (ii) grow a ball $B_\Gamma(a, R/2)$, and (iii) grow two balls $B_H(u_a, R/4)$ and $B_H(v_a, R/4)$ in H (recall that node $a \in \Gamma$ corresponds to edge $\{u_a, v_a\} \in H$).

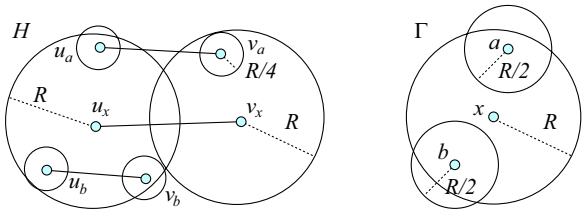


Figure 3.7: The metric space defined by d_Γ has doubling dimension.

Let $a, b \in \Gamma$ be be such that $d_\Gamma(a,b) > R/2$ (see Fig. 3.7). Assume w.l.o.g. that $d_\Gamma(a,b) = d_H(u_a,u_b) + d_H(v_a,v_b)$. Then either $d_H(u_a,u_b) > R/4$, or $d_H(v_a,v_b) > R/4$, or both. Assume w.l.o.g. that $d_H(u_a,u_b) > R/4$. If $d(u_a,u_b) \leq 1$, then there exists $i \geq 0$ such that $d(u_a,u_b) \leq \gamma r_i$ and for which Lemma 35 guarantees that $d(u_a,u_b) \geq d_H(u_a,u_b)/(1+\varepsilon)$. This implies that $d(u_a,u_b) \geq \min\{1, \frac{R}{4(1+\varepsilon)}\}$ and therefore u_a and u_b cannot be too close to each other in the metric space defined by d. This along with the fact that the metric space defined by d has constant doubling dimension implies that only a constant number of such pairs (u_a,u_b) (and therefore quadruples (u_a,u_b,v_a,v_b)) exist in $B_H(u_x,R) \cup B_H(v_x,R)$. This further implies that only a constant number of balls of radii $R/2$ are used to cover $B_\Gamma(x,R)$, proving that the metric space defined by d_Γ has constant doubling dimension. □

Lemma 41 implies that a Maximal Independent Set I of Γ can be constructed in $O(\log^* n)$ communication rounds in Step (2.3) of the Leapfrog-Spanner algorithm, using the MIS algorithm in [51]. In Step (2.4), each node u removes from E_i all incident edges corresponding to nodes not in I. This completes the proof that Step (3) of the Leapfrog-Spanner algorithm takes $O(\log^* n)$ communication rounds.

3.5 Extension to Quasi-Unit Ball Graphs

It can be verified that the Spanner algorithm described at the end of Section 3.2.3 works for β-QUBG as well, for fixed $0 < \beta \leq 1$. Let $\delta = \lceil \log_\alpha \frac{\gamma}{\beta} \rceil$ be the new value of δ to be used by these methods (in place of the value $\lceil \log_\alpha \gamma \rceil$ defined in Section 3.2). Then all properties of H listed in Section 3.2.1 hold unaltered: δ is

used only in the proof of Lemma 28, but since $\delta = \log_\alpha \frac{\gamma}{\beta} > \log_\alpha \gamma$, the lemma holds for the new δ value as well. The Edge Replacement Procedure from Section 3.2.2 is carried out only for edges in E_i, with $i \leq h - \delta - 3$; these are all real edges of length no greater than $\beta/\alpha^3 < \beta$. It follows from the proof of Lemma 30 that such edges get replaced by real edges as well (if edge $\{v, u\}$ gets replaced by edge $\{v, w\}$, then $d(v, w) \leq \beta$). Other proofs in Section 3.2.2 use δ as a constant only and therefore they hold for the new δ value as well. The Virtual Edge Replacement Procedure from Section 3.2.3 is carried out for each edge in J that is not in the input β-QUBG (which may include edges of length in the interval $(\beta, 1]$), producing the spanner G' with the properties listed in Theorem 33.

3.6 Conclusion

The question of whether the leapfrog property for the edge set of a spanner H implies that the total weight of H is bounded above by $O(wt(MST))$ for doubling metric spaces, remains open. Were this question to be positively resolved, the $(1+\varepsilon)$-spanner constructed in this chapter would have not only degree bounded above by a constant, but also low weight bounded above by $O(wt(MST))$. Alternately, constructing a $(1+\varepsilon)$-spanner of bounded degree and $O(wt(MST))$ weight for doubling metric spaces, without relying on the leapfrog property, is an interesting open problem.

CHAPTER 4
FACILITY LOCATION IN UNIT DISK GRAPHS

4.1 Introduction

The widespread use of wireless multi-hop networks such as ad hoc and sensor networks pose numerous algorithmic challenges. In the previous chapters we saw examples of topology control problems. Another such challenge is posed by the need for efficient *clustering* algorithms. Clustering can play a critical role in increasing the performance and lifetime of wireless networks and has been proposed as a way to improve MAC layer protocols (e.g., [32, 82]), higher level routing protocols (e.g., [78, 79, 80]), and energy saving protocols (e.g., [18, 39]). Clustering problems can be modeled as combinatorial or geometric optimization problems of various kinds; the *minimum dominating set (MDS)* problem, the *k-median* problem, etc. are some popular abstractions of the clustering problem. Since wireless networks reside in physical space and since transmission ranges of nodes can be modeled as geometric objects (e.g., disks, spheres, fat objects, etc.), wireless networks can be modeled as geometric graphs, especially as intersection graphs of geometric objects. This has motivated researchers to consider a variety of clustering problems for geometric graphs [77, 5, 10, 2, 20] and attempt to develop efficient distributed algorithms for these. Most of these clustering problems are NP-hard even for fairly simple geometric graphs and this has motivated attempts to design fast distributed *approximation* algorithms. In this chapter, we present the first constant-factor approximation algorithm for the

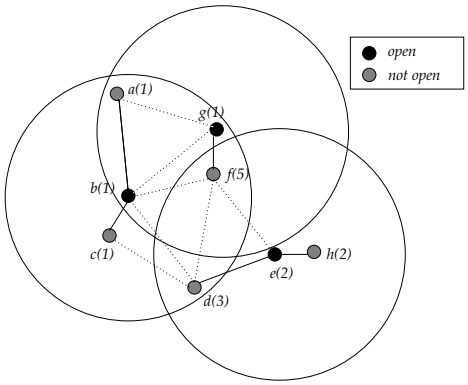


Figure 4.1: A UDG with eight vertices. Opening costs are integers shown next to the vertex names and connection costs of edges are assumed to be Euclidean lengths. Vertices b, g, and e have been opened as facilities. The solid lines indicate the assignments of vertices (clients) to open facilities and the dotted lines indicate edges in the UDG that are not being used for any facility-client connection. Only the disks around the three open facilities are shown in the figure. The cost of this solution is 4 units (for opening facilities) plus $|fg| + |ab| + |cb| + |de| + |he|$.

facility location problem on *unit disk graphs (UDGs)*. For points u and v in Euclidean space we use $|uv|$ to denote the Euclidean distance in L_2 norm between u and v. A graph $G = (V, E)$ is a *unit disk graph (UDG)* if there is an embedding of the vertices of G in $\mathbb{R}^2$ (the 2-dimensional Euclidean space) such that $\{u, v\} \in E$ iff $|uv| \leq 1$. The *facility location problem on UDGs* (in short, `UDG-FacLoc`) takes as input a UDG $G = (V, E)$, *opening costs* $f : V \to \mathbb{R}^+$ associated with vertices, and *connection costs* $c : E \to \mathbb{R}^+$ associated with the edges. The problem is to find a subset $I \subseteq V$ of vertices to open (as "facilities") and a function $\phi : V \to I$ that assigns every vertex ("client") to an open facility *in its neighborhood* in such a way that the total cost of opening the facilities and connecting clients to open facilities is minimized. In other words, the problem seeks to minimize the objective function $\sum_{i \in I} f(i) + \sum_{j \in V} c(j, \phi(j))$. See Fig. 4.1 for an illustration. The opening

cost $f(i)$ reflects the available battery power at node i; less the battery power, greater the cost $f(i)$. The connection cost $c(j, \phi(j))$ represents the power needed for j to communicate with $\phi(j)$. Hence it is safe to assume that the connection costs of edges are determined by their Euclidean lengths via a fairly general function. More precisely, let $g : [0, 1] \to \mathbb{R}^+$ be a monotonically increasing function with *bounded growth*, i.e., for some constant $B \geq 1$, $g(x) \leq B \cdot g(x/3)$ for all $x \in [0, 1]$. We assume that each edge $\{i, j\} \in E$ get assigned a connection cost $c(i, j) = g(|ij|)$. Note that the restriction that g has bounded growth still permits cost functions that are quite general from the point of view of wireless networks. For example, if $g(x) = \beta \cdot x^\gamma$ for constants β and γ (as might be the case if connection costs represent power usage), then $B = 3^\gamma$. It should be noted that every vertex in G is a "client" and every vertex has the potential to be a "facility." Furthermore, a vertex ("client") can only be connected to (i.e., "serviced" by) another vertex ("facility") in its neighborhood and thus the set of open facilities forms a dominating set.

Note that `UDG-FacLoc` is inherently *non-metric*, i.e., connection costs of edges do not satisfy the triangle inequality. This is because a vertex cannot be connected to a non-neighbor, implying that the connection cost of a vertex to a non-neighbor is ∞. There are no known constant-factor approximation algorithms for the non-metric version of facility location, even for UDGs. In one sense, this is not surprising because `UDG-FacLoc` is a generalization of the *weighted minimum dominating set (`WMDS`)* problem on UDGs. This can be seen by noting that an instance of `WMDS`, namely $G = (V, E)$, $w : V \to \mathbb{R}^+$, can be interpreted as a `UDG-FacLoc` instance in

which the connection costs (of edges) are set to 0 and each opening cost $f(i)$ is set to the vertex weight $w(i)$. Unlike the WMDS problem that ignores the cost of connecting to *dominators*, the facility location problem explicitly models *connection costs*. As a result, solutions to WMDS may lead to clustering that is quite poor. There have been no constant-factor approximation algorithms for WMDS on UDGs until recently, with the result of Ambühl et al. [2] being the first constant-factor approximation for WMDS on UDGs. Subsequently, Huang et al. [34] have improved the approximation ratio significantly. Our technique combines the well known *primal-dual* algorithm of Jain and Vazirani [36] with these recent constant-factor approximation algorithms for WMDS on UDGs, to obtain a constant-factor approximation for UDG-FacLoc. Applicability of our technique to more general models of wireless networks, for example, *unit ball graphs* in higher dimensional spaces or *doubling metric spaces*, *disk graphs*, *growth-bounded graphs* etc. is only limited by the availability of good approximation algorithms for the WMDS problem on these graph classes. Using our technique, a constant-factor approximation algorithm for WMDS on any of these graph classes would immediately imply a constant-factor approximation for facility location on that graph class.

UDGs are simple and popular models of wireless networks and the facility location problem on UDGs is a general abstraction of the clustering problem on wireless networks. To be more specific consider one common application of dominating sets in wireless networks, which is to save energy by sending all dominatees into a low power *sleep mode* and having the network be serviced exclusively by the domi-

nators. While it makes sense to keep the size or weight of the dominating set small so that most nodes are in the sleep mode, ignoring the connection costs could yield a dominating set in which each dominator has to spend a lot of energy in order to reach its dominatees. By using an objective function that takes opening costs as well as connection costs into account, `UDG-FacLoc` yields a set of cluster heads that can service the network with smaller overall cost and for a longer duration. For more background see the recent survey by Frank [21] on the facility location problem as it arises in the context of wireless and sensor networks.

4.1.1 Related work

Facility location is an old and well studied problem in operations research ([48, 76, 3, 43, 12]), that arises in contexts such as locating hospitals in a city or locating distribution centers in a region. A *standard* instance of the facility location problem takes as input a complete bipartite graph $G = (F, C, E)$, where F is the set of facilities and C is the set of cities, opening costs $f : F \to \mathbb{R}^+$, and connection costs $c : E \to \mathbb{R}^+$. The goal, as mentioned before, is to find a set of facilities $I \subseteq F$ to open and a function $\phi : C \to I$ that assigns every city to an open facility so as to minimize $\sum_{i \in I} f(i) + \sum_{j \in C} c(j, \phi(j))$. In this context, the connection costs are said to satisfy the *triangle inequality* if for any $i, i' \in F$ and $j, j' \in C$, $c(i, j) \leq c(i, j') + c(i', j') + c(i', j)$. In the *metric facility location* problem the connection costs satisfy the triangle inequality; when they don't we have the more general *non-metric facility location* problem. `UDG-FacLoc` can be seen as an instance of the non-metric

facility location problem by setting $F = V$, $C = V$, setting connection costs between a facility and a city that correspond to non-adjacent vertices to ∞, setting $c(i, i) = 0$ for all $i \in V$, and inheriting the remaining connection costs and opening costs in the natural way. $O(\log n)$-approximation algorithms for the non-metric facility location problem are well known [33, 60]. Starting with an algorithm due to Shmoys, Tardos and Aardal [75] the metric facility location problem has had a series of constant-factor approximation algorithms, each improving on the approximation factor of the previous. We make use of an elegant primal-dual schema algorithm due to Jain and Vazirani [36] that provides a 3-approximation to the metric facility location problem. Since `UDG-FacLoc` is not a metric version of the facility location problem, we cannot use the Jain-Vazirani algorithm directly. We use the Jain-Vazirani algorithm to get a "low cost," but infeasible solution to `UDG-FacLoc` and then "repair" this solution via the use of a "low weight" dominating set and show that the resulting solution is within a constant-factor of `OPT`.

Several researchers have attempted to devise distributed algorithms for the facility location problem; these attempts differ in the restrictions placed on the facility location problem and in the network and distributed computing models. For example, Moscibroda and Wattenhofer [65] present a distributed algorithm for the standard non-metric facility location problem. The network on which their algorithm runs is the complete bipartite graph on F, the set of facilities and C, the set of cities. Since this network has diameter 2, one way to solve the problem would be for a node to gather information about the entire network in constant number of communication rounds

and just run a known sequential algorithm locally. Thus this problem is uninteresting in the $\mathcal{LOCAL}$ model [71] of distributed computation. The problem becomes more interesting in the $\mathcal{CONGEST}$ model, where a reasonable bound, such as $O(\log n)$ bits, is imposed on each message size. The message size of $O(\log n)$ bits allows each message to contain at most a constant number of node identifiers and constants that are polynomial in n. In such a model, exchanging a lot of information costs a lot of rounds and Moscibroda and Wattenhofer [65] present an approximation algorithm for non-metric facility location that, for every k, achieves an $O(\sqrt{k}(m\rho)^{1/\sqrt{k}}\log(m+n))$-approximation in $O(k)$ communication rounds. Here m is the number of facilities, n is the number of clients, and ρ is a coefficient that depends on the numbers (i.e., opening costs and connection costs) that are part of the input. The main thrust of this result is that even with a constant number of communication rounds, a non-trivial approximation factor can be achieved. However, it should be noted that no matter how large k is (e.g., $k = \text{polylog}(n)$), the approximation factor of this algorithm is $\Omega(\log(m+n))$.

Frank and Römer [22] consider facility location on multi-hop networks (like we do), but assume that given edge weights, the connection cost $c(i,j)$ for any pair of vertices i and j is simply the shortest path distance between i and j. This turns their problem into a *metric* problem and thus they can use known sequential algorithms; in particular, they use the 1.61-approximation due to Jain et al. [35]. Frank and Römer [22] show how to implement the sequential algorithm of Jain et al. [35] in a distributed setting without any degradation in the approximation factor, but they do

not provide any non-trivial running time guarantees. These authors [22] do mention the version of the problem in which connection costs between non-neighboring vertices is ∞, but they just observe that since this is a non-metric problem, constant-factor approximation algorithms are not known.

Gehweiler et al. [23] present a constant-approximation, constant-round distributed algorithm using only $O(\log n)$-bits per message, for the *uniform* facility location problem. In this problem, all opening costs are identical and the underlying network is a clique. The authors make critical use of the fact that all facility opening costs are identical in order to obtain the constant-approximation. The uniform opening costs assumption is restrictive for certain settings. For example, if we want opening costs to reflect the amount of battery power that nodes have available – more the available power at a node, cheaper it is to open that node, then this assumption requires the battery power at all nodes to remain identical through the life of the network. This may be untenable because nodes will tend to expend different amounts of power as they perform different activities. The interesting aspect of the Gehweiler et al. [23] algorithm is that all message sizes are bounded above by $O(\log n)$.

4.1.2 Main results

We assume that we are given a UDG along with its geometric representation. Let $g : [0, 1] \to \mathbb{R}^+$ be a monotonically increasing function with *bounded growth*, i.e., there exists a constant B such that $g(x) \leq B \cdot g(x/3)$ for all $x \in [0, 1]$. Each edge $\{i, j\} \in E$ gets assigned a connection cost $c(i, j) = g(|ij|)$, representing the

dependence of the connection cost on the Euclidean distance between the involved vertices. For any $\varepsilon > 0$, we present a $(6 + B + \varepsilon)$-approximation algorithm for `UDG-FacLoc`. To put this result in context, observe that if connection costs are exactly Euclidean distances, i.e., $g(x) = x$, then $B = 3$ and we have a $(9+\varepsilon)$-approximation. If the connection costs are meant to represent energy usage, then a function such as $g(x) = \beta \cdot x^{\gamma}$ for constants β and $2 \leq \gamma \leq 4$ may be reasonable. In this case, $B = 3^{\gamma}$ and we get a $(3^{\gamma} + 6 + \varepsilon)$-approximation, still a constant-factor approximation. We then present a distributed implementation of our algorithm that runs in just $O(1)$ rounds and yields an $O(B)$-approximation.

In the $\mathcal{LOCAL}$ model, every node $v \in V$ can send an arbitrarily large message to ever neighbor u in each round of communication. This model abstracts away all the restricting factors (e.g. congestion and asynchronicity) and focus on the impact of *locality* on distributed computation. To obtain this result we show that `UDG-FacLoc` can be solved "locally" with only a constant-factor degradation in the quality of the solution. One aspect of our result, namely the constant approximation factor, depends crucially on the availability of a geometric representation of the input UDG. If we are given only a combinatorial representation of the input n-vertex UDG, then our algorithm runs in $O(\log^* n)$ rounds yielding an $O(\log n)$-approximation. This result depends on two recent results: (i) an $O(\log^* n)$-round algorithm for computing a *maximal independent set (MIS)* in growth-bounded graphs [74] and (ii) an algorithm that partitions a UDG, given without geometry, into relatively small number of cliques [72]. Overall, our results indicate that `UDG-FacLoc` is as "local" a problem as MIS is,

provided one is willing to tolerate a constant-factor approximation. Our techniques extend in a straightforward manner to the *connected* UDG-FacLoc problem, where it is required that the facilities induce a connected subgraph; we obtain an $O(1)$-round, $O(B)$-approximation for this problem also.

4.2 Sequential Algorithm

Now we present a high level *three step* description of our algorithm for finding a constant-factor approximation for UDG-FacLoc. Let $G = (V, E)$ be the given UDG with an opening cost $f(i)$ for each vertex $i \in V$ and connection cost $c(i, j)$ for each edge $\{i, j\} \in E$. We assume that there is a monotonically increasing function $g : [0, 1] \to \mathbb{R}^+$ satisfying $g(x) \leq B \cdot g(x/3)$ for all $x \in [0, 1]$ for some $B \geq 1$, such that $c(i, j) = g(|ij|)$.

Step 1. Convert the given instance of UDG-FacLoc into a standard non-metric instance of facility location. This transformation is as described in the previous section. Run the primal-dual algorithm of Jain and Vazirani [36] on this instance to obtain a solution S. The solution S may contain connections that are infeasible for UDG-FacLoc; these connections have connection cost ∞ and they connect pairs of non-adjacent vertices in G.

Step 2. Assign to each vertex i of G a weight equal to $f(i)$. Compute a dominating set of G with small weight. For this we can use the $(6 + \varepsilon)$-approximation algorithm due to Huang et al. [34]. Let D^* denote the resulting solution.

Step 3. For each vertex $i \in V$ that is connected to a facility by an edge of cost ∞,

reconnect i to an arbitrarily chosen neighbor $d \in D^*$. Think of the vertices $d \in D^*$ as facilities and declare them all open. Let the new solution to `UDG-FacLoc` be called S^*.

We will prove the following theorem in the next subsection.

Theorem 42. *Let OPT denote the cost of an optimal solution to a given instance of `UDG-FacLoc`. Then $cost(S^*) \leq (6 + B + \varepsilon) \cdot OPT$.*

4.2.1 Analysis

To analyze our algorithm we need some details of the Jain-Vazirani primal-dual algorithm used in Step 1. For a more complete description see [36]. The starting point of this algorithm is the following Integer Program (IP) representation of facility location. Here y_i indicates whether facility i is open and x_{ij} indicates if city j is connected to facility i. The first set of constraints ensure that every city is connected to a facility and the second set of constraints guarantee that each city is connected to an open facility.

$$\begin{aligned}
\text{minimize} \quad & \sum_{i \in F, j \in C} c(i,j) \cdot x_{ij} + \sum_{i \in F} f(i) \cdot y_i & \\
\text{subject to} \quad & \sum_{i \in F} x_{ij} \geq 1, & j \in C \\
& y_i - x_{ij} \geq 0, & i \in F, j \in C \\
& x_{ij} \in \{0,1\}, & i \in F, j \in C \\
& y_i \in \{0,1\}, & i \in F
\end{aligned}$$

As is standard, we work with the LP-relaxation of the above IP obtained by replacing the integrality constraints by $x_{ij} \geq 0$ for all $i \in F$ and $j \in C$ and $y_i \geq 0$ for all $i \in F$. The dual of this LP-relaxation is the following:

$$\begin{aligned}
\text{maximize} \quad & \sum_{j \in C} \alpha_j & \\
\text{subject to} \quad & \alpha_j - \beta_{ij} \leq c(i,j), & i \in F,\, j \in C \\
& \sum_{j \in C} \beta_{ij} \leq f(i), & i \in F \\
& \alpha_j \geq 0, & j \in C \\
& \beta_{ij} \geq 0, & i \in F,\, j \in C
\end{aligned}$$

The dual variable α_j can be interpreted as the amount that city j is willing to pay in order to connect to a facility. Of this amount, $c(i,j)$ goes towards paying for connecting to facility i, whereas the "extra," namely β_{ij}, is seen as the contribution of city j towards opening facility i. Initially all the α_j and β_{ij} values are 0. The Jain-Vazirani algorithm initially raises all of the α_j values in sync. When α_j reaches $c(i,j)$ for some edge $\{i,j\}$, then the connection cost $c(i,j)$ has been paid for by j and any subsequent increase in α_j is accompanied by a corresponding increase in β_{ij} so that the first dual constraint is not violated. The quantity β_{ij} is j's contribution towards opening facility i and when there is enough contribution, i.e., $\sum_j \beta_{ij} = f(i)$, then the facility i is declared *temporarily open*. Furthermore, all unconnected cities j that make positive contribution towards $f(i)$, i.e., $\beta_{ij} > 0$, are declared *connected* to i. Also, any unconnected city j that has completely paid its connection cost $c(i,j)$, but has not yet started paying towards β_{ij}, i.e., $\beta_{ij} = 0$, is also declared *connected*

to j. The opening of a facility i corresponds to setting $y_i = 1$ and declaring a city j connected to i corresponds to setting $x_{ij} = 1$. Once a facility i is open and cities connected to it, then the dual variables of these cities are no longer raised; otherwise the dual constraint $\sum_{j \in C} \beta_{ij} \leq f(i)$ would be violated. The algorithm proceeds in this way until every city has been connected to some open facility. This is the end of Phase 1 of the algorithm.

It is easy to check that at the end of Phase 1, $\{\alpha_j, \beta_{ij}\}$ define a feasible dual solution and $\{y_i, x_{ij}\}$ define a feasible *integral* solution. If the cost of the primal solution is not too large compared to the cost of the dual solution, then by the Weak Duality Theorem, we would have a solution to facility location that is not too far from a lower bound on OPT. However, the gap between the costs of the dual and the primal solutions can be quite high because a single city may be contributing towards the connection costs and opening costs of many facilities. To fix this problem, Phase 2 of the algorithm is run. Let F_t be the set of temporarily open facilities. Define a graph H on this set of vertices with edges $\{i, i'\}$ whenever there is a city j such that $\beta_{ij} > 0$ and $\beta_{i'j} > 0$; in other words, city j is contributing a positive amount towards the opening of both facilities i and i'. Compute a *maximal independent set (MIS) I* of H and declare all facilities in I open (permanently) and close down all facilities in $F_t \setminus I$, i.e., set $y_i = 0$ for all $i \in F_t \setminus I$. Due to the shutting down of some facilities, some cities may be connected to closed facilities implying that the primal solution may be infeasible, due to violation of the $y_i - x_{ij} \geq 0$ constraints. Call a city j a *Class I city* if it is connected to an open facility. Denote the set of Class I cities by

C_1. We will call cities outside of C_1, *Class II cities*. At this point in the algorithm the primal and the dual solution satisfy the following properties.

Lemma 43. ***[Jain-Vazirani [36]]*** *The dual solution $\{\alpha_j, \beta_{ij}\}$ is feasible. The primal solution $\{y_i, x_{ij}\}$ is integral, but may not be feasible. Furthermore,*

$$\sum_{j \in C_1} \alpha_j = \sum_{j \in C_1} c(j, \phi(j)) + \sum_{i \in I} f(i).$$

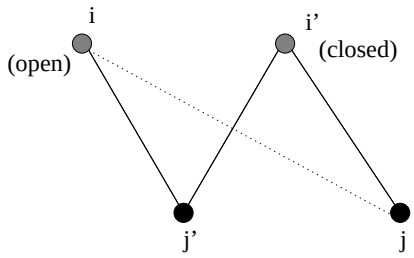


Figure 4.2: Client j is connected to temporarily open facility i at the end of Phase 1. Client j' contributes positively to the opening cost of both i and i'. Facility i' is closed at the beginning of Phase 2 and facility i becomes a candidate for connecting j to.

The above Lemma is essentially saying that the Class I cities completely pay for connections to and the opening of facilities in I. The goal now is to fix the infeasibility of the primal solution, i.e., find connections for cities outside C_1, without increasing the cost of the primal solution too much relative to the cost of the dual. Let j be a city that is connected to a closed facility. If there is a open city i to which j has already paid connection cost, i.e., $\alpha_j \geq c(i, j)$, then simply connect j to one

such city. Since $\alpha_j \geq c(i,j)$, the connection cost is paid for by α_j and furthermore the opening cost of i has been paid for by other cities. This leaves a set C' of cities such that for each $j \in C'$, $\alpha_j < c(i,j)$ for all open cities i. This may happen, for example, if none of j's neighbors in G have been opened as facilities and therefore for every open facility i, $c(i,j) = \infty$. Note that at the end of Phase 1, there was a temporarily open facility, say i, to which j was connected and in Phase 2, i was shut down. This implies that (i) $\alpha_j \geq c(i,j)$ and (ii) there exists a city j' that is paying a positive amount towards the opening of two facilities i and i' and this "double payment" is responsible for i being shut down. See Fig. 4.2 for an illustration. In such a case, the Jain-Vazirani algorithm simply connects j to i'. In the *metric* facility location case, Jain and Vazirani are able to show that the connection cost $c(j,i')$ is not too big relative to $c(i,j)$ (they show, $c(j,i') \leq 3 \cdot c(i,j)$). In our case, i' may be outside the neighborhood of j and therefore $c(j,i') = \infty$ and therefore connecting j to i' is too costly. This possible mistake is fixed in the subsequent two steps of our algorithm, via the use of a `WMDS` solution. We now include the last two steps of our algorithm in the analysis to show that we are able to find a facility that is not too costly for i to connect to. More precisely, if j' is in the neighborhood of i, then $c(i,j') < \infty$ and we are able to show that connecting i to j' is a good idea. On the other hand, if j' is not a neighbor of i, then we show that connecting i to some neighbor in the `WMDS` solution D^* will not increase the cost of the solution too much. We make use of the following inequalities that Jain and Vazirani prove. The first inequality was mentioned earlier in this paragraph, but the remaining two inequalities take a little

bit of work to prove and we refer the reader to the Jain-Vazirani paper [36].

Lemma 44. ***[Jain-Vazirani [36]]*** *$\alpha_j \geq c(i,j)$, $\alpha_j \geq c(i,j')$ and $\alpha_j \geq c(i',j')$.*

Lemma 45. *Let B satisfy $g(x) \leq B \cdot g(x/3)$ for all $x \in [0,1]$. If j' is a neighbor of i, then i is connected to j' in Step 1 and $c(i,j') \leq B \cdot \alpha_j$. If j' is not a neighbor of i, then i is connected to some neighbor $j^* \in D^*$ in Step 3 and $c(i,j^*) \leq B \cdot \alpha_j$.*

Proof. Since Euclidean distances satisfy triangle inequality, we have

$$|ij| + |i'j| + |i'j'| \geq |ij'|.$$

Let y denote the largest of the three terms on the left hand side above. Then $y \geq |ij'|/3$. Suppose that j' is a neighbor of i. Then $|ij'| \leq 1$ and $c(i,j') = g(|ij'|) < \infty$. Then,

$$\begin{aligned} c(i,j') = g(|ij'|) &\leq B \cdot g\left(\frac{|ij'|}{3}\right) \qquad \text{(due to bounded growth of } g\text{)} \\ &\leq B \cdot g(y) \qquad \text{(due to monotonicity of } g\text{)} \\ &\leq B \cdot \alpha_j \qquad \text{(due to Lemma 44).} \end{aligned}$$

Now suppose that j' is not a neighbor of i. Then $|ij'| > 1$ and for any neighbor j^* of i, $|ij'| > |ij^*|$. Since $y \geq |i'j|/3$, it follows that $y > |ij^*|/3$. Then, by the same reasoning as above, we get

$$\begin{aligned} c(i,j^*) = g(|ij^*|) &\leq B \cdot g\left(\frac{|ij^*|}{3}\right) \\ &\leq B \cdot g(y) \\ &\leq B \cdot \alpha_j. \end{aligned}$$

□

Lemma 46. *Let S^* be the solution produced by our algorithm. Then, $cost(S^*) \leq (6 + B + \varepsilon) \cdot OPT$, where OPT is the cost of an optimal solution to* `UDG-FacLoc`.

Proof. The cost of the entire solution can be expressed as

$$\left(\sum_{i \in I} f(i) + \sum_{j \in C_1} c(\phi(j), j) \right) + \left(\sum_{i \in D^*} f(i) + \sum_{j \in C_2} c(\phi(j), j) \right).$$

By Lemma 43, the first term in the above sum equals $\sum_{j \in C_1} \alpha_j$. Let OPT_{DOM} denote the weight of an optimal dominating set when each vertex i of G is assigned weight $f(i)$. Then,

$$\sum_{i \in D^*} f(i) \leq (6 + \varepsilon) \cdot OPT_{DOM} \leq (6 + \varepsilon) \cdot OPT \tag{4.1}$$

because we use the $(6+\varepsilon)$-approximation algorithms of Huang et al. [34] to compute a dominating set of small weight. Also, by Lemma 45,

$$\sum_{j \in C_2} c(\phi(j), j) \leq B \cdot \sum_{j \in C_2} \alpha_j. \tag{4.2}$$

Together the above inequalities yield

$$\begin{aligned} cost(S^*) &= \left(\sum_{i \in I} f(i) + \sum_{j \in C_1} c(\phi(j), j) \right) + \left(\sum_{i \in D^*} f(i) + \sum_{j \in C_2} c(\phi(j), j) \right) \\ &\leq \sum_{j \in C_1} \alpha_j + (6 + \varepsilon) \cdot OPT + B \cdot \sum_{j \in C_2} \alpha_j \\ &\leq B \cdot \sum_{j \in C} \alpha_j + (6 + \varepsilon) \cdot OPT \qquad (\text{since } B \geq 1) \\ &\leq (B + 6 + \varepsilon) \cdot OPT \qquad (\text{by Weak Duality Theorem}). \end{aligned}$$

□

By making simple modifications to an example due to Jain and Vazirani ([36]), we can show that the above analysis is tight in the sense that there exists a UDG

along with an assignment of opening and connection costs for which our algorithm produces a solution with cost at least $B \cdot OPT$, independent of the quality of the WMDS used.

4.3 Distributed Algorithm

In this section, we present an $O(1)$-round distributed implementation of the above algorithm in the $\mathcal{LOCAL}$ model [71]. In this model there is no upper bound placed on the message size and due to this, a node can collect all possible information (i.e., node IDs, topology, interactions) about its k-neighborhood in k communication rounds. We show in this section that `UDG-FacLoc` is inherently a "local" problem provided we are willing to tolerate a constant-factor approximation in the cost of the solution. This property of `UDG-FacLoc` allows us to solve a version of the problem independently on *small squares* and combine the solutions in a simple way to get the overall solution. We partition the plane into squares by placing on the plane an infinite grid of $1/\sqrt{2} \times 1/\sqrt{2}$ squares. This is a standard and simple way of partitioning a UDG with geometric representation into cliques. The square S_{ij} for $i, j \in \mathbb{Z}$, contains all the points (x, y) with $\frac{i}{\sqrt{2}} \leq x < \frac{i+1}{\sqrt{2}}$ and $\frac{j}{\sqrt{2}} \leq y < \frac{j+1}{\sqrt{2}}$. Let $G = (V, E)$ be the given UDG. For a square S_{ij} that has at least one node in V, let $V_{ij} \subseteq V$ be the set of vertices whose centers lie in S_{ij}. Let $N(V_{ij})$ denote the set of all vertices in $V \setminus V_{ij}$ that are adjacent to some vertex in V_{ij}. Now consider the subproblem, denoted $\texttt{UDG-FacLoc}_{ij}$, in which we are allowed to open facilities from the set $V_{ij} \cup N(V_{ij})$ with the aim of connecting all the nodes in V_{ij} as clients to these facilities. The objective

function of the problem remains the same: minimize the cost of opening facilities plus the connection costs.

Let $\{F_{ij}, \phi_{ij}\}$ denote a solution to $\texttt{UDG-FacLoc}_{ij}$, where $F_{ij} \subseteq V_{ij} \cup N(V_{ij})$ is the set of open facilities and $\phi_{ij} : V_{ij} \to F_{ij}$ is the assignment of clients to open facilities. Let $\cup_{ij}\{F_{ij}, \phi_{ij}\}$ denote a solution to `UDG-FacLoc` in which the set of open facilities is $\cup_{ij} F_{ij}$ and the assignment $\phi : V \to \cup_{ij} F_{ij}$ is defined by $\phi(v) = \phi_{ij}(v)$ if $v \in V_{ij}$. Thus $\cup_{ij}\{F_{ij}, \phi_{ij}\}$ defines a simple way of combining solutions of $\texttt{UDG-FacLoc}_{ij}$ to obtain a solution of `UDG-FacLoc`. The following Lemma shows that if the small square solutions $\cup_{ij}\{F_{ij}, \phi_{ij}\}$ are good then combining them in this simple way yields a solution to `UDG-FacLoc` that is also quite good. This Lemma is a generalization of a result due to Ambühl et al. [2] that was proved in the context of the `WMDS` problem for UDGs. The proof of this Lemma is omitted due to space constraints, but it appears in [67].

Lemma 47. *For each $i, j \in \mathbb{Z}$, let OPT_{ij} denote the cost of an optimal solution to $\texttt{UDG-FacLoc}_{ij}$ and let $\{F_{ij}, \phi_{ij}\}$ be a solution to $\texttt{UDG-FacLoc}_{ij}$ such that for some c, $cost(\{F_{ij}, \phi_{ij}\}) \leq c \cdot OPT_{ij}$. Then $cost(\cup_{ij}\{F_{ij}, \phi_{ij}\}) \leq 16c \cdot OPT$. Here OPT is the cost of an optimal solution to `UDG-FacLoc`.*

The above Lemma implies the following simple distributed algorithm.

Step 1. Each node v gathers information (i.e., coordinates of nodes, opening costs of nodes, and connection costs of edges) about the subgraph induced by its 2-neighborhood.

Step 2. Each node v in S_{ij} then identifies V_{ij} and $N(V_{ij})$. Recall that $V_{ij} \subseteq V$ is the

set of nodes that belong to square S_{ij} and $N(V_{ij}) \subseteq V \setminus V_{ij}$ is the set of nodes outside of V_{ij} that have at least one neighbor in V_{ij}.

Step 3. Each node v locally computes the solution of $\texttt{UDG-FacLoc}_{ij}$, thereby determining whether it should be opened as a facility and if not which neighboring facility it should connect to.

Based on the above description, it is easily verified that the algorithm takes 2 rounds of communication. Note that the instance of $\texttt{UDG-FacLoc}_{ij}$ solved in Step 3 is slightly different from UDG-FacLoc, in that only certain vertices (namely, the vertices in V_{ij}) need to connect to open facilities, whereas every vertex (both in V_{ij} and in $N(V_{ij})$) is a potential client. This difference is minor and the $(6 + B + \varepsilon)$-approximation algorithm described in the previous section, can be essentially used without any changes, to solve $\texttt{UDG-FacLoc}_{ij}$. Lemma 47 then implies that the distributed algorithm above would yield a $16 \cdot (6 + B + \varepsilon)$-approximation algorithm. We can do better by making use of an intermediate result due to Ambühl et al. [2] that presents a 2-approximation algorithm for the WMDS problem on each square S_{ij}. Using arguments from the previous section, we can use this to obtain a $(B + 2)$-approximation for $\texttt{UDG-FacLoc}_{ij}$ and a $16 \cdot (B + 2)$-approximation for UDG-FacLoc.

4.4 Conclusion

One open question implied by this work is whether we can obtain a constant-factor approximation algorithm for facility location on more general classes of wireless network models. We believe that a first step towards solving this problem would

be to obtain a constant-factor approximation algorithm for `UDG-FacLoc` when the input UDG is given without any geometry. The only obstacle to obtaining such an approximation, using our techniques, is the lack of a constant-factor approximation to `WMDS` on UDGs given without geometry. Without geometry, the best algorithm we could design is one that, in $O(\log^* n)$ rounds, outputs an $O(\log n)$-approximation to the `UDG-FacLoc` problem. That algorithm appears in the full version of the ICDCN paper [67].

Furthermore, the distributed algorithm we present runs in $O(1)$ rounds in the $\mathcal{LOCAL}$ model, which assumes that message sizes are unbounded. We wanted to extend our distributed algorithm to the $\mathcal{CONGEST}$ model where only messages of size $O(\log n)$ are allowed. The results in the next chapter describes our findings in that area.

CHAPTER 5
A PRIMAL-DUAL APPROACH FOR METRIC FACILITY LOCATION

5.1 Introduction

Recent research in the area of distributed approximation algorithms [19, 44, 49, 50, 54, 55, 65, 23] has led to interesting and sometimes optimal trade-offs between the amount of resources used (e.g., number of communication rounds, size of messages, etc.) and the quality of solution (i.e., approximation factor) obtained. In recent years, such trade-offs have been especially well-studied for the minimum spanning tree problem [44] and the dominating set problem [50, 54, 55]. One theme in this research is motivated by the question: can one design distributed approximation algorithms that provide non-trivial approximation guarantees even when run for very few (e.g., constant) number of rounds? The earliest example of such an algorithm, as far as we know, is the dominating set algorithm due to Kuhn and Wattenhofer [54] that runs in k^2 rounds, for any k, and outputs a dominating set whose expected size is within $O(k^2\Delta^{2/k}\log\Delta)$ of OPT. Here Δ is the maximum degree of the network. In this paper, we investigate this question for the *metric facility location* problem. For instances with m facilities and n clients, we first present a 7-approximation algorithm running in $O(\log m + \log n)$ rounds and then generalize this algorithm to a k-round algorithm that yields an $O(m^{2/\sqrt{k}} \cdot n^{2/\sqrt{k}})$-approximation, for any constant k. In fact, the k-round algorithm yields this approximation factor for all $k = O((\frac{\log m}{\log\log m})^2)$. A key constraint of our model of distributed computation is that message sizes are

bounded above by $O(\log(m+n))$ bits.

Although we have already defined the facility location problem (from the UDG point of view) in previous chapters, it is worth revisiting the traditional version of the problem for this chapter. The traditional facility location problem takes as input a complete bipartite graph $G = (F, C, E)$, where F is the set of *facilities* and C is the set of *clients* (or *cities*), facility opening costs $f : F \to \mathbb{R}^+$, and connection costs $c : E \to \mathbb{R}^+$. As usual, the goal is to find a set of facilities $I \subseteq F$ to open and a function $\phi : C \to I$ that assigns every client to an open facility so as to minimize $\sum_{i \in I} f(i) + \sum_{j \in C} c(j, \phi(j))$. In other words, the goal is to minimize the total cost of opening facilities and connecting clients to open facilities. This version of the problem is old and well studied in operations research [12], that arises in contexts such as locating hospitals in a city or locating distribution centers in a region. However, this problem has also been recently used as an abstraction for the problem of locating resources in wireless networks [21, 68]. The facility location problem comes in two main versions: the *non-metric* version and the *metric* version. In Chapter 4 we have seen the *metric facility location* problem. In the metric version, the connection costs satisfy the triangle inequality; when they don't, we have the more general *non-metric facility location* problem. See Figure 5.1 for an illustration. This distinction is important from an approximation point of view because there are a number of sequential constant-factor approximation algorithms for the metric facility location problem ([9, 36, 75] are some examples), whereas for the non-metric facility location problem, the best known approximation factor is $O(\log n)$ and this is optimal [33, 60].

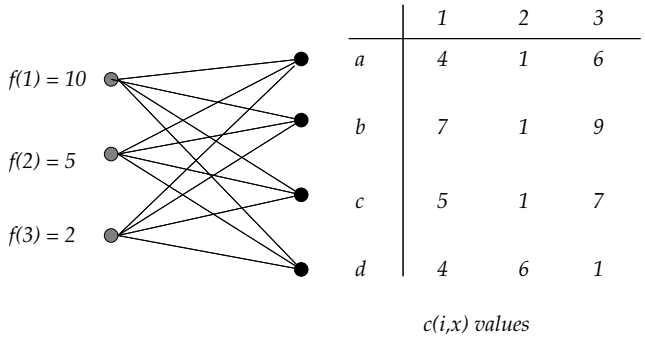


Figure 5.1: An instance of standard `FacLoc`. The table shows the pairwise connection costs between clients and facilities. `OPT` consists of open facilities 2 and 3 with clients a, b and c connected to facility 2 and client d to facility 3. Total cost of `OPT` is 11. Note that any solution with a single open facility or with all the facilities open, will have cost more than 11. So is the case for any solution that opens facility 1.

This work is partly motivated by the results of Moscibroda and Wattenhofer [65] who consider the *non-metric* facility location problem and design an approximation algorithm that in $O(k)$ rounds, for any constant k, yields a non-trivial approximation factor that depends on k and the input size. The underlying network on which this algorithm runs is the complete bipartite graph $G = (F, C, E)$, induced by the facilities and the clients. Given that the network has diameter 2, the problem is trivial (i.e., simply reduces to the sequential setting) if arbitrary amounts of information are allowed to be exchanged between neighbors in each round. Placing a reasonable bound, such as $O(\log(m + n))$, on the message sizes makes the problem challenging and allows Moscibroda and Wattenhofer [65] to highlight an interesting trade-off between the number of communication rounds of the algorithm and the approximation factor achieved. Specifically, they present an algorithm that runs in $O(k)$ rounds and yields an approximation of $O(\sqrt{k}(m\rho)^{1/\sqrt{k}} \log(m + n))$, where m is the

number of facilities, n is the number of clients, and ρ is a coefficient that depends on the numbers in the given instance (i.e., the opening costs and the connection costs). To focus more clearly on the growth of this approximation factor as a function of k, let us assume that $\rho = m = n$. Assuming that ρ is equal to m or n is reasonable since Moscibroda and Wattenhofer [65] assume that ρ can be stored in a message of size $O(\log n)$ bits, implying that the value of ρ is bounded above by a polynomial in n. With this simplification we see that the Moscibroda-Wattenhofer algorithm runs in $O(k)$ rounds yielding an approximation factor of $O(\sqrt{k}n^{2/\sqrt{k}}\log(n))$. Since the metric facility location problem is a special case of the non-metric problem, this leads to question of whether it is possible to design k-round algorithms for *metric* problem that give a better approximation factor.

This question is motivated by the expectation that in a distributed setting such as this, in which network diameter is not a concern (since it is just a constant), one should be able to obtain, in polylogarithmic number of rounds, approximation factors that are as good as the best approximation factors possible in a sequential setting. Since the metric facility location problem has a sequential constant-factor approximation, ideally we should be able to obtain an approximation factor $f(m, n, k)$ that tends to a constant as k tends to some polylogarithmic quantity in m and n. The Moscibroda-Wattenhofer [65] falls short in this regard because no matter how large we make k, the smallest value achieved by their approximation factor of $O(\sqrt{k}n^{2/\sqrt{k}}\log(n))$ is just $O(\log^2 n)$. At a high level, the reason for this is that the Moscibroda-Wattenhofer algorithm [65] uses a 2-step approach that involves solving

an LP-relaxation in the first step and doing an independent randomized rounding of the fractional solution, in the second step; unfortunately, in both of these steps there is a logarithmic "overhead" in the approximation factor. These overheads seem fairly fundamental to their approach and to do better, it seems that different techniques are needed.

Our results. Our algorithms use a model of distributed computing that is similar to the Moscibroda-Wattenhofer model [65]. Specifically, we assume that the underlying network is the complete bipartite graph $G = (F, C, E)$ induced by facilities and clients. Also, the network executes algorithms synchronously and in each round nodes receive messages from neighbors, perform polynomial-time local computations, and send $O(\log(m+n))$-sized messages to neighbors. This message passing model with a logarithmic bound (in the network size) on the message size has been called the $\mathcal{CONGEST}$ model by Peleg [71]. The $\mathcal{CONGEST}$ model is a "point-to-point" communication model and allows a node to send a different $O(\log n)$-bit message to each of its neighbors, but in our algorithm it suffices for each node to send the same $O(\log n)$-sized message to *all* of its neighbors. This is the *local broadcast* model that is commonly used in distributed algorithms on wireless networks. To focus more clearly of the challenges of using the $\mathcal{CONGEST}$ model, we assume that each piece of information that a node possesses (IDs, f_i's, c_{ij}'s, etc.) can fit into an $O(\log(m+n))$-bit message. We consider the metric facility location problem in this model and present two results: (i) a 7-approximation running in $O(\log m + \log n)$ rounds and (ii) a k-round algorithm that yields an $O(m^{2/\sqrt{k}} \cdot n^{2/\sqrt{k}})$-approximation.

These results answer a question on fast distributed approximation algorithms for the metric facility location problem, posed by Moscibroda and Wattenhofer [65]. Our techniques are based on the Jain-Vazirani primal-dual algorithm [36] and a rapid, distributed, randomized algorithm for sparsification of graphs due to Gfeller and Vicari [24]. In fact the success of the primal-dual approach in this setting contradicts the claim by Moscibroda and Wattenhofer [65] that the Jain-Vazirani primal-dual approach works in the distributed setting only if "either message-size is unbounded, or the algorithm's time-complexity depends on the size of the problem instance."

Another motivation for our work comes from the elegant *constant-round*, constant-factor approximation algorithm for the *uniform* metric facility location problem by Gehweiler et al. [23]. The adjective "uniform" in the title refers to the assumption that all facility opening costs are identical. The Gehweiler et al. [23] algorithm also assumes the $\mathcal{CONGEST}$ model and further assumes that the underlying network is a clique, with each node being a client as well as a possible location for a facility. The results in this paper are partly motivated by the desire to extend the Gehweiler et al. algorithm [23] to instances that allow arbitrary facility opening costs. While we describe our algorithm for a bipartite network, the algorithm works more or less unchanged, in the clique setting of Gehweiler et al. [23].

Organization. The rest of the chapter is organized in three sections. Section 5.2 is devoted to our constant-factor Logarithmic-round algorithm, Section 5.3 is devoted to our k-round algorithm, and Section 5.4 contains a brief discussion of related open

problems and future work.

5.2 Logarithmic-round Algorithm

For the remainder of this chapter, let $m = |F|$, $n = |C|$, and $N = m + n$. This section is devoted to our $O(\log n + \log m)$-round, 7-approximation algorithm. This algorithm will set the stage for the more complicated k-round approximation algorithm described in the next section. We assume that each message can be $O(\log N)$ bits long and that all node IDs can be represented in $\log N$ bits. Note that in a complete bipartite graph, every node can learn the total number of facilities and clients in two rounds of communication. So for all algorithms described in this paper, we assume that all nodes know m and n. As the formulation of the primal and dual programs and their relaxations, as well as the idea behind the primal-dual approach have already been described in Chapter 4, we move on to the algorithm. Unless otherwise mentioned, the related notations (primal and dual variables, constants etc.) are identical to Chapter 4.

5.2.1 Overview of Algorithm

The algorithm consists of the three phases, each of which is overviewed in Algorithm 5.1.

At a high level this algorithm is quite similar to the Jain-Vazirani algorithm, but several challenges have to be dealt with in order for the algorithm to run in logarithmic number of rounds in the $\mathcal{CONGEST}$ model. For example, we have to quickly find an initial feasible solution to the DP that is not too far away from the

Algorithm 5.1 Algorithm overview

1: **Initialization phase.** Runs in constant rounds. We identify the *low-paying* clients and *cheap* facilities. Cheap facilities are opened and low-paying clients are connected to cheap facilities. Initial α_j-values for the remaining (i.e., non-low-paying) clients are computed.

2: **Primal-Dual phase.** Runs in $O(\log n)$ rounds. Each client j increments its dual variable (i.e., α_j) geometrically in each round until a facility i is found such that the connection cost c_{ij} is "paid for" by α_j and the facility opening cost f_i is paid for by the payments of some clients. Such facilities are declared *temporarily open.* an j is connected to i.

3: **Sparsification phase.** Runs in $O(\log m)$ rounds. We use Luby's MIS algorithm [63] to shut down many of the temporarily open facilities and declare the rest permanently opened. Some clients are reconnected as the result of the partial shut down.

"final" feasible solution so that the Primal-Dual Phase can terminate in $O(\log n)$ rounds.

5.2.2 Initialization Phase

Set $\alpha_j := \frac{1}{n}\min_i(f_i + c_{ij})$ for all $j \in C$. For each $j \in C$, if i^* is such that $f_{i^*} + c_{i^*j} = \min_i(f_i + c_{ij})$, then set $\phi(j) := i^*$. For each $i \in F$, $j \in C$, set $\beta_{ij} := \max\{\alpha_j - c_{ij}, 0\}$.

Algorithm 5.2 Initialization Phase

1: Each facility i broadcasts f_i

2: Each client j, knowing f_i and c_{ij} for all i and knowing n sets $\alpha_j \leftarrow \frac{1}{n}\min_i(f_i + c_{ij})$ and computes $\phi(j)$. Each client j then broadcasts α_j

3: Each facility i then calculate $\alpha^* \leftarrow \max_j \alpha_j$ and broadcasts α^*

4: Each client j checks if $\alpha_j \leq \alpha^*/n$ and determines if it is *low-paying.* Each client j that is low-paying sets $color(j) \leftarrow grey$ and broadcasts $ID(j)$ and $\phi(j)$

5: For each facility $i \in F | i = \phi(j)$ for some j hears this and sets $status(i) \leftarrow open$

Lemma 48. *The set of α_j-values and β_{ij}-values defined above constitute a feasible solution to the DP.*

Proof. It is easy to verify the constraints $\alpha_j - \beta_{ij} \leq c_{ij}\ \forall i \in F, \forall j \in C$, $\alpha_j \geq 0\ \forall j \in F$ and $\beta_{ij} \geq 0\ \forall i \in F, \forall j \in C$. Hence, we will focus on the constraints $\sum_{j \in C} \beta_{ij} \leq f_i$, $\forall i \in F$. Let us fix an $i \in F$ and examine β_{ij} for each $j \in C$. If $\beta_{ij} = 0$, it is clear that $\beta_{ij} - f_i \leq 0$ and more to the point,

$$\beta_{ij} \leq \frac{f_i}{n} \tag{5.1}$$

On the other hand, if $\beta_{ij} > 0$, then it must be the case that $\beta_{ij} = \alpha_j - c_{ij}$. For this j, suppose that $\phi(j) = i^*$, i.e.,

$$\alpha_j = \frac{1}{n}\left(\min_i (f_i + c_{ij})\right) = \frac{1}{n}\left(f_{i^*} + c_{i^*j}\right).$$

Hence, $\beta_{ij} = \frac{1}{n}(f_{i^*} + c_{i^*j} - nc_{ij})$. By choice of i^*,

$$\beta_{ij} - \frac{f_i}{n} = \frac{1}{n}\Big(f_{i^*} + c_{i^*j} - nc_{ij} - f_i\Big) = \frac{1}{n}\Big((f_{i^*} + c_{i^*j}) - (f_i + c_{ij}) - (n-1)c_{ij}\Big) \leq 0. \tag{5.2}$$

Summing over all j, using inequalities (5.1) and (5.2), we get $\sum_{j \in C} \beta_{ij} - f_i \leq 0$. □

Now, let us define $\alpha^* = \max_j \alpha_j$ and let $j^* = \arg\max \alpha_j$. Any $j \in C$ with $\alpha_j \leq \alpha^*/n$ will be called a *low-paying* client and the facility $\phi(j)$ (for low-paying j) will be called a *cheap* facility. The reason for identifying these low-paying clients and cheap facilities is that their overall opening and connection cost is so low that they can be dealt with much further ado (so shown in the next lemma). Furthermore, eliminating these clients is critical in guaranteeing that the next phase (i.e., the

primal-dual phase) runs in $O(\log n)$ rounds. In preparation for the next lemma, let L be the set of low-paying clients and OPT be the cost of an optimal solution to the facility location problem.

Lemma 49. *The total cost of opening all cheap facilities and connecting each low-paying client j to facility $\phi(j)$ is at most OPT.*

Proof. The total cost of opening the cheap facilities and connecting each low-paying client j to cheap facility $\phi(j)$ is bounded above by

$$\sum_{j \in L} \left(c_{\phi(j)j} + f_{\phi(j)} \right) = \sum_{j \in L} \alpha_j \leq \sum_{j \in L} \frac{\alpha^*}{n}.$$

Since the total number of clients is n, the right hand side above is bounded above by α^*. Note that there is a client j^* is such that $\alpha_{j^*} = \alpha^*$ and furthermore the α_j's are all part of a feasible solution to the DP. Therefore, by weak duality $\alpha^* \leq \sum_{j \in C} \alpha_j \leq OPT$. □

The aforementioned process (Algorithm 5.2) of identifying the low-paying clients and cheap facilities can be accomplished quite easily in just $O(1)$ rounds of communication with messages of size $O(\log n)$.

5.2.3 The Primal-Dual Phase

Here we describe the primal-dual phase of the algorithm in which α_j's are geometrically raised in a synchronous manner and a set of facilities are "temporarily" opened. To prevent low-paying clients and cheap facilities from getting in the way of the primal-dual phase, for each low-paying client j, we set $color(j) := grey$ and

$status(\phi(j)) := open$. The *color* of a client j will indicate to the primal-dual phase whether j is still active or not; the color *grey* will denote inactivity (i.e., α_j is not raised any more). The *status* of a facility can be one of $\{closed, temporarily\text{-}open, open\}$ and the cheap facilities are all declared open and do not participate in the primal-dual phase.

Recalling that L denotes the set of low-paying facilities, let $\alpha_{min} := \min_{j \notin L} \alpha_j$. Thus α_{min} is the smallest α_j value, excluding the "very small" α_j values belonging to the low-paying clients. Also, note that $\alpha^*/n < \alpha_{min} \leq \alpha^*$.

Algorithm 5.3 Primal-Dual Client j Algorithm

1: **Init:** If j is not a low-paying client: $\alpha_j \leftarrow \alpha_{min}$; $color(j) \leftarrow white$.
2: {**Activity in iteration** s**:**}
3: **if** $(color(j) = white)$ **then**
4: $\quad Send[\alpha_j]$
5: $\quad \forall i$: $\gamma_{ij} \leftarrow \alpha_j - c_{ij}$
6: $\quad \forall i$: $Receive[status(i)]$
7: $\quad$ **if** $(\exists i$ such that $(status(i) \in \{temporarily\text{-}open, open\} \wedge \gamma_{ij} \geq 0))$ **then**
8: $\quad\quad color(j) \leftarrow grey$
9: $\quad\quad \phi(j) \leftarrow i$
10: $\quad Send[color(j)]$
11: $\quad \alpha_j \leftarrow 2 \cdot \alpha_j$

We next describe a typical iteration (say, iteration s) of Client j's Primal-Dual Algorithm (see Algorithm 5.3). Initially, every client j (that is not low-paying) sets its α_j-value to α_{min} and sets its *color* to *white* indicating that it is ready to be active. The clients increase their α_j-values synchronously, in every iteration, by a

constant multiplicative factor. These α_j-values should be viewed as "payments" and the payment α_j of client j first goes towards paying off the connection costs c_{ij} for all $i \in F$. The sign of γ_{ij} (Line 5) indicates whether j has paid for a connection cost c_{ij} or not. If, during an iteration , j discovers that it has paid enough towards a connection cost c_{ij}, then there are two possibilities: (i) i is already open, in which case the client j is connected to i, by setting $\phi(j)$ to i (Line 9) and (ii) i is not open and in this case j's payments will go towards opening the facility i, i.e., towards β_{ij}.

The *color* of a client denotes whether it has been connected or not. Initially all active clients are *white*. Each client changes its own color to *grey* as soon as they get connected.

All facilities that are not cheap, are initialized to have a *closed status*. During the course of a typical iteration (say, iteration s) of facility i's algorithm (shown in Algorithm 5.4), the facility receives payments α_j from clients, determines their residual payments β_{ij} (after accounting for payment towards connection costs), and checks if the residual payments β_{ij}, for all $j \in C$ are sufficient to pay for the cost of opening f_i (in Line 6). When the opening cost of a facility is completely paid for, it sets its status to *temporarily-open*.

Note that, corresponding to the three message transactions (*Send-Receive-Send*) in a typical iteration of a client's algorithm, there are three message transactions (*Receive-Send-Receive*) in a facility's algorithm. We assume that these transactions take place in a synchronous manner. All activities in the primal-dual algorithm cease when all clients become *grey*. This is because when clients become *grey* they no

Algorithm 5.4 Primal-Dual Facility i Algorithm

1: **Init:** $status(i) \leftarrow closed$ if i is not a cheap facility
2: {**Activity in iteration** s**:**}
3: **if** $(status(j) = closed)$ **then**
4: $\quad \forall j : Receive[\alpha_j]$
5: $\quad \forall j$ such that $color(j) = white$: $\beta_{ij} \leftarrow \max\{\alpha_j - c_{ij}, 0\}$
6: $\quad$ **if** $(\sum_{j \in C} \beta_{ij} \geq f_i)$ **then**
7: $\quad\quad status(i) \leftarrow temporarily\text{-}open$
8: $\quad Send[status(i)]$
9: $\quad \forall j$: $Receive[color(j)]$

longer increase their α_j-values and without any change in the α_j-values no new facilities will be paid for. We now prove two important properties of the Primal-Dual Algorithm.

Lemma 50. *The Primal-Dual Algorithm runs for at most* $6 \log n$ *communication rounds.*

Proof. For client j, let ${\alpha_j}^{(s)}$ denote the value of α_j after iteration s. We know ${\alpha_j}^{(0)} = \alpha_{min}$ and also $\alpha_{min} > \alpha^*/n$. Since α_j's double in each iteration, $\alpha_j^{(t)} > n \cdot \alpha^*$ for $t := 2 \log n$. Now recall from the initialization phase that $\alpha^* = \max_j \min_i (f_i + c_{ij})/n$. Therefore, $\alpha_j^{(t)}$ will exceed $\min_i (f_i + c_{ij})$, which means that client j, all by itself, is paying for more than the amount needed to open facility i^*, where i^* is $\text{argmin}_i (f_i + c_{ij})$. Therefore client j, for all $j \in C$, turns grey in round $t = 2 \log n$ or earlier. Each iteration of the primal-dual algorithms corresponds to 3 communication rounds, yielding the result. □

Lemma 51. *After the Primal-Dual Algorithm has terminated,* $\sum_{j \notin L} \alpha_j \leq 2 \cdot OPT$.

Proof. Set $\overline{\alpha}_j = 0$ for all $j \in L$ and $\overline{\alpha}_j = \alpha_j/2$ for all $j \notin L$. Set $\overline{\beta}_{ij} := \max\{c_{ij} - \overline{\alpha}_j, 0\}$ for all $i \in F, j \in C$. It is easy to check that this is a feasible solution to the DP and therefore by weak duality $\sum_{j \in C} \overline{\alpha}_j \leq OPT$. This yields the claim. □

5.2.4 The Sparsification Phase

At the end of the Primal-Dual Phase we have a solution (albeit, a costly one, possibly) to the facility location problem, given by the open and temporarily open facilities and each client connecting to an open or temporarily open facility. The goal of the sparsification phase is to obtain a cheaper solution that can be "charged" to the dual variables and since the sum of the dual variables is within a constant times OPT (Lemma 51), we will get a constant-factor approximation. This is similar to the last phase in the Jain-Vazirani algorithm [36]; our contribution here is to show that it can be implemented in $O(\log m)$ rounds in the $\mathcal{CONGEST}$ model.

Let F_t be the set of temporarily open facilities at the end of the Primal-Dual Phase. Define a graph $H = (F_t, E_t)$, where E_t consists of edges connecting pairs of temporarily open facilities i and i' for which there is a client j such that $\beta_{ij} > 0$ and $\beta_{i'j} > 0$; in other words, j makes a positive payment towards the opening of both i and i'. The high level algorithm for the sparsification phase is shown in Algorithm 5.5.

Note that since M is an MIS of H, for every client j that finds out that the facility $\phi(j)$ is now closed, there is another facility $i' \in M$, that is a neighbor of

Algorithm 5.5 Sparsification Phase

1: Compute a maximal independent set (MIS) M of H
2: Permanently open each facility in M, i.e., $\forall i \in M : status(i) \leftarrow open$
3: Close each facility in $F_t \setminus M$, i.e., $\forall i \in F_t \setminus M : status(i) \leftarrow closed$
4: **for all** clients $j \in C$ **do**
5: **if** $(status(\phi(j)) = closed)$ **then**
6: $\phi(j) \leftarrow i'$, where $i' \in M$ and is a neighbor in H of $\phi(j)$

$\phi(j)$ in H, to which j can be "reconnected." If a client j retains its connection from the Primal-Dual Phase, we say that j is *directly connected* to $\phi(j)$; otherwise, if j is reconnected in the Sparsification Phase, we say that j is *indirectly connected* to $\phi(j)$.

In the Sparsification Phase, Steps 2 and beyond are all easy to implement in constant number of communication rounds in the $\mathcal{CONGEST}$ model. So we focus on Step 1. The difficulty with using Luby's algorithm [1, 63] "as is" is to compute the MIS of H is that nodes in H do not have edges to each other in the underlying network and will have to communicate via clients. Given the restriction on the message sizes, this may be hard to do in constant rounds. For example, it is impossible for a node $i \in F_t$ to quickly (i.e., in constant or even polylogarithmic number of rounds) find out the IDs of all its neighbors in H since all this information may have to arrive at i via a single client. Consider a typical stage of Luby's algorithm:

1. Each, as yet undecided node $i \in F_t$ marks itself with probability $1/degree(i)$. Here $degree(i)$ refers to the degree of i in the subgraph of H induced by the undecided vertices.

2. Each node i that is marked in Step 1 unmarks itself if it finds a neighbor j with

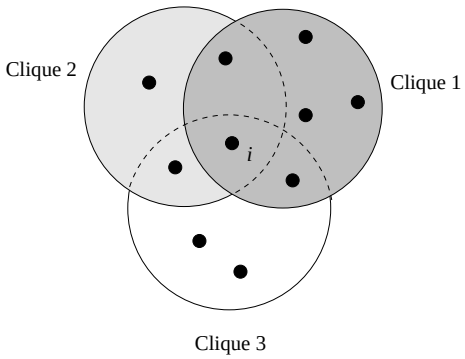


Figure 5.2: Suppose that a temporarily open facility i is positively paid for by Clients 1, 2 and 3 (i.e., $\beta_{i1}, \beta_{i2}, \beta_{i3} > 0$). Further suppose that client 1 positively pays for 6 temporarily open facilities, client 2 pays for 4, and client 3 for 5 temporarily open facilities. This leads to 3 cliques in H of sizes 6, 4, and 5 respectively. If each facility i thinks of itself as belonging to the clique of the client with lowest ID, then Clique 1 will have size 6, Clique 2 will have size 2, and Clique 3 will have size 2. This allows i to figure out that its degree is $6 + 2 + 2 = 10$.

lower degree that has marked itself. Ties can be resolved via the use of IDs.

The nodes that mark themselves in a stage of Luby's algorithm have "decided" to join the MIS and neighbors of such nodes have "decided" not to join the MIS; the undecided nodes continue to the next phase. For Step 1, node $i \in F_t$ needs to know its degree in the subgraph of H induced by currently undecided nodes and for Step 2 node i needs to know the lexicographically smallest tuple $(ID(i'), degree(i'))$ over all marked nodes i' in its neighborhood. Both pieces of information can be computed in constant number of rounds in the $\mathcal{CONGEST}$ model and to do this we take advantage of the fact that H is essentially composed of a number of interacting cliques, with each clique being "controlled" by a client (see Figure 5.2).

In Algorithm 5.6, we present in details how each undecided node $i \in F_t$ to compute $degree(i)$ in a constant number of rounds. The information needed for Step

2 can similarly be computed in constant rounds.

Algorithm 5.6 Calculating Degrees

1: {This algorithm is illustrated by Figure 5.2}
2: Each client j broadcasts its ID.
3: Each undecided $i \in F_t$ computes $c(i) \leftarrow \min_{j:\beta_{ij}>0}(ID_j)$. Node i broadcasts $c(i)$.
4: Each client j computes $d(j)$, which is the total number of facilities with $c(i) = ID_j$. Node j broadcasts $d(j)$.
5: Each undecided $i \in F_t$ computes $degree(i) \leftarrow \sum_{j:\beta_{ij}>0} d(j)$.

5.2.5 Analysis

We now we analyze the quality of the computed solution as well as the number of rounds it takes to compute the solution in the $\mathcal{CONGEST}$ model. We start off with a lemma showing that the cost of reconnecting j to an open facility in the Sparsification Phase can be charged to α_j. A proof such as this appears in Jain and Vazirani's analysis [36] as well; our proof is slightly different since events in our algorithm happen in discrete rounds whereas the Jain-Vazirani algorithm grows dual variables continuously.

Lemma 52. *If j is indirectly connected to i, then $c_{ij} \leq 3\alpha_j$.*

Proof. Consider a facility i' that is responsible for j turning grey. That is, client j finds, in some round that $status(i')$ is either *open* or *temporarily-open* and $\gamma_{i'j} = \alpha_j - c_{i'j} \geq 0$. This implies that $\alpha_j \geq c_{i'j}$. Since j is indirectly connected to i, it must be the case that i' was shut down during the Sparsification Phase and facilities i and

i' are neighbors in H. Hence, there exists a client j' such that $\beta_{ij'} > 0$ and $\beta_{i'j'} > 0$ (refer to Fig. 5.3). We will show $\alpha_j \geq c_{i'j'}$ and $\alpha_j \geq c_{ij'}$. The rest will follow from triangle inequality.

Suppose that facility i and i' we temporarily opened in rounds t_1 and t_2 respectively. Note that at the beginning of these rounds all white clients' α-values were $\alpha^{(t_1)}$ and $\alpha^{(t_2)}$ respectively. We have also argued before that j become grey either in round t_2 or after. Hence $\alpha_j \geq \alpha^{(t_2)}$. As $\beta_{ij'} > 0$ and $\beta_{i'j'} > 0$, $\alpha_{j'} > c_{i'j'}$ and $\alpha_{j'} > c_{ij'}$. Note that $\alpha_{j'}$ cannot be growing after round $\min\{t_1, t_2\}$. Hence

$$\alpha_{j'} \leq \min\{\alpha^{(t_1)}, \alpha^{(t_2)}\} \leq \alpha^{(t_2)} \leq \alpha_j$$

Therefore, $\alpha_j \geq c_{i'j'}$ and $\alpha_j \geq c_{ij'}$ as well. □

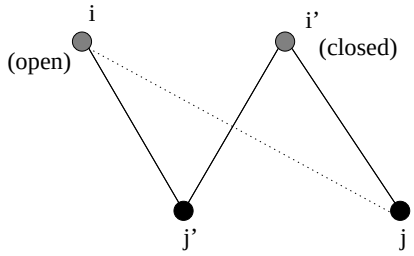


Figure 5.3: Client j is indirectly connected to i because i' was closed after the MIS computation.

The following lemma can be proved by routine accounting and we skip its proof.

Lemma 53. *The algorithm presented above terminates in $O(\log m + \log n)$ rounds and uses messages of size $O(\log N)$ bits.*

Proof. We can assign unique IDs to all clients and facilities using $\log_2 N$ bits. We also assume that the constants involved (f_i and c_{ij} values) and α_j and β_{ij} values can also be represented by $O(\log N)$ bits. We show that the bounds on total number of rounds and message size are respected in every stage of the algorithm.

Initialization phase. We have already shown that the "low-paying" facilities can be identified in $O(1)$ rounds. The messages sent contain f_i values, α_j values and node IDs. Hence the maximum message size is $O(\log N)$.

Primal-dual phase. Lemma 50 shows it is enough to let the facility process and the client process run for $O(\log n)$ rounds. The messages involved contain α_j values, status values (2 bits) and color values (1 bit).

Sparsification phase. First compute the degree of a temporarily-open facility i in H in $O(1)$ rounds. The messages involved are client IDs, $c(i)$ values and $d(j)$ values. All these values are at most N and can be represented in $O(\log N)$ bits. Finally, we use Luby's MIS algorithm that runs in $O(\log m)$ rounds using messages of size $O(\log m)$. □

Our final lemma shows that our solution is a 7-approximation.

Lemma 54. *Let I be the set of facilities opened by our algorithm. Then,*

$$\sum_{i \in I} f_i + \sum_{j \in C} c_{\phi(j)j} \leq 7 \cdot OPT.$$

Proof. Let $I' \subseteq I$ denote the facilities that are not cheap and were therefore opened in the Primal-Dual Phase. Let $COST$ denote the total cost of opening facilities in I' and the connection cost of clients that are not low-paying. Clients that are not low-paying can be partitioned into sets, A, consisting of clients that directly connect to a facility and B, consisting of clients that indirectly connect to a facility. Note that for each $i \in I'$, $f_i \leq \sum_{j \in A} \beta_{ij}$. Therefore,

$$\begin{aligned}
COST &= \sum_{i \in I'} f_i + \sum_{j \in A} c_{\phi(j)j} + \sum_{j \in B} c_{\phi(j)j} \\
&\leq \sum_{i \in I'} \sum_{j \in A} \beta_{ij} + \sum_{j \in A} (\alpha_j - \beta_{\phi(j)j}) + 3 \cdot \sum_{j \in B} \alpha_j \qquad (5.3) \\
&\leq \sum_{j \in A} \beta_{\phi(j)j} + \sum_{j \in A} (\alpha_j - \beta_{\phi(j)j}) + 3 \cdot \sum_{j \in B} \alpha_j \qquad (5.4) \\
&\leq \sum_{j \in A} \alpha_j + 3 \cdot \sum_{j \in B} \alpha_j \\
&\leq 3 \cdot \sum_{j \notin L} \alpha_j \leq 6 \cdot OPT \qquad (5.5)
\end{aligned}$$

Inequality (5.3) is obtained by using the upper bound on f_i, $i \in I'$, mentioned earlier and the fact that if client j is directly connected to $\phi(j)$, then $\alpha_j \geq c_{\phi(j)j}$ and so $\beta_{\phi(j)j} = \alpha_j - c_{\phi(j)j}$. Inequality (5.4 is obtained by rewriting $\sum_{i \in I'} \sum_{j \in A} \beta_{ij}$ as $\sum_{j \in A} \sum_{i \in I'} \beta_{ij}$ and then noting that for each $j \in A$, $\beta_{ij} > 0$ for exactly one i, namely $i = \phi(j)$. This is the result of the Sparsification Phase. The first part of Inequality (5.5) follows from the fact that $A \cup B$ is the set of clients that are not low-paying. Earlier we had used L to denote the set of low-paying clients. The second part of this inequality follows from Lemma 51.

The total cost of the solution, including opening cost of cheap facilities and connection cost of low-paying clients, is $7 \cdot OPT$. □

5.3 k-round Algorithm

Motivated by the goal of achieving a non-trivial approximation ratio even while using a very small (e.g., a constant) number of rounds, we present in this section a generalization of the Logarithmic-round algorithm. For two tunable, positive, integer parameters k_1 and k_2, we run the algorithm for $k := k_1 \cdot k_2$ rounds to obtain an $O(n^{2/k_1} \cdot m^{2/k_2})$-approximation algorithm. Recall that m is the number of facilities and n is the number of clients. Here k_1 can take any value whereas k_2 can take any value bounded by $O(\frac{\log m}{\log \log m})$. The bound on k_2 comes from our randomized Sparsification Algorithm, which does not exhibit the desired behavior for larger values of k_2. This is because for larger k_2 the degree of the graph being sparsified falls below $\log m$ and at that point it is no longer possible to prove properties of the Sparsification Algorithm with high probability.

The high-level structure of the k-round algorithm is shown in Algorithm 5.7.

The Initialization Phase is identical to the one used in the Logarithmic-round algorithm. However, the other two phases (especially the Sparsification Phase) are quite different and furthermore, how the Primal-Dual Phase and the Sparsification Phase interact is also quite different. Rather than running the Sparsification Phase after the Primal-Dual Phase is completed, we now interleave the iterations of the two

Algorithm 5.7 k-round Algorithm

1: Initialization Phase.
2: **for** $p \leftarrow 1$ **to** k_1 **do**
3: Run an iteration of the Primal-Dual Algorithm (for each client and each facility)
4: **if** ($\exists$ facility i that is temporarily opened in iteration p) $\wedge$
($\exists$ facility i' that is temporarily opened in iteration $p-1$ or earlier) $\wedge$
($\exists$ client j: $\beta_{ij} > 0 \wedge \beta_{i'j} > 0$) **then**
5: $status(i) \leftarrow closed$
6: **for** $q \leftarrow 1$ **to** k_2 **do**
7: Run an iteration of the Sparsification Algorithm
8: {**for each facility temporarily opened in this Primal-Dual iteration**}

phases, running k_2 iterations of a Sparsification Algorithm after each iteration of the Primal-Dual Algorithm.

To have the Primal-Dual Algorithm make sufficient progress in each iteration, client j raises the α_j-value by a multiplicative factor of n^{2/k_1}. Recall from Section 5.2.3 that each α_j for $j \notin L$ is initialized to α_{min}, where $\alpha_{min} \cdot n^2 > \min_i(f_i + c_{ij})$ for every client j. Therefore, if client j were to run k_1 iterations of the Primal-Dual Algorithm, then its α_j value would grow from α_{min} to $\alpha_{min} \cdot (n^{2/k_1})^{k_1} = \alpha_{min} \cdot n^2 > \min_i(f_i + c_{ij})$. At this point the α_j-value is more than sufficient to pay for the opening cost of a facility all by itself and therefore client j would have turned grey and been connected to some facility $i = \phi(j)$ that is temporarily open. This discussion is encapsulated in the following lemma.

Lemma 55. *Every client that enters the Primal-Dual Phase colored white turns grey during one of the k_1 iterations of the Primal-Dual Phase and gets connected to some temporarily open facility.*

Also, similar to Lemma 51, we can show that the α_j-values, when scaled down

by a factor of n^{2/k_1} and the corresponding β_{ij} values form a feasible solution to the DP, yielding the following lemma; the proof is similar to the proof of Lemma 51 and is skipped.

Lemma 56. *After the k-round Algorithm has terminated,* $\sum_{j \notin L} \alpha_j \leq n^{2/k_1} \cdot OPT$.

The Sparsification Phase is the fundamental challenge for the k-round algorithm. Recall from the analysis of the Logarithmic-round algorithm that the goal of the Sparsification Phase is to bound, for each client j, the size of $K_j = \{i \in F \mid \beta_{ij} > 0 \text{ and } i \text{ is open}\}$. If we permanently open all the nodes in F_t (the set of temporarily open facilities), K_j can become arbitrarily large. This motivated the definition of the graph $H = (F_t, E_t)$, where E_t contains all edges connecting pairs of facilities $i, i' \in F_t$ such that there is a client j with $\beta_{ij} > 0$ and $\beta_{i'j} > 0$. Then an MIS M on H is computed and only nodes in M are permanently opened. As a result, $|K_j| \leq 1$ for all j. Furthermore, it is guaranteed that if for a client j, facility $i = \phi(j)$ is shut down (due to not being in M) then some neighbor i' of i in H is open and j can be reconnected to i' without the cost of the reconnection being too high relative to α_j.

Given that we now have a very small number of rounds in which to do sparsification, we can no longer use a Luby-like algorithm because it does not sparsify the graph rapidly enough to make adequate progress in a small number of rounds. Since the goal of the k-round algorithm is not to achieve a constant-factor approximation, we can relax the requirement that we need to shut down enough temporarily open facilities to ensure that $|K_j| = O(1)$ for all clients j. Instead, we would like to use k_2 rounds to sparsify H and select a set of nodes M that induces a sparse subgraph

with maximum degree $O(m^{1/k_2})$. M needs to satisfy the additional "maximality" requirement that each node in H has a neighbor in M. This maximality property will guarantee that if for a client j, facility $i = \phi(j)$ is shut down (due to not being in M) then some neighbor i' of i in H is open and j can be reconnected to i' without the cost of the reconnection being too high relative to α_j. Unfortunately, we do not know how to rapidly sparsify in a controlled manner so as to satisfy both requirements. In the next subsection we present an algorithm that runs for k_2 rounds and selects a node subset M of H and guarantees, with high probability, the following two properties:

1. $\Delta(H[M]) \leq 6 \cdot m^{2/k_2}$.

2. $\mathit{distance}(i, M) \leq 2 \cdot k_2$ for all nodes i in H.

Here $\Delta(\cdot)$ refers to the maximum degree of a given graph and $\mathit{distance}(i, M)$ is the shortest number of hops in H between a facility i and the set of selected vertices M. Note that our algorithm only satisfies a relaxed version of the maximality requirement — a node in H may not find a node in M in its neighborhood, but will find a node in M within $2 \cdot k_2$ hops. This means that there may be a client j for which the facility $i = \phi(j)$ is shut down (due to not being in M) and now client j has to be reconnected to a facility i' that is relatively far away. Given this, we are no longer able to ensure that the cost of reconnecting j to i' is small enough. However, if all the α_j-values are identical, then we can guarantee that the cost of reconnecting j to i' is $O(\alpha_j \cdot k_2)$. This bound on the reconnection cost suffices for our analysis and in order to ensure that all of the α_j-values are identical, we run the Sparsification Algorithm repeatedly and

separately, on just the facilities that were temporarily opened in each Primal-Dual Algorithm iteration (Lines 7-10). This modification by itself would still allow K_j to be quite large because a client j could be making positive payments to facilities that were temporarily opened in different iterations. To ensure that $|K_j|$ is bounded, we also additionally shut down (in Lines 4-6) each facility i that was temporarily opened in an iteration p but which has a facility i' "nearby" that was temporarily opened in a previous iteration.

Our Sparsification Phase is inspired by a recent randomized MIS algorithm due to Gfeller and Vicari [24]. These authors are interested in quickly computing an MIS on growth-bounded graphs. While our graphs are not growth-bounded, the first phase of the Gfeller-Vicari algorithm runs on arbitrary graphs and here we essentially show that even a few iterations of a variant of this algorithm guarantees enough progress.

5.3.1 The Sparsification Phase

We will describe the Sparsification Phase with respect to an arbitrary input graph $G = (V, E)$. It is worth noting that in the context of Algorithm 5.7, the Sparsification Phase is run on a graph $H = (F_t, E_t)$ where F_t is the set of facilities temporarily opened in an iteration p of the Primal-Dual Phase (Line 3) and not immediately shut down in Line 5; as usual E_t is the set of edges connecting pairs of facilities i, i' in F_t for which there is a client j with $\beta_{ij} > 0$ and $\beta_{i'j} > 0$.

Let $M_0 := V$ and $G_0 := G$. For $s \geq 1$, let M_s be the set of nodes selected

Algorithm 5.8 Sparsification Algorithm

Input: $G = (V, E)$, $\mid V \mid = m$
{**Activity for node** u **in iteration** s**:**}
$\tau \leftarrow m^{\frac{2}{r}}$
$\rho \leftarrow \frac{1}{m^{\frac{1}{r}}}$
if $(d_u \leq \tau) \wedge (\Delta_u \leq \tau)$ **then**
    $type(u) \leftarrow interior$
    u joins M_s
if $(d_u \leq \tau) \wedge (\Delta_u > \tau)$ **then**
    $type(u) \leftarrow boundary$
if $(d_u > \tau)$ **then**
    $type(u) \leftarrow exterior$
if $type(u) \in \{boundary, exterior\}$ **then**
    u independently joins M_s with probability ρ

in iteration s of the Sparsification Phase when run on graph G_{s-1} and let $G_s := G[M_s]$. We now describe a typical iteration s of the Sparsification Phase. For ease of presentation, set $r = s + 2$. The goal of iteration s is to select a set M_s of nodes so that the maximum degree in $G[M_s]$ is at most $m^{2/r}$. Let $\tau := m^{2/r}$. Any node in G_{s-1} with degree no greater that τ is called a *small degree* node. Any small degree node, all of whose neighbors are also small degree nodes, is called an *interior* node; all other small degree nodes are called *boundary* nodes. Nodes that are not small degree nodes are called *exterior* nodes. See Figure 5.4 for an illustration of these definitions.

The Sparsification Algorithm is shown in Algorithm 5.8. Given the goal of ensuring that all node degrees are bounded by $m^{2/r}$, the interior vertices are not of concern because of their small degrees and their neighbors' small degrees and they can simply be included in the output. The boundary vertices and the exterior vertices probabilistically include themselves in the output. Since boundary vertices

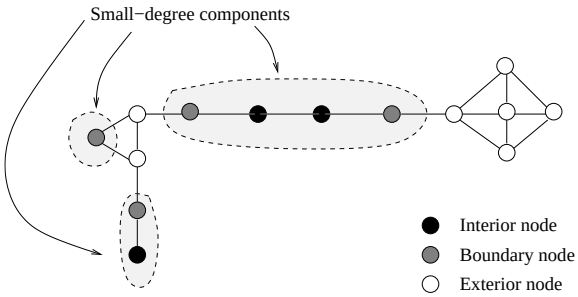


Figure 5.4: Shows the partition of the nodes of a graph into interior, boundary, and exterior nodes with degree threshold τ set to 2.

also have small degrees, we do have to worry about their degrees. We can inductively upper bound the degree of exterior vertices entering iteration s and if we choose the probability ρ of inclusion in M_s carefully, then we can bound the degrees of exterior vertices by $m^{2/r}$ also. This is proved in the next two lemmas. Of course, if the goal was solely to bound the degrees, then we could achieve this trivially by including no vertices in M_s. An additional an equally important goal in the context of Algorithm 5.7 is to show that after k_2 iterations of the executed, the output $M := M_{k_2}$ satisfies $\mathit{distance}(i, M) \leq 2 \cdot k_2$ for all $i \in V$. Here $\mathit{distance}(\cdot, \cdot)$ refers to hop distance in G. Lemma 59 shows that in each iteration the selected vertices can get at most 2 hops farther away from all vertices of G.

Lemma 57. *For some constant $c_1 > 0$, for all $s \leq c_1 \cdot \frac{\log m}{\log \log m}$, if the maximum degree in G_{s-1} is at most $m^{\frac{3}{(s+2)}}$, then with high probability, the maximum degree of G_s is at most $6 \cdot m^{\frac{2}{(s+2)}}$.*

Proof. Recall that we use r to denote $s + 2$. For a node u in G_s, if u is either an interior or a boundary node during iteration s, then the degree of u in G_{s-1} is at most

$m^{2/r}$. As vertices are only dropped during the iteration, the degree of u in G_s is also at most $m^{2/r}$. Hence we can simply consider the case when u is an exterior node in iteration s. Let X_u be the random variable denoting the number of neighbors of u in G_{s-1} who joined M_s. Since u has at most $m^{\frac{3}{r}}$ neighbors in G_{s-1} and each of them can join M_s with probability $\frac{1}{m^{\frac{1}{r}}}$,

$$E[X_u] \leq m^{\frac{3}{r}} \cdot \frac{1}{m^{\frac{1}{r}}} = m^{\frac{2}{r}}$$

Since X_u is the sum of independent binary random variables, using a simple version of Chernoff bounds, we get that

$$Prob[X_u \geq 6m^{\frac{2}{r}}] \leq 2^{-6m^{\frac{2}{r}}}.$$

Since $r \leq c_1 \cdot \frac{\log m}{\log \log m} - 2$,

$$Prob[X \geq 6m^{\frac{2}{r}}] \leq 2^{-6m^{\frac{\log \log m}{c_1 \log m}}}.$$

It is easy to verify that for some constant $c_1 > 0$, the right hand side above is bounded by $1/m^2$. This proves that with probability at least $1 - 1/m^2$, the maximum degree of node u in G_s is at most $6m^{\frac{2}{r}}$. Using the union bound, we see that the probability that all vertices (at most m) have degree at most $6m^{\frac{2}{r}}$ in G_s is at least $(1 - m \cdot \frac{1}{m^2}) = (1 - \frac{1}{m})$. □

The hypothesis of the above lemma required that the maximum degree of G_{s-1} be upperbounded by $m^{\frac{3}{s+2}}$. We now inductively show that this is true.

Lemma 58. *For some constant $c_2 > 0$, for all $s \leq c_2 \cdot \log m$, the maximum degree in G_{s-1} is at most $m^{\frac{3}{s+2}}$, with high probability.*

Proof. We show this by induction. The base case of $s = 1$ is trivial since every node has degree at most m at all times. For the inductive step, consider iteration $s > 1$. Assuming, using the inductive hypothesis, that the maximum degree in G_{s-1} is $m^{\frac{3}{(s+2)}}$, we get by Lemma 57, that the maximum degree in G_s is at most $6m^{\frac{2}{(s+2)}}$. It is easy to verify that for large enough m, there exists a constant $c_2 > 0$ such that for all $s \leq c_2 \cdot \log m$, $6m^{\frac{2}{(s+2)}} \leq m^{\frac{3}{(s+3)}}$. □

The next lemma shows that with each iteration of the Sparsification algorithm, the selected set of nodes "move" at most 2 hops further away from the rest of the graph.

Lemma 59. *For some constant c_3 and for $s \leq c_3 \cdot \frac{\log m}{\log \log m}$, if $u \in M_{s-1}$, then with probability at least $1 - 1/m^2$, at least one node in its 2-neighborhood is in M_s.*

Proof. Consider a node u in M_{s-1}. If u is an interior node in iteration s, it is included in M_s. If u is an exterior node in iteration s, we know that the degree of u in G_{s-1} is greater than $m^{2/r}$. The probability that none of u's neighbors are selected for M_s is at most

$$(1 - \frac{1}{m^{\frac{1}{r}}})^{m^{\frac{2}{r}}} \leq e^{-m^{\frac{3}{r}}}.$$

It is easy to verify that there exists a constant $c_3 > 0$, such that for all $r \leq c_3 \cdot \frac{\log m}{\log \log m} - 2$, this probability is at most $1/m^2$. If u is a boundary node, then it has at least one exterior node as a neighbor and therefore, using what we have shown for exterior nodes, we conclude that with probability at least $1 - 1/m^2$, there is node from M_2 in u's 2-neighborhood. □

5.3.2 Wrapping Things Up

Recall that the Sparsification Phase is applied to a graph $H = (F_t, E_t)$, where F_t is the set of facilities temporarily opened in an iteration p of the Primal-Dual Phase (Line 3) and not immediately shut down in Line 5 and E_t is the set of edges connecting pairs of facilities i, i' in F_t for which there is a client j with $\beta_{ij} > 0$ and $\beta_{i'j} > 0$. The following lemma is obtained in a straightforward way by repeatedly applying Lemma 59.

Lemma 60. *There is a constant $c > 0$ such that for any $k_2 \leq c \cdot \frac{\log m}{\log \log m}$, after k_2 iterations of the Sparsification Algorithm on input H, the output $M := M_{k_2}$ satisfies, with high probability, the property that distance$(i, M) \leq 2k_2$ for all nodes i in H. Here, distance(i, M) is the shortest path distance in hops in H from node i to some node in M.*

Proof. Let T_i be the nodes selected in the ith iteration and let us set $T_0 = V$. Due to Lemma 59, T_i is a 2-ruling set of T_{i-1}. Note that T_0 is a 0-ruling set. Hence the claim follows by induction over i. □

The next lemma shows that our efforts at careful sparsification have paid off in terms of guaranteeing that when a client j is reconnected, its reconnection cost can still be charged to α_j.

Lemma 61. *At the end of the k-round algorithm, each client j, with high probability, will connect to open facility i, with $c_{ij} \leq (4k_2 + 1) \cdot \alpha_j$.*

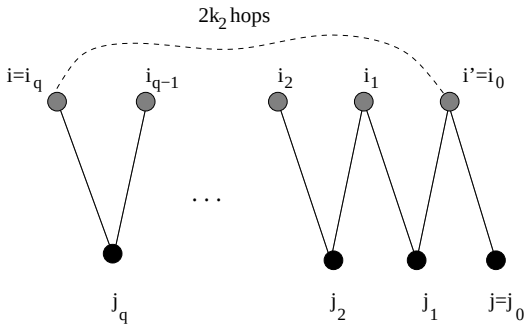


Figure 5.5: Client j was initially connected to facility i' that was closed during sparsification. In that case, j will find a facility i to connect to so that $distance(i, i') \leq 2k_2$.

Proof. If a client j is directly connected to a facility i, then we simply have $c_{ij} \leq \alpha_j$. If client j is indirectly connected to facility i, there could be two possible reasons for it:

(i) j was directly connected to a facility i' that was shut down in Line 5 of the k-round algorithm (Algorithm 5.7). Say this happened in the p-th iteration of the outer loop. Hence i' just became temporarily open in that iteration. From the conditions stated in Line 4 of Algorithm 5.7, there must also be another facility i that became temporarily open at an earlier iteration (say p', $p' < p$) and a client j' such that $\beta_{ij'} > 0$ and $\beta_{i'j'} > 0$. Hence, $c_{ij'} \leq \alpha_{j'}$ and $c_{i'j'} \leq \alpha_{j'}$. Notice that $\alpha_{j'}$ cannot increase after the end iteration p' as client j would have had a temporarily open facility that it has positively contributed to. Hence, $\alpha_{j'} < \alpha_j$. Also, j was directly connected to i', hence $c_{i'j} \leq \alpha_j$. Using the triangle inequality,

$$c_{ij} \leq c_{i'j} + c_{i'j'} + c_{ij'} \leq \alpha_j + \alpha_{j'} + \alpha_{j'} < 3\alpha_j$$

(ii) j was directly connected to a facility i' that was shut down during an iteration of the Sparsification Algorithm (Line 8) of the k-round algorithm (Algorithm 5.7). From Lemma 60, it follows that $distance(i, i') \leq 2k_2$. See Figure 5.5. For each pair i_m and i_{m+1}, to be neighbors in H, they must have had a client j_{m+1} such that $c_{i_m j_{m+1}} \leq \alpha_{j_{m+1}}$ and $c_{i_{m+1} j_{m+1}} \leq \alpha_{j_{m+1}}$ and also $\alpha_{j_{m+1}} \leq \alpha_j$. Given, j was directly connected to i', hence $c_{i'j} \leq \alpha_j$. Again, using the triangle inequality we get $c_{ij} \leq (4k_2 + 1) \cdot \alpha_j$.

□

Our final lemma proves the approximation guarantee for the solution produced by the k-round algorithm. The proof is similar to the proof of Lemma 54 and is skipped.

Lemma 62. *For any $k = k_1 \cdot k_2$, with $k_2 = O(\frac{\log m}{\log \log m})$, the k-round algorithm computes a solution that is an $O(n^{2/k_1} m^{2/k_2})$-approximation to the metric facility location problem.*

5.4 Conclusion

Here our main contribution is the demonstration of a trade-off between the approximation factor and the number of rounds of communication for the metric facility location problem in the $\mathcal{CONGEST}$ model. Specifically, we present an algorithm that runs in k rounds and yields an $O(m^{2/\sqrt{k}} \cdot n^{2/\sqrt{k}})$-approximation. Setting $k = c \cdot \log^2(m+n)$ for some constant c, would make this approximation factor reduce

to a constant. However, our randomized, rapid Sparsification Algorithm does not exhibit the desired behavior when $k > c' \cdot \left(\frac{\log m}{\log \log m}\right)^2$, for some constant c'. So there is a small gap at the upper end in the desired range of values that k can take and plugging this gap is a problem that we intend to tackle.

Gehweiler et al. [23] obtain a constant-factor approximation for the metric facility location problem with *uniform* facility opening costs. The Logarithmic-round Algorithm in this paper solves the non-uniform version of Gehweiler's problem in $O(\log n)$ rounds. Are $O(\log n)$ rounds necessary? Is it possible to obtain non-trivial lower bounds in this setting? Moscibroda and Wattenhofer [65] call the problem of finding lower bounds in this model an "outstanding" open problem.

Another question that interested us is whether it is possible to obtain a better approximation factor versus communication rounds trade-off for the *non-metric* facility location, improving on the approximation factor achieved by Moscibroda and Wattenhofer [65] for k-round algorithms. This is our most current work and the results are described in the next chapter. The results in this chapter and the next also highlight how the primal-dual approach can be a powerful tool in order to obtain a non-trivial approximation factor, even in small number of rounds.

CHAPTER 6
DISTRIBUTED APPROXIMABILITY OF NONMETRIC FACILITY LOCATION

6.1 Introduction

Simple sequential greedy algorithms provide optimal approximations for a variety of optimization problems such as *minimum dominating set* (MDS) and *facility location*. Despite their simplicity, these greedy algorithms are challenging to implement efficiently in a distributed setting because their success depends on decisions being made sequentially. In a breakthrough result, Jia et al. [37] showed that it is possible to implement a "relaxed" version of the greedy MDS algorithm in a distributed setting to achieve an $O(\log \Delta)$-approximation (in expectation) in $O(\log n \log \Delta)$ rounds (with high probability), Δ being the maximum degree of the input graph. This algorithm balances the need for parallelism with the need to control the size of the dominating set by first allowing several "good" choices to be picked independently (as opposed to picking one "best" choice at a time) and then efficiently pruning away some of these choices. In our view, the key contribution of the Jia et al. paper [37] is showing that the pruning process can be implemented efficiently, i.e., in $O(\log n)$ rounds with high probability, while pruning enough candidates so that only a constant overhead is incurred with respect to the approximation factor.

In this paper, we model the pruning process abstractly as involving sellers on one side and buyers on the other. The goal is to quickly select a subset of the sellers

[0]This research was partially supported by NSF Grant CCF 0915543.

so that the products that these sellers bring to the market has small *cost ratio*, i.e., the ratio of the total cost of the selected sellers' products to amount that interested buyers are willing to pay. See Section 6.1.1 for details. As modeled here, the pruning process applies not just to the MDS greedy algorithm, but to greedy algorithms in general (e.g., we describe how this pruning process applies to a greedy algorithm for facility location). We present an instance of the pruning process that, for any positive k, runs in $O(k)$ communication rounds with $O(\log N)$-sized messages, yielding a cost ratio of $O(N^{c/k})$. Here N is the product of the number of sellers and number of buyers and c is a small constant. The pruning process in the Jia et al. algorithm [37] should be viewed as a special case (with $k = \Theta(\log N)$ and cost ratio $O(1)$) of our algorithm. Using this k-round pruning algorithm as the basis, we derive several simple, greedy, k-round approximation algorithms for MDS and facility location. Our approximation algorithms shave a "logarithmic factor" off the best, known, approximation factor, typically achieved using LP-rounding techniques.

To place these results in context, we refer to the 2003 result due to Kuhn and Wattenhofer [53], that described, for any parameter k, an MDS algorithm running in $O(k)$ rounds and yielding an approximation factor of $O(k\Delta^{2/\sqrt{k}} \log \Delta)$. This was the first example of an algorithm achieving a non-trivial approximation ratio for MDS in a constant number of rounds (e.g., $k = 16$ implies an approximation factor of $O(\sqrt{\Delta} \log \Delta)$). Also note that by setting $k = \Theta(\log^2 \Delta)$, one obtains an $O(\log^2 \Delta)$-approximation algorithm for MDS, running in $O(\log^2 \Delta)$ communication rounds. The Kuhn-Wattenhofer algorithm [53] used the technique of first approxi-

mately solving the LP relaxation of MDS and then doing (independent) randomized rounding on the fractional MDS solution. Moscibroda and Wattenhofer [65] use the same technique to solve the facility location problem in the $\mathcal{CONGEST}$ model; they achieve, for every constant k, an $O(\sqrt{k}(mn)^{1/\sqrt{k}}\log(m+n))$-approximation[1] in $O(k)$ communication rounds. The greedy, pruning-based algorithms that we derive in this paper improve both of the above results by logarithmic-factor and achieve optimal approximation ratios in polylogarithmic rounds. However we note that [52] presents a $O(\Delta^{c/k})$ for solving LPs of general *covering-packing* problems (including MDS) in $O(k^2)$ rounds. This, combined with a standard randomized rounding technique can achieve a $O(\Delta^{c/k}\log\Delta)$-approximation. This is asymptotically the same as the approximation factor achieved by our MDS algorithm. However, in case of Facility Location, our result is at least a logarithmic factor improvement over previously known results. Also, our results indicate that the original Jia et al. greedy algorithm [37] contained all the ingredients of a k-round MDS algorithm.

6.1.1 The Pruning Process

The *pruning process* works on a bipartite graph $H = (S, B, E)$. Think of S as the set of sellers, B as the set of buyers, and E as the set of edges representing buyers' interest in sellers products. We assume that H has no isolated vertices. Vertices in S have associated *costs* given by the function $f : S \to \mathbb{R}^+$. There is a fixed *payment* p

[1]The approximation factor, as stated in Moscibroda and Wattenhofer [65], depends also on ρ, a quantity that is a function of the numbers in the given instance (i.e., the opening costs and the connection costs). To focus more clearly on the growth of this approximation factor as a function of k, we assume that $\rho = m = n$.

that each vertex $j \in B$ makes. For any $i \in S \cup B$, let $N(i)$ denote the set of neighbors of vertex i. The costs and the payments satisfy the property that for each $i \in S$, $p \cdot |N(i)| \geq f(i)$. In other words, the cost of every vertex $i \in S$ is "paid for" by all of its neighbors. A typical iteration of the pruning process starts with vertices in some subset $S' \subseteq S$ being *opened*. Each $j \in B$ that has a neighbor in S' is then said to be *covered*. After this, several vertices are deleted from H. First, all vertices in S' and all neighbors of vertices in S' are deleted. As a result of the deletion of some vertices in B, the "paid for" condition $p \cdot |N(i)| \geq f(i)$ may no longer be satisfied for some $i \in S$. That is, the number of neighbors of i may have fallen below $f(i)/p$. All such vertices $i \in S$ are now deleted. Finally, all isolated vertices in B are deleted. This ends one iteration of the pruning process and iterations of the pruning process continue until H becomes empty. See Figure 6.1 for an illustration.

We evaluate the pruning process along two dimensions: (i) *running time*, i.e., the number of iterations of the process before H becomes empty and (ii) *cost ratio*, i.e., the ratio of the total cost of the vertices in S that were opened to the total payment by the vertices that were covered. For randomized versions of the pruning process, we defined the cost ratio as ratio of the expected total cost of the open vertices to the expected total payment by the covered vertices. Assume that Figure 6.1 illustrated the first iteration of the pruning process and in the second iteration vertex 2 (in S) was opened and as a result, vertex 3 (in B) was covered. The total cost is $2 + 3 + 1 = 6$ and the total payment is $1 \cdot 5 = 5$, yielding a cost ratio of $6/5$. Note that the pruning process, as described above, is incompletely specified in the sense

that it does not prescribe how vertices in S are picked for opening. We are interested in designing an instance of the pruning process that minimizes both running time and cost ratio. It is easy to see how to minimize one of these measures, while ignoring the other. For example, if in the first iteration all vertices in S are opened then we have a pruning process that completes in one iteration, but could have cost ratio $|S|$. On the other hand, if we opened exactly one vertex in S in each iteration, then the pruning process would have cost ratio one but could take $|S|$ iterations to terminate. In this paper we describe a randomized instance of the pruning process that, for every positive integer k, has running time $O(k)$ (with high probability) and expected cost ratio $O(m^{2/k} \cdot n^{2/k})$, where $m = |S|$ and $n = |B|$.

To see how this abstraction of the pruning process makes sense in the context of distributed greedy algorithms, the reader might want to consider the primal-dual interpretation of the greedy algorithms for MDS [11, 62] and facility location [27, 35]. According to this interpretation, the sequential greedy algorithm for MDS (in which highest degree vertices are selected) is equivalent to the primal-dual algorithm in which vertices raise their dual variable synchronously and vertices get selected as dominators in the order in which their dual constraints become tight. Thus a vertex that gets selected as a dominator is one whose cost of selection has been "paid for" by the dual variables in its closed neighborhood. Once a vertex v is selected as a dominator, all vertices u (in the closed neighborhood of v) whose dual variables were making contributions towards v "recall" their contributions from other vertices. A standard way to speed up this algorithm by allowing parallelism, is to raise the dual

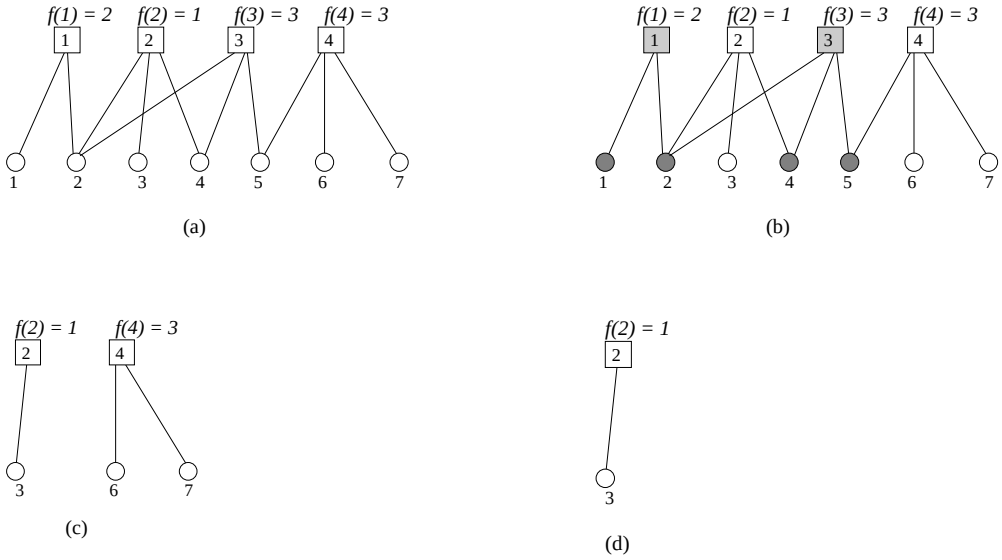


Figure 6.1: Example execution of one iteration of the pruning process. **Fig. (a)** shows the original graph H with S (square vertices) and B (round vertices). Suppose $p = 1$. **Fig. (b)** shows two vertices, 1 and 3 of S, being opened. These two open vertices cover 4 vertices in B. Then the two open vertices and the four covered vertices are deleted from H. **Fig. (c)** shows the resulting graph. Vertex 4 in S is no longer being paid for and is also deleted. This renders vertices 6 and 7 (in B) isolated and so these are also deleted. **Fig. (d)** shows the graph at the end of the iteration.

variables by a big amount in each step. As a result, when the dual variables move from one value to the next higher value, several vertices may find themselves "paid for" simultaneously. Selecting all of these vertices as dominators can lead to an arbitrarily bad dominating set. The pruning process models the problem of selecting a subset of these "paid for" vertices whose total cost does not exceed the payment made by the dual variables by too large a factor. More specifically, the vertices $i \in S$ that are opened in the pruning process are exactly the ones that are selected as dominators.

6.1.2 Related Work

As mentioned earlier, our work draws inspiration from the elegant work of Jia et al. [37]. Our results are most closely comparable to results due to Kuhn and Wattenhofer [53] on a k-round algorithm for MDS and Moscibroda and Wattenhofer [65] on a k-round algorithm for non-metric facility location. Our own recent work (see Chapter 5), presents a k-round primal-dual algorithm for metric facility location. Since there is a greedy $O(1)$-approximation for metric facility location [27], even this result can be obtained via techniques in the current paper.

Greedy algorithms for MDS and non-metric facility location are quite old [11, 62] and are often discussed in the context of set cover. Lovász [62] and Chvatal [11] present a dual-fitting-based analysis of the greedy set cover algorithm and this provides a natural primal-dual framework within which to view the algorithms. The first constant-factor approximation algorithm for metric facility location via a greedy algorithm is due to Guha and Khuller [27]. A dual-fitting-based analysis of this

algorithm along with a better approximation ratio is due to Jain et al. [35].

6.1.3 Main Results

Our main result is a randomized instance of the pruning process that, for any positive k, runs in $O(k)$ communication rounds, with high probability, uses messages of size $O(\log m)$ and has cost ratio $O((mn)^{2/k})$. Here m is the number of sellers and n is the number of buyers. Using this as a basis, we present simple, $O(k)$-round greedy algorithms for MDS and facility location. On a graph with n vertices and maximum degree Δ, for any positive k, our algorithm for MDS runs in $O(k)$ rounds, uses messages of size $O(\log n)$ and yields a solution of size $O(\Delta^{3/\sqrt{k}} \log \Delta)$ times OPT. For instances with m facilities and n clients, for any positive k, our algorithm for non-metric facility location runs in $O(k)$ rounds using messages of size $O(\log m + \log n)$ and yields a solution of size $O((mn)^{5/\sqrt{k}} \cdot \log n)$ times OPT. We also show that this algorithm produces a solution of size $O((mn)^{5/\sqrt{k}})$ times OPT if the connection costs of the instance form a metric. All our running times are "with high probability" and all our approximation ratios are in expectation. Also note that we have not paid particular attention to minimizing the constants, 3, 5, etc. that appear in the exponent of our approximation ratios.

6.2 Rapid Randomized Pruning

Here we describe an instance of the pruning process that we call *Rapid Randomized Pruning* (RRP). Recall that RRP runs on a bipartite graph $H = (S, B, E)$. Here we let $m = |S|$, $n = |B|$, and for a given positive k, we let $d = (mn)^{1/k}$. As

mentioned before, RRP is a generalization of the pruning process described in Jia et al. [37].

Algorithm 6.1 Rapid Randomized Pruning (one iteration)

Calculate Coverage. For each $j \in B$, define the *coverage*, $c(j)$ of j as $|N(j)|$.

Split Coverage. For each $i \in S$, arrange the coverage values of all the neighbors of i in non-decreasing order. Call this sequence $L(i)$ and call each item in $L(i)$ an *entry*. Let r, $1 \leq r \leq |L(i)|$, be the largest integer such that $\frac{|L(i)|-r}{r-1} \geq d$. Let split$(i)$ denote the item of rank r in $L(i)$ and let $Bot(i)$ denote the "bottom entries," i.e., the subsequence of $L(i)$ consisting of all lower ranked elements. Similarly, let $Top(i)$ denote the "top entries," i.e., the subsequence of $L(i)$ consisting of all the entries of rank greater than k.

Open Vertices. For each vertex $i \in S$, we open it independently with probability

$$P_o(i) := \min\left\{\frac{4\ln d}{\text{split}(i)}, 1\right\}.$$

Let S' be the set of vertices just opened. All neighbors of vertices in S' are said to be *covered*.

Prune Graph. Delete from H all vertices in S' and all neighbors of vertices in S'. Then delete all vertices $i \in S$ that no longer satisfy the "paid for" condition $p \cdot |N(i)| \geq f(i)$. Finally, delete all isolated vertices in B.

In order to speed up the Jia et al. algorithm to a k-round algorithm, we make two key changes to their algorithm.

1. In the **Split Coverage** step, we choose an item split(i) from $L(i)$ so that there are roughly d times as many elements in $Top(i)$ as there are in $Bot(i)$. This generalizes how Jia et al. split $L(i)$; they choose split(i) to be the median of $L(i)$ (i.e., take $d = 1$). Choosing to split in the middle provides for them the right balance between running time and cost ratio. Choosing split(i) so that

$|Top(i)| \approx d \cdot |Bot(i)|$ ensures that for larger values of d, split(i) will be a lower ranked element and as a result $1/\text{split}(i)$ will tend to be larger. The probability that vertex $i \in S$ is opened grows with $1/\text{split}(i)$ and thus, as d increases, we get a controlled boost in the probability that i opens and this in turn speeds up the pruning process, while only incurring a factor d overhead in the cost ratio.

2. In the **Open Vertices** step, we open each vertex in $i \in S$ with probability $4 \ln d/\text{split}(i)$. This provides an important additional boost to the probability.

We also note that RRP has a simple distributed implementation in a synchronous message passing model if we assume that the underlying network is H. In such an implementation, each iteration takes $O(1)$ communication rounds using messages of size $O(\log m)$.

6.2.1 Analysis

We start with a technical lemma (whose easy proof is skipped) that shows that the choice of split(i) is such that $Bot(i)$ is roughly a fraction $1/(d+1)$ of $N(i)$ in size.

Lemma 63. *The choice of split(i) is such that for any $i \in S$*

$$(d+1)|Bot(i)| \leq |N(i)| \leq (2d+3)|Bot(i)|.$$

Lemma 64. *Let f be the total cost of vertices in S that were opened during RRP and let c be the number of vertices in B that were covered during RRP. Then $E[f] \leq 8(2d+3) \ln d \cdot p \cdot E[c]$.*

Proof. For $i \in F$, let X_i be an indicator variable indicating if $i \in S$ is opened. Then,

$$E[f] = E\left[\sum_{i\in S} f(i)X_i\right] \leq E\left[\sum_{i\in S}\left(\sum_{j\in N(i)} p\right)\cdot X_i\right] \leq E\left[\sum_{i\in S}\left((2d+3)\sum_{j\in Bot(i)} p\right)\cdot X_i\right].$$

The last inequality follows from Lemma 63. Now define $b(j)$ to be the set $\{i \in N(j) \mid j \in Bot(i)\}$. By changing the order of the summations we obtain

$$E[f] \leq (2d+3)\cdot p\cdot E\left[\sum_{j\in B}\sum_{i\in b(j)} X_i\right] \leq (2d+3)\cdot p\cdot \sum_{j\in C} E\left[\sum_{i\in b(j)} X_i\right].$$

Now let $t(j)$ denote the sum $\sum_{i\in b(j)} X_i$. Then,

$$E[f] \leq (2d+3)\cdot p\cdot \sum_{j\in B} E\left[t(j)\right] = (2d+3)\cdot p\cdot \sum_{j\in B} E\left[t(j) \mid t(j) > 0\right]\cdot \text{Prob}\left[t(j) > 0\right].$$

In Lemma 65 below we show an upper bound of $8\ln d$ on $E\left[t(j) \mid t(j) > 0\right]$. Using this we obtain

$$E[f] \leq 8(2d+3)\cdot \ln d\cdot p\cdot \sum_{j\in B} \text{Prob}[t(j) > 0] \leq 8(2d+3)\cdot \ln d\cdot p\cdot E[c].$$

The last inequality above follows from the fact that if $t(j) > 0$ then vertex j is covered. □

Lemma 65. *$E[t(j) \mid t(j) > 0] \leq 8\ln d$.*

Proof. For $j \in B$ being considered in the lemma, let $W = \{i \mid j \in Bot(i)\}$. Then,

$$E[t(j) \mid t(j) > 0] = \sum_{i\in W} \text{Prob}\left[i \text{ is opened } \mid t(j) \geq 1\right] = \sum_{i\in W} \frac{\text{Prob}[i \text{ is opened }]}{\text{Prob}[t(j) \geq 1]}.$$

Now recall that a vertex $i \in S$ is opened with probability $P_o(i) = \min\left\{\frac{4\ln d}{\text{split}(i)}, 1\right\}$. Let $q(i) = \frac{1}{\text{split}(i)}$. Then $q(i) \leq P_o(i) \leq 4\ln d\cdot q(i)$. Thus,

$$E[t(j) \mid t(j) > 0] = \frac{\sum_{i\in W} P_o(i)}{1-\prod_{i\in W}(1-P_o(i))} \leq \frac{4\ln d\sum_{i\in W} q(i)}{1-\prod_{i\in W}(1-q(i))}.$$

Using the algebraic inequality $\prod_{i\in W}(1-q(i)) \leq 1-\sum_{i\in W} q(i)+\sum_{x,y\in W,x\neq y} q(x)q(y)$, we get

$$E[t(j) \mid t(j) > 0] \leq 4 \ln d \left(\frac{\sum_{i\in W} q(i)}{\sum_{i\in W} q(i) - \sum_{x,y\in W,x\neq y} q(x)q(y)} \right). \tag{6.1}$$

Now note that for $i \in W$, since $j \in Bot(i)$, we have $q(i) = \frac{1}{\text{split}_{(i)}} \leq \frac{1}{|N(j)|} \leq \frac{1}{|W|}$. Thus $\sum_{i\in W} q(i) \leq 1$ and from this we obtain the inequality $\sum_{x,y\in W,x\neq y} q(x)q(y) \leq \sum_{i\in W} q(i)/2$. Substituting this into (6.1), we obtain $E[t(j) \mid t(j) > 0] \leq 8 \ln d$. □

Note that $E[f]/(p \cdot E[c])$ is the cost ratio of RRP. Therefore Lemma 64 leads to the following.

Corollary 66. *The cost ratio of RRP is* $8(2d+3)\ln d = O(d^2)$.

Next we show that RRP terminates in k rounds with high probability. Recall that for each $i \in S$, each item in the sequence $S(i)$ is called an entry. As vertices get deleted from H the total number of entries, i.e., $\sum_{i\in F} |L(i)|$, falls. Determining this rate of decrease is one way of bounding the running time of RRP.

Fix a particular iteration of RRP. For a vertex $i \in S$ and a vertex $j \in B$, we say i is *good* for j if $j \in Top(i)$. Also, we say a vertex $j \in B$ is *nice* if at least $\frac{|N(j)|}{4}$ facilities in $N(j)$ are good for j. For a vertex $j \in B$, let $P_c(j)$ denote the probability that j will be covered in this iteration. The following lemma places a lower bound on the $P_c(j)$ for nice vertices j.

Lemma 67. *If j is a nice vertex, then $P_c(j) > 1 - \frac{1}{d}$.*

Proof. Recall that, for a vertex $i \in S$,

$$P_o(i) = \min\left\{\frac{4\ln d}{\text{split}(i)}, 1\right\}$$

Now consider an $i \in N(j)$ and suppose that i is good for j. If $P_o(i) = 1$ then $P_c(j) = 1$. Otherwise,

$$P_o(i) = \frac{4\ln d}{\text{split}(i)} \geq \frac{4\ln d}{|N(j)|}$$

Thus for a nice vertex j, we have

$$\begin{aligned}
P_c(j) &= 1 - \prod_{i \in N(j)} (1 - P_o(i)) \\
&\geq 1 - \prod_{i \text{ is good for } j} (1 - P_o(i)) \\
&\geq 1 - \left(1 - \frac{4\ln d}{|N(j)|}\right)^{\frac{|N(j)|}{4}} \\
&> 1 - \left(e^{\frac{-4\ln d}{|N(j)|}}\right)^{\frac{|N(j)|}{4}} \\
&= 1 - \frac{1}{d}
\end{aligned}$$

□

For each $i \in S$, entries in $L(i)$ are classified in two ways. An entry j in $L(i)$ is a *top* entry if j occurs in $Top(i)$; otherwise j is a *bottom* entry. An entry j in $L(i)$ is a *nice* entry if vertex j is nice; otherwise entry j is a *non-nice* entry.

Lemma 68. *At least* $(1 - \frac{4}{3d})$*-fraction of the total entries are nice top entries.*

Proof. Let x denote the total number of entries, i.e., $\sum_{i \in S} |L(i)|$, and w denote the number of non-nice top entries. By definition, for a nice entry j, there are at least $\frac{|N(j)|}{4}$ vertices in $N(j)$ such that $j \in Top(i)$. Hence, there are more than $3w$ non-nice

bottom entries. We know that the total number of bottom entries is $x/(d+1)$. Using the fact that this is more than $3w$, we obtain the inequality $w < x/3d$. Thus, the total number of nice top entries is

$$(1 - \frac{1}{d})x - w \geq (1 - \frac{1}{d})x - (\frac{1}{3d})x = (1 - \frac{4}{3d})x.$$

□

Lemma 69. *Let Φ and Φ' denote the total number of covered entries, respectively before and after an iteration of RRP. Then,*

$$E[\Phi'] \leq \frac{\Phi}{d/3}.$$

Proof. Let E denote the set of entries before the iteration. Thus, $\Phi = |E|$. Let E_1 be the subset of E consisting of nice top entries; denote $E \setminus E_1$ by $\overline{E_1}$. Then,

$$E[\Phi'] = \sum_{j \in E} Prob[\text{entry } j \text{ is deleted}] = \sum_{j \in E_1} \text{Prob[entry } j \text{ is deleted]} + \sum_{j \in \overline{E_1}} \text{Prob[entry } j \text{ is deleted]}.$$

The first term on the right hand side is bounded above by $\Phi \cdot \frac{1}{d}$, since $|E_1| \leq \Phi$ and by Lemma 67, the probability that a nice top entry is deleted in an iteration is at most $\frac{1}{d}$. The second term is bounded above by $\Phi \cdot \frac{4}{3d}$ since the size of $\overline{E_1}$ is at most the fraction $\frac{4}{3d}$ of Φ according to Lemma 68. These bounds yield the claimed probabilistic recurrence. □

We now invoke a result on probabilistic recurrence relations due to Karp [42] to obtain a bound on the running time of RRP. Let, $T(P)$ denote the number of rounds remaining when the total number of entries at the beginning of the round is P. Note

that, $T(P)$ is a random variable. We have the following probabilistic recurrence for $T(\cdot)$.

$$T(P) = 1 + T(P'),$$

where, P' is the random variable denoting the number of entries at the end of the round. P can be at most mn at the beginning. Also, $T(P) = 0$ for $P < c$ for some positive constant c. Note that the solution to the deterministic counter part of this probabilistic recurrence relation, $\tau(P) = 1 + \tau(\frac{P}{d/3})$, is αk for some constant $\alpha > 0$. Hence, applying Theorem 1.3 of [42] and noting that the maximum number of entries can be mn, we get the inequality

$$\text{Prob}[T(mn) \geq \alpha k + w] \leq \left(\frac{3}{d}\right)^w.$$

If we set $w = \Theta(k)$, the right hand side of the above inequality is $O\left(\frac{1}{\text{poly}(mn)}\right)$. Hence the running time RRP is $O(k)$ with high probability. Here, the running time denotes the number of iterations of RRP before it terminates. Since each iteration can be implemented in $O(1)$ communication rounds in a synchronous message passing model, we obtain the following theorem.

Theorem 70. *For any positive k, RRP runs in $O(k)$ rounds (with high probability), yielding a cost ratio of $O(m^{\frac{2}{k}} n^{\frac{2}{k}})$.*

6.3 Facility Location

A simple, greedy approximation algorithm for facility location is obtained by repeatedly picking the most *effective stars*. For each facility $i \in F$ and subset of

clients $C' \subseteq C$, we call the ordered pair (i, C') a *star*. The *effectiveness* of a star (i, C'), denoted $e(i, C')$, is defined as

$$e(i, C') = \frac{f_i + \sum_{j \in C'} c_{ij}}{|C'|}.$$

The greedy algorithm for facility location simply picks the most effective star iteratively. Once a star (i, C') is picked, facility i is opened, all clients in C' are connected to i. The clients in C' are no longer considered, but facility i still participates in the algorithm, but with its opening cost reset to 0. This algorithm is just an instance of the greedy set cover algorithm with C being the ground set and each star (i, C') covering the subset $C' \subseteq C$ and having weight $f_i + \sum_{j \in C'} c_{ij}$. By the well-known analysis of the greedy set cover algorithm [62, 11] this produces an $O(\log n)$-approximation, where n is number of clients.

There is a simple way to view this greedy algorithm in the *primal-dual* framework. To see this consider the following Integer Program (IP) for facility location. Here y_i indicates whether facility i is open and x_{ij} indicates if client j is connected to facility i. The first set of constraints ensure that every client is connected to a facility and the second set of constraints guarantee that each client is connected to an open facility. As is standard, we work with the LP-relaxation (LP) of the IP obtained by replacing the integrality constraints by $x_{ij} \geq 0$ for all $i \in F$, $j \in C$ and $y_i \geq 0$ for all $i \in F$. We also construct the dual (DP) of this LP-relaxation. For convenience of the reader we restate the IP and the DP below:

Facility Location IP

$$\begin{aligned}
\text{minimize} \quad & \sum_{i \in F, j \in C} c_{ij} \cdot x_{ij} + \sum_{i \in F} f_i \cdot y_i \\
\text{subject to} \quad & \sum_{i \in F} x_{ij} \geq 1, && j \in C \\
& y_i - x_{ij} \geq 0, && i \in F, j \in C \\
& x_{ij} \in \{0, 1\}, && i \in F, j \in C \\
& y_i \in \{0, 1\}, && i \in F
\end{aligned}$$

Dual of Facility Location LP

$$\begin{aligned}
\text{maximize} \quad & \sum_{j \in C} \alpha_j \\
\text{subject to} \quad & \alpha_j - \beta_{ij} \leq c_{ij}, && i \in F,\ j \in C \\
& \sum_{j \in C} \beta_{ij} \leq f_i, && i \in F \\
& \alpha_j \geq 0, && j \in C \\
& \beta_{ij} \geq 0, && i \in F,\ j \in C
\end{aligned}$$

The dual variable α_j can be interpreted as the amount that client j is willing to pay in order to connect to a facility. Of this amount, c_{ij} goes towards paying for connecting to facility i, whereas the "extra", namely β_{ij}, is seen as the *payment* of client j towards opening facility i. Now consider the following procedure that maintains an integral primal solution (implicitly given by a set F' of open facilities) and an *infeasible* dual solution (α_j, β_{ij}).

1. Initialize $F' \leftarrow \emptyset$ (open facilities), $C' \leftarrow C$ (active clients), for all $j \in C$: $\alpha_j \leftarrow 0$ and for all $i \in F, j \in C$: $\beta_{ij} \leftarrow 0$.

2. While $C' \neq 0$, uniformly raise the duals α_j ($\forall j \in C'$) and β_{ij} ($\forall \{i, j\}$ pair such that $j \in C'$ and $i \in F \backslash F'$ and $\alpha_j \geq c_{ij}$) until one of the following happens:

- $\alpha_j = c_{ij}$ for some $j \in C'$ and $i \in F'$. In this case, set $C' \leftarrow C' \backslash j$ and $\beta_{ij} \leftarrow 0$ for all $iinF$. *Client j is assigned to already open facility i.*
- $\sum_j \beta_{ij} = f_i$ for some $i \in F \backslash F'$. In this case, let $C(i) = \{j \in C' | \beta_{ij} > 0\}$. Set $\beta i' j \leftarrow 0$ for all $i' \in F \backslash \{i\}$ and $j \in C(i)$. Set $F' \leftarrow F' \cup \{i\}$ and $C' \leftarrow C' \backslash C(i)$. *Clients in $C(i)$ are assigned to newly open facility i.*

Note that in the second type of event (where i is a newly opened facility), any $j \in C(i)$ withdraws its payments from any other facility that j may have positively contributed to. It is easy to check that the outcome of this process are identical to the outcomes of the greedy algorithm that picks the most effective star. In particular, the "stars" induced by the primal-dual algorithm and the order in which they arise is exactly as the aforementioned greedy algorithm.

6.3.1 Distributed Facility Location

Our goal is the same as that of Moscibroda and Wattenhofer [65]: to design a distributed approximation algorithm for facility location in the $\mathcal{CONGEST}$ model assuming that the underlying network is the complete bipartite graph induced by facilities and clients. The $\mathcal{CONGEST}$ model of distributed computing is a message passing model, where nodes communicate with each other in synchronous rounds. In each round, a node is allowed to send a message of $O(\log n)$ bits to its neighbors. It is a realistic bound as $\log n$ bits are necessary to assign unique IDs to n nodes. This bound on message size is also significant in terms of the hardness of the problem. If nodes were allowed to send messages of unlimited size, they could exchange all

the neighborhood information in each round. In that case it would take each node at most D rounds to have all the knowledge about the whole network, where D is the diameter of the network graph. And if we were allowed D rounds of message communication, designing a distributed algorithm would be trivial. Given that in our case, the diameter of the input graph is 2, this restriction on message size is what primarily makes the problem much more challenging.

The second challenge is that any fast distributed implementation of a greedy algorithm needs to pick many "good" choices instead of the one "best" choice at every step (round). In case of a primal-dual algorithm this translates into increasing the dual variables faster, for example, in a geometric manner (say, by a factor b). Of course, the larger b is, the less closely we expect to approximate the behavior of the sequential algorithm. Recall that in the sequential primal-dual algorithm, each client pays positively to exactly one open facility as it withdraws its payments from all the other facilities as soon as it is assigned to some open facility. But in the distributed algorithm, clients not only can end up paying b times more than what was necessary, but also may end up positively paying to many facilities. This is where the pruning process is necessary. We use RRP to "undo" some of the "imperfect" choices that we have made. During each round some temporarily open facilities can be permanently opened. Clients that paid positively towards any of these facilities will be connected to them and withdraw their contribution from all other facilities. Note that the clients do not get a chance to withdraw their contributions from the facilities that opened in the same round.

6.3.1.1 Algorithm

Our distributed algorithm combines the primal-dual framework with RRP Algorithm. Refer to Algorithm 6.2 for an overview.

Algorithm 6.2 Facility Location

1: **Initialization Stage.** (runs in $O(1)$ rounds). This is similar to the description in Chapter 5. We identify *low-paying* clients and *cheap* facilities. Cheap facilities are opened and low-paying clients are connected to cheap facilities. All the remaining clients are denoted *unassigned* and their initial α_j-values are computed. See below for details.

2: **Primal-Dual Stage.** (runs in k phases, each phase runs in $O(k)$ rounds whp). Each unassigned client j increases its dual variable (i.e., α_j) every phase by a multiplicative factor $b = n^3/k$ until a facility i is found such that $\alpha_j \geq c_{ij}$ and the facility opening cost f_i is *paid-for* by some clients. Note that these *payments* are tracked by the β_{ij} values. Facility i is declared *temporarily open* and included in F_t and each j with $\alpha_j \geq c_{ij}$ and $\beta_{ij} \geq 0$ is included in $C(i)$. Let C_t be the set of clients such that $j \in C(i)$ for some $i \in F_t$ and E_t be the set of edges from each facility $i \in F_t$ to each client $j \in C(i)$. We run RRP (Algorithm 6.1) on the bipartite graph $H = (F_t, C_t, E_t)$ and decide which facilities in F_t will open permanently. Any client that is in $C(i)$ for an open facility i will be considered *assigned*. The phase ends here and the dual variables of yet-unassigned clients are increased again to start the next phase. See Algorithms 6.3 and 6.4 for a distributed implementation of the primal-dual stage.

Initialization. Set $\alpha_j := \frac{1}{n} \min_i(f_i + c_{ij})$ for all $j \in C$. For each $j \in C$, let i^* be such that $f_{i^*} + c_{i^*j} = \min_i(f_i + c_{ij})$, and set $\phi(j) := i^*$. For each $i \in F$, $j \in C$, set $\beta_{ij} := \max\{\alpha_j - c_{ij}, 0\}$. Now, let us define $\alpha^* = \max_j \alpha_j$ and let $j^* = \arg\max \alpha_j$. Any $j \in C$ with $\alpha_j \leq \alpha^*/n$ will be called a *low-paying* client and the facility $\phi(j)$ (for low-paying j) will be called a *cheap* facility. Let OPT be the cost of an optimal solution to the facility location problem.

Lemma 71. *The total cost of opening all cheap facilities and connecting each low-paying client j to facility $\phi(j)$ is at most OPT.*

This proof is the same as in the metric case described in Chapter 5.2.2. It can also be easily verified that the aforementioned process of identifying the low-paying clients and cheap facilities can be accomplished in just $O(1)$ rounds of communication with messages of size $O(\log n)$.

Primal-Dual. The primal-dual phase is broken up in k *phases*. Each execution of the for-loops ($p = 1$ to k) at the beginning of Algorithms 6.3 and 6.4 describes one phase. The cheap facilities are already opened in the Initialization phase and the low-paying clients do not participate in this process. Let $b = n^{\frac{3}{k}}$. Every client, that is not low-paying starts with an initial α_j value set at the largest power of b that is less than or equal to α^*/n^2. In each phase the α_j values are increased geometrically by factor b and the set of facilities that are paid for are temporarily open. Then we run the pruning algorithm described below and a subset of these facilities are permanently opened.

Let us fix a phase (say, $p = t$) and let F_t be the set of temporarily open facilities in that phase. Let, H denote the graph induced by the *stars* of all facilities in F_t. Recall that, a client j is said to be in the star $C(i)$ of a facility i, if it has positively contributed towards that facility. The stars are identified via 3 rounds of message communication between the facilities and the clients. Note that due to the geometric increase of dual variables, a client j can belong to multiple stars. Finally,

we run the RRP algorithm (Algorithm 6.1) on induced (bipartite) graph H.

Algorithm 6.3 Primal-Dual Stage: Facilities

1: **Init:** States of all facilities that are not *cheap* are *active*, $C(i) \leftarrow \phi$
2: **for** $(p = 1, 2, \ldots, k)$ **do**
3: `Receive` α_j from all $j|state(j) = unassigned$
4: **if** $(state(i) = open)$ **then**
5: **for** $(\forall j \in C)$ **do**
6: **if** $(\alpha_j \geq c_{ij}$ and $state(j) = unassigned)$ **then**
7: $C(i) \leftarrow C(i) \cup \{j\}$
8: **if** $(state(i) = active)$ **then**
9: **for** $(\forall j \in C)$ **do**
10: $\beta_{ij} \leftarrow 0$
11: **if** $(state(j) = unassigned)$ **then**
12: $\beta_{ij} \leftarrow \max\{0, \alpha_j - c_{ij}\}$
13: **if** $(\sum_j \beta_{ij} \geq f_i)$ **then**
14: $state(i) \leftarrow paidfor$
15: **for** $(\forall j|state(j) = unassigned)$ **do**
16: **if** $(\beta_{ij} > 0)$ **then**
17: $C(i) \leftarrow C(i) \cup \{j\}$
18: `Send` $state(i)$ to all $j \in C(i)$
19: **if** $(state(i) = paidfor)$ **then**
20: i participates in RRP with neighbors $C(i)$
21: **if** (RRP selects i) **then**
22: $state(i) \leftarrow open$
23: `Send` $state(i)$ to all $j \in C(i)$
24: **else**
25: $state(i) \leftarrow active$

6.3.1.2 Analysis

This section is devoted to the proof of the following theorem. Note that if we set $k = \Theta(\log(mn))$, we obtain an $O(\log n)$-approximation in $O(\log^2(mn))$ rounds whp.

Algorithm 6.4 Primal-Dual Stage: Clients

Init: States of all clients that are not *lowpaying* are *unassigned*, $b \leftarrow \frac{n^3}{k}$, $\alpha_j \leftarrow$ Largest power of b less than α^*/n^2
for $(p = 1, 2, \ldots, k)$ **do**
  if $(state(j) = unassigned)$ **then**
    `Send` α_j to all $i \in F$
    $\alpha_j \leftarrow b \cdot \alpha_j$
  `Receive` $state(i)$ from all possible i
  {Note: j will only *hear* all i such that $j \in C(i)$}
  if $(\exists i | state(i) = open)$ **then**
    $state(j) \leftarrow assigned$
  else
    $F(j) \leftarrow \{i | state(i) = paidfor\}$
  if $(state(j) = unassigned)$ **then**
    j participates in RRP with neighbors $F(j)$
    {Note: If $F(j) = \phi$, j does not play any role in the outcome of the RRP}
    `Receive` $state(i)$ from all possible i
    if $(\exists i | state(i) = open)$ **then**
      $state(j) \leftarrow assigned$

Theorem 72. *For any positive k, our distributed algorithm runs in $O(k^2)$ communication rounds (with high probability) and yields a solution of expected cost $O((mn)^{\frac{5}{k}} H_n) \cdot OPT$. All messages are of size $O(\log m)$.*

Approximation Ratio. Let a_j be the final α_j value of client j. Note that a_j must be a power of b. In the primal-dual scheme the sum of all the a_j values can be seen as the dual objective function. We show a logarithmic bound on this sum in Lemma 73.

Lemma 73. $\sum_{j \in \mathcal{C}} a_j \leq bH_n \cdot OPT$.

Proof. Consider a facility i in the optimal OPT and the star $(i, C(i))$. Recall that, s_i denote the total cost of opening the star, i.e. $s_i = f_i + \sum_{j \in C(i)} c_{ij}$ and let $l = \mid C(i) \mid$.

We sort all $j \in C(i)$ in non-decreasing order of a_j values, i.e. in the order they were assigned. If two or more clients were assigned in the same phase, they will have the same a_j value. Let, $j_1, j_2, \ldots, j_l$ be the sequence obtained this way. First, we claim $a_{j_k} \leq b \cdot \frac{s_i}{l-k+1}$, $1 \leq k \leq l$. We prove this claim by induction.

We start by considering the base case (j_1) and showing that $a_{j_1} \leq b\frac{s_i}{l}$. Let, j_1 was assigned in phase p. If a_{j_1} exceeds the value $b \cdot \frac{s_i}{l}$ then, at the end of phase p, the payment from all the remaining $l-1$ clients also exceed $b \cdot \frac{s_i}{l}$ by the end of phase p. This means the star $(i, C(i))$ should have been completely paid for and all $j \in C(i)$ should have been assigned to i by the end of phase $p-1$, which presents a contradiction. Hence $a_{j_1} \leq b \cdot \frac{s_i}{l}$.

Let our inductive hypothesis be, for some k, $1 \leq k < l$, $a_{j_k} \leq b \cdot \frac{s_i}{l-k+1}$. We show, $a_{j_{k+1}} \leq b \cdot \frac{s_i}{l-k}$. By definition, the client j_{k+1} cannot be connected before j_k is connected. However, they both can be connected in the same round. If they are connected in the same round, $a_{j_{k+1}} = a_{j_k} \leq b \cdot \frac{s_i}{l-k+1} \leq b \cdot \frac{s_i}{l-k}$. If the client j_{k+1} is connected in a round after j_k is connected, let us consider the set of clients $C'(i) = C(i) \backslash \{j_1 \cup j_2 \cup \cdots \cup j_k\}$. If $a_{j_{k+1}}$ reach $b \cdot \frac{s_i}{l-k}$ in some phase p', all the remaining α_{j_r} values ($k < r \leq l$ will also reach $b \cdot \frac{s_i}{l-k}$ in that phase. Hence, at the end of phase p' the total contribution from these $l-k$ clients is going to be $b \cdot s_i$, which will be enough to open the star $(i, C(i))$ and hence, enough to open the star $(i, C'(i))$. Also, j_k must have paid for its connection cost to i by this phase. Thus, j_{k+1} will have to be connected to i before the phase can end and $\alpha_{j_{k+1}}$ will not be raised further. Hence, $a_{j_{k+1}} \leq b \cdot \frac{s_i}{l-k}$. This concludes the inductive proof that $a_{j_k} \leq b \cdot \frac{s_i}{l-k+1}$.

Hence we have,

$$\sum_{j \in C(i)} a_j = b \sum_{k=1}^{l} a_{j_k} \leq b(\frac{s_i}{l} + \frac{s_i}{l-1} + \ldots \frac{s_i}{1}) = bH_l \cdot s_i \leq bH_n \cdot s_i$$

Summing over all the stars $(i, C(i))$ in OPT, we have

$$\sum_{j \in \mathcal{C}} a_j \leq b \sum_{i \in OPT} \sum_{j \in C(i)} a_j \leq b \sum_{i \in OPT} H_n \cdot s_i = bH_n \cdot OPT$$

□

Lemma 74. *Let R be the set of stars open in a phase and J be the clients that get connected in this particular phase. Let s_R denote the total cost of the set pf stars R. Then $E[s_R] \leq 8d \ln d \cdot E[\sum_{j \in J} a_j]$*

Proof. For any facility $i \in R$, $\hat{d}(i) \geq \frac{s_i}{b|C(i)|}$. So we have,

$$s_R = \sum_{i \in R} s_i \leq \sum_{i \in R} \sum_{j \in C(i)} a_j$$

The rest follows from Lemma 64 in Section 6.2.1. □

Lemma 74 (adding over all phases) and Lemma 73 yield the expectation results in Theorem 72. The high-probability results are obtained from this via Chernoff Bounds. We refer the reader to [37] for the details.

Running Time. Next we will verify that, after the initialization phase, the primal-dual phase needs at most k phases to connect all the clients. We already know that the RRP algorithm is executed in each phase, which takes $O(k)$ rounds with high probability. Also it takes a constant number of $O(\log n)$ bit messages per client per phase. The following theorem summarizes the claims in this section.

Theorem 75. *Our distributed algorithm in a synchronous message passing model terminates in $O(k^2)$ rounds with high probability with messages of size $O(\log n)$.*

First, is easy to see that the initialization phase takes only a constant number of message transactions per facility or client and, assuming the input constants can be represented by $O(\log n)$ bits, each of those messages are at most of size $O(\log n)$ bits. Hence, focus on the primal-dual phase.

Lemma 76. *The primal-dual phase contains at most k phases.*

Proof. For a client j, let α_{min} denote the initial value of α_j. We know $\alpha_{min} \geq \alpha^*/n^2$ and hence $\alpha_{min} \cdot n^3 > \min_i(f_i + c_{ij})$. Therefore, if client j were to run k phases, then its α_j value would grow from α_{min} to $\alpha_{min} \cdot (n^{3/k})^k = \alpha_{min} \cdot n^3 > \min_i(f_i + c_{ij})$. At this point the α_j-value is more than sufficient to pay for the opening cost of a facility i all by itself and therefore if client j is not connected to any other facility, it would have to connect facility i by the end of this phase. □

Lemma 76, along with the fact that RRP terminates in $O(k)$ rounds with high probability, gives us the proof of Theorem 72.

6.3.2 Metric Facility Location

Based on the nature of the connection costs, the facility location problem has two main versions: the *non-metric* version and the *metric* version. The connection costs in a facility location instance are said to satisfy the *triangle inequality* if for any $i, i' \in F$ and $j, j' \in C$, $c(i,j) \leq c(i,j') + c(i',j') + c(i',j)$. In the *metric facility location* problem the connection costs satisfy the triangle inequality; when they

don't, we have the more general *non-metric facility location* problem. This distinction is important from an approximation point of view because there are a number of sequential constant-factor approximation algorithms for the metric facility location problem ([9, 36, 75] are some examples), whereas for the non-metric facility location problem, the best known approximation factor is $O(\log n)$ and this is optimal [33, 60]. In Chapter 5, we presented the first constant factor distributed algorithm (in the $\mathcal{CONGEST}$ model) for the metric case. We claim that, in the metric case, our hyper-local algorithm achieves a constant-factor approximation in polylogarithmic round. More generally, the approximation factor of our hyper-local algorithm can improved by a factor of $O(\log n)$ if the input instance is a metric. This can be accounted for by Lemma 78, which is the metric version of Lemma 73. But first we need to prove the following helping lemma.

Lemma 77. *For a facility i and clients $j, j' \in C$ such that $a_j \geq c_{ij}$ and $a_{j'} \geq c_{ij'}$, $\frac{a_j}{b} \leq c_{ij} + 2a_{j'}$.*

Proof. If $a_j \leq ba_{j'}$, the proof is trivial. Let us assume, $a_j > ba_{j'}$. In this case, j was still unassigned when j' got assigned in phase p (to some facility i'). However as per our assumption $a_j > ba_{j'}$, j could not have been assigned phase p. Let j was assigned (to i) at a later phase p'. By the end of the phase $p' - 1$, i' was completely paid for and the payment of j has reached a_j/b. Hence $a_j/b \leq c_{i'j}$, as otherwise $\beta_{i'j}$ would be positive and j would be connected to i' instead. Using triangle inequality,

$$c_{i'j} \leq c_{i'j'} + c_{ij'} + c_{ij} \leq a_{j'} + a_{j'} + c_{ij} = 2a_{j'} + c_{ij}.$$

Hence, $\frac{a_j}{b} \leq c_{i'j} \leq 2a_{j'} + c_{ij}$. □

Lemma 78. *If the connection costs form a metric,* $\sum_{j \in \mathcal{C}} a_j \leq 3b \cdot OPT$.

Proof. As in Lemma 73, let us start with a facility $i \in OPT$ and the star $(i, C(i))$. We intend to show that

$$\sum_{j \in C(i)} a_j \leq 3bs_i.$$

Again we let $j_1, j_2, \ldots, j_l$ be the sequence obtained by sorting all $j \in C(i)$ in non-decreasing order of a_j values, i.e. in the order they were assigned to some facility. Note that a_j values will be same for all the clients assigned in the same phase of the primal-dual stage. Let j_1 was assigned in phase p. First we assume that $\sum_{j \in C(i)} \max\{0, a_{j_1} - c_{ij}\} \leq f_i$. Because otherwise, at the end of phase p, i will have enough payments from all $j \in C(i)$ to be opened and all $j \in C(i)$ would be assigned to i. In that case, $\sum_{j \in C(i)} a_j \leq bs_i$ and the proof is trivial. Hence, we consider the case where $\sum_{j \in C(i)} \max\{0, a_{j_1} - c_{ij}\} \leq f_i$.

$$\begin{aligned}
\sum_{j \in C(i)} a_j &\leq b \sum_{j \in C(i)} (2a_{j_1} + c_{ij}) \\
&= 3b \sum_{j \in C(i)} c_{ij} + 2b \sum_{j \in C(i)} (a_{j_1} - c_{ij}) \\
&\leq 3b \sum_{j \in C(i)} c_{ij} + 2bf_i \\
&\leq 3b\Big(\sum_{j \in C(i)} c_{ij} + f_i \Big)
\end{aligned}$$

This completes the proof. The first inequality is obtained by substituting $j' = j_1$ in Lemma 77. □

As in the non-metric case, Lemma 74 (adding over all phases) and Lemma 78 yield the metric results in Theorem 79.

Theorem 79. *If the connection costs form a metric, then the output of our distributed algorithm has expected cost* $O((mn)^{\frac{5}{k}})$ *times* OPT.

6.3.3 Minimum Dominating Set

A *dominating set* of a graph $G = (V, E)$ is defined to be a set $D \subseteq V$, such that each vertex j is either in D, or there exists at least one vertex $i \in D$ such that $(i, j) \in E$. The *minimum dominating set* problem (MDS) seeks to find the smallest such D. MDS can be seen as a special case of the facility location problem, where each vertex is as facility (with unit opening cost), as well as a client and all connection costs are zero. However let us consider the MDS problem independently and examine how it can fit into our RRP/Primal-dual framework. Note that the underlying network in this case is slightly different than the facility location cases. Here each vertex can be a seller if it is paid for by other vertices in its closed neighborhood. Hence, once it is reduced to the stage when the RRP routine is called, we can "view" it as a bipartite graph. Note that unlike the facility location instance, this virtual bipartite graph is not necessarily a complete bipartite graph. This could pose some challenges for a distributed implementation. However, as can be seen (both intuitively and from the IP formulation below) that nodes only need information from their direct neighbors. We also assume that each node has knowledge of Δ, the maximum closed degree of any vertex in the graph.

To begin with let us describe the integer program (IP_{MDS}) representing MDS.

$$\begin{aligned} \text{minimize} \quad & \sum_{i \in V} x_i & \\ \text{subject to} \quad & \sum_{i \in N[j]} x_i \geq 1, & j \in V \\ & x_i \in \{0, 1\}, & i \in V \end{aligned}$$

Here, $N[j]$ denotes the closed neighborhood of j. Note that, any $j \in V$ is capable of dominating itself. The variable $x_i = 1$ indicates vertex i is selected to be a dominator. Let us consider the natural LP-relaxation (LP_{MDS}) and its dual (DP_{MDS}). We do not need an initialization phase. Each vertex starts their dual variable (payment) at $1/\Delta$ and increase it by a factor of $b = \Delta^{1/k}$ in every phase, where Δ is the maximum degree of the graph. At the beginning of a phase each vertex checks if the sum payments from its neighbors reach or exceed 1 and temporarily opens itself if that is the case. Let F_t be the set of temporarily open vertices and C_t be the yet uncovered neighbors of vertices in F_t. This sets us up for the RRP, which takes $O(k)$ rounds as discussed earlier. Note that after k phases the payment of each yet uncovered vertex reaches 1, which is enough to temporarily open itself. Such vertices will open with probability 1 during RRP. Hence, following the same analysis, we obtain a $O(\Delta^{\frac{3}{k}} \log \Delta) \cdot OPT$ in $O(k^2)$ rounds with high probability.

6.4 Conclusion

In this paper we introduced the primal-dual approach, aided by the RRP algorithm, to be a valuable tool for obtaining k-round algorithms for some covering problems. We believe this opens up the possibility of obtaining such algorithms for more *covering-packing* problems. Please see Chapter 7 for more details.

CHAPTER 7
FUTURE WORK

For the remainder of the thesis, we intend to focus on two problems. Both of these problems are related to our current work on facility location and we have some intuition and ideas on how to effectively approach these problems. We conclude this report by describing these two problems, the challenges we face and our ideas on how to overcome them.

7.1 k-round Algorithms for Other Covering-Packing Problems

Our current result is achieved by using a primal-dual algorithm. It is clear from recent results (by us [69, 70] and others [54, 65]) that the primal-dual scheme is an effective tool for exposing the inherent trade-off between the approximation factor and the running-time of distributed algorithms, and as a result, achieving non-trivial approximation factors even in small number of rounds. We believe this opens up the possibility of obtaining such algorithms for more *covering-packing* problems. Usually, if we have a covering (minimization) problem (e.g. *minimum set cover*, *minimum vertex cover*, *minimum edge cover*), the dual of it will be a packing (maximization) problem (e.g. *maximum set packing*, *maximum matching*, *maximum independent set* respectively). Covering problems can be reduced (after relaxation) to the following general LP form:

minimize $\quad \mathbf{b^T x}$

subject to $\quad \mathbf{A^T x} \geq \mathbf{c}$

$$\mathbf{x} \geq \mathbf{0}$$

such that matrix $\mathbf{A}$ and vectors $\mathbf{b}$ and $\mathbf{c}$ are non-negative. The dual takes the general form:

$$\begin{aligned} \text{maximize} \quad & \mathbf{c^T y} \\ \text{subject to} \quad & \mathbf{A y} \leq \mathbf{b} \\ & \mathbf{y} \leq \mathbf{0} \end{aligned}$$

In the future, we would like to claim that fast increase of the dual variables ($\mathbf{y}$), each increase followed by fast pruning of the *covering* set would yield non-trivial approximation factors in k rounds. Furthermore, for a polylogarithmic value of k we expect the approximation factor to reach or improve the best known results so far.

7.2 Capacitated Facility Location on UDGs

We also want to venture into the capacitated versions of facility location problems. There are two types of *capacitated facility location problems.* In *hard capacitated facility location problem* each facility i, if opened, can serve at most u_i clients. The term u_i denotes the *capacity* of facility i. As usual, F being the set of facilities and C being the set of clients, the goal is to find a set of facilities $I \subseteq F$ to open and a function $\phi : C \to I$ that assigns every client to an open facility, while maintaining the capacity constraint for each facility, so as to minimize $\sum_{i \in I} f(i) + \sum_{j \in C} c(j, \phi(j))$.

However, our intention is to focus on the *soft capacitated facility location.* Facilities have limited capacities in this variant as well. But we allow each facility to

be opened an unbounded number of times if necessary. If facility i is opened y_i times, it can serve at most $u_i y_i$ clients. From the primal-dual standpoint, this provision introduces the following additional constraint to the LP-relaxation:

$$\forall i \in F : u_i y_i - \sum_{j \in C} x_{ij} \geq 0.$$

Jain and Vazirani [36] obtained a 4-approximation for the metric case via primal-dual method. But we want to obtain a $O(1)$-approximation for UDGs. To our knowledge, no constant factor approximation exists for this problem in UDGs, even in the sequential setting.

We have already obtained a $O(1)$-approximation for `UDG-FacLoc` (described in Chapter 4). The idea behind that result was to run a primal-dual algorithm followed by pruning of facilities. If, due to pruning, a client looses its connections to all temporarily open facilities, then we *reconnect* that client. To ensure a small (constant-factor) overhead of reconnection, the final step is based on a $O(1)$-approximation [2] for `UDG-WMDS`. Following the same footsteps, we hope to use a primal-dual algorithm in order to obtain a $O(1)$-approximation for the soft capacitated facility location problem in UDGs (`UDG-CapFacLoc`). The rough idea is as follows:

- Run the primal-dual algorithm for `CapFacLoc`. We can use an existing algorithm [36]. Let F_p be the set of open facilities.

- Compute some capacitated version of `UDG-MDS` and open a facility at each node belonging to that dominating set (say, F_d).

- Connect each client to a facility in F_p that it has positively contributed to. If a client does not have such a facility in its neighborhood, connect them to any facility in F_d.

The bottleneck is that, so far, there is no known $O(1)$-approximation for soft capacitated MDS in UDGs (`UDG-CapMDS`). As shown by Kuhn and Moscibroda [49], in general graphs and even with uniform capacities, `CapMDS` is inherently *non-local* i.e., every distributed algorithm achieving a non-trivial approximation ratio must have a time complexity that essentially grows linearly with the network diameter. However, if for some parameter $\epsilon > 0$, capacities can be violated by a factor of $1 + \epsilon$, `CapMDS` becomes much more local. Additionally we hope to utilize the geometric properties of unit disks for better approximability. In summary, we intend to study and solve the `UDG-CapFacLoc` problem, and in the process, expect to get a clear idea of the locality of the problem.

REFERENCES

[1] Noga Alon, László Babai, and Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *J. Algorithms*, 7(4):567–583, 1986.

[2] Christoph Ambühl, Thomas Erlebach, Matús Mihalák, and Marc Nunkesser. Constant-factor approximation for minimum-weight (connected) dominating sets in unit disk graphs. In *APPROX-RANDOM*, pages 3–14, 2006.

[3] M. L. Balinski. On finding integer solutions to linear programs. In *Proceedings of IBM Scientific Computing Symposium on Combinatorial Problems*, pages 225–248, 1966.

[4] L. Barriére, P. Fraigniaud, and L. Narayanan. Robust position-based routing in wireless ad hoc networks with unstable transmission ranges. In *DIALM*, pages 19–27, 2001.

[5] Vittorio Bilò, Ioannis Caragiannis, Christos Kaklamanis, and Panagiotis Kanellopoulos. Geometric clustering to minimize the sum of cluster sizes. In *ESA*, pages 460–471, 2005.

[6] H-T.H. Chan and A. Gupta. Small hop-diameter sparse spanners for doubling metrics. In *SODA '06: Proceedings of the 17th annual ACM-SIAM symposium on Discrete algorithm*, pages 70–78, 2006.

[7] Hubert T-H. Chan, Anupam Gupta, Bruce M. Maggs, and Shuheng Zhou. On hierarchical routing in doubling metrics. In *SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 762–771, 2005.

[8] T-H. Hubert Chan. Personal Communication, 2006.

[9] Moses Charikar and Sudipto Guha. Improved combinatorial algorithms for the facility location and k-median problems. In *FOCS '99: Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, page 378, Washington, DC, USA, 1999. IEEE Computer Society.

[10] Xiuzhen Cheng, Xiao Huang, Deying Li, Weili Wu, and Ding-Zhu Du. A polynomial-time approximation scheme for the minimum-connected dominating set in ad hoc wireless networks. *Networks*, 42(4):202–208, 2003.

[11] Vasek Chvátal. A greedy heuristic for the set cover problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[12] G. Cornuejols, G. Nemhouser, and L. Wolsey. *Discrete Location Theory*. Wiley, 1990.

[13] A. Czumaj and H. Zhao. Fault-tolerant geometric spanners. *Discrete & Computational Geometry*, 32(2):207–230, 2004.

[14] Mirela Damian, Saurav Pandit, and Sriram V. Pemmaraju. Distributed spanner construction in doubling metric spaces. In *OPODIS*, pages 157–171, 2006.

[15] Mirela Damian, Saurav Pandit, and Sriram V. Pemmaraju. Local approximation schemes for topology control. In *PODC*, pages 208–217, 2006.

[16] G. Das, P. Heffernan, and G. Narasimhan. Optimally sparse spanners in 3-dimensional Euclidean space. In *SCG '93: Proc. of the ninth annual symposium on Computational geometry*, pages 53–62, New York, NY, USA, 1993. ACM Press.

[17] G. Das and G. Narasimhan. A fast algorithm for constructing sparse euclidean spanners. *Int. J. Comput. Geometry Appl.*, 7(4):297–315, 1997.

[18] Budhaditya Deb and Badri Nath. On the node-scheduling approach to topology control in ad hoc networks. In *MobiHoc '05: Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, pages 14–26, 2005.

[19] Michael Elkin. Distributed approximation: a survey. *SIGACT News*, 35(4):40–57, 2004.

[20] Thomas Erlebach and Erik Jan van Leeuwen. Domination in geometric intersection graphs. *Lecture Notes in Computer Science*, 4957:747–758, 2008.

[21] Christian Frank. *Algorithms for Sensor and Ad Hoc Networks*. Springer, 2007.

[22] Christian Frank and Kay Römer. Distributed facility location algorithms for flexible configuration of wireless sensor networks. In *Proceedings of the 3rd IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS 2007)*, pages 124–141, Santa Fe, NM, USA, jun 2007.

[23] Joachim Gehweiler, Christiane Lammersen, and Christian Sohler. A distributed o(1)-approximation algorithm for the uniform facility location problem. In *SPAA*, pages 237–243, 2006.

[24] Beat Gfeller and Elias Vicari. A randomized distributed algorithm for the maximal independent set problem in growth-bounded graphs. In *PODC '07: Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pages 53–60, 2007.

[25] J. Gudmundsson, C. Levcopoulos, and G. Narasimhan. Fast greedy algorithms for constructing sparse geometric spanners. *SIAM J. Comput.*, 31(5):1479–1500, 2002.

[26] J. Gudmundsson, C. Levcopoulos, and G. Narasimhan. Fast greedy algorithms for constructing sparse geometric spanners. *SIAM J. Comput.*, 31(5):1479–1500, 2002.

[27] Sudipto Guha and Samir Khuller. Improved methods for approximating node weighted steiner trees and connected dominating sets. *Inf. Comput.*, 150(1):57–74, 1999.

[28] M. Hajiaghayi, N. Immorlica, and V. S. Mirrokni. Fault-tolerant and 3-dimensional distributed topology control algorithms in wireless multi-hop networks. In *Proc. of the 11th IEEE International Conference on Computer Communications and Networks (IC3N)*, pages 392–398, 2002.

[29] M. Hajiaghayi, N. Immorlica, and V. S. Mirrokni. Power optimization in fault-tolerant topology control algorithms for wireless multi-hop networks. In *MobiCom*, pages 300–312, 2003.

[30] M. Hajiaghayi, G. Kortsarz, V.S. Mirrokni, and Z. Nutov. Power optimization for connectivity problems. In *IPCO*, pages 349–361, 2005.

[31] S. Har-Peled and M. Mendel. Fast construction of nets in low dimensional metrics, and their applications. In *SCG'05: Proceedings of the 21st annual symposium on Computational geometry*, pages 150–158, 2005.

[32] Wendi Rabiner Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *HICSS*, volume 8, page 8020, 2000.

[33] Dorit S. Hochbaum. Heuristics for the fixed cost median problem. *Mathematical Programming*, 22(1):148–162, 1982.

[34] Yaochun Huang, Xiaofeng Gao, Zhao Zhang, and Weili Wu. A better constant-factor approximation for weighted dominating set in unit disk graph. *Journal of Combinatorial Optimization*, 2008.

[35] Kamal Jain, Mohammad Mahdian, Evangelos Markakis, Amin Saberi, and Vijay V. Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing lp. *J. ACM*, 50(6):795–824, 2003.

[36] Kamal Jain and Vijay V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *J. ACM*, 48(2):274–296, 2001.

[37] Lujun Jia, Rajmohan Rajaraman, and Torsten Suel. An efficient distributed algorithm for constructing small dominating sets. *Distributed Computing*, 15(4):193–205, 2002.

[38] David S. Johnson. Approximation algorithms for combinatorial problems. In *STOC '73: Proceedings of the fifth annual ACM symposium on Theory of computing*, pages 38–49, 1973.

[39] Jaewon Kang, Yanyong Zhang, and Badri Nath. Analysis of resource increase and decrease algorithm in wireless sensor networks. In *ISCC '06: Proceedings of the 11th IEEE Symposium on Computers and Communications*, pages 585–590, 2006.

[40] Iyad A. Kanj, Ge Xia, and Fenghui Zhang. Local construction of spanners in the 3-d space. In *DCOSS '09: Proceedings of the 5th IEEE International Conference on Distributed Computing in Sensor Systems*, pages 315–328, 2009.

[41] B. Karp and H. T. Kung. Greedy perimeter stateless routing for wireless networks. In *MobiCom*, pages 243–254, 2000.

[42] Richard M. Karp. Probabilistic recurrence relations. *J. ACM*, 41(6):1136–1150, 1994.

[43] Leon Kaufman, Marc Vanden Eede, and Pierre Hansen. A plant and warehouse location problem. *Operational Research Quarterly*, 28(3):547–554, 1977.

[44] M. Khan and G. Pandurangan. A fast distributed approximation algorithm for minimum spanning trees. *Distributed Computing*, 20(6):391–402, 2008.

[45] David Kotz, Calvin Newport, and Chip Elliot. The mistaken axioms of wireless-network research. Technical Report TR2003-467, Dartmouth College, Department of Computer Science, 2003.

[46] R. Krauthgamer, A. Gupta, and J.R. Lee. Bounded geometries, fractals, and low-distortion embeddings. In *FOCS '03: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, pages 534–543, 2003.

[47] R. Krauthgamer and J.R. Lee. Navigating nets: simple algorithms for proximity search. In *SODA '04: Proceedings of the 15th annual ACM-SIAM symposium on Discrete algorithms*, pages 798–807, 2004.

[48] Alfred A. Kuehn and Michael J. Hamburger. A heuristic program for locating warehouses. *Management Science*, 9(4):643–666, 1963.

[49] Fabian Kuhn and Thomas Moscibroda. Distributed approximation of capacitated dominating sets. In *SPAA*, pages 161–170, 2007.

[50] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. What cannot be computed locally! In *PODC '04: Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 300–309, New York, NY, USA, 2004. ACM.

[51] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. On the locality of bounded growth. In *PODC '05: Proceedings of the twenty-fourth annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pages 60–68, 2005.

[52] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. The price of being near-sighted. In *SODA*, pages 980–989, 2006.

[53] Fabian Kuhn and RogerWattenhofer. Constant-time distributed dominating set approximation. *Distributed Computing*, 17(4):303–310, 2005.

[54] Fabian Kuhn and Roger Wattenhofer. Constant-time distributed dominating set approximation. In *PODC '03: Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 25–32, New York, NY, USA, 2003. ACM.

[55] Christoph Lenzen, Yvonne Anne Oswald, and Roger Wattenhofer. What can be approximated locally?: case study: dominating sets in planar graphs. In *SPAA '08: Proceedings of the twentieth annual symposium on Parallelism in algorithms and architectures*, pages 46–54, New York, NY, USA, 2008. ACM.

[56] C. Levcopoulos and A. Lingas. There are planar graphs almost as good as the complete graphs and almost as cheap as minimum spanning trees. *Algorithmica*, 8:251–256, 1992.

[57] X. Y. Li, G. Calinescu, and P. Wan. Distributed construction of planar spanner and routing for ad hoc wireless networks. In *Proc. of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 2002.

[58] X. Y. Li, G. Calinescu, P. J. Wan, and Y. Wang. Localized delaunay triangulation with application in ad hoc wireless networks. *IEEE Trans. Parallel Distrib. Syst.*, 14(10):1035–1047, 2003.

[59] Xiang-Yang Li and Yu Wang. Efficient construction of low weighted bounded degree planar spanner. *International Journal of Computational Geometry and Applications*, 14(1–2):69–84, 2004.

[60] Jyh-Han Lin and Jeffrey Scott Vitter. e-approximations with minimum packing constraint violation (extended abstract). In *STOC '92: Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 771–782, 1992.

[61] Nathan Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21(1):193–201, 1992.

[62] L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13:383–390, 1975.

[63] M Luby. A simple parallel algorithm for the maximal independent set problem. In *STOC '85: Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 1–10, 1985.

[64] Madhav V. Marathe, Heinz Breu, Harry B. Hunt III, S. S. Ravi, and Daniel J. Rosenkrantz. Simple heuristics for unit disk graphs. *Networks*, 25:59–68, 1995.

[65] Thomas Moscibroda and Roger Wattenhofer. Facility location: distributed approximation. In *PODC '05: Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, pages 108–117, 2005.

[66] Moni Naor and Larry Stockmeyer. What can be computed locally? In *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 184–193, 1993.

[67] Saurav Pandit and Sriram Pemmaraju. Finding facilities fast. In *Full Paper*, 2009. http://cs.uiowa.edu/ spandit/research/icdcn2009.pdf.

[68] Saurav Pandit and Sriram V. Pemmaraju. Finding facilities fast. In *ICDCN*, pages 11–24, 2009.

[69] Saurav Pandit and Sriram V. Pemmaraju. Return of the primal-dual: Distributed metric facility location. In *PODC*, pages 180–189, 2009.

[70] Saurav Pandit and Sriram V. Pemmaraju. Rapid randomized pruning for fast distributed greedy algorithms. In *PODC (to appear)*, 2010.

[71] David Peleg. *Distributed computing: a locality-sensitive approach*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[72] Sriram Pemmaraju and Imran Pirwani. Good quality virtual realizations of unit ball graphs. In *Algorithms - ESA 2007*, pages 311–322, 2007.

[73] R. Rajaraman. Topology control and routing in ad hoc networks: A survey. *SIGACT News*, 33:60–73, 2002.

[74] Johannes Schneider and Roger Wattenhofer. A Log-Star Distributed Maximal Independent Set Algorithm for growth-Bounded Graphs. In *27th ACM Symposium on Principles of Distributed Computing (PODC), Toronto, Canada*, August 2008.

[75] David B. Shmoys, Éva Tardos, and Karen Aardal. Approximation algorithms for facility location problems (extended abstract). In *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 265–274, 1997.

[76] John F. Stollsteimer. A working model for plant numbers and locations. *Management Science*, 45(3):631–645, 1963.

[77] Kunal Talwar. Bypassing the embedding: algorithms for low dimensional metrics. In *STOC '04: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 281–290, 2004.

[78] Peng-Jun Wan, Khaled M. Alzoubi, and Ophir Frieder. Distributed construction of connected dominating set in wireless ad hoc networks. *Mob. Netw. Appl.*, 9(2):141–149, 2004.

[79] Yu Wang and Xiang-Yang Li. Localized construction of bounded degree and planar spanner for wireless ad hoc networks. In *DIALM-POMC '03: Proceedings of the 2003 joint workshop on Foundations of mobile computing*, pages 59–68, 2003.

[80] Yu Wang, Weizhao Wang, and Xiang-Yang Li. Distributed low-cost backbone formation for wireless ad hoc networks. In *MobiHoc*, pages 2–13, 2005.

[81] R. Wattenhofer and A. Zollinger. XTC: A practical topology control algorithm for ad-hoc networks. In *4th International Workshop on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks (WMAN)*, 2004.

[82] Tao Wu and Subir Biswas. Minimizing inter-cluster interference by self-reorganizing mac allocation in sensor networks. *Wireless Networks*, 13(5):691–703, 2007.

[83] 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.