UNIVERSITY OF CALIFORNIA,
IRVINE

Geometric Spanners: New Algorithms and Frameworks

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Hadi Khodabandeh

2024

DEDICATION

To my family.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ALGORITHMS

ACKNOWLEDGMENTS

I would like to extend my gratitude to my advisor and defense committee chair, Professor David Eppstein, for introducing me to spanners and guiding me throughout my academic adventure. It has been an honor to learn from him over the years.

I am also deeply thankful to my other defense committee members: Professors Michael Goodrich and Michael Shindler for their meticulous review of this dissertation and for providing invaluable insights.

My appreciation extends to my co-authors and colleagues, whose knowledge and expertise have greatly enriched my understanding. Working with them—Daniel Frishberg, Pedro Matias, Nil Mamano, Haleh Havvaei, Sujoy Bhore, Arnold Filtser, Csaba D. Tóth, Amir Hossein Zargari, and Seyd Amir Hossein Aqajari—has been an absolute pleasure.

I owe a debt of gratitude to my family for their unwavering support from across the globe. Their sacrifices have been immeasurable, and they truly deserve the best.

I am also grateful to my colleagues and friends in the theory lab at UCI, who have been instrumental in shaping my PhD experience and career. Special thanks to Evrim Ozel, Thorben Tröbst, Shion Fukuzawa, Ryuto Kitagawa, Alvin Chiu, Martin Bullinger, Abraham Mathew Illickan, and all other lab members and visitors.

This work has partially been supported by US National Science Foundation under grants CCF-1616248 and CCF-2212129.

VITA

Hadi Khodabandeh

EDUCATION

Doctor of Philosophy in Computer Science	**2024**
University of California, Irvine	*Irvine, California*
Master of Science in Computer Science	**2022**
University of California, Irvine	*Irvine, California*
Bachelor of Science in Computer Engineering	**2018**
Sharif University of Technology	*Tehran, Iran*

RESEARCH EXPERIENCE

Part-time Student Researcher	**2023–2024**
Google	*Sunnyvale, California*
Graduate Research Assistant	**2018–2024**
University of California, Irvine	*Irvine, California*
Undergraduate Research Intern	**2017**
Max Planck Institute for Informatics	*Saarbrücken, Germany*
Undergraduate Research Intern	**2016**
Hong Kong University of Science and Technology	*Clear Water Bay, Hong Kong*

TEACHING EXPERIENCE

Teaching Assistant	**2019–2024**
University of California, Irvine	*Irvine, California*

REFEREED JOURNAL PUBLICATIONS

- A. H. Zargari, S. A. Aqajari, H. Khodabandeh, A. M. Rahmani, and F. Kurdahi. An accurate non-accelerometer-based ppg motion artifact removal technique using cyclegan. *ACM Transactions on Computing for Healthcare*, 2022
- P. Choudhary, M. T. Goodrich, S. Gupta, H. Khodabandeh, P. Matias, and V. Raman. Improved kernels for tracking paths. *Information Processing Letters*, 181:106360, 2023
- S. Bhore, A. Filtser, H. Khodabandeh, and C. D. Tóth. Online spanners in metric spaces. *SIAM Journal on Discrete Mathematics*, 38(1):1030–1056, 2024

REFEREED CONFERENCE PUBLICATIONS

- M. J. Golin, H. Khodabandeh, and B. Qin. Non-approximability and polylogarithmic approximations of the single-sink unsplittable and confluent dynamic flow problems. In *28th International Symposium on Algorithms and Computation (ISAAC 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017
- D. Eppstein and H. Khodabandeh. On the edge crossings of the greedy spanner. In *37th International Symposium on Computational Geometry*, volume 12, page 37, 2021
- M. T. Goodrich, S. Gupta, H. Khodabandeh, and P. Matias. How to catch marathon cheaters: New approximation algorithms for tracking paths. In *Workshop on Algorithms and Data Structures*, pages 442–456. Springer, 2021
- D. Eppstein and H. Khodabandeh. Brief announcement: Distributed lightweight spanner construction for unit ball graphs in doubling metrics. In *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 57–59, 2022
- D. Eppstein and H. Khodabandeh. Distributed construction of lightweight spanners for unit ball graphs. In *36th International Symposium on Distributed Computing*, 2022
- S. Bhore, A. Filtser, H. Khodabandeh, and C. D. Tóth. Online spanners in metric spaces. In *30th Annual European Symposium on Algorithms (ESA 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022
- D. Eppstein and H. Khodabandeh. Maintaining light spanners via minimal updates. *36th Canadian Conference on Computational Geometry (CCCG 2024)*, 2024

ABSTRACT OF THE DISSERTATION

Geometric Spanners: New Algorithms and Frameworks

By

Hadi Khodabandeh

Doctor of Philosophy in Computer Science

University of California, Irvine, 2024

Distinguished Professor David Eppstein, Chair

In this dissertation, we investigate a multitude of novel algorithms aimed at efficiently constructing geometric spanners across various computational scenarios. In addition to devising new algorithms, we also develop several methodologies and frameworks for analyzing geometric spanners and their attributes, such as sparsity, weight, and separation properties, offering valuable tools for readers engaged in related research areas.

We present our results in three main chapters. We begin by investigating the construction of the greedy t-spanner for a set of points in the plane. This spanner is an undirected graph constructed by considering pairs of points in order by distance, and connecting a pair by an edge when there does not already exist a path connecting that pair with length at most t times the Euclidean distance. Our analysis reveals that for any $t > 1$, these graphs exhibit a linear number of crossings and possess bounded degeneracy in their intersection graphs. These properties lead to the development of a separator theorem for greedy spanners, which demonstrates our capability to form a recursive separator hierarchy from the planarizations of these spanners in linear time, or in near-linear time if the planarization is unknown.

Expanding on these fundamental concepts, we address an unresolved question concerning the existence of lightweight bounded-degree spanners for unit ball graphs in metrics with bounded doubling dimension. We present a distributed algorithm operating within $\mathcal{O}(\log^* n)$

rounds in the LOCAL model of computation, achieving significant improvements over prior methodologies. We extend the applicability of this algorithm to the CONGEST model while maintaining its round complexity. Our investigations extend to the two-dimensional Euclidean plane, where we devise constructions with a constant average number of edge intersections per node. Experimental evaluations validate the efficiency of our distributed algorithm compared to centralized counterparts.

Finally, we focus on the dynamic maintenance of lightweight bounded-degree $(1+\varepsilon)$-spanners in d-dimensional Euclidean spaces. In a fully-dynamic setting, where points can be inserted or deleted, we aim to minimize the recourse while preserving essential spanner properties. We introduce a novel fully-dynamic algorithm with amortized constant recourse for point insertion and $\mathcal{O}(\log \Delta)$ recourse for point deletion, representing the first advancement on lightweight dynamic spanners.

Throughout this dissertation, our aim is to also offer insightful discussions on the open problems inherent in each addressed scenario, providing many opportunities for readers seeking to engage with and explore further challenges in this field.

Chapter 1

Introduction

Geometric spanners are fundamental structures in computational geometry designed to approximate the connectivity between a set of points in a geometric space while satisfying certain distance constraints. Specifically, a geometric spanner is a sparse graph that connects pairs of points such that the shortest path between any two points in the spanner is not significantly longer than their Euclidean distance. These structures are crucial in various computational tasks where efficient communication, path planning, or proximity queries are required.

Spanners can be defined in any metric space, but they are often located in a geometric space, where a heavy or undesirable network is given and finding a sparse and light-weight spanner and working with it instead of the actual network makes the computation easier and faster. Finding sparse and light-weight geometric spanners has been a topic of interest in many areas of computer science, including communication network design and distributed computing. These subgraphs have few edges and are easy to construct, leading them to appear in a wide range of applications since they were introduced [23, 66, 81]. In wireless ad hoc networks t-spanners are used to design sparse networks with guaranteed connectivity and guaranteed bounds on routing length [5]. In distributed computing spanners provide communication-efficiency and time-efficiency through the sparsity and the bounded stretch property [10, 38, 8, 39]. There has also been extensive use of geometric spanners in the analysis of road networks [40, 1, 21]. In robotics, geometric spanners helped motion planners to design near-optimal plans on a sparse and light subgraph of the actual network [33, 77, 29]. Spanners have many other applications including computing almost shortest paths [36, 25, 83, 50], and overlay networks [17, 86, 64].

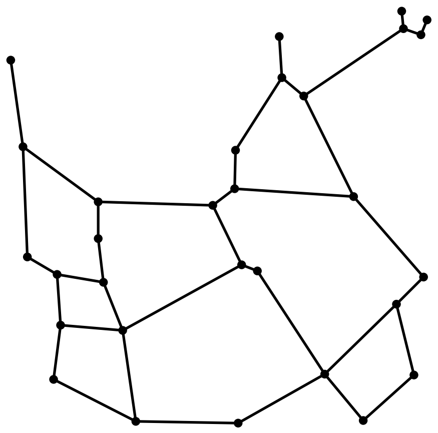
1.1 Definition

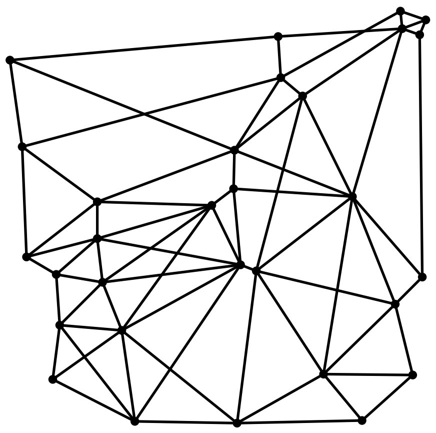
Given an input graph $G = (V, E)$ and a distance function $d : V \times V \to \mathbb{R}^+$, an (α, β)-spanner of G is a subgraph $H = (V, E_H)$ of G that

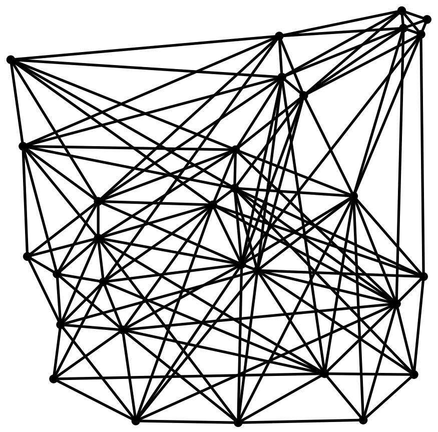
$$d_H(u, v) \leq \alpha \cdot d_G(u, v) + \beta \tag{1.1}$$

Here, d_G and d_H are the shortest path metrics of G and H, respectively. Meaning that $d_H(u, v)$ (resp. $d_G(u, v)$) is the length of the shortest weighted path between u and v in H (resp. G), where the edge weights come from the distance function d. Figure 1.1 shows a few examples of spanners with various values of α on a random point set in the two dimensional Euclidean plane. The impact of the α on the sparsity of the spanner is clear from this figure.

A spanner algorithm is an algorithm that operates based on assumptions regarding the characteristics of the point set and the distance function. It takes as input a graph that conforms to these assumptions and finds a sparse spanner of the input graph. The aim of a spanner algorithm is to produce a spanner that provides good bounds on its total weight or maximum degree while preserving connectivity and approximation properties. Different spanner algorithms may prioritize specific qualities, e.g. sparsity, lightness, or low degree bound, depending on the intended application or optimization criteria. Some spanner algorithms focus on optimizing a subset of these qualities, while others strive to achieve near-optimal performance across all measures. We explore some of these spanner algorithms and their performance trade-offs in subsequent sections.


(a) $(2, 0)$-spanner


(b) $(1.2, 0)$-spanner


(c) $(1.05, 0)$-spanner

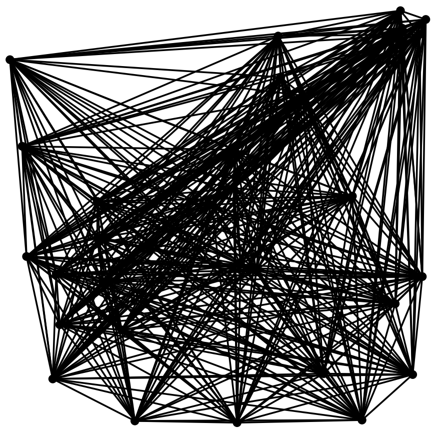

(d) Complete graph

Figure 1.1: A comparison of the complete graph on 30 random points on the plane with spanners of stretch 2, 1.2, and 1.05 on the same point set.

1.2 Additive vs Multiplicative

Geometric spanners come in various types, each tailored to different distance constraints and applications. Two common spanner types are additive and multiplicative spanners. Additive spanners connect pairs of points such that the length of any path in the spanner is at most a fixed constant plus the Euclidean distance between the points, i.e. when $\alpha = 1$ and $\beta > 0$ in eq. (1.1). Formally, a β-additive spanner H satisfies

$$d_H(u, v) \leq d_G(u, v) + \beta$$

for all pairs (u, v) of the vertices.

In contrast, multiplicative spanners ensure that the length of any path in the spanner is at most a constant factor times the Euclidean distance between the points, i.e. when $\alpha > 1$ and $\beta = 0$. Formally, an α-multiplicative spanner (or in short, an α-spanner) satisfies

$$d_H(u, v) \leq \alpha \cdot d_G(u, v)$$

for all pairs (u, v) of the vertices. The term α is called the *stretch-factor* of the multiplicative spanner.

Additive spanners are well-suited for applications where the input graph is unweighted (e.g. social networks), while multiplicative spanners can be useful for both weighted (e.g. road networks) and unweighted input graphs. An interesting example where both additive and multiplicative spanners would fit is packet routing. If the processing time of a packet in a router is the dominant part of the routing time complexity, e.g. in a local network, one can ignore the travel time of the packet and represent the input graph as an unweighted graph. Then look for an additive spanner to simplify the network. On the other hand if the travel time of a packet is the dominant factor, e.g. in the internet network, then one can ignore the processing time of the packet in the routers, and aim to minimize the travel time on a multiplicative spanner of the input.

In this dissertation, we focus on multiplicative spanners due to their wider range of use and the intriguing open problems they present. Another reason for our focus on multiplicative spanners would be the recent advancements in spanners which have mostly evolved around finding light-weight spanners with low stretch-factor. Analyzing the lightness only makes sense in the context of weighted graphs and hence multiplicative spanners.

1.3 Early history

Low-stretch (also known as low dilation) graphs have been studied very early on in the literature [41]. Chew [22] studied the Delaunay triangulations and presented them as one of the first examples of planar graphs that can approximate complete graphs up to a constant factor. Keil [66] showed by relaxing the planarity property and allowing edge intersections to happen one can get arbitrarily close to 1 stretch-factors via linear number of edges, using the well-known Yao-graph construction. Das and Joseph [30] introduced the weight of the spanner as an important factor for the first time, and showed a set of conditions that would be sufficient to show that a planar graph has a constant bound on its lightness. A few years later, Althöfer, Das, Dobkin, and Joseph [3] came up with the greedy spanner construction as an extremely sparse spanner. Callahan and Kosaraju [18] then introduced well-separated pair decompositions and since then they have been used in many applications, including constructing geometric spanners. While all of these constructions are very interesting and technical, our focus is mainly on the greedy spanner algorithm, and to some extent on WSPD-based spanners.

1.4 Distance Function

The first spanner results in an real-world application would be to model the problem space using a graph and a distance function. The choice of the distance function plays a crucial role in determining the strength of the guarantees we can expect from a spanner algorithm. Different distance functions lead to varying structures in the space, thereby influencing the types of proof arguments we can use and therefore impacting the strengths of the bounds we can show on the quality of a spanner algorithm.

In the context of Euclidean metric spaces, where the Euclidean distance metric is commonly used, spanner algorithms aim to approximate the Euclidean distances between points while minimizing the number of edges in the spanner subgraph. The analysis of these algorithms typically rely on geometric properties of the Euclidean metric to find creative ways of proving their desired sparsity bounds and stretch-factor. It is within this domain that spanners with the most robust guarantees reside, effectively fulfilling their purpose in real-world applications. In Euclidean spaces, these spanners often achieve an arbitrarily close to 1 stretch factor. Notably, the greedy spanner algorithm is renowned for its ability to generate spanner graphs with arbitrarily low stretch factors in Euclidean spaces.

However, in more general metric spaces, such as those satisfying the doubling property, the impact of the distance function on spanner algorithms can be more pronounced. Doubling metric spaces are characterized by the following property: for any point x in the space and any radius $r > 0$, the ball $B(x, 2r)$ of radius $2r$ centered at x can be covered by at most C balls of radius r, where C is a constant. For example, a d-dimensional Euclidean space under an L_p norm (for $p > 0$) is a $\Theta(d)$ doubling metric space. The geometric arguments and proofs that are based on angles and trigonometric functions do not naturally extend to doubling metric spaces. The proofs of the spanners in doubling spaces are, in some sense, even more creative and versatile, since they only use the triangle inequality and the doubling property of the space. In such spaces, spanner algorithms usually provide a slightly weaker trade-off between the stretch factor and other qualities such as weight and maximum degree, compared to Euclidean spaces.

Moreover, in general metric spaces with arbitrary distance functions, the full impact of the distance function on spanner algorithms becomes evident. Spanner algorithms in general metric spaces are constrained to a very limited set of tools to ensure their stretch and sparsity bounds. Consequently, this often leads to significantly weaker bounds on the stretch factor as well as other qualities of the spanner. These limitations arise from the inherent constraints that general metrics impose on such algorithms, highlighting the challenges in designing and analyzing spanner algorithms under such general assumptions.

The choice of the distance function in a spanner problem is driven by the characteristics of the real-world problem motivating the analysis: When real-world measurements are precise and the point set exhibits well-defined structure, using a Euclidean distance function is appropriate for modeling purposes.

When confronted with noisy measurements and uncertainty regarding the potential impact of measurement errors on the efficacy of a Euclidean spanner algorithm, e.g. when the impact of the Euclidean dimension on the output bounds is high, opting for a spanner algorithm tailored for doubling spaces potentially offers a better alternative. The inherent resilience of the doubling dimension to small perturbations in measurements ensures robustness against inaccuracies, making them preferable for such scenarios.

In situations where the problem domain lacks geometric patterns or adherence to Euclidean principles, modeling the world with general metrics becomes unavoidable. General metric spanners accommodate the complex nature of the problem space without relying on specific geometric assumptions, providing a flexible solution for approximation and analysis.

1.5 Spanner Qualities

As we mentioned earlier, spanner algorithms often aim to provide various qualities besides the the basic stretch bound and sparsity. Although the extra qualities that a spanner algorithm provides are limited by the type of metric space they are dealing with, they also greatly depend on the algorithm itself and the way it processes the input graph.

For example, a greedy spanner processes the edges of the input graph in the increasing order of their lengths, and as the name suggests, only adds edges when they are essential to the guarantee of the stretch property. Unnecessary edges will be dropped and as a consequence, the spanner possesses great sparsity and weight bounds, as far as the metric space allows.

While spanner algorithms are typically straightforward, delving into their properties often demands intricate techniques and methods. For instance, the weight analysis of the greedy spanner mentioned above serves as a prime example; although the algorithm itself is straightforward, its weight analysis fills an entire publication in a leading theory conference.

1.5.1 Sparsity and Lightness

Sparsity is a core property of a spanner, a property that the spanner algorithms strongly focus on and try to achieve near optimal results. Informally, sparsity measures how sparse a spanner looks if it is drawn on a plane. Formally, we can measure the number of edges of a spanner and compare it against the total possible number of edges between the input vertices, $n(n-1)/2$, and if the ratio of the former to the latter is low, call the spanner sparse. There is no well-agreed threshold for sparsity, but when people call their spanners sparse, they often mean the number of edges of the spanner is bounded by a linear function of the number of vertices.

Lightness, conversely, has emerged as a central focus in much of the recent research on spanners, despite being introduced early in the development of these structures. As the name implies, lightness assesses the weight efficiency of a spanner. Unlike sparsity, lightness can be precisely quantified. Since a spanner must connect all the vertices, its weight must be at least equal to that of the minimum spanning tree of the input graph. Consequently, the lightness of the spanner, defined as the ratio below, is lower bounded by 1.

$$\text{Lightness} := \frac{\text{Weight of the spanner}}{\text{Weight of the minimum spanning tree}}$$

Both sparsity and lightness are integral for defining a good quality spanner. In contexts where the quantity of edges directly impacts the operational cost of a spanner, such as in wireless networks, sparsity takes precedence. On the other hand, in scenarios where a physical cost is incurred for each unit of length of spanner edges utilized, as in physical networks, lightness becomes the more practical factor to consider.

1.5.2 Degree Bound

Degree bound is another essential property of spanners that is inherently linked to sparsity. This connection arises from the fact that maintaining a constant maximum degree would result in a linear constraint on the number of edges, thereby ensuring sparsity. The significance of the degree bound originates from its implications in various applications of spanners. For instance, in wireless networks, where each node corresponds to a wireless router and each edge represents a wireless connection between two routers, the memory constraints of routers dictate a limit on the number of connections they can accommodate. This imposes a physical upper limit on the number of edges that can be linked to a node within the spanner.

Not all spanner constructions guarantee a constant degree bound. There are many spanner constructions, e.g. Theta-graphs and greedy spanners, that offer good sparsity bounds (linear number of edges) but do not guarantee satisfactory degree bounds, either in Euclidean spaces or in doubling metrics.

1.5.3 Running Time

We are concerned about the running time of any algorithm, and this holds true for spanner algorithms as well. The running time of an algorithm can often be the primary limiting factor for the input size feasible for computation within a reasonable duration. While the discussed greedy spanner algorithm exhibits very high qualities, its running time is not the best. For instance, the best known implementation of this algorithm operates in $\mathcal{O}(n^2 \log n)$ time. Realistically, running this algorithm on a conventional computer for more than a few thousand vertices would likely take at least a minute. Recent efforts have been directed towards devising near-linear time spanner algorithms. In contrast, a linear algorithm would potentially process around a million nodes within a similar amount of time.

1.5.4 Adaptation to Change

Another aspect to consider for certain applications is the spanner's adaptability to changes in the input set. An offline algorithm takes advantage of having complete knowledge of the input and can manipulate it in any arbitrary manner. Conversely, an online algorithm observes the input gradually (e.g., point by point) and must maintain a valid output at all times. This functionality is valuable in applications such as mobile networks, where the usefulness of a spanner relies on its continuous maintenance even following a user addition. Dynamic algorithms extend this capability further by allowing modifications to the previously provided input, such as the removal of existing points or edges in the input graph.

In applications where algorithm responsiveness is key, the running time of an online or dynamic algorithm assumes critical significance. In contrast, in other applications, the actual physical cost of changes made to the output takes precedence. Here, a new parameter known as *recourse* becomes important. Recourse is defined as the number of alterations an online or dynamic algorithm can make to its previous output to adapt to a single change in the input.

1.6 Summary of Results

In chapters 2-4, we investigate spanners across different contexts, developing novel algorithms and introducing fresh analytical approaches. Our exploration begins with an examination of the renowned greedy spanner, wherein we establish an upper bound on the number of edge intersections associated with these spanners.

1.6.1 Separators for Greedy Spanners

The greedy spanner algorithm iterates through the pairs of points in the input set in the ascending order of their distance, and adds the current pair to the spanner only if the existing shortest path inside the spanner between the pair is bad, i.e. more than $(1 + \varepsilon)$ times the distance of the pair. Although this algorithm looks very simple, Farshi and Gudmundsson [49] observed that in practice, it performs surprisingly good in terms of the number of edges, weight, maximum vertex degree, and also the number of edge crossings. There has been studies of many of these properties since then, and most of them have been proven rigorously. Filster and Solomon [51] proved that the size and the lightness of the greedy spanner is optimal to within a constant factor for worst-case instances. Borradaile, Le, and Wulff-Nilsen [14] proved lightness optimality in doubling metrics, and Le and Solomon [71] showed that no geometric t-spanner can do asymptotically better than the greedy spanner in terms of number of edges and lightness. However, past work has not proven rigorous bounds on the number of crossings of greedy spanners, which is the focus of our first result in this dissertation.

In Chapter 2, we establish that greedy t-spanners in the Euclidean plane exhibit few crossings for any $t > 1$. Leveraging this finding, we demonstrate that greedy spanners in the Euclidean plane possess small separators. Specifically, our proofs yield the following assertions:

- **Claim 1.** Each edge in a greedy spanner may be crossed by only $O(1)$ edges of equal or greater length, where the constant in the $O(1)$ term depends solely on t, the stretch factor of the spanner. More precisely, as $t \to 1$, there exist $O(1/(t-1)^2)$ edges that intersect the given edge and exceed it in length by a factor of $\Omega(1/(t-1))$, and $1/(t-1)^{O(1)}$ edges that cross the given edge with a length of at least ε times its length, for any constant $\varepsilon > 0$.

- **Claim 2.** For certain selections of t, there exist greedy spanners wherein certain edges are intersected by a linear number of (considerably shorter) edges.

- **Claim 3.** Every n-vertex greedy spanner, as well as every n-vertex subgraph of a greedy spanner, can be decomposed into connected components of size at most cn for a constant $c < 1$ by the removal of $O(\sqrt{n})$ vertices. Again, the constant factor in the $O(\sqrt{n})$ term depends exclusively on the stretch factor of the spanner. Additionally, a separator hierarchy for the greedy spanner can be constructed from its planarization in nearly-linear time.

It is established that spanners constructed via alternate methods, such as well-separated pair decomposition [2] and hierarchical decomposition [53], possess small $\mathcal{O}(\sqrt{n})$-separators in two dimensions. Although experimental observations by Farshi and Gudmundsson on greedy spanners of random point sets had suggested a limited number of crossings in practice [49], our results represent the first theoretical insights into this property, offering analysis on worst-case scenarios rather than just random instances, and providing evidence that greedy spanners feature small $\mathcal{O}(\sqrt{n})$-separators.

1.6.2 Distributed Spanners for Unit Ball Graphs

In Chapter 3, motivated by wireless and ad-hoc networks, we focus on the special case of spanners where the underlying graph is a unit disk graph (UDG). In wireless and ad-hoc networks the communication of the nodes are limited by their physical distances. Therefore, two nodes are connected if their geometric distance is less than a constant R. When modeling, we can assume this constant is equal to 1, because otherwise, we can scale down (or up) the network for that to happen. The necessity of a connected and energy-efficient topology for high-level routing protocols led researchers to develop many spanning algorithms for ad-hoc networks and in particular, UDGs. And the decentralized nature of ad-hoc networks demands that these algorithms be local instead of centralized.

On the other hand, we are interested to study this problem in a more generalized metric space than the Euclidean spaces, which are called the *doubling metric spaces*. Doubling metric spaces are metric spaces with the extra doubling property: For any $R > 0$, any ball of radius R in the metric space should be coverable with at most 2^d balls of radius $R/2$; for some constant d which is referred to as the *doubling dimension* of the metric space. Doubling spaces not only include Euclidean spaces of finite dimension, but also include any Euclidean space under any L_p norm (for $p > 0$, including the L_∞ norm). They also include cases like sphere points with spatial distance as the metric distance, which makes them useful in a variety of scenarios. Another reason for the importance of the spaces of bounded doubling dimension comes from the fact that a small perturbation in the pairwise distances does not affect the doubling dimension of the point set by much, while it can change their Euclidean dimension significantly, or the resulting distances might not even be embeddable in Euclidean metrics at all [19].

The state-of-the-art for this problem goes back to 2006, where Damian, Pandit, and Pemmaraju [28] designed a distributed construction for $(1 + \varepsilon)$-spanners of the UBGs lying in *d-dimensional Euclidean space.* Their algorithm runs in $\mathcal{O}(\log^* n)$ rounds of communication and produces a $(1+\varepsilon)$-spanner with constant bounds on its maximum degree and lightness. They used the so-called *leapfrog property* to prove the constant bound on the lightness of the spanner, which does not hold for the spaces of bounded doubling dimension in general. Instead, they showed in another work [27] that the weight of their spanner in the spaces of bounded doubling dimension is bounded by a factor $\mathcal{O}(\log \Delta)$ of the weight of the minimum spanning tree, where Δ is the ratio of the length of the longest edge in the unit ball graph divided by the length of its shortest edge. Besides these, their algorithm requires the knowledge of $\mathcal{O}(\frac{1}{\alpha-1})$-hop neighborhood of the nodes, which is costly in the CONGEST model of distributed computing, the more accepted and practical model than the LOCAL model of computation.

In Chapter 3, we devise and analyze novel spanner algorithms tailored for unit ball graphs in the spaces of bounded doubling dimensions under the LOCAL and CONGEST models of distributed algorithms. More specifically, we present two primary contributions.

First, we settle an outstanding open question by demonstrating the existence of lightweight bounded-degree $(1+\varepsilon)$-spanners for unit ball graphs in spaces with bounded doubling dimensions. Our construction maintains constant bounds on its maximum degree and lightness, and it can be executed in $\mathcal{O}(\log^* n)$ rounds of communication in the LOCAL model of computation, where n denotes the number of vertices.

Second, we introduce the inaugural lightweight spanner construction for unit ball graphs in the CONGEST model of computation. Even in the restricted scope of the two-dimensional Euclidean plane, where the majority of applications involving unit disk graphs are observed, there previously existed no known CONGEST algorithm for generating light spanners of unit disk graphs. We accomplish this feat by adapting our construction for the LOCAL model to function within the CONGEST model in the same asymptotic number of rounds. The lightness and maximum degree bounds of our spanner remain consistent in this model.

In addition to these primary results, we adapt these constructions for the two-dimensional Euclidean plane to ensure a linear number of total edge intersections, thereby implying a constant average number of edge intersections per node. This adaptation is motivated by the observation that a higher intersection per edge increases the likelihood of interference between the corresponding endpoints. To the best of our knowledge, this represents the first distributed low-stretch low-intersection spanner construction for unit disk graphs.

For further elaboration on our findings, we present the following theorems. Initially, we introduce a centralized algorithm, Centralized-Spanner, which resolves the following theorem:

Theorem 3.8. Given a weighted unit ball graph G in a metric of bounded doubling dimension and a constant $\varepsilon > 0$, the spanner returned by Centralized-Spanner(G,ε) is a $(1+\varepsilon)$-spanner of G and has constant bounds on its lightness and maximum degree. These constant bounds only depend on ε and the doubling dimension.

We employ this centralized construction to propose the distributed construction, Distributed-Spanner, within the LOCAL model of computation:

Theorem 3.16. Given a weighted unit ball graph G with n vertices in a metric of bounded doubling dimension and a constant $\varepsilon > 0$, the algorithm Distributed-Spanner(G,ε) runs in $\mathcal{O}(\log^* n)$ rounds of communication in the LOCAL model of computation, and returns a $(1+\varepsilon)$-spanner of G that has constant bounds on its lightness and maximum degree. These constant bounds only depend on ε and the doubling dimension.

Next, we explore the problem within the CONGEST model of computation. Our distributed construction, Distributed-Spanner, necessitates complete information about the 2-hop neighborhood of a selected set of vertices, a task that is challenging to achieve in the CONGEST model. A similar issue arises in the distributed algorithm proposed by [27], where they aggregate information about nodes that are $\mathcal{O}(\frac{1}{\alpha-1})$ hops away, for some constant α. A straightforward approach for aggregating 2-hop neighborhoods would require $\mathcal{O}(d)$ rounds of communication in the CONGEST model, which can be as large as $\Omega(n)$ if the input graph is dense. In the subsequent theorem, we surmount this barrier by refining our algorithm to function within the CONGEST model of computation. Despite adding complexity to the algorithm itself, we demonstrate that the round complexity of our new algorithm, CONGEST-Spanner, remains bounded by $\mathcal{O}(\log^* n)$.

Theorem 3.21. Given a weighted unit ball graph G with n vertices in a metric of bounded doubling dimension and a constant $\varepsilon > 0$, the algorithm CONGEST-Spanner(G,ε) runs in $\mathcal{O}(\log^* n)$ rounds of communication in the CONGEST model of computation, and returns a $(1+\varepsilon)$-spanner of G that has constant bounds on its lightness and maximum degree. These constant bounds only depend on ε and the doubling dimension.

Subsequently, we delve into the problem within the two-dimensional Euclidean plane, where the greedy spanner on a complete weighted graph is known to exhibit constant upper bounds on its lightness [51], maximum degree, and average number of edge intersections per node [45]. We observe that a simple modification to this algorithm can extend these results to unit disk graphs as well. We refer to this modified algorithm as Centralized-Euclidean-Spanner, and we establish that

Theorem 3.22. Given a weighted unit disk graph G in the two dimensional Euclidean plane and a constant $\varepsilon > 0$, the spanner returned by Centralized-Euclidean-Spanner(G,ε) is a $(1 + \varepsilon)$-spanner of G and has constant bounds on its lightness, maximum degree, and the average number of edge intersections per node. These constant bounds only depend on ε and the doubling dimension.

We utilize the aforementioned construction to propose Distributed-Euclidean-Spanner, a specific distributed low-intersection construction for the two-dimensional Euclidean plane that preserves the aforementioned properties and incorporates the low-intersection property.

Theorem 3.32. Given a weighted unit disk graph G with n vertices in the two dimensional Euclidean plane and a constant $\varepsilon > 0$, the algorithm Distributed-Euclidean(G,ε) runs in $\mathcal{O}(\log^* n)$ rounds of communication and returns a bounded-degree $(1+\varepsilon)$-spanner of G that has constant bounds on its lightness, maximum degree, and the average number of edge intersections per node. These constant bounds only depend on ε and the doubling dimension.

Furthermore, we demonstrate that this final construction exhibits sublinear separators and a separator hierarchy in the two-dimensional Euclidean plane. We extend this result to higher dimensions of Euclidean spaces. Finally, we present experimental results on random point sets in the two-dimensional Euclidean plane, confirming the efficacy of our distributed construction.

1.6.3 Fully Dynamic Spanners with Small Recourse

Finally, in Chapter 4, we focus on spanner algorithms in a fully dynamic settings, where points can be inserted as well as removed, and the algorithm has to maintain a lightweight spanner with minimal recourse after each such update on the point set. Lightweight fully dynamic spanners have not been thoroughly examined in existing literature, to the best of our knowledge. Currently, there are no known algorithms that yield a spanner with constant lightness without rebuilding the entire spanner.

However, in the online model, where input points are given to the algorithm one at a time (and the algorithm can only insert new edges in the output), there are a few related results. Online Euclidean spanners were first studied by Bhore and Tóth [13], who showed an upper bound of $\mathcal{O}(\varepsilon^{-(d+1)}\log n)$ on the competitive ratio of the deformable spanner algorithm of Gao, Guibas, and Nguyen [55], for points in any dimension d. Later, Bhore, Filtser, Khodabandeh, and Tóth [11] improved the upper bound to $\mathcal{O}(\varepsilon^{-d}\log n)$ and achieved a lower bound of $\Omega(\varepsilon^{-d})$ under the L_1 metric. However, the maximum degree of a node in deformable spanner could be as large as $\Omega(\log n)$ which implies a logarithmic recourse per point insertion and removal.

In the dynamic setting, for n points in d-dimensional Euclidean space, Arya, Mount, and Smid [7] designed a spanner construction with a linear number of edges and $\mathcal{O}(\log n)$ diameter under the assumption that a point to be deleted is chosen randomly from the point set, and a point to be inserted is chosen randomly from the new point set. Bose, Gudmundsson, and Morin [16] presented a semi-dynamic $(1+\varepsilon)$-spanner construction with $\mathcal{O}(\log n)$ maximum degree and diameter. Gao, Guibas, and Nguyen's deformable spanner was a fully-dynamic construction with $\mathcal{O}(\log \Delta)$ maximum degree and $\mathcal{O}(\log \Delta)$ lightness, where Δ is the aspect ratio of the point set, defined as the ratio of the length of the largest edge divided by the length of the shortest edge.

In the spaces of bounded doubling dimension, Roditty [82] provided the first dynamic spanner construction whose update time (and therefore recourse) depended solely on the number of points ($\mathcal{O}(\log n)$ for point insertion and $\tilde{\mathcal{O}}(n^{1/3})$ for point removal). This was later improved by Gottlieb and Roditty [60], who extended this result in doubling metrics and provided a better update time as well as the bounded-degree property. The same authors further improved this construction to have an asymptotically optimal insertion time (and therefore recourse) of $\mathcal{O}(\log n)$ under the algebraic decision tree model [61] but logarithmic lightness.

In Chapter 4, we devise a fully dynamic spanner with the aim of minimizing recourse, defined as the number of edge updates following a point insertion or removal. Our spanner maintains constant bounds on its lightness and maximum degree at all times. Our maintenance approach achieves amortized constant recourse per point insertion and amortized $\mathcal{O}(\log \Delta)$ recourse per point deletion. The details of our bounds are provided in theorem 4.24.

Theorem 4.24. Our construction of fully dynamic spanners in d-dimensional Euclidean spaces exhibits a stretch factor of $1+\varepsilon$ and lightness bounded by a constant. Moreover, this construction incurs an amortized $\mathcal{O}(1)$ edge updates after a point insertion and an amortized $\mathcal{O}(\log \Delta)$ edge updates following a point deletion.

The hidden constants in our bounds depend solely on ε and d. While our amortized bound for recourse after point insertion is optimal, we do not claim optimality for point deletion. However, it is noteworthy that our recourse bound for deletion is no worse than the bounds achieved in prior work. To attain our recourse bounds, we introduce novel techniques for iteratively enhancing the weight of the spanner without compromising its other characteristics.

Chapter 2

Separators for Greedy Spanners

2.1 Background

There are various spanner algorithms designed for different applications, depending on the specific additional properties needed in those applications. Well-separated pair decomposition, θ-graphs, and greedy spanners are among the most well-known of these geometric spanner constructions. In this chapter, we focus on the greedy spanner. It was first introduced by Althöfer [3, 4] and Bern, generalizing a pruning strategy used by Das and Joseph [30] on a triangulation of the planar graph [40].

A greedy spanner can be constructed by running the greedy spanner algorithm (Algorithm 1) on a set of points on the Euclidean plane. This short procedure adds edges one at a time to the spanner it constructs, in ascending order by length. For each pair of vertices, in this order, it checks whether that pair already satisfies the bounded stretch inequality using the edges already added. If not, it adds a new edge connecting the pair. Therefore, by construction, each pair of vertices satisfies the inequality, either through previous edges or (if not) through the newly added edge. The resulting graph is therefore a t-spanner. Examples of the results

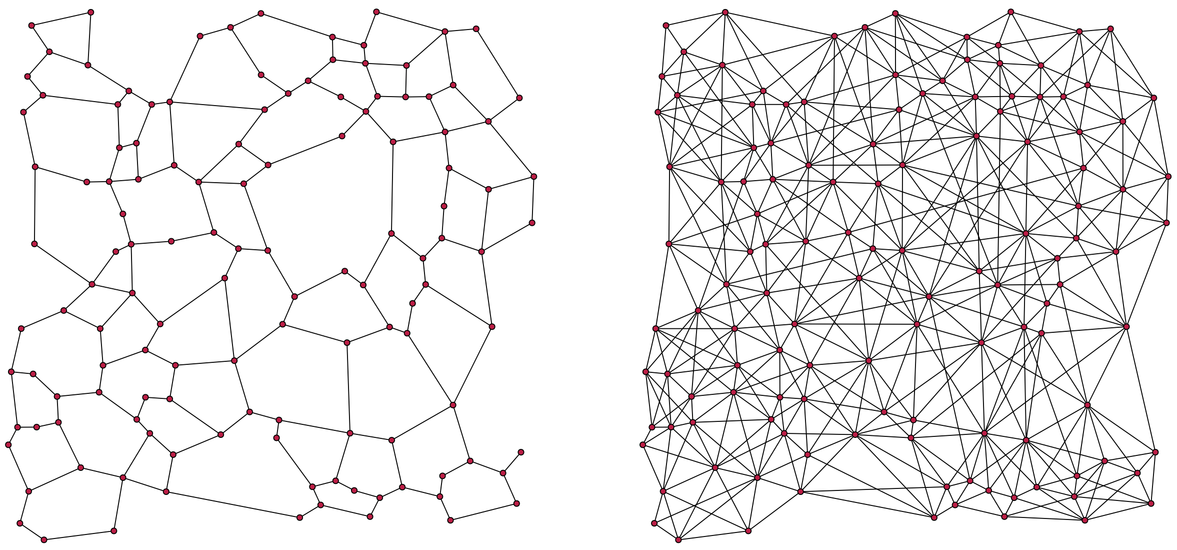


Figure 2.1: Greedy spanners of 128 random points with stretch factor 2 (left) and 1.1 (right)

of this algorithm, for two different stretch factors, are shown in Figure 2.1. Although the 2-spanner in the figure is planar, this is not true for 2-spanners in general: there exist point sets with non-planar greedy t-spanners for arbitrarily large values of t (Figure 2.2), and by placing widely-spaced copies of the same construction within a single point set, one can construct point sets whose greedy t-spanners have linearly many crossings, for arbitrarily large values of t.

Algorithm 1 The naive greedy spanner algorithm.

```
1: procedure NAIVE-GREEDY($V$)
2:     Let $S$ be a graph with vertices $V$ and edges $E = \{\}$
3:     for each pair $(P, Q) \in V^2$ in increasing order of $d(P, Q)$ do
4:         if $d_S(P, Q) > t \cdot d(P, Q)$ then
5:             Add edge $PQ$ to $E$
       return S
```

A naïve implementation of the greedy spanner algorithm runs in time $\mathcal{O}(n^3 \log n)$, where n is the number of given points [15]. Bose et al. [15] improved the running time of Algorithm 1 to near-quadratic time using a bounded version of Dijkstra's algorithm. Narasimhan et al. proposed an approximate version of the greedy spanner algorithm that reached a running time of $\mathcal{O}(n \log n)$, based on the use of approximate shortest path queries [31, 62, 79].

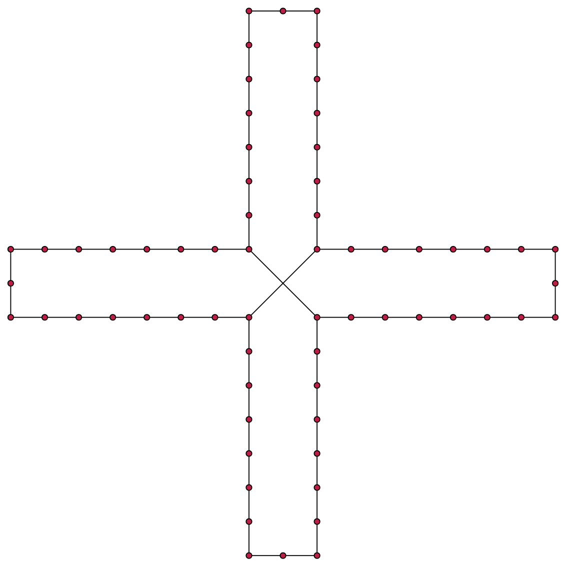


Figure 2.2: Nonplanar greedy spanner with stretch factor 11.3

Despite the simplicity of Algorithm 1, Farshi and Gudmundsson [49] observed that in practice, greedy spanners are surprisingly good in terms of the number of edges, weight, maximum vertex degree, and also the number of edge crossings. Many of these properties have been proven rigorously. Filster and Solomon [51] proved that greedy spanners have size and lightness that is optimal to within a constant factor for worst-case instances. They also achieved a near-optimality result for greedy spanners in spaces of bounded doubling dimension. Borradaile, Le, and Wulff-Nilsen [14] recently proved optimality for doubling metrics, generalizing a result of Narasimhan and Smid [79], and resolving an open question posed by Gottlieb [59], and Le and Solomon showed that no geometric t-spanner can do asymptotically better than the greedy spanner in terms of number of edges and lightness [71]. However, past work has not proven rigorous bounds on the number of crossings of greedy spanners.

One reason for particular interest in bounds on the number of crossings is the close relation, for geometric graphs in the plane, between crossings and *separators*. The well-known planar separator theorem of Lipton and Tarjan [76] states that any planar graph (that is, a geometric graph with no crossings) can be partitioned into subgraphs whose size is at most a constant fraction of the total by the removal of $O(\sqrt{n})$ vertices. This property is central to the efficiency of many algorithms on planar graphs [57, 35, 43, 42, 68], and applied as well in multiple computational geometry problems [52, 6, 67]. Analogous separator theorems have been extended from planar graphs to graphs with few crossings per edge [34], or more generally to graphs with sparse patterns of crossings [44, 9]. Past work has not shown that greedy spanners have small separators, but as we will show, bounds on their crossings can be used to show that they do.

2.2 Overview

In this chapter we prove that greedy t-spanners in the Euclidean plane have few crossings, for any $t > 1$, and we use this result (together with a result of Eppstein and Gupta [44] on graphs with sparse patterns of crossings) to prove that greedy spanners in the Euclidean plane have small separators. In particular, we prove:

- **Claim 1.** Each edge in a greedy spanner can be crossed by only $O(1)$ edges of equal or greater length, where the constant in the $O(1)$ depends only on t, the stretch factor of the spanner. More precisely as $t \to 1$ there are $O(1/(t-1)^2)$ edges that cross the given edge and are longer than it by a factor of $\Omega(1/(t-1))$ (Theorem 2.14), and $1/(t-1)^{O(1)}$ edges that cross the given edge and have length at least ε times it, for any constant $\varepsilon > 0$ (Theorem 2.17).
- **Claim 2.** For some choices of t, there exist greedy spanners in which some edges are crossed by a linear number of (significantly shorter) edges (Theorem 2.24).
- **Claim 3.** Every n-vertex greedy spanner, and every n-vertex subgraph of a greedy spanner, can be partitioned into connected components of size at most cn for a constant $c < 1$ by the removal of $O(\sqrt{n})$ vertices. Again, the constant factor in the $O(\sqrt{n})$ term depends only on the stretch factor of the spanner. Moreover, a separator hierarchy

for the greedy spanner can be constructed from its planarization in near-linear time (Theorem 2.20).

It is known that the spanners that are constructed by some other methods, i.e. semi-separated pair decomposition [2] and hierarchical decomposition [53], have small $\mathcal{O}(\sqrt{n})$-separators in two dimensions. Although experimental results of Farshi and Gudmundsson on greedy spanners of random point sets had shown the number of crossings to be small in practice [49] our results are the first theoretical results on this property, the first to study crossings for worst-case and not just random instances, and the first to prove that greedy spanners have small $\mathcal{O}(\sqrt{n})$-separators.

2.3 Intuition

Our proof that edges can be crossed by only a bounded number of edges of greater or equal length splits into two cases, one for crossings by edges of significantly greater length and another for crossings by edges of similar length.

For edges of significantly greater length, we divide the greedy spanner edges that might cross the given edge into a constant number of nearly-parallel sets of edges, and prove the bound separately within each such set. We show that, within a set of nearly-parallel long edges that all cross the given edge, the edges can be totally ordered by their projections onto a base line, because edges whose endpoints project to nested intervals would contradict the greedy property of the spanner (the inner of two nested edges could be used to shortcut the outer one). By similar reasoning, the endpoints of any two nearly-parallel long crossing edges are separated by a distance that is at least a constant fraction of the length of the smaller edge. This geometric growth in the separation of the endpoints leads to a system of inequalities on the lengths of the edges that can only be satisfied when the number of crossing edges is bounded by a constant.

For edges of comparable length to the crossed edge, we use a grid to partition the crossing edges into a constant number of subsets of edges, such that within each subset all edges have pairs of endpoints that are close to each other relative to the length of the edge, and we show that each of these subsets can contain only a unique edge.

Our construction showing that a single edge can be crossed linearly many times is based on the combination of three "zig-zag" sets of points, evenly spaced in their x-coordinates and alternating between two different y-coordinates. In the top and bottom zig-zag, the distance along the zigzag between two consecutive points with the same y-coordinates is exactly t times the difference between their x-coordinates, while in the middle zig-zag it is slightly greater. The greedy spanner for this point set contains the zig-zag edges, plus a single long edge crossing all of the middle edges, for a pair of points that are far enough from each other along the middle zig-zag for their Euclidean distance to be almost the same as their difference in x-coordinates (differing by a number smaller than the amount by which a single edge of the middle zig-zag exceeds t times its difference in x-coordinates).

The results on separators follow from previous results on the existence of separators in graphs whose edge intersection graphs have bounded degeneracy [44].

2.4 Preliminaries

As we mentioned earlier, t-spanners can be defined in any metric space. For a given graph G, a t-spanner is defined in the following way,

Definition 2.1 (t-spanner)**.** Given a metric graph $G = (V, E, d)$, i.e. weighted graph with distances as weights, a t-spanner is a spanning subgraph S of G such that for any pair of vertices $u, w \in V$,

$$d_G(u, w) \leq t \cdot d(u, w)$$

where $d_G(u, w)$ is the length of the shortest path in G between u and w.

Then the greedy spanner on a given set of points V can be defined in the following way,

Definition 2.2 (greedy spanner)**.** Given a set of points V in any metric space, a greedy spanner on V is a t-spanner that is an output of Algorithm 1.

Here we restrict the problem to geometric graphs and we take advantage of inequalities that hold in geometric space.

We consider the natural embedding that the greedy spanner inherits from its vertices. Edges are drawn as straight segments between the two points corresponding to the two endpoints of the edge. We say two edges of the spanner cross or intersect if their corresponding segments intersect at some interior point. The crossing graph of a given embedding can be defined in this way,

Definition 2.3 (crossing graph)**.** Given a graph $G(V, E)$ and its Euclidean embedding, the crossing graph $Cr(G)$ is a graph $G'(E, C)$ whose vertices are the edges of the original graph and for each two vertices $e, f \in E$ there is an edge between them if and only if they intersect with each other in the embedding given for G.

Most of the proofs here use a lemma that we call the *short-cutting lemma*, which is simple but very useful in greedy spanners. The lemma is proven in [79] and it states that a t-spanner edge cannot be shortcut by some other edges of the spanner by a factor of t. Formally,

Lemma 2.4 (short-cutting)**.** *An edge AB of a greedy t-spanner cannot be shortcut by some other spanner edges by a factor of t, i.e. there is no constant k and points $A =$*

$P_0, P_1, \ldots, P_k = B$ that $P_0P_1, P_1P_2, \ldots, P_{k-1}P_k$ are all spanner edges distinct from AB, and

$$\sum_{i=0}^{k-1} |P_iP_{i+1}| \leq t \cdot |AB|$$

Proof. Suppose on the contrary that such points exist. If AB is larger than all other segments P_iP_{i+1}, then it should be added the last by the greedy algorithm, so when AB is being added all P_iP_{i+1}s are already included in the spanner, and

$$\sum_{i=0}^{k-1} |P_iP_{i+1}| \leq t \cdot AB$$

By the definition AB should not be added to the graph because there is a path in the spanner with length at most $t \cdot AB$, which contradicts the assumption.

So assume that AB is not larger than all P_iP_{i+1}s. Denote the largest among P_iP_{i+1}s by $P_{i_0}P_{i_0+1}$. Then by the assumption

$$\begin{aligned}\sum_{i \neq i_0} |P_iP_{i+1}| + |AB| &\leq \sum_{i \neq i_0} |P_iP_{i+1}| + |P_{i_0}P_{i_0+1}| \\ &= \sum_{i} |P_iP_{i+1}| \leq t \cdot |AB| \leq t \cdot |P_{i_0}P_{i_0+1}|\end{aligned}$$

which shows that $P_{i_0}P_{i_0+1}$ can be shortcut by some smaller segments by a factor of t, which is impossible according to what we proved earlier in this lemma. □

If some of the segments P_iP_{i+1} are not included in the spanner, the same argument still works but a factor t appears before the term $|P_iP_{i+1}|$ in the summation. So

Corollary 2.5 (Extended short-cutting)**.** *Given a greedy t-spanner S and an edge AB of*

S, there cannot be a constant k and points $A = P_0, P_1, \ldots, P_k = B$ such that

$$\sum_{P_iP_{i+1} \in S} |P_iP_{i+1}| + t \cdot \sum_{P_iP_{i+1} \notin S} |P_iP_{i+1}| \leq t \cdot |AB|$$

Proof. Assume to the contrary that such points exist. For any non-spanner segment P_iP_{i+1} there exists a path $\mathcal{P}(P_iP_{i+1})$ from P_i to P_{i+1} that has length at most $t \cdot |P_iP_{i+1}|$. So by replacing each non-spanner segment P_iP_{i+1} by its own path $\mathcal{P}(P_iP_{i+1})$ in the shortcut path $P_0P_1 \ldots P_k$ the length of the resulting path would be

$$\sum_{P_iP_{i+1} \in S} |P_iP_{i+1}| + \sum_{P_iP_{i+1} \notin S} |\mathcal{P}(P_iP_{i+1})| \leq \sum_{P_iP_{i+1} \in S} |P_iP_{i+1}| + t \cdot \sum_{P_iP_{i+1} \notin S} |P_iP_{i+1}| \leq t \cdot |AB|$$

which shows that the new path is also a shortcut for AB by a factor of t. But the new path only consists of the spanner segments, which is impossible by Lemma 2.4 and leads to a contradiction. □

In the following section we consider intersections between an arbitrary edge of a greedy spanner and sufficiently larger edges, and we show a constant bound on the number of intersections per edge. In Section 2.5.5 we again prove a constant bound for the number of intersections between a spanner edge and other edges of almost the same length. Finally, in section 2.7 we introduce an example in which the number of intersections with smaller edges can be more than any constant bound, completing our analysis. In section 2.6 we introduce some new results and improvements based on the constant bound we provided earlier.

2.5 Few intersections with long edges

In this section, we prove an upper bound on the number of intersections of an edge with sufficiently larger edges. We will specifically show that the number of intersections, in this case, has a constant bound that only depends on t. Later in Section 2.5.5 we prove a constant bound also exists for the intersections with the edges that have almost the same length of the intersecting edge. Hence we prove our first claim.

In this setting, we consider an arbitrary edge AB of the spanner, and we are interested in counting the number of intersections that AB may have with sufficiently larger edges, i.e. edges PQ that intersect AB at some interior point with $|PQ| > c \cdot |AB|$ for some constant $c > 1$ which we will specify later.

First, we only consider a set of *almost-parallel* spanner segments that cross AB, where we define the term *almost-parallel* below, and we put a bound on the number of these segments. Then we generalize the bound to hold for all large spanner segments that cross AB.

2.5.1 Definitions

Definition 2.6 (almost-parallel)**.** We say a pair of arbitrary segments PQ and RS in the plane are *almost-parallel* or *θ-parallel* if there is an angle of at most θ between them. We say a set of segments are *almost-parallel* if every pair of segments chosen from the set are almost-parallel.

For any set of almost-parallel segments, we define a baseline to measure the angles and distances with respect to that line.

Definition 2.7 (baseline)**.** Given a set of almost-parallel (or θ-parallel) segments in the plane, denoted by S, the *baseline* $b(S)$ of the set of segments S is the segment with the smallest slope.

We use the uniqueness of the segment chosen in Definition 2.7 and we emphasize that any other definition works if it determines a unique segment for any almost-parallel set of segments.

In Section 2.5.2, we define a total ordering on a set of almost-parallel segments that cross a spanner segment AB. Once we have sorted these segments based on the ordering, in Section 2.5.3 we prove the distance between the endpoints of two consecutive segments is at least a constant fraction of the length of the smaller segment. Putting together these two parts, in Section 2.5.4 we prove there cannot be more than a constant number of segments in the sequence.

2.5.2 A total ordering on almost-parallel intersecting segments

In this section, we define an ordering on a set of almost-parallel segments of the t-spanner. The ordering is based on the order of the projections of the endpoints of the segments on the baseline corresponding to the segments. We first define the ordering and then we use Lemma 2.9 and Lemma 2.10 to prove that it is a total ordering when the set of almost-parallel segments are all crossing a given segment of the spanner.

Consider a set of almost-parallel spanner segments that cross some spanner segment. One can define an ordering on this set of almost-parallel segments, which we call the *endpoint-ordering*, based on how their endpoints are ordered along the direction they are aligned to. We formulate the definition in the following way,

Definition 2.8 (endpoint-ordering)**.** Let $S = \{P_iQ_i : i = 1, 2, \dots, k\}$ be a set of almost-parallel segments. Also let l be the baseline of S, $b(S)$. Define the *endpoint-ordering* $\mathcal{R}$ between two segments P_iQ_i and P_jQ_j by projecting the endpoints P_i, P_j, Q_i, Q_j to the base-

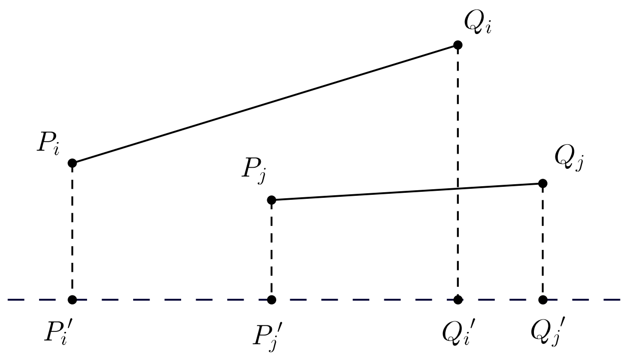


Figure 2.3: Ordering segments by projecting on the baseline l, here $P_iQ_i <_{\mathcal{R}} P_jQ_j$.

line l and comparing the order of the projected points P_i', P_j', Q_i', Q_j' along an arbitrary direction of the baseline l,

- $P_iQ_i <_{\mathcal{R}} P_jQ_j$ if the projections are ordered as $P_i'P_j'Q_i'Q_j'$ or $P_i'Q_i'P_j'Q_j'$.
- $P_iQ_i >_R P_jQ_j$ if they are ordered as $P_j'P_i'Q_j'Q_i'$ or $P_j'Q_j'P_i'Q_i'$. (Figure 2.3)

We claim that the endpoint-ordering is a total ordering on the set of almost-parallel segments. This basically means that after projecting two almost-parallel segments on the baseline, none of the resulting projections would lie completely inside the other one. Other cases correspond to a valid endpoint-ordering.

In order to prove this, first, we prove a simpler case when the two segments intersect with each other. This assumption will help to significantly simplify the proof. Later we use this lemma to show the original claim is also true.

Lemma 2.9. *Let MN and PQ be two intersecting segments from a set of θ-parallel spanner segments. Also assume that $\theta < \frac{t-1}{2t}$ where t is the stretch factor of the spanner. Then MN and PQ are endpoint-ordered, i.e. the projection of one of the segments on the baseline of the set cannot be included in the projection of the other one.*

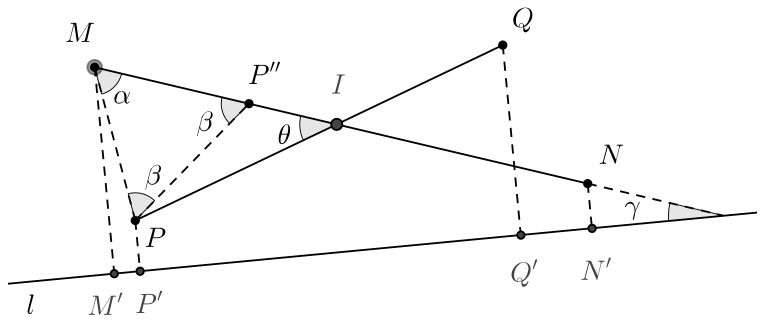


Figure 2.4: Proof of Lemma 2.9.

Proof. We prove the lemma by contradiction. Without loss of generality suppose that the projections of P and Q on some baseline l are both between the projections of M and N (on the same baseline). We show that MN can be shortcut by PQ by a factor of t, i.e.

$$t \cdot |MP| + |PQ| + t \cdot |QN| \leq t \cdot |MN|$$

Let P', Q', M', and N' be the corresponding projections of P, Q, M, and N on l, respectively (Figure 2.4). Also let I be the intersection point and $\alpha = \angle PMI$, and also γ to be the angle between MN and the baseline, according to the figure. By the assumption P' is between M' and N', so $\alpha \leq \pi/2 + \gamma \leq \pi/2 + \theta$. Let also P'' be the point on MN s.t. $|MP''| = |MP|$ and $\beta = \angle MPP'' = \angle MP''P$. Then by sine law,

$$\frac{|MI| - |MP|}{|PI|} = \frac{|P''I|}{|PI|} = \frac{\sin(\beta - \theta)}{\sin\beta} = \frac{\sin(\pi/2 - \alpha/2 - \theta)}{\sin(\pi/2 - \alpha/2)} = \frac{\cos(\alpha/2 + \theta)}{\cos(\alpha/2)} \\ = \cos\theta - \sin\theta\tan(\alpha/2) \tag{2.1}$$

but we have,

$$\cos\theta \geq 1 - \theta^2/2 \geq 1 - \theta/4 \tag{2.2}$$

as $\theta < \frac{t-1}{2t} < 1/2$. Also,

$$\tan(\alpha/2) \le \tan(\pi/4 + \theta/2) = \tan(\pi/4 + 1/4) < \frac{7}{4} \tag{2.3}$$

Putting together Equation 2.1, Equation 2.2, and Equation 2.3, also using $\sin\theta \le \theta$,

$$\frac{|MI| - |MP|}{|PI|} \ge (1 - \theta/4) - (\frac{7}{4})\theta = 1 - 2\theta > \frac{1}{t}$$

which is equivalent to $t \cdot |MI| - t \cdot |MP| \ge |PI|$. Similarly, $t \cdot |NI| - t \cdot |NQ| \ge |QI|$. Adding together,

$$t \cdot |MN| - t \cdot |MP| - t \cdot |NQ| \ge |PQ|$$

which is what we are looking for. □

Lemma 2.9 assumes that segments intersect at some interior point. In order to prove the totality of the ordering, we also need to prove the claim when the segments do not intersect with each other. Instead, in this case, both segments intersect some spanner edge. We use Lemma 2.9 to prove this in the Lemma 2.10.

Lemma 2.10. *Let MN and PQ be two segments chosen from a set of θ-parallel spanner segments that cross a spanner edge AB. Also assume that $\theta < \frac{t-1}{2(t+1)}$, and $\min(|MN|, |PQ|) \ge \frac{3t(t+1)}{t-1}|AB|$, where t is the spanner parameter. Then MN and PQ are endpoint-ordered.*

Proof. Again, the proof goes by contradiction. Without loss of generality suppose that the projections of P and Q on some baseline l are both between the projections of M and N (on the baseline). We use Lemma 2.9 to show that MN can be shortcut by PQ by a factor of t, i.e.

$$t \cdot |MP| + |PQ| + t \cdot |QN| \le t \cdot |MN|$$

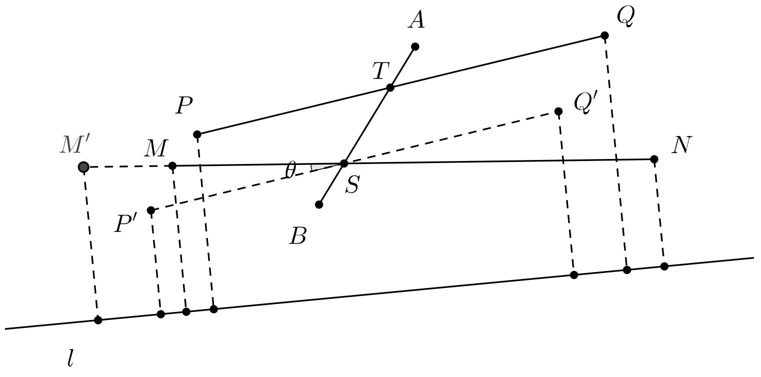


Figure 2.5: Proof of Lemma 2.10.

The idea is to move PQ by a small amount with respect to its length, so that the new segment intersects MN, and then use Lemma 2.9. We also keep track of the changes in both sides of the inequality during this movement to show the inequality holds for original points.

Let the segments MN and PQ intersect AB at S and T, respectively. One can move PQ by vector $\overrightarrow{TS}$ in order to intersect MN. Let the new segment be $P'Q'$. But the projections of P' and Q' on the baseline may not be between M and N anymore. In order to preserve this, we can extend MN on one side by $|\overrightarrow{TS}|$ to get a new segment $M'N'$. Extending by this amount is enough to preserve the betweenness. For example, in Figure Figure 2.5), we moved PQ by $\overrightarrow{TS}$ to get $P'Q'$. Now $P'Q'$ intersects MN (at S), but the projection of P' on the baseline is not between the projections of M and N anymore. So we extend MN from M by $|\overrightarrow{TS}|$ to get M'. Now the projection of P' on the baseline is between the projections of M' and N. Before the movement the projections of P and Q are both between the projections of M and N, so after movement at most one of the projections of P' or Q' can be outside of the projections of M and N. So extending on one side will be sufficient.

Now $P'Q'$ and $M'N'$ intersect each other and the projections of P' and Q' are between the projections of M' and N', we can use Lemma 2.9. By the assumption $\theta = \frac{t'-1}{2t'}$ where

$t' = (t+1)/2$, so Lemma 2.9 implies that,

$$t' \cdot |M'P'| + |P'Q'| + t' \cdot |Q'N'| \le t' \cdot |M'N'| \tag{2.4}$$

By the triangle inequality after this movement MP and NQ each will decrease by at most $|\overrightarrow{TS}| \le |AB|$. So,

$$|M'P'| \ge |MP| - |AB|,\ |N'Q'| \ge |NQ| - |AB| \tag{2.5}$$

Also length of MN will increase by at most $|\overrightarrow{TS}| \le |AB|$, so

$$|M'N'| \le |MN| + |AB| \tag{2.6}$$

The length of PQ does not change though. Putting together Equation 2.4, Equation 2.5, and Equation 2.6,

$$\begin{aligned}|PQ| = |P'Q'| &\le t' \cdot (|M'N'| - |M'P'| - |N'Q'|)\\ &\le \frac{t+1}{2} \cdot (|MN| - |MP| - |NQ| + 3|AB|)\\ &\le \frac{t+1}{2} \cdot (|MN| - |MP| - |NQ|) + \frac{t+1}{2} \cdot (\frac{t-1}{t(t+1)}|PQ|)\end{aligned}$$

So

$$|PQ| \le t \cdot (|MN| - |MP| - |NQ|)$$

which is the result we are looking for. □

Based on Lemma 2.10 it is easy to prove the main result of this section, Proposition 2.11.

Proposition 2.11. *Given an arbitrary edge AB of a t-spanner, for a set of sufficiently large almost-parallel spanner edges that intersect AB, the* endpoint-ordering *we defined in*

Definition 2.8 is a total ordering.

Proof. Totality requires reflexivity, anti-symmetry, transitivity, and comparability. Reflexivity and transitivity are trivial because of the projection. Anti-symmetry and comparability follow directly from Lemma 2.10. □

Now that we have ordered the set of almost-parallel spanner segments, we can prove a lower bound on the distance of two ordered segments. Later we prove a bound on the number of these segments based on the resulting distance lower bound.

2.5.3 Lower bounding the distance of endpoints of two crossing segments

In Section 2.5.2 we restricted the problem to a set of almost-parallel spanner segments that intersect another spanner segment, and we defined an ordering on these segments. The next step is to find a lower bound on the distance of two almost-parallel segments that intersect some spanner segment AB. The idea is to show that both endpoints of two ordered segments cannot be arbitrarily close, and hence there cannot be more than a constant number of them in a sequence.

More specifically, we show in Proposition 2.13 that the corresponding endpoints of two almost-parallel spanner segments that both cross the same spanner segment should have a distance of at least a constant fraction of the length of the smaller segment, otherwise the longer segment could be shortcut by the smaller one, which is indeed a contradiction. A weaker version of this lemma is proven in [79] and it is called "gap property", but the inequality we show here is actually stronger.

First we propose a geometric inequality in Lemma 2.12 that helps to prove the proposition.

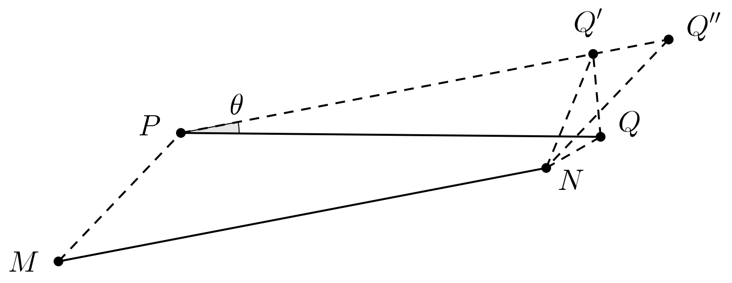


Figure 2.6: Proof of Lemma 2.12.

Then we complete the proof of the proposition at the end of this section.

Lemma 2.12. *Let MN and PQ be two segments in the plane with angle θ. Then*

$$||MN| - |PQ|| > ||MP| - |NQ|| - 2\sin(\theta/2) \cdot |PQ|$$

Proof. By swapping MN and PQ, it turns out that the case where $|MN| \geq |PQ|$ is stronger than $|MN| \leq |PQ|$. So without loss of generality, let $|MN| \geq |PQ|$ and by symmetry $|MP| \geq |NQ|$. Let Q' be the rotation of Q around P by θ, so that PQ' and MN are parallel, and $|PQ'| = |PQ|$ (Figure 2.6). Let Q'' be the point on the ray PQ' where $|PQ''| = |MN|$. As a result Q'' and P will be on different sides of Q'. By the triangle inequality,

$$\begin{aligned} |MN| - |PQ| = |PQ''| - |PQ'| = |Q'Q''| &\geq |NQ''| - |NQ'| \\ &= |MP| - |NQ'| \geq |MP| - (|NQ| + |QQ'|) \\ &= |MP| - |NQ| - 2|PQ| \cdot \sin(\theta/2) \end{aligned}$$

□

Now we state and prove Proposition 2.13. As we mentioned earlier, the idea is to show one of the segments can be shortcut by the other one if one of the matching endpoints is very close. In the simplest case when the segments are two opposite sides of a rectangle, it is easy to see that a distance of $\frac{t-1}{2}|PQ|$ on both sides is required to prevent short-cutting. In the general case, when the segments are placed arbitrarily, Proposition 2.13 holds.

Proposition 2.13. *Let MN and PQ be two θ-parallel spanner segments. The matching endpoints of these two segments cannot be closer than a constant fraction of the length of the smaller segment. More specifically,*

$$\min(|MP|, |NQ|) \geq \frac{t-1-2\sin(\theta/2)}{2t} \min(|MN|, |PQ|)$$

Proof. Without loss of generality and by symmetry, let $|NQ| \leq |MP|$. Suppose, on the contrary, that $|NQ| < \frac{t-1-2\sin(\theta/2)}{2t}|PQ|$. Then,

$$\begin{aligned} t \cdot |MP| + |PQ| + t \cdot |NQ| &\leq t \cdot (|MN| - |PQ| + |NQ| + 2\sin(\theta/2) \cdot |PQ|) + |PQ| + t \cdot |NQ| \\ &= t \cdot |MN| - (t-1-2\sin(\theta/2))|PQ| + 2t \cdot |NQ| \\ &< t \cdot |MN| \end{aligned}$$

So MN can be shortcut by PQ within a factor of t which contradicts the extended short-cutting lemma for the edge MN and the path $MPQN$. □

So far, in Proposition 2.13 we proposed an ordering on the set of almost-parallel spanner segments that cross a given edge and we proved each of these segments has a significant distance from the other ones. In the next section we put together these results and we find a constant upper bound on the number of these segments.

2.5.4 Putting things together

Based on the ordering proposed in Section 2.5.2, and the lower bound we proved in Section 2.5.3, we can show that the following constant upper bound on the number of intersections with sufficiently large edges holds.

If we look at one of the endpoints of the endpoint-ordered sequence of almost-parallel spanner segments, and we project them on the baseline, the distance of every two consecutive projected points cannot be smaller than a constant fraction of the length of the smaller segment, i.e. $|P_i'P_{i+1}'| \geq C \cdot \min(|P_iQ_i|, |P_{i+1}Q_{i+1}|)$ for all values of $i = 0, 1, \ldots, k-1$. Summing up these inequalities leads to a bound on k, the number of segments.

Theorem 2.14. *For sufficiently small θ, the number of sufficiently large θ-parallel segments that intersect a given edge AB of a t-spanner is limited by*

$$\frac{4t}{(t-1-2\sin(\theta/2))\cos\theta} + 1$$

By sufficiently large we specifically mean larger than $\frac{3t(t+1)}{t-1}|AB|$.

Proof. Let P_iQ_is be the segments larger than AB that intersect AB at some angle in $[\alpha, \alpha+\theta)$. Let P_0Q_0 be the shortest edge among P_iQ_is. Because of the total ordering, at least half of the segments are larger than P_0Q_0 with respect to the ordering $\mathcal{R}$, or at least half of them are smaller than P_0Q_0 with respect to $\mathcal{R}$. Without loss of generality, assume that half of the segments are larger than P_0Q_0 with respect to $\mathcal{R}$, and they are indexed by $i = 1, 2, \ldots, (k-1)/2$. Also let P_i's and Q_i's be the projections of P_is and Q_is on the base line l. By Proposition 2.13, for all i, P_{i+1} is farther than P_i by a constant fraction of

$\min(|P_iQ_i|, |P_{i+1}Q_{i+1}|)$, so

$$\sum_{i=0}^{(k-3)/2} |P_iP_{i+1}| > \frac{t-1-2\sin(\theta/2)}{2t} \sum_{i=0}^{(k-3)/2} \min(|P_iQ_i|, |P_{i+1}Q_{i+1}|)$$
$$\geq \frac{t-1-2\sin(\theta/2)}{2t} \cdot \frac{k-1}{2}|P_0Q_0|$$

If $k \geq \frac{4t}{(t-1-2\sin(\theta/2))\cos\theta} + 1$,

$$\sum_{i=0}^{(k-3)/2} |P_iP_{i+1}| > \frac{1}{\cos\theta}|P_0Q_0|$$

or equivalently

$$|P_0Q_0| < \sum_{i=0}^{(k-3)/2} |P_iP_{i+1}|\cos\theta \leq \sum_{i=0}^{(k-3)/2} |P'_iP'_{i+1}| = |P'_0P'_{\frac{k-1}{2}}|$$

which is not possible, because $P'_0P'_{\frac{k-1}{2}}$ lies inside $P'_0Q'_0$ and so $|P'_0P'_{\frac{k-1}{2}}| \leq |P'_0Q'_0| \leq |P_0Q_0|$ which contradicts the last inequality above. □

The constraints on θ imposed by our earlier lemmas imply that, as $t \to 1$, we should choose θ proportional to $t-1$. Asymptotically, as $t \to 1$, the number of large segments of all angles that intersect AB is $O(1/(t-1)^2)$, with one factor of $1/(t-1)$ coming from the bound in the theorem and the second factor coming from the number of different classes of nearly-parallel segments.

2.5.5 Almost-equal length edges

In the previous subsections, we proved a bound on the number of intersections with relatively larger edges. Here we prove a constant bound on the number of intersections with edges that are nearly the same length as the length of the intersecting edge. Later in section 2.7 we consider intersections with relatively smaller edges, which completes our analysis for this problem.

For same-length intersections Lemma 2.10 does not hold anymore, hence the endpoint-ordering is not necessarily a total ordering in this case. Since totality is a key requirement for the rest of the proof the same proof will not work anymore. But Proposition 2.13 still holds as it has no assumption on the ordering of the segments.

Our idea is to partition the neighborhood of AB into a square network, such that no two spanner segments can have both endpoints in the same squares (Figure 2.7). If this happens, then by Proposition 2.13 one of the segments should be shortcut by the other one, leading to a contradiction because both segments are already included in the spanner.

We first prove a simpler version of Proposition 2.13 that does not include θ in the inequality, as we are not using the almost-parallel assumption and the value of θ can be large enough to make the inequality in Proposition 2.13 trivial. We will use this modified version to prove our claim.

Lemma 2.15. *Given a greedy spanner with parameter t and two spanner segments MN and PQ,*

$$\max(|MP|, |NQ|) \geq \frac{t-1}{2t} \min(|MN|, |PQ|)$$

Proof. Suppose on the contrary that

$$\max(|MP|, |NQ|) < \frac{t-1}{2t} \min(|MN|, |PQ|)$$

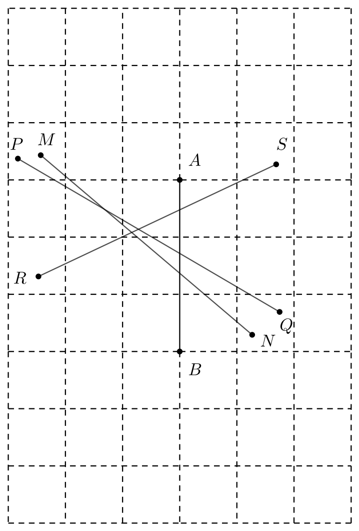


Figure 2.7: Partition of the area around AB

Also, without loss of generality assume that $|MN| \geq |PQ|$. Then,

$$t \cdot |MP| + |PQ| + t \cdot |NQ| \leq (t-1)\min(|MN|, |PQ|) + |PQ| = t \cdot |PQ| \leq t \cdot |MN|$$

which contradicts the extended short-cutting lemma for the edge MN and the path $MPQN$.

$\square$

Proposition 2.16. *The number of spanner segments PQ that cross a segment AB of a t-spanner and that have length within $\alpha \cdot |AB| \leq |PQ| \leq \beta \cdot |AB|$ is limited by*

$$\left\lceil \frac{2\beta(2\beta+1)}{\alpha^2} \cdot \frac{8t^2}{(t-1)^2} \right\rceil^2$$

where t is the spanner parameter.

Proof. Partition the area around AB with squares of edge length $\frac{t-1}{2\sqrt{2}t} \cdot \alpha|AB|$ with edges parallel or perpendicular to AB. The area that an endpoint of a crossing segment can lie in is a rectangle of size $(2\beta+1)|AB|$ by $2\beta|AB|$ (Figure 2.7). The total number of squares in this area would be

$$\frac{2\beta(2\beta+1)}{\alpha^2} \cdot \frac{8t^2}{(t-1)^2}$$

But for each crossing segment the pair of squares that contain the two endpoints of the segment is unique. Otherwise two segments, e.g. MN and PQ, will have both endpoints at the same pair, which means

$$\max(|MP|, |NQ|) < (\sqrt{2})(\frac{t-1}{2\sqrt{2}t} \cdot \alpha|AB|) = \frac{t-1}{2t} \cdot \alpha|AB| \leq \frac{t-1}{2t} \min(|MN|, |PQ|)$$

which cannot happen due to Lemma 2.15. So the total number of pairs, and hence the total number of crossing segments, would be

$$\left[\frac{2\beta(2\beta+1)}{\alpha^2} \cdot \frac{8t^2}{(t-1)^2}\right]^2$$

□

In Proposition 2.16 both α and β can be chosen arbitrarily, and the bound is a strictly increasing function of β and a strictly decreasing function of α. The bound tends to infinity when β is large enough, and also when α is small enough. So it basically does not prove any constant bound for the cases that edges are very small or very large. But for the edges of almost the same length, it gives a constant upper bound.

Putting together the main results of section 2.5 and Section 2.5.5 we can prove the following bound for the number of intersections with not-relatively-small spanner segments.

Theorem 2.17. *Given a spanner segment AB in the Euclidean plane and a positive constant ε, the number of edges of length at least $\varepsilon \cdot |AB|$ of the spanner that intersect AB is $\mathcal{O}(\frac{t^{12}}{\varepsilon^4(t-1)^8})$.*

Proof. By Theorem 2.14 the number of intersections with edges PQ such that $|PQ| \geq \frac{3t(t+1)}{t-1}|AB|$ is bounded by

$$C_1 = \frac{4t}{(t-1-2\sin(\theta/2))\cos\theta} + 1 \in \mathcal{O}(\frac{t}{t-1})$$

On the other side, putting $\alpha = \varepsilon$ and $\beta = \frac{3t(t+1)}{t-1}$ into Proposition 2.16 implies that the number of intersections with edges larger than AB and smaller than $\frac{3t(t+1)}{t-1}|AB|$ is at most

$$C_2 = \left[\frac{2}{\varepsilon^2}\left(\frac{3t(t+1)}{t-1}\right)\left(2\frac{3t(t+1)}{t-1}+1\right)\left(\frac{8t^2}{(t-1)^2}\right)\right]^2 \in \mathcal{O}(\frac{t^{12}}{\varepsilon^4(t-1)^8})$$

Hence the number of intersections with edges larger than AB is at most $C_1 + C_2$, which is $\mathcal{O}(\frac{t^{12}}{\varepsilon^4(t-1)^8})$. □

In section 2.5 we proved the number of intersections with sufficiently large edges is bounded by a constant and now we completed the proof for all larger edges. In Appendix 2.7, we show that the same argument does not work for intersections with arbitrarily smaller edges, and we provide an example that shows there can be an arbitrarily large number of intersections with smaller edges. This completes our analysis of the problem. In the following section, we show some of the applications of this result, most importantly, the sparsity of the crossing graph of the greedy spanner.

2.6 Separators

In this section, we use the crossing bound that we proved in Theorem 2.17 to show that greedy spanners have small separators. First, we start with the definition of *degeneracy*, which is a measure of sparsity of a graph.

Definition 2.18 (degeneracy)**.** A graph G is called k-degenerate, if each subgraph of G has a vertex of degree at most k. The smallest value of k for which a graph is k-degenerate is called the *degeneracy* of the graph.

The first important consequence of Theorem 2.17 is the constant degeneracy of the crossing graph of the greedy spanner, implying its sparsity and linearity of the number of edges, i.e. crossing.

Theorem 2.19. *The crossing graph of a greedy t-spanner has a constant degeneracy.*

Proof. In any subgraph of the crossing graph, by Theorem 2.17 the node corresponding to the smallest edge has at most a constant number of neighbours. □

This, together with the result of [44] implies the existence of sublinear separators for greedy spanners. A separator is a subset of vertices whose removal splits the graph into smaller pieces. A sublinear separator is a sublinear number of vertices with the same property. The splitting can be recursively performed on the smaller parts and a separator hierarchy can be constructed in this way, which effectively helps in the design of new recursive algorithms. The planarization of a graph, which is obtained by adding new vertices on the edge intersections of the graph, would help us to find such hierarchy in linear time, otherwise, a near-linear time algorithm would be used.

Theorem 2.20. *Greedy spanners have separators of size $\mathcal{O}(\sqrt{n})$. Also, a separator hierarchy for them can be constructed from their planarization in linear time.*

Proof. By Theorem 2.19 the crossing graph of the greedy t-spanner has a constant degeneracy, so by Theorem 6.9 of [44] they have separators of size $\mathcal{O}(\sqrt{n})$. Also by the same theorem, a separator hierarchy for them can be constructed from their planarization in linear time. □

One of the basic algorithms that can be improved using the separator hierarchy is Dijkstra's single-source shortest path algorithm, which runs in $\mathcal{O}(n \log n)$ time on a graph with n vertices. As a result of Theorem 2.20 linear algorithms exist for finding single-source shortest path on greedy spanners, If the planarization has not already been found, it can be constructed in time $O(n \log^{(i)} n)$ for any constant i, where $\log^{(i)}$ denotes the i-times iterated logarithm, e.g. $\log^{(3)} n = \log\log\log n$ [43].

Corollary 2.21. *Single source shortest paths can be computed in time $O(n \log^{(i)} n)$ on a greedy spanner.*

Proof. This follows from the planarization algorithm and from the existence and construction of separators from planarizations by Corollary 6.10 of [44]. □

2.7 Many intersections with short edges

We proved in sections 2.5 and 2.5.5 that, in greedy spanners, each edge has $O(1)$ crossings with edges of greater or equal length. It is natural to ask whether this holds more generally for all crossings, regardless of length. That is, is the total number of crossings for each edge bounded by a constant, depending only on t? In this section we will show that this is not true, by constructing a family of arrangements of points in the plane that have arbitrarily many intersections between a long edge and a set of smaller edges.

2.7.1 Zig-zags

The building block of our construction is an arrangement of points which form a zig-zag shape, as in Figure 2.8. After running the greedy spanner algorithm on a horizontal zig-zag like this, denoted by Z, if Z is not stretched too much along the vertical axis, the first set of edges that will be added to the graph by the greedy algorithm are actually the zig-zag edges that are drawn in Figure 2.8. Then, depending on the shape of the zig-zag and parameter

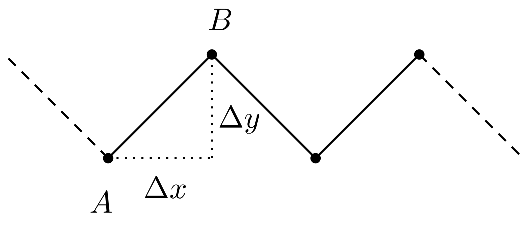


Figure 2.8: A horizontal zig-zag, and its stretch factor $\Delta y/\Delta x$

t, other edges may or may not be added in the future iterations. More specifically, we will show that this only depends on a parameter we call the *stretch-factor* of the zig-zag.

Definition 2.22 (zig-zag)**.** Let $Z = P_0P_1\dots P_k$ be a sequence of points on the Euclidean plane. We say Z forms a *Zig-Zag* if there exist two perpendicular vectors $\overrightarrow{\Delta x}$ and $\overrightarrow{\Delta y}$ that

$$P_i = P_0 + i\overrightarrow{\Delta x} + (i \bmod 2)\overrightarrow{\Delta y}$$

The direction of the vector $\overrightarrow{\Delta x}$ is called the *direction* of the zigzag and the ratio $|\overrightarrow{\Delta y}|/|\overrightarrow{\Delta x}|$ is called the *stretch factor* of the zig-zag, and is denoted by $s(Z)$. (Figure 2.8)

Hence a zig-zag which is more stretched toward the $\overrightarrow{\Delta y}$ vector will have a larger stretch-factor, and a zig-zag which is more stretched along the $\overrightarrow{\Delta x}$ vector will have a smaller stretch-factor.

Lemma 2.23 (zig-zag spanner)**.** *Consider a zig-zag $Z = P_0P_1\dots P_k$ with more than two vertices ($k > 2$) in which the consecutive pairs P_iP_{i+1} are connected to each other ($0 \le i < k$). For any $t > 1$, the zig-zag forms a t-spanner if and only if $s(Z) \le \sqrt{t^2-1}$.*

Proof. For $i < j$, the length of the path between P_i and P_j is

$$d_Z(P_i, P_j) = (j-i)|\overrightarrow{\Delta x} + \overrightarrow{\Delta y}|$$

while the Euclidean distance between P_i and P_j is

$$d(P_i, P_j) = |(j-i)\overrightarrow{\Delta x} + (j - i \bmod 2)\overrightarrow{\Delta y}|$$

The zig-zag forms a t-spanner if and only if $d_Z(P_i, P_j) \leq t \cdot d(P_i, P_j)$ for all $i < j$. Assume that $(j-i) \bmod 2 = 0$, this inequality turns into

$$(j-i)|\overrightarrow{\Delta x} + \overrightarrow{\Delta y}| \leq t \cdot (j-i)|\overrightarrow{\Delta x}|$$

which is equivalent to $s(Z) \leq \sqrt{t^2-1}$. So this is a necessary condition, and it can be shown that it is a sufficient condition too. Because assuming $s(Z) \leq \sqrt{t^2-1}$, in a similar way,

$$(j-i)|\overrightarrow{\Delta x} + \overrightarrow{\Delta y}| \leq t \cdot (j-i)|\overrightarrow{\Delta x}|$$

The left side of the inequality is $d_Z(P_i, P_j)$ and the right side is no more than $t \cdot d(P_i, P_j)$ because it is missing the term $(j - i \bmod 2)\overrightarrow{\Delta y}$, so $d_Z(P_i, P_j) \leq t \cdot d(P_i, P_j)$. $\square$

2.7.2 Introducing the arrangement

Now we introduce the arrangement. Consider two horizontal zig-zags U on the top and B on the bottom which are connected together using a middle zig-zag M (Figure 2.9). U is colored by green, B is colored by blue, and M is colored by red. So there are four rows of points and three zig-zags U, M, and B, which connect these points together. U and M share the second row, while M and B share the third row. The first row is only included in U, and the last row is only included in B. For now, suppose that there are enough points in each row. Later we will see that if the number of points is larger than a specific amount, then a large edge appears at some point in the greedy algorithm, intersecting many edges in between.

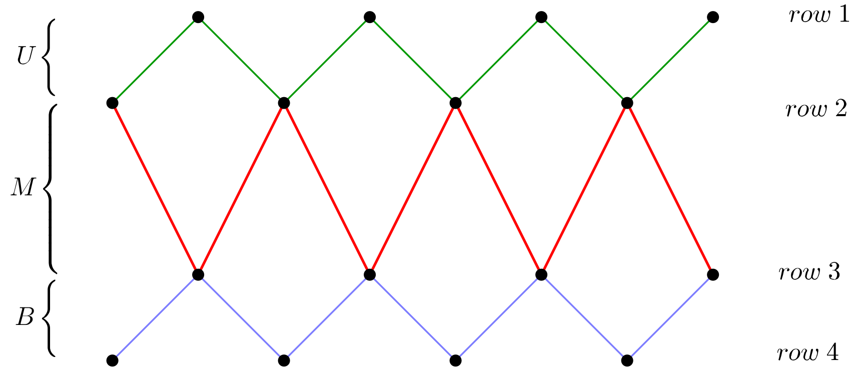


Figure 2.9: Example with more than constant intersections with smaller edges

All of the zig-zags U, M, and B can have arbitrary stretch-factors as we can move the rows up or down to adjust the stretch-factor of each zig-zag independently. So assume that $s(U) = s(B) = \sqrt{t^2 - 1}$ and $s(M) = \sqrt{(t + \delta)^2 - 1}$, for some small positive δ which will be specified later. In other words, U and B are the most stretched zig-zags that form a t-spanner and M is a slightly more stretched zig-zag, which is not a t-spanner by itself anymore.

With this choice of stretch-factors, it is not hard to see, by the Pythagorean theorem, that the length of the zig-zag path between two points on U, say a and b, is exactly $t \cdot |x_a - x_b|$. And the length of the path between two points on B is also the same expression. But in a similar way, the length of the zig-zag path between two points on M would be slightly more, $(t + \delta) \cdot |x_a - x_b|$.

2.7.3 Simulating the greedy algorithm on the arrangement

For an appropriate choice of t (one causing the angles of all zig-zags to lies strictly between $60°$ and $120°$), the greedy spanner algorithm will first add the zig-zag edges in U, B, and M, as they are the closest pairs of vertices. According to the chosen stretch-factors, no edges

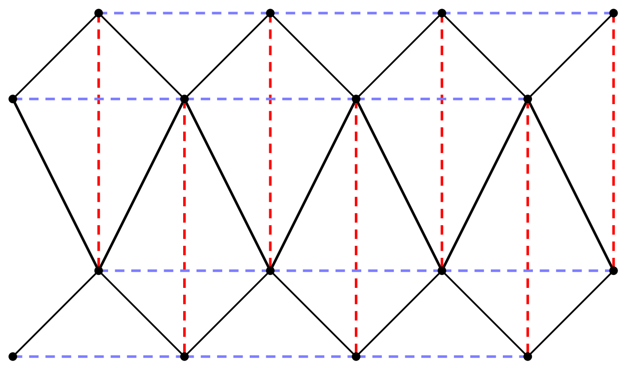


Figure 2.10: Vertical dashed segments are included in the graph but not horizontal ones.

will be added to U and B in the future. For example, the horizontal dashed blue edges in Figure 2.10 will not be added as the endpoints of these segments both belong to U or B, which are t-spanners by themselves. So any potential edge must be between U and B.

The next set of edges that may be added by the algorithm are the vertical edges between rows 1 and 3, and 2 and 4 (red dashed segments in Figure 2.10). These are the closest pairs across U and B which are not connected, so they will be included first. The edges between rows 1 and 4 which connect the points in consecutive columns (dashed blue segments in Figure 2.11) may also be added in the next iteration, depending on how small the value of t is, but we will see that they do not affect the length of the shortest paths between pairs of points in U and B that much.

2.7.4 Sufficiency of small edges for close pairs

Now we claim that the edges we found until now are the only local, i.e. small, edges between these points, and the next edge that is going to be added by the greedy algorithm, would be a large one which intersects many of the zig-zag edges in M. The greedy algorithm may

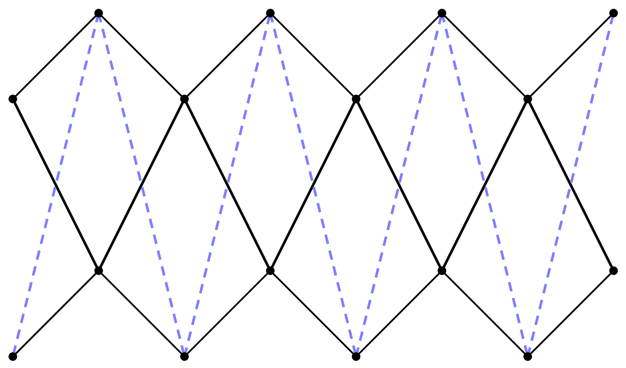


Figure 2.11: The big dashed zig-zag might be included in the graph or might not.

stop here and do not add any edges, but we will prove later that this is not possible. We are not going to address this issue in this section.

Intuitively, one can use edges in U and B, and only one edge in M to build a path from any point in U to any point in B (see Figure 2.12). Again, intuitively, zig-zags are defined in a way that the length of this path is more than $t \cdot |x_u - x_b|$ by a small constant. But when u and b are not far away $|x_u - x_b|$ is much less than $d(u, b)$ and hence the length of the path is no more than $t \cdot d(u, b)$. On the other hand, when u and b are far away, $|x_u - x_b|$ is closer than any constant to $d(u, b)$ (because here $|y_u - y_b|$ is bounded), hence the length of the path becomes more than $t \cdot d(u, b)$ and a long edge appears.

In order to prove this formally, as stated above, any potential edge must be between U and B. So let $u \in U$ and $b \in B$ be two arbitrary points in the top and the bottom zig-zags, respectively. Also assume that u is the i-th point in U ($i = 0, 1, \dots$), and b is the j-th point in B ($j = 0, 1, \dots$), counting from left (Figure 2.12).

We assume that no edges other than the ones we stated above have been added so far, and we compute the length of a path we propose between u and b that uses these edges and we show that it is less than $t \cdot d(u, b)$ if $d(u, b)$ is not very large. In this way, we prove that the

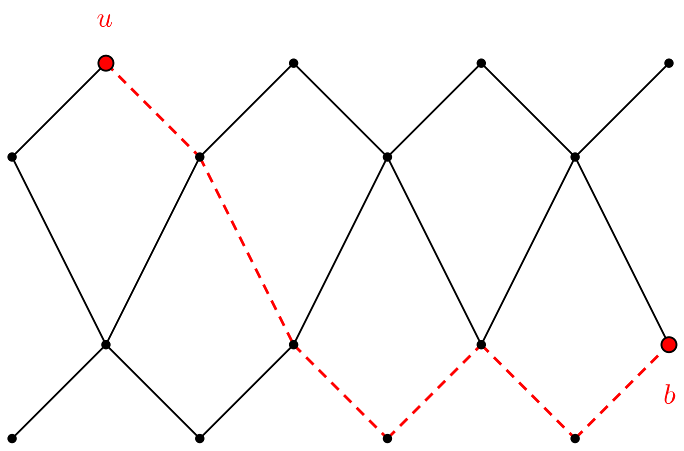


Figure 2.12: $P(u, b)$, which uses some of the edges in U and B and only one edge in M. Here $i = 1$ and $j = 7$.

next edge which is going to be added would be a large one.

Without loss of generality, assume that $i \leq j$. Consider a path that uses zig-zag edges of U and B and only one of the edges in M to reach from u to b. Denote this path by $P(u, b)$. Such a path is drawn by a red dashed line for two sample points in Figure 2.12. Clearly, we do not use any edge twice and we only use zig-zag edges in U, B, or M.

We will show that $|P(u, b)|$, the length of the red path, is not more than $t \cdot d(u, b)$ when $d(u, b)$ is not very large. By the definition, $P(u, b)$ uses $j - i - 1$ edges of U and B, and one edge in M, so

$$|P(u, b)| = (j - i - 1)l + l' \tag{2.7}$$

where l is the edge length in U (and B), and l' is the edge length in M. On the other side, the distance along the x-axis between u and b is $(i - j)\Delta x$, where Δx is defined in Definition 2.22. The distance along the y-axis between u and b is at least the height of the zig-zag M, which is by the definition $s(M)\Delta x$. This distance can be strictly more than $s(M)\Delta x$ when

u is in the first row or b is in the last row. So,

$$d(u,b) \geq \sqrt{(j-i)^2(\Delta x)^2 + s(M)^2(\Delta x)^2} = \sqrt{(j-i)^2 + (t+\delta)^2 - 1}\Delta x \tag{2.8}$$

In order to show $|P(u,b)| \leq t \cdot d(u,b)$, we use Equation 2.7 and Equation 2.8 to show $|P(u,b)|^2 - t^2 \cdot d(u,b)^2$ is non-positive,

$$\begin{aligned}|P(u,b)|^2 - t^2 \cdot d(u,b)^2 &\leq [(j-i-1)l + l')]^2 - \left[(j-i)^2 + (t+\delta)^2 - 1\right](t\Delta x)^2 \\ &= [(j-i) + (l'/l - 1))]^2 l^2 - \left[(j-i)^2 + (t+\delta)^2 - 1\right] l^2 \\ &= \left[2(j-i)(l'/l-1) + (l'/l-1)^2 - (t+\delta)^2 + 1\right] l^2\end{aligned}$$

We used $t\Delta x = l$ in the first equality. Now by putting $l'/l = \frac{t+\delta}{t}$, when $j - i \leq t(t^2-1)/(2\delta)$,

$$\begin{aligned}|P(u,b)|^2 - t^2 \cdot d(u,b)^2 &\leq \left[2(j-i)\frac{\delta}{t} + (\frac{\delta}{t})^2 - (t+\delta)^2 + 1\right] l^2 \\ &\leq \left[(t^2-1) + \delta^2 - (t+\delta)^2 + 1\right] l^2 \leq 0\end{aligned}$$

So no edge is required between u and b and if there is any edge between them, it must be the case that $j - i > t(t^2-1)/(2\delta)$. On the other side, the edge (u,b), if exists, will intersect at least $j-i-2$ of the zig-zag edges which separate u and b. So one can choose δ to be sufficiently small to increase the number of intersections.

2.7.5 Existence of a large edge

Now we address the issue we mentioned earlier, that the greedy algorithm may stop after adding the small edges we discussed in section 3.3 and never add any large edges. We need to prove the existence of such a large edge to complete the proof.

Again, let u be the i-th point in U and b be the j-th point in B, counting from left. We will show that when $j - i$ is large enough an edge is required between u and b. None of the edges that we mentioned so far connects two points whose x-distance is more than Δx. So the shortest path between u and b, denoted by $P^*(u, b)$, needs at least $j - i$ edges to reach from u to b. At least one of these edges should be across U and B, hence having a length at least l'. The other edges have lengths of at least l, as it is the smallest edge in the graph. Thus

$$|P^*(u,b)| \geq (j-i-1)l + l' \tag{2.9}$$

Again, the x-distance of u and b is $(i - j)\Delta x$, and the y-distance of them is at most the height of the whole figure, which is the sum of the height of the three zig-zags, $(s(U) + s(M) + s(B))\Delta x$. So,

$$\begin{aligned} d(u,b) &\leq \sqrt{(j-i)^2(\Delta x)^2 + (s(U)+s(M)+s(B))^2(\Delta x)^2} \\ &\leq \sqrt{(j-i)^2 + (3s(M))^2}\Delta x = \sqrt{(j-i)^2 + 9(t+\delta)^2 - 9}\Delta x \end{aligned} \tag{2.10}$$

The second inequality follows from the fact that $s(M)$ is the maximum among $s(U)$, $s(M)$, and $s(B)$. Similarly, we use Equation 2.9 and Equation 2.10 to show that $|P(u, b)|^2 - t^2 \cdot d(u, b)^2$ is positive,

$$|P^*(u,b)|^2 - t^2 \cdot d(u,b)^2 \geq \left[2(j-i)\frac{\delta}{t} + (\frac{\delta}{t})^2 - 9((t+\delta)^2 - 1)\right] l^2$$

When $j - i \geq 9t((t + \delta)^2 - 1)/(2\delta)$,

$$|P^*(u,b)|^2 - t^2 \cdot d(u,b)^2 \geq \left[9((t+\delta)^2 - 1) + (\frac{\delta}{t})^2 - 9((t+\delta)^2 - 1)\right] l^2 > 0$$

Hence the result.

Theorem 2.24. *For some values of t, there is no constant bound (depending only on t) on the number of crossings between an edge of a greedy t-spanner and other smaller edges.*

Proof. This follows from the existence of the example above. □

2.8 Conclusions

We have shown that greedy t-spanners in the plane have linearly many crossings, and that the intersection graphs of their edges have bounded degeneracy but can have unbounded (and even linear) degree. As a consequence, we proved that these graphs have small separators.

Given these results, it is natural to ask whether higher-dimensional Euclidean greedy t-spanners also have small separators. This cannot be achieved through bounds on crossings, because in dimensions greater than two, graphs whose vertices are in general position can have no crossings. We leave this question open for future research.

Chapter 3

Distributed Spanners for Unit Ball Graphs

3.1 Background

Given a collection of points V in a metric space with doubling dimension d, the weighted *unit ball graph (UBG)* on V is defined as a weighted graph $G(V, E)$ where two points $u, v \in V$ are connected if and only if their metric distance $\|uv\| \leq 1$. The weight of the edge uv of the UBG is $\|uv\|$ if the edge exists. Unit ball graphs in the Euclidean plane are called unit disk graphs (UDGs) and are frequently used to model ad-hoc wireless communication networks, where every node in the network has an effective communication range R, and two nodes are able to communicate if they are within a distance R of each other.

The special case of spanners where the underlying graph is a unit ball graph is motivated by the application of unit ball graphs in modeling wireless and ad-hoc networks, where the communication of the nodes are limited by their physical distances. The problem of finding sparse lightweight spanners for unit ball graphs in this settings translates into efficient topology control algorithms. Thus the necessity of a connected and energy-efficient topology for high-level routing protocols led researchers to develop many spanning algorithms for ad-hoc networks and in particular, UDGs. But the decentralized nature of ad-hoc networks demands that these algorithms be local instead of centralized. In these applications, it is important that the resulting topology is connected, has a low weight, and has a bounded degree, implying also that the number of edges is linear in the number of vertices.

Several known *proximity graphs* have been studied for this purpose, including the relative neighborhood graph (RNG), Gabriel graph (GG), Delaunay graph (DG), and Yao graph (YG). It is well-known that these proximity graphs are sparse and they can be calculated locally, using only the information from a node's neighborhood. But further analysis shows that they have poor bounds on at least one of the important criteria: maximum vertex degree, total weight, and stretch-factor [75].

Researchers have modified these constructions to fulfill the requirements. Li, Wan, and Wang [75] introduced a modified version of the Yao graph to resolve the issue of unbounded in-degree while preserving a small stretch-factor, but they left as an open question whether there exists a construction whose total weight is also bounded by a constant factor of the weight of the minimum spanning tree. The localized Delaunay triangulation (LDT) [74] and local minimum spanning tree (LMST) [73] were two other efforts in this way which failed to bound the total weight of the spanner. Hence bounding the weight became the main challenge in designing efficient spanners. The commonly used measure for the weight of the spanners is *lightness*, which is defined as the weight of the spanner divided by the weight of the minimum spanning tree.

In the distributed setting in particular, Gao, Guibas, Hershberger, Zhang, and Zhu [54] introduced restricted Delaunay graph (RDG), a planar distributed spanner construction for unit disk graphs in the two dimensional Euclidean plane that possessed a constant stretch-factor, leaving the weight of the spanner unstudied. Later Kanj, Perković, and Xia [65] presented the first local spanner construction for unit disk graphs in the two dimensional Euclidean plane, which also was planar and had constant bounds on its stretch-factor, maximum degree, and lightness. Their construction was also based on the Delaunay triangulation of the point set and required information from k-th hop neighbors of every node, for some constant k that depended on the input parameters.

In 2006, Damian, Pandit, and Pemmaraju [28] designed a distributed construction for $(1+\varepsilon)$-spanners of the UBGs lying in *d-dimensional Euclidean space*. Their algorithm ran in $\mathcal{O}(\log^* n)$ rounds of communication and produced a $(1+\varepsilon)$-spanner with constant bounds on its maximum degree and lightness. They used the so-called *leapfrog property* to prove the constant bound on the lightness of the spanner, which does not hold for the spaces of bounded doubling dimension in general. Instead, they showed in another work [27] that the weight of their spanner in the spaces of bounded doubling dimension is bounded by a factor $\mathcal{O}(\log \Delta)$ of the weight of the minimum spanning tree, where Δ is the ratio of the length of the longest edge in the unit ball graph divided by the length of its shortest edge. Besides these, their algorithm requires the knowledge of $\mathcal{O}(\frac{1}{\alpha-1})$-hop neighborhood of the nodes, which is costly in the CONGEST model of distributed computing, the more accepted and practical model than the LOCAL model of computation.

In the 3D Euclidean space, Jenkins, Kanj, Xia, and Zhang [63] designed the first localized bounded-degree $(1+\varepsilon)$-spanner for unit ball graphs. They also presented a lightweight construction which possessed constant bounds on its stretch-factor and maximum degree. These algorithms again required k-th hop neighborhood information for every node, for a constant k that depended on the input parameters. Although these constructions were local, i.e. they ran in constant rounds of communication, they relied heavily on Euclidean transformations which made them inapplicable for other metric spaces.

Finally, Elkin, Filtser, and Neiman [37] studied the topic of lightweight spanners for general graphs and doubling graphs in the CONGEST model of distribution. For general graphs, they presented $(2k-1)\cdot(1+\varepsilon)$-spanners with lightness $\mathcal{O}(k\cdot n^{1/k})$ in $\tilde{\mathcal{O}}(n^{0.5+1/(4k+2)}+D)$ rounds, where n is the number of vertices and D is the hop-diameter of the graph. For doubling graphs, they presented a $(1+\varepsilon)$-spanner with lightness $\varepsilon^{-\mathcal{O}(1)}\log n$ in $(\sqrt{n}+D)\cdot n^{o(1)}$ rounds of communication. Although these constructions are more general than the constructions of [27] and they perform in a more restricted model (CONGEST), they do not imply a superior result in the specific case of unit ball graphs in doubling metrics.

Apart from being a generalization of the Euclidean space, the importance of the spaces of bounded doubling dimension comes from the fact that a small perturbation in the pairwise distances does not affect the doubling dimension of the point set by much, while it can change their Euclidean dimension significantly, or the resulting distances might not even be embeddable in Euclidean metrics at all [19]. This makes these metrics of bounded doubling dimension to be more applicable in real-world scenarios. On the other hand, geometric arguments are considered as a strong tool for proofs of sparsity and lightness bounds in Euclidean spaces, but in doubling spaces the only available tool besides metric properties, is the packing argument which is directly followed from the definition of the doubling dimension. Therefore, the sparsity and lightness results are more difficult to achieve in the spaces of bounded doubling dimension.

Since the work of Damian, Pandit, and Pemmaraju [27] in 2006, it has remained open whether UBGs in the spaces of bounded doubling dimension possess lightweight bounded-degree $(1+\varepsilon)$-spanners and whether they can be found efficiently in a distributed model of computation. On the other hand, the construction of [27] requires complete information about the nodes in $\mathcal{O}(\frac{1}{\alpha-1})$ hops away, for some constant α. Acquiring this information is costly in the CONGEST model of computation, which is a more accepted model in distributed computing. Therefore, another open question arising from this line of work is to study the round complexity of the aforementioned problem in the CONGEST model. In this chapter, we resolve both of these long-standing open questions by presenting centralized and distributed algorithms, both in the LOCAL, and the CONGEST model, for the purpose of finding such spanners.

3.2 Overview

We have two main contributions in this chapter. First, we resolve the proposed open question that has remained open for more than a decade, and we prove the existence of light-weight bounded-degree $(1+\varepsilon)$-spanners of unit ball graphs in the spaces of bounded doubling dimension. Our construction has constant bounds on its maximum degree and its lightness, and it can be built in $\mathcal{O}(\log^* n)$ rounds of communication in the LOCAL model of computation, where n is the number of vertices.

Second, we propose the first lightweight spanner construction for unit ball graphs in the CONGEST model of computation. Even if we restrict our scope to the two dimensional Euclidean plane, where we see most of the applications of unit disk graphs, prior to this work there was no known CONGEST algorithm for finding light spanners of unit disk graphs. We achieve this construction by making adjustments on our construction for the LOCAL model to make it work in the CONGEST model in the same asymptotic number of rounds. The bounds on the lightness and maximum degree of our spanner remain the same in this model.

Besides these main results, we modify these constructions for the two dimensional Euclidean plane in order to have a linear number of edge intersections in total, implying a constant average number of edge intersections per node. This is motivated by the observation that a higher intersection per edge causes a higher chance of interference between the corresponding endpoints. To the best of our knowledge, this is the first distributed low-stretch low-intersection spanner construction for unit disk graphs.

A more detailed version of our results can be found in the following theorems. First, we introduce a centralized algorithm Centralized-Spanner that,

Theorem 3.8. Given a weighted unit ball graph G in a metric of bounded doubling dimension and a constant $\varepsilon > 0$, the spanner returned by Centralized-Spanner(G,ε) is a $(1+\varepsilon)$-spanner of G and has constant bounds on its lightness and maximum degree. These constant bounds only depend on ε and the doubling dimension.

We use this centralized construction to propose the distributed construction Distributed-Spanner in the LOCAL model of computation,

Theorem 3.16. Given a weighted unit ball graph G with n vertices in a metric of bounded doubling dimension and a constant $\varepsilon > 0$, the algorithm Distributed-Spanner(G,ε) runs in $\mathcal{O}(\log^* n)$ rounds of communication in the LOCAL model of computation, and returns a $(1+\varepsilon)$-spanner of G that has constant bounds on its lightness and maximum degree. These constant bounds only depend on ε and the doubling dimension.

Next, we study the problem in the CONGEST model of computation. Our distributed construction Distributed-Spanner requires complete information about 2-hop neighborhood of a selected set of vertices, which is not easy to acquire in the CONGEST model. The same issues exists in the distributed algorithm of [27], where they aggregate information about the nodes that are $\mathcal{O}(\frac{1}{\alpha-1})$ hops away, for some constant α. A simple approach for aggregating 2-hop neighborhoods would require $\mathcal{O}(d)$ rounds of communication in the CONGEST model, which can be as large as $\Omega(n)$ if the input graph is dense. In our next theorem, we break this barrier by making some adjustments for our algorithm to work in the CONGEST model of computation. Despite adding to the complexity of the algorithm itself, we prove that the round complexity of our new algorithm, CONGEST-Spanner, would still be bounded by $\mathcal{O}(\log^* n)$.

Theorem 3.21. Given a weighted unit ball graph G with n vertices in a metric of bounded doubling dimension and a constant $\varepsilon > 0$, the algorithm CONGEST-Spanner(G,ε) runs in $\mathcal{O}(\log^* n)$ rounds of communication in the CONGEST model of computation, and returns a $(1+\varepsilon)$-spanner of G that has constant bounds on its lightness and maximum degree. These constant bounds only depend on ε and the doubling dimension.

Furthermore, we study the problem in the case of the two dimensional Euclidean plane, where the greedy spanner on a complete weighted graph is known to have constant upper bounds on its lightness [51], maximum degree, and average number of edge intersections per node [45]. We observe that a simple change on the this algorithm can extend these results for unit disk graphs as well. We call this modified algorithm Centralized-Euclidean-Spanner and we show that

Theorem 3.22. Given a weighted unit disk graph G in the two dimensional Euclidean plane and a constant $\varepsilon > 0$, the spanner returned by Centralized-Euclidean-Spanner(G,ε) is a $(1+\varepsilon)$-spanner of G and has constant bounds on its lightness, maximum degree, and the average number of edge intersections per node. These constant bounds only depend on ε and the doubling dimension.

We use the aforementioned construction to propose Distributed-Euclidean-Spanner, a specific distributed low-intersection construction for the case of the two dimensional Euclidean plane that preserves the previously mentioned properties and adds the low-intersection property.

Theorem 3.32. Given a weighted unit disk graph G with n vertices in the two dimensional Euclidean plane and a constant $\varepsilon > 0$, the algorithm Distributed-Euclidean(G,ε) runs in $\mathcal{O}(\log^* n)$ rounds of communication and returns a bounded-degree $(1+\varepsilon)$-spanner of G that has constant bounds on its lightness, maximum degree, and the average number of edge intersections per node. These constant bounds only depend on ε and the doubling dimension.

Besides these, we also prove that the last construction possesses sublinear separators and a separator hierarchy in the two dimensional Euclidean plane. We generalize this result to work for higher dimensions of Euclidean spaces. Finally, in section 3.8, we provide experimental results on random point sets in the two dimensional Euclidean plane that confirm the efficiency of our distributed construction.

3.3 Preliminaries

3.3.1 Doubling metrics

We start by recalling the definition of the doubling dimension of a metric space,

Definition 3.1 (doubling dimension)**.** The doubling dimension of a metric space is the smallest d such that for any $R > 0$, any ball of radius R can be covered by at most 2^d balls of radius $R/2$.

We say a metric space has *bounded doubling dimension* if its doubling dimension is upper bounded by a constant. Besides the triangle inequality, which is intrinsic to metric spaces, the *packing lemma* is an essential tool for the metrics of bounded doubling dimension. This lemma states that it is impossible to pack more than a certain number of points in a ball of radius $R > 0$ without making a pair of points' distance less than some $r > 0$.

Lemma 3.2 (Packing Property)**.** *In a metric space of bounded doubling dimension d, let X be a set of points with minimum distance r, contained in a ball of radius R. Then $|X| \leq \left(\frac{4R}{r}\right)^d$.*

Proof. This is a well-known fact, see e.g. [85]. □

3.3.2 Spanners for complete graphs

For a weighted graph G in a metric space, where every edge weight is equal to the metric distance of its endpoints, a t-spanner is defined in the following way,

Definition 3.3 (t-spanner)**.** A t-spanner of a weighted graph G is a subgraph S of G that for every pair of vertices x, y in G,

$$\text{dist}_S(x, y) \leq t \cdot \text{dist}_G(x, y)$$

where $\text{dist}_A(x, y)$ is the length of a shortest path between x and y in A. The lightness of S is defined as $\mathbf{w}(S)/\mathbf{w}(MST)$ where $\mathbf{w}$ is the weight function and MST is the minimum spanning tree in G.

In other words, a t-spanner approximates the pairwise distances within a factor of t. Spanners were studied for complete weighted graphs first, and several constructions were proposed to optimize them with respect to the number of edges and total weight. Among these constructions, *greedy spanners* [3] are known to out-perform the others.

We recall the greedy spanner algorithm (Algorithm 2): this short procedure adds edges one at a time to the spanner it constructs, in ascending order by length. For each pair of vertices, in this order, it checks whether the pair already satisfies the distance inequality using the edges already added. If not, it adds a new edge connecting the pair. Therefore, by construction, each pair of vertices satisfies the inequality, either through previous edges or (if not) through the newly added edge. The resulting graph is therefore a t-spanner.

In Chapter 2, we showed that the number of edge crossings of the greedy spanner in the two dimensional Euclidean plane is linear in the number of vertices. Moreover, we proved that the crossing graph of the greedy spanner has bounded degeneracy, implying the existence of sub-linear separators for these graphs [44]. This, together with the well-known fact that

Algorithm 2 Recalling the naive greedy spanner algorithm.

```
1: procedure NAIVE-GREEDY($V$)
2:     Let $S$ be a graph with vertices $V$ and edges $E = \{\}$
3:     for each pair $(P, Q) \in V^2$ in increasing order of $\|PQ\|$ do
4:         if $\text{dist}_S(P, Q) > t \cdot \text{dist}(P, Q)$ then
5:             Add edge $PQ$ to $E$
       return S
```

greedy spanners have bounded-degree in the two dimensional Euclidean plane, makes greedy spanners more practical in this particular metric space.

Although the degree of the greedy spanner is bounded in the two dimensional Euclidean plane, it is known that there exist n-point metric spaces with doubling dimension 1 where the greedy spanner has maximum degree $n - 1$ [51]. Gudmundsson, Levcopoulos, and Narasimhan [62] devised a faster algorithm that was later proven to have bounded degree as well as constant lightness and linear number of edges [51]. We call this algorithm APPROXIMATE-GREEDY in this chapter, and we make use of it in our algorithms for the metrics of bounded doubling dimension, while we take advantage of the extra low-intersection property of NAIVE-GREEDY in the two dimensional Euclidean plane.

3.3.3 Unit ball graphs

We formally define a unit ball graph on a set of points V in the following way,

Definition 3.4 (unit ball graph)**.** Given a set of points V in a metric space, the unit ball graph G on V contains V as its vertex set and every two vertices $x, y \in V$ are connected if and only if $\|xy\| \leq 1$. The weight of an edge (x, y) is equal to $\|xy\|$ if the edge exists.

Unit ball graphs are an important subclass of the graphs called *growth-bounded graphs*, which only limit the number of independent nodes in every neighborhood, a property that holds for UBGs due to the packing property.

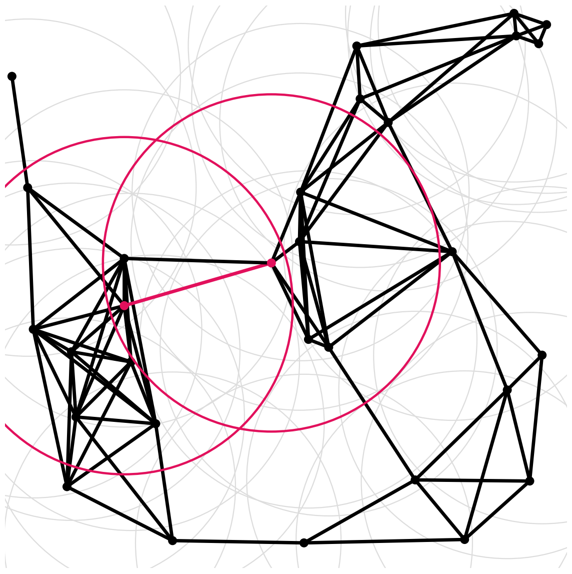


Figure 3.1: The unit disk graph on the same point set introduced earlier. The red disks intersect, therefore there is an edge between their centers.

Kuhn, Moscibroda, and Wattenhofer [70] presented a $\mathcal{O}(\log^* n)$-round distributed algorithm for finding a maximal independent set (MIS) of a unit ball graph graph in a space with bounded doubling dimension. This result was later generalized by Schneider and Wattenhofer [84] for growth-bounded graphs. Throughout the chapter we refer to their algorithm by Maximal-Independent. It turns out that Maximal-Independent will be a key ingredient of our distributed algorithms, as well as their bottleneck in terms of the number of rounds. This means that if a maximal independent set is known beforehand, our algorithms can be executed fully locally, in constant number of rounds.

In section 3.4 we prove the existence of $(1 + \varepsilon)$-spanners with constant bounds on the maximum degree and the lightness by introducing an algorithm that finds such spanners in a centralized manner. In section 3.5 we propose a distributed construction that delivers the same features through a $\mathcal{O}(\log^* n)$-round algorithm. Finally, in section 3.7 we consider the special case of two dimensional Euclidean plane and we design centralized and distributed algorithms to construct a spanner that has the extra low-intersection property, making it more suitable for practical purposes.

3.4 Centralized Construction

In this section we propose a centralized construction for a light-weight bounded-degree $(1+\varepsilon)$-spanner for unit ball graphs in a metric of bounded doubling dimension. Later in section 3.5 we use this centralized construction to design a distributed algorithm that delivers the same features.

It is worth mentioning that the greedy spanner would be a $(1+\varepsilon)$-spanner of the UBG if the algorithm stops after visiting the pairs of distance at most 1, and it even has a lightness bounded by a constant, but as we mentioned earlier, there are metrics with doubling dimension 1 in which its degree may be unbounded.

To construct a lightweight bounded-degree $(1+\varepsilon)$-spanner of the unit ball graph, we start with the spanner of [62], called Approximate-Greedy, which is returns a spanner of the complete graph. It is proven in [79] that Approximate-Greedy has the desired properties, i.e. bounded-degree and lightness, for complete weighted graphs in Euclidean metrics, but as stated in [51], the proof only relies on the triangle inequality and packing argument which both work for doubling metrics as well. Therefore, we may safely assume that Approximate-Greedy finds a light-weight bounded-degree $(1+\varepsilon)$-spanner of the complete weighted graph defined on the point set. The main issue is that the edges of length more than 1 are not allowed in a spanner of the unit ball graph on the same point set. Therefore, a replacement procedure is needed to substitute these edge with edges of length at most 1. Peleg and Roditty [80] introduced a refinement process which removes the edges of length larger than 1 from the spanner and replaces them with three smaller edges to make the output a subgraph of the UBG. The main issue with their approach is that it can lead to vertices having unbounded degrees in the spanner, therefore missing an important feature. Here, we introduce our own refinement process that not only replaces edges of larger than 1 with smaller edges and makes the spanner a subgraph of the unit ball graph, but also guarantees a constant bounded on the degrees of the resulting spanner.

3.4.1 The algorithm

In the first step of the algorithm (Algorithm 3) we choose $\varepsilon' = \varepsilon/36$, a smaller stretch parameter than ε, to cover the errors that future steps might inflict to the spanner. Then we call the procedure Approximate-Greedy on the set of vertices V to calculate a light-weight bounded-degree $(1+\varepsilon')$-spanner S of the complete weighted graph on V. This spanner might contain edges of length larger than 1, which we will replace by some edges of length at most 1 in the future steps.

Since an edge of length larger than $1+\varepsilon'$ in S cannot participate in the shortest path between any two adjacent vertices in G, we simply remove and discard them from the spanner. Then for every remaining edge $e = (u, v)$ of length in the range $(1, 1 + \varepsilon']$ we find an edge (x, y) of the original graph G so that $\|ux\| \leq \varepsilon'$ and $\|vy\| \leq \varepsilon'$. We then replace such an edge e by the edge (x, y). We call the pair (x, y) the *replacement edge* or the *replacement pair* for the edge e. Since this procedure can end up assigning too many replacement edges to a single vertex (x or y in this case) and hence increasing its degree significantly, we perform a simple check before adding a replacement edge; we store the set R of previously added replacement pairs in the memory and if a *weak* replacement pair $(x', y') \in R$ exists, then we prefer it over a newly found replacement pair $(x, y) \notin R$. By weak replacement pair we mean a pair $(x', y') \in R$ that $\|ux'\| \leq 2\varepsilon'$ and $\|vy'\| \leq 2\varepsilon'$, which is weaker than the definition of the replacement pair. As we later see this weaker notion of replacement pair will help us to bound the degree of the vertices.

After removing the edges of length larger than 1 and replacing the ones in the range $(1, 1+\varepsilon']$, we return the spanner to the output.

Algorithm 3 A centralized spanner construction.

Input. A unit ball graph $G(V, E)$ in a metric with doubling dimension d.

Output. A light-weight bounded-degree $(1 + \varepsilon)$-spanner of G.

```
1: procedure CENTRALIZED-SPANNER($G$, $\varepsilon$)
2:     $\varepsilon' \leftarrow \varepsilon/36$
3:     $S \leftarrow$ APPROXIMATE-GREEDY($V$, $\varepsilon'$)
4:     R $\leftarrow \varnothing$
5:     for $e = (u, v)$ in $S$ do
6:         if $|e| > 1$ then
7:             Remove $e$ from $S$
8:         if $|e| \in (1, 1 + \varepsilon']$ then
9:             if $\exists (x, y) \in E$ that $\|ux\| \leq \varepsilon'$ and $\|vy\| \leq \varepsilon'$ then
10:                if $\nexists (x', y') \in R$ that $\|ux'\| \leq 2\varepsilon'$ and $\|vy'\| \leq 2\varepsilon'$ then
11:                    $S \leftarrow S \cup \{(x, y)\}$
12:                    $R \leftarrow R \cup \{(x, y)\}$
13:    return S
```

3.4.2 The analysis

Now we prove that the output S of the algorithm is a light-weight bounded-degree $(1 + \varepsilon)$-spanner of the unit ball graph G. Clearly, after the refinement is done the spanner S is a subgraph of G, so we need to analyze the lightness, the stretch factor, and the maximum degree of the spanner.

First we prove that the stretch-factor of the spanner is indeed bounded by $1 + \varepsilon$.

Lemma 3.5. *The spanner returned by* CENTRALIZED-SPANNER *has a stretch factor of* $1+\varepsilon$.

Proof. We recall that the output of APPROXIMATE-GREEDY is a light-weight bounded-degree $(1 + \varepsilon')$-spanner of the complete weighted graph on the point set. So an edge e of length $|e| > 1 + \varepsilon'$ cannot be used to approximate any edges in the UBG, i.e. if $(x, y) \in E$ then e cannot belong to the shortest path between x and y in S; otherwise the length of the path would exceed $1 + \varepsilon'$ which cannot happen. So we may safely remove these edges in the first step of the refinement procedure without replacing them.

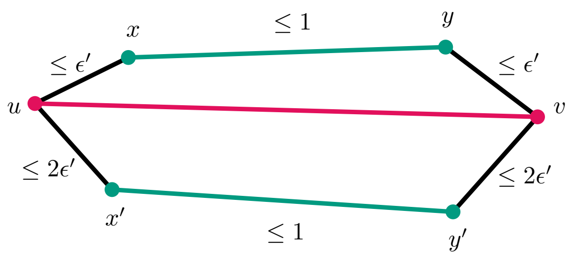


Figure 3.2: An edge (x, y) of the UBG that uses a longer than unit length edge (u, v) of the spanner on its shortest path, which is then replaced by (x', y') during the replacement procedure.

Also, any edge of length in the range $(1, 1+\varepsilon']$ that is not used in a shortest path between any two endpoints of an edge of UBG can be removed as well, because removing them does not change the stretch-factor of the spanner. Now consider an edge $(x, y) \in E$ of the UBG that uses a spanner edge $e = (u, v) \in S$ that $|e| \in (1, 1+\varepsilon']$ on its shortest path. We want to prove that after the replacement of e, the shortest path between x and y remains within $1+\varepsilon$ factor of their distance. Clearly, we have $\|ux\| \leq \varepsilon'$ and $\|vy\| \leq \varepsilon'$; otherwise the length of the path $xuvy$ would be more than $1+\varepsilon'$, contradicting the fact that it is approximating an edge of length at most 1. This shows that (x, y) would be a valid replacement edge for e. So we can safely assume that the algorithm finds a (possible weak) replacement edge $(x', y') \in E$ for e (Figure 3.2). This replacement edge might be a normal replacement edge or a weak replacement edge. Either way, we have $\|ux'\| \leq 2\varepsilon'$ and $\|vy'\| \leq 2\varepsilon'$. By the triangle inequality

$$\|x'x\| \leq \|x'u\| + \|ux\| \leq 2\varepsilon' + \varepsilon' = 3\varepsilon'$$

Similarly, $\|yy'\| \leq 3\varepsilon'$. Therefore

$$\|x'y'\| \leq \|x'x\| + \|xy\| + \|yy'\| \leq \|xy\| + 6\varepsilon' \tag{3.1}$$

Denote the shortest spanner path between x and x' by $P_{xx'}$ and similarly define $P_{yy'}$. Consider the spanner path $P = P_{xx'}x'y'P_{y'y}$ that connects x and y. Using Equation 3.1 the length of the path P is

$$|P| = |P_{xx'}| + \|x'y'\| + |P_{y'y}| \leq \|xy\| + 6\varepsilon' + |P_{xx'}| + |P_{y'y}| \tag{3.2}$$

The changes that we make in the refinement process do not affect the length of short paths like $P_{xx'}$ and $P_{y'y}$. So we have

$$|P_{xx'}| \leq (1+\varepsilon')\|xx'\| \leq 3\varepsilon'(1+\varepsilon')$$

Similarly, $|P_{yy'}| \leq 3\varepsilon'(1+\varepsilon')$. Putting these into Equation 3.2 and using $\varepsilon = 36\varepsilon'$,

$$|P| \leq \|xy\| + 6\varepsilon' + 6(1+\varepsilon')\varepsilon' \leq \|xy\| + \frac{\varepsilon}{1+\varepsilon'} \tag{3.3}$$

But since e was previously approximating the edge (x, y), we know that $(1+\varepsilon')\|xy\| \geq |e| > 1$ or equivalently $\|xy\| > 1/(1+\varepsilon')$. Substituting this into Equation 3.3,

$$|P| \leq \|xy\| + \varepsilon\|xy\| = (1+\varepsilon)\|xy\|$$

So S is a $(1+\varepsilon)$-spanner of G. $\square$

Now we analyze the weight of the spanner, proving its constant lightness.

Lemma 3.6. *The spanner returned by* Centralized-Spanner *has a weight of* $\mathcal{O}(1)\mathbf{w}(MST)$.

Proof. Again, we use the fact that the output of Approximate-Greedy has weight $\mathcal{O}(1)\mathbf{w}(MST(G))$. During the refinement process, every edge is replaced by an edge of smaller length, so the whole weight of the graph does not increase during the refinement process. Therefore in the end $\mathbf{w}(S) = \mathcal{O}(1)\mathbf{w}(MST(G))$. $\square$

In the final step, we bound the maximum degree of the spanner.

Lemma 3.7. *The spanner returned by* Centralized-Spanner *has bounded degree.*

Proof. It is clear from the algorithm that immediately after processing an edge $e = (u, v)$, the degree of u and v does not increase; it may decrease due to the removal of the edge which is fine. But if a replacement edge (x, y) is added after the removal of e then the degree of x and y is increased by at most 1. We need to make sure this increment is bounded for every vertex.

Let x be an arbitrary vertex of G and let (x, y) and (x, z) be two replacement edges that have been added to x in this order as a result of the refinement process. We claim that $\|yz\| > \varepsilon'$ holds. Assume, on the contrary, that $\|yz\| \leq \varepsilon'$, and also assume that (x, z) has been added in order to replace an edge (u, v) of the spanner. Then by the triangle inequality

$$\|vy\| \leq \|vz\| + \|zy\| \leq 2\varepsilon'$$

Also $\|ux\| \leq \varepsilon' < 2\varepsilon'$ because (x, z) is added to replace (u, v). But the last two inequalities contradict the fact that (x, y) cannot be a weak replacement for (u, v).

Now that we have proved $\|yz\| > \varepsilon'$ we can use the packing property of the bounded doubling dimension to bound the number of such replacement edges around x. All the other endpoints of such replacement edges are included in ball of radius 1 around x, and the distance between every two such points is at least ε'. Thus by the packing property there can be at most $(\frac{4}{\varepsilon'})^d = \varepsilon^{-\mathcal{O}(d)}$ many replacement edges incident to x. □

Putting these together, we can prove Theorem 3.8.

Theorem 3.8 (Centralized Spanner)**.** *Given a weighted unit ball graph G in a metric of bounded doubling dimension and a constant $\varepsilon > 0$, the spanner returned by* Centralized-

SPANNER*(G,ε) is a* $(1+\varepsilon)$*-spanner of* G *and has constant bounds on its lightness and maximum degree. These constant bounds only depend on* ε *and the doubling dimension.*

Proof. Follows directly from Lemma 3.5, Lemma 3.6, and Lemma 3.7. □

3.5 Distributed Construction

In this section we propose our distributed construction for finding a $(1+\varepsilon)$-spanner of a unit ball graph using only 2-hop neighborhood information. The spanner returned by our algorithm has constant bounds on its maximum degree and its lightness. This is the first light-weight distributed construction for unit ball graphs in doubling metrics, to the best of our knowledge.

In our distributed construction, we run our centralized algorithm on the 2-hop neighborhoods of a an independent set of the unit ball graph, and we prove that putting these local spanners together will achieve a spanner that possesses the desired properties.

3.5.1 The algorithm

For the distributed construction we propose Algorithm 4. There is a preprocessing step of finding a maximal independent set I of G, which can be done using the distributed algorithm of [70] in $\mathcal{O}(\log^* n)$ rounds. We refer to this algorithm by MAXIMAL-INDEPENDENT. Then the LOCAL-GREEDY subroutine is run on every vertex $w \in I$ to find a $(1+\varepsilon)$-spanner S_w of the 2-hop neighborhood of w, denoted by $\mathcal{N}^2(w)$. At the final step, every $w \in I$ sends its local spanner edges to the corresponding endpoints of every edge. Symmetrically, every vertex listens for the edges sent by the vertices in I and once a message is received, it stores the edges in its local storage. In other words, the final spanner is the union of all these local spanners. We use the centralized algorithm of section 3.4 for every local neighborhood $\mathcal{N}^2(w)$ to guarantee the bounds that we need.

Algorithm 4 The localized greedy algorithm.

Input. A unit ball graph $G(V, E)$ in a metric with doubling dimension d and an $\varepsilon > 0$.
Output. A light-weight bounded-degree $(1+\varepsilon)$-spanner of G.

```
 1: procedure DISTRIBUTED-SPANNER($G$, $\varepsilon$)
 2:     Find a maximal independent set $I$ of $G$ using [70]
 3:     Run LOCAL-GREEDY on the vertices of $G$
 4: function LOCAL-GREEDY(vertex $w$)
 5:     Retrieve $\mathcal{N}^2(w)$, the 2-hop neighborhood information of $w$
 6:     if $w \in I$ then
 7:         $\mathcal{S}_w \leftarrow$ CENTRALIZED-SPANNER($\mathcal{N}^2(w)$, $\varepsilon$)
 8:         for $e = (u, v)$ in $\mathcal{S}_w$ do
 9:             Send $e$ to $u$ and $v$
10:     Listen to incoming edges and store them
```

Similar to the aforementioned greedy algorithm (Algorithm 2), our algorithm seems very simple in the first sight. But as we see later in this section, proving its properties, particularly its lightness, is a non-trivial task.

3.5.2 The analysis

Now we show that the spanner introduced in Algorithm 4 possesses the desired properties. First, we show the round complexity of $\mathcal{O}(\log^* n)$.

Lemma 3.9. DISTRIBUTED-SPANNER *can be done in* $\mathcal{O}(\log^* n)$ *rounds of communication.*

Proof. The pre-processing step of finding the maximal independent set takes $\mathcal{O}(\log^* n)$ rounds of communication [70]. Retrieving the 2-hop neighborhood information can be done in $\mathcal{O}(1)$ rounds of communication. Computing the greedy spanner is done locally, and the edges are sent to their endpoints, which again can be done in $\mathcal{O}(1)$ rounds of communication. Overall, the algorithm requires $\mathcal{O}(\log^* n)$ rounds of communication. □

Next we bound the stretch-factor of the spanner.

Lemma 3.10. *The spanner returned by* Distributed-Spanner *has a stretch factor of* $1 + \varepsilon$.

Proof. From section 3.4 we know that $\mathcal{S}_w$ is a light-weight bounded-degree $(1 + \varepsilon)$-spanner of $\mathcal{N}^2(w)$. Let $u, v \in V$ be chosen arbitrarily. We need to make sure there is a path of length at most $(1 + \varepsilon)d_G(u, v)$ between u and v in the output.

First we prove this for the case that $(u, v) \in E$. So let $e = (u, v) \in E$. Then u is either in I or has a neighbor in I, according to choice of I. In any case, the edge e belongs to $\mathcal{N}^2(w)$ for some $w \in I$, which means that there is a path $P \subset \mathcal{S}_w$ of length at most $(1+\varepsilon)|e|$ that connects u and v. The edges of P are all included in the final spanner according to the algorithm, so the output includes this path between u and v, which has a length at most $(1 + \varepsilon)|e|$ and so the distance inequality is satisfied.

If $(u, v) \notin E$, we can take the shortest path $u = p_0, p_1, \cdots, p_k = v$ between them in G and append the corresponding $(1 + \varepsilon)$-approximate paths $P_0, P_1, \cdots, P_{k-1}$ of the edges $p_0p_1, p_1p_2, \cdots, p_{k-1}p_k$, respectively, to get a $(1 + \varepsilon)$-approximate path for $p_0p_1 \cdots p_k$. This implies that the stretch factor of the output is indeed $1 + \varepsilon$. □

Now we bound the maximum degree of the spanner.

Lemma 3.11. *The spanner returned by* Distributed-Spanner *has a bounded degree.*

Proof. First we use the packing lemma to prove that any vertex $v \in V$ appears at most a constant number of times in different neighborhoods, $\mathcal{N}^2(w)$ for $w \in I$. Because $v \in \mathcal{N}^2(w)$ implies that $\|vw\| \leq 2$, any vertex $w \in I$ such that $v \in \mathcal{N}^2(w)$ should be contained in the ball of radius 2 around v. But all such ws are chosen from I, which is an independent set of

G, so the distance between every two such vertex is at least 1. By the packing property, the maximum number of such vertices would be $8^d = \mathcal{O}(1)$.

Now that every vertex appears in at most in 8^d different sets $\mathcal{N}^2(w)$, for $w \in I$, and from section 3.4 we already knew that every vertex has bounded degree in any of $\mathcal{S}_w$s, it immediately follows that every vertex has bounded degree in the final spanner. □

In order to bound the lightness of the output, we assume that $\varepsilon \leq 1$ and we make a few comparisons. First, for any $w \in I$ we compare the weight of $\mathcal{S}_w$ to the weight of the minimum spanning tree on $\mathcal{N}^2(w)$. Then we compare the weight of the minimum spanning tree on $\mathcal{N}^2(w)$ to the weight of the minimum Steiner tree on $\mathcal{N}^3(w)$, where the required vertices are $\mathcal{N}^2(w)$ and 3-hop vertices are optional. Finally, we compare the weight of this minimum Steiner tree to the weight of the induced subgraph of CENTRALIZED-SPANNER(G, ε) on the subset of vertices $\mathcal{N}^3(w)$, which later implies that the overall weight of $\mathcal{S}_w$s is bounded by a constant factor of the weight of the minimum spanning tree on G.

Our first claim is that the weight of $\mathcal{S}_w$ is bounded by a constant factor of the weight of the MST on $\mathcal{N}^2(w)$.

Corollary 3.12. $\mathbf{w}(\mathcal{S}_w) = \mathcal{O}(1)\mathbf{w}(MST(\mathcal{N}^2(w)))$

Proof. Follows from the properties of the centralized algorithm in section 3.4. □

Next we compare $\mathbf{w}(MST(\mathcal{N}^2(w)))$ to the weight of the minimum Steiner tree of $\mathcal{N}^3(w)$ on the required vertices $\mathcal{N}^2(w)$.

Lemma 3.13. *Define $\mathcal{T}$ to be the optimal Steiner tree on the set of vertices $\mathcal{N}^3(w)$, where only vertices in $\mathcal{N}^2(w)$ are required and the rest of them are optional. Then*

$$\mathbf{w}(MST(\mathcal{N}^2(w))) \leq 2\mathbf{w}(\mathcal{T})$$

Proof. This is a well-known fact that implies a 2-approximation for minimum Steiner tree problem. The idea is if we run a full DFS on the vertices of $\mathcal{T}$ and we write every vertex once we open and once we close it, then we get a cycle whose shortcut on optional edges will form a path on the required vertices. The weight of the cycle is at least $\mathbf{w}(MST(\mathcal{N}^2(w)))$ and at most $2\mathbf{w}(\mathcal{T})$, which proves the result. □

We then compare the weight of $\mathcal{T}$ to the weight of induced subgraph of Centralized-Spanner(G, ε) on the subset of vertices $\mathcal{N}^3(w)$. The main observation here is that when $\varepsilon \leq 1$ the induced subgraph of the centralized spanner on $\mathcal{N}^3(w)$ would be a feasible solution to the minimum Steiner tree problem on $\mathcal{N}^3(w)$, with the required vertices being the vertices in $\mathcal{N}^2(w)$. This will imply that the weight of the induced subgraph is at least equal to the weight of the minimum Steiner tree.

Lemma 3.14. *Let $\mathcal{S}^*$ be the output of* Centralized-Spanner*(G, ε) and let $\mathcal{S}^*_w$ be the induced subgraph of $\mathcal{S}^*$ on $\mathcal{N}^3(w)$. Then*

$$\mathbf{w}(\mathcal{T}) \leq \mathbf{w}(\mathcal{S}^*_w)$$

Proof. We prove that for $\varepsilon \leq 1$, $\mathcal{S}^*_w$ forms a forest that connects all the vertices in $\mathcal{N}^2(w)$ in a single component. So $\mathcal{S}^*_w$ is a feasible solution to the minimum Steiner tree problem on the set of vertices $\mathcal{N}^3(w)$ with required vertices being $\mathcal{N}^2(w)$. Thus $\mathbf{w}(\mathcal{T}) \leq \mathbf{w}(\mathcal{S}^*_w)$.

Now we just need to prove that the vertices in $\mathcal{N}^2(w)$ are connected in $\mathcal{S}^*_w$. Let u be an i-hop neighbor of w and v be an $i+1$-hop neighbor of w for some $w \in I$ and $i = 0, 1$. Assume that $(u, v) \in E$. It is enough to prove that u and v are connected in $\mathcal{S}^*_w$. In order to do so, we observe that there is a path of length at most $(1+\varepsilon)\|uv\|$ between u and v in $\mathcal{S}^*$. We show that this path is contained in $\mathcal{N}^3(w)$ and we complete the proof in this way, because $\mathbf{w}(\mathcal{S}^*_w)$ is nothing but the induced subgraph of $\mathcal{S}^*$ on $\mathcal{N}^3(w)$.

Assume, on the contrary, that there is a vertex $x \notin \mathcal{N}^3(w)$ on the $(1+\varepsilon)$-path between u

and v. This means that x is not a 1-hop neighbor of any of u and v, because otherwise x would have been in $\mathcal{N}^3(w)$. So $\|ux\| > 1$ and $\|vx\| > 1$. Thus the length of the path would be at least $\|ux\| + \|xv\| > 2 \geq (1+\varepsilon) \geq (1+\varepsilon)\|uv\|$ which is a contradiction. □

This lemma concludes our sequence of comparisons. By putting together what we proved so far, we have

Proposition 3.15. *The spanner returned by* Distributed-Spanner *has a weight of* $\mathcal{O}(1)\mathbf{w}(MST)$.

Proof. By Corollary 3.12, Lemma 3.13, and Lemma 3.14,

$$\mathbf{w}(\mathcal{S}_w) = \mathcal{O}(1)\mathbf{w}(\mathcal{S}^*_w)$$

Summing up together these inequalities for $w \in I$,

$$\mathbf{w}(\text{output}) = \mathcal{O}(1)\sum_{w \in I} \mathbf{w}(\mathcal{S}^*_w)$$

But we recall that every vertex, and hence every edge of $\mathcal{S}^*$, is repeated $\mathcal{O}(1)$ times in the summation above, so

$$\mathbf{w}(\text{output}) = \mathcal{O}(1)\mathbf{w}(\mathcal{S}^*) = \mathcal{O}(1)\mathbf{w}(MST(G))$$

□

Therefore we have all the ingredients to prove Theorem 3.16.

Theorem 3.16 (Distributed Spanner). *Given a weighted unit ball graph G with n vertices in a metric of bounded doubling dimension and a constant $\varepsilon > 0$, the algorithm* Distributed-Spanner*(G,ε) runs in $\mathcal{O}(\log^* n)$ rounds of communication in the LOCAL model of computation, and returns a $(1+\varepsilon)$-spanner of G that has constant bounds on its lightness and* *maximum degree. These constant bounds only depend on ε and the doubling dimension.*

Proof. It directly follows from Lemma 3.9, Lemma 3.10, Lemma 3.11, and Proposition 3.15. □

3.6 Adjustments for the CONGEST Model

In this section we study the problem of finding a bounded-degree $(1 + \varepsilon)$-spanner in the CONGEST model of computation, for a point set that is located in a doubling metric space. In the CONGEST model, every node can send a message of bounded size to every other node in a single round of communication. This makes it hard to gather any global information about the graph.

The maximal independent set algorithm of [70] still works in $\mathcal{O}(\log^* n)$ rounds of communication in the CONGEST model. But our proposed distributed algorithm (Algorithm 4) needs to gather 2-hop neighborhood information of every center in the MIS, which requires $\mathcal{O}(D)$ rounds in the CONGEST model, where D is the maximum degree of a vertex in the unit ball graph. The rest of the algorithm is performed locally and the number edges sent to every neighbor in the end is bounded by a constant, so the remaining of the algorithm only requires a constant number of rounds.

It is natural to ask whether our algorithm can be adapted to the CONGEST model, and if it requires more communication rounds compared to the LOCAL model. In this section we show how to modify our algorithm to work in the CONGEST model, and surprisingly, have no asymptotic change on its number of communication rounds.

As we mentioned earlier, the only step of our algorithm that requires more than constant rounds of communication is the aggregation of the 2-hop neighborhood information for every center in the MIS. We passed the 2-hop neighborhoods to our centralized algorithm to find an asymptotically optimal spanner on them, which was later distributed among the vertices in the neighborhood to form the final spanner. Here, for our construction in the CONGEST model, we directly address the problem of finding an asymptotically optimal spanner for the 2-hop neighborhoods, without the need to access all of the points in those neighborhoods.

Let $w \in I$ be a center in the maximal independent set. We partition the edges of the UBG on $\mathcal{N}^2(w)$ into two sets, depending on whether their length is larger than 1/2 or not. We aim to find asymptotically optimal spanners for each partition separately. We use the notation $G_{\leq \alpha}$ to refer to the subgraph of the unit ball graph that consists of edges of length at most α. We similarly define $G_{>\alpha}$. Therefore, we can refer to the subgraphs induced by the two partitions by $G_{\leq 1/2}$ and $G_{>1/2}$.

First, we show that in constant rounds of communication, we can find a covering of the points in $\mathcal{N}^2(w)$ with at most a constant number of balls of radius 1/2. The existence of such covering trivially follows from the definition of a doubling metric space, but finding such covering in the distributed setting is not trivial. Therefore, we introduce the following procedure: Every center $v \in \mathcal{N}^1(w)$ (including w itself) finds a maximal independent set $I_{1/4}(v)$ of the vertices $\mathcal{N}^1(v)$ in $G_{\leq 1/4}$, and sends it to w, all centers at the same time. Recall that $\mathcal{N}^1(v)$ is the set of neighbors of v in the UBG, and a maximal independent set in $G_{\leq 1/4}$ is simply a maximal set of vertices where the pair-wise distance of each two vertex is at least 1/4. Finding this maximal independent set for each v can be easily done using a (centralized) greedy algorithm, and the size of such maximal independent set would be bounded by a constant according to the packing lemma. Therefore, this step can be done in constant number of rounds. Afterwards, w calculates a maximal independent set $\mathcal{I}(w)$ of the vertices $\cup_{v \in \mathcal{N}^1(w)} I_{1/4}(v)$ in $G_{\leq 1/4}$. We show that the centers in $\mathcal{I}$ satisfy our desired properties.

Lemma 3.17. *The union of the balls of radius* $1/2$ *around the centers in* $\mathcal{I}(w)$ *cover* $\mathcal{N}^2(w)$.

Furthermore, the size of $\mathcal{I}(w)$ is bounded by a constant.

Proof. Let $v \in \mathcal{N}^2(w)$ be an arbitrary point. Thus there exists $u \in \mathcal{N}^1(w)$ that $v \in \mathcal{N}^1(u)$. Let $I_{1/4}(u)$ be the maximal independent set of the vertices $\mathcal{N}^1(u)$ in $G_{\leq 1/4}$, that u calculates and sends to w in the first step. There exists $v' \in I_{1/4}(u)$ that $\|vv'\| \leq 1/4$. Similarly, there exists $v'' \in \mathcal{I}(w)$ that $\|v'v''\| \leq 1/4$. By the triangle inequality, $\|vv''\| \leq 1/2$, i.e. v is covered by a ball of radius $1/2$ around v''.

On the other hand, $\mathcal{I}(w)$ is contained in a ball of radius 2 and every pair of points in $\mathcal{I}(w)$ have a distance of at least $1/4$. Thus, by the packing lemma, he size of $\mathcal{I}(w)$ is bounded by a constant. □

Next, every center $v \in \mathcal{I}(w)$ calculates a $(1+\varepsilon)$-spanner $\mathcal{S}_{\leq 1/2}(v)$ of the point set $\mathcal{N}^1(v)$ using the centralized algorithm, and notifies its neighbors about their connections. We prove that the union of these spanners, would be a spanner for one of the partitions, i.e. the edges of length at most $1/2$ in $\mathcal{N}^2(w)$. The pseudo-code of this procedure is available in Algorithm 6

Lemma 3.18. *The union of the spanners $\mathcal{S}_{\leq 1/2}(v)$ for $v \in \mathcal{I}(w)$ is a $(1+\varepsilon)$-spanner of $\mathcal{N}^2(w)$ in $G_{\leq 1/2}$. The maximum degree and the lightness of this spanner are both bounded by constants.*

Proof. First, we prove the $1+\varepsilon$ stretch-factor. Let (u, v) be a pair in $\mathcal{N}^2(w)$ such that $\|uv\| \leq 1/2$. By Lemma 3.17 we know there exists $u' \in \mathcal{I}(w)$ that $\|uu'\| \leq 1/2$. Thus $\|vu'\| \leq 1$ which means that $u, v \in \mathcal{N}^1(u')$ and there would be a $(1+\varepsilon)$-path for this pair in $\mathcal{S}_{\leq 1/2}(u')$, which would be present in the union of the spanners.

The degree bound follows from the fact that, by the packing lemma, every point in $\mathcal{N}^2(w)$ is appeared in at most a constant number of one-hop neighborhoods and therefore in at most a constant number of spanners constructed the elements in $\mathcal{I}(w)$. Since in every spanner it has a bounded degree, in the union it will have a bounded degree as well.

To prove the lightness bound we follow a similar approach to the proof of Proposition 3.15. The weight of each spanner $\mathcal{S}_{\leq 1/2}(v)$ is $\mathcal{O}(1)\mathbf{w}(MST(\mathcal{N}^1(v)))$. The weight of the MST is at most twice the weight of the optimal Steiner tree on $\mathcal{N}^2(v)$ with the required vertices being $\mathcal{N}^1(v)$. And the weight of this optimal Steiner tree is at most equal to the weight of the induced sub-graph of an (asymptotically) optimal $(1+\varepsilon)$-spanner of G on the subset of vertices $\mathcal{N}^2(v)$. Summing up these subgraphs for different vs and different ws would end up with adding at most a constant factor to the weight of the optimal spanner, which proves that the lightness would be bounded by a constant. □

Algorithm 5 The CONGEST spanner algorithm.

Input. A unit ball graph $G(V, E)$ in a metric with doubling dimension d and an $\varepsilon > 0$.
Output. A light-weight bounded-degree $(1+\varepsilon)$-spanner of G.

```
1: procedure CONGEST-SPANNER($G$, $\varepsilon$)
2:     Find a maximal independent set $I$ of $G$ using [70]
3:     Run SPAN-SHORT-EDGES on the vertices of $G$
4:     Run SPAN-LONG-EDGES on the vertices of $G$
```

Algorithm 6 Finding a spanner of the edges of length smaller than 1/2 in $\mathcal{N}^2(w)$.

```
1: function SPAN-SHORT-EDGES(vertex $u$)
2:     if $u \in I$ then
3:         Send a signal of type 1 to every $v \in \mathcal{N}^1(u)$.
4:         Wait for their maximal independent sets, $I_{1/4}(v)$s.
5:         Calculate a maximal independent set of $\cup_{v\in\mathcal{N}^1(u)} I_{1/4}(v)$ in $G_{<1/4}$ greedily.
6:         Store this maximal independent set in $\mathcal{I}(u)$.
7:         Send a signal of type 2 to every $v \in \mathcal{I}(u)$.
8:     if received type 1 signal from some $w$ then
9:         Calculate a maximal independent set of $\mathcal{N}^1(w)$ in $G_{1/4}$ greedily.
10:        Send this maximal independent set to $w$.
11:    if received type 2 signal from some $w$ then
12:        Calculate $\mathcal{S}_{\leq 1/2}(u) \leftarrow$ CENTRALIZED-SPANNER($\mathcal{N}^1(u)$, $\varepsilon$)
13:        for $e = (a, b)$ in $\mathcal{S}_{\leq 1/2}(u)$ do
14:            Send $e$ to $a$ and $b$
15:    Receive and store the edges sent by other centers
```

Now we find a spanner for the other partition, the edges of length larger than 1/2 in $\mathcal{N}^2(w)$. The procedure is as follows: First, every center $v \in \mathcal{N}^1(w)$ calculates a maximal independent set $I_{\varepsilon/40}(v)$ of $\mathcal{N}^1(v)$ in $G_{\leq\varepsilon/40}$ and sends it to w. Again, the size of each maximal independent set is $\mathcal{O}(\varepsilon^{-d})$ by the packing lemma, which is constant. Therefore, this step takes only constant number of rounds. Afterwards, w finds a maximal independent set $\mathcal{I}'(w)$ of $\cup_{v\in\mathcal{N}^1(w)} I_{\varepsilon/40}(v)$ in $G_{\leq\varepsilon/40}$. Then w constructs a $(1+\varepsilon/5)$-spanner $\mathcal{S}'(w)$ of $\mathcal{I}'(w)$ in G using the centralized algorithm, and announces the edges of the spanner to their corresponding endpoints. Finally, every center $v \in \mathcal{I}'(w)$ calculates a $(1+\varepsilon)$-spanner $\mathcal{S}''(v)$ of its $\varepsilon/20$ neighborhood and announces its edges to their endpoints. We show that the union of $S'(w)$ and $S''(v)$s for $v \in \mathcal{I}'(w)$ would form a $(1+\varepsilon)$-spanner of the second partition, i.e. the edges of larger than $1/2$. The pseudo-code of this procedure is available in Algorithm 7

Lemma 3.19. *The union of the balls of radius $\varepsilon/20$ around the centers in $\mathcal{I}'(w)$ cover $\mathcal{N}^2(w)$. Furthermore, the size of $\mathcal{I}'(w)$ is bounded by a constant.*

Proof. Similar to the proof of Lemma 3.17. □

Lemma 3.20. *The union of the spanners $\mathcal{S}'(w)$ and $\mathcal{S}''(v)$ for $v \in \mathcal{I}'(w)$ forms a $(1+\varepsilon)$-spanner of $\mathcal{N}^2(w)$ in $G_{>1/2}$. The maximum degree and the lightness of this spanner are btoh bounded by constants.*

Proof. Again, we first prove the $1+\varepsilon$ stretch-factor of the spanner. Let (u,v) be a pair in $\mathcal{N}^2(w)$ that $\|uv\| > 1/2$. Let u' and v' be centers in $\mathcal{I}'(w)$ that are at distance of at most $\varepsilon/20$ from u and v, respectively. Such centers exist according to Lemma 3.19. Consider the $(1+\varepsilon)$-path connecting u to u' in $\mathcal{S}''(u')$ and the $(1+\varepsilon)$-path connecting v to v' in $\mathcal{S}''(v')$. We can attach these paths together with the $(1+\varepsilon/5)$-path between u' and v' in $\mathcal{S}'(w)$ to get a path between u and v. The stretch of this path would be at most

$$\frac{(1+\varepsilon)(\|uu'\|+\|vv'\|)+(1+\varepsilon/5)\|u'v'\|}{\|uv\|} = \frac{(1+\varepsilon)(\|uu'\|+\|vv'\|)}{\|uv\|} + \frac{(1+\varepsilon/5)\|u'v'\|}{\|uv\|}$$

But,

$$\frac{(1+\varepsilon)(\|uu'\|+\|vv'\|)}{\|uv\|} \leq \frac{(1+\varepsilon)\varepsilon/10}{1/2} \leq \frac{2\varepsilon}{5}$$

Also,

$$\frac{(1+\varepsilon/5)\|u'v'\|}{\|uv\|} \leq \frac{(1+\varepsilon/5)(\|uv\|+\|uu'\|+\|vv'\|)}{\|uv\|} \leq 1+\varepsilon/5+\frac{(1+\varepsilon/5)\varepsilon/10}{\|uv\|}$$

Bounding the last term,

$$\frac{(1+\varepsilon/5)\varepsilon/10}{\|uv\|} \leq \frac{(6/5)\varepsilon/10}{1/2} = \frac{6\varepsilon}{25}$$

Therefore, the stretch of the uv-path would be upper bounded by,

$$\frac{2\varepsilon}{5}+1+\varepsilon/5+\frac{6\varepsilon}{25} < 1+\varepsilon$$

An approach similar to the proof of Lemma 3.18 shows that the degree of every vertex in the union of $\mathcal{S}''(v)$s would be bounded by a constant. We do not repeat the details of the proof here. From the properties of our centralized construction, the degree of every vertex would be bounded in $\mathcal{S}'(w)$ as well. Thus, the degree of every vertex in the union of these spanners would be bounded by a constant.

To prove the lightness bound, we bound the weight of each spanner separately. First, we bound the total weight of $\mathcal{S}'(w)$. We know from the properties of our centralized construction, that $\mathbf{w}(\mathcal{S}'(w)) = \mathcal{O}(1)\mathbf{w}(MST(\mathcal{I}'(w)))$. But $\mathbf{w}(MST(\mathcal{I}'(w))) \leq 2MST(\mathcal{N}^2(w))$, so $\mathbf{w}(\mathcal{S}'(w)) = \mathcal{O}(1)\mathbf{w}(MST(\mathcal{N}^2(w)))$. Therefore, by Lemma 3.13 and Lemma 3.14 the total weight of $\mathcal{S}'(w)$ spanners for different centers w would sum up to at most a constant factor of the weight of the optimal spanner.

Next, we bound the total weight of $\mathcal{S}''(v)$ spanners. Again, we know from the properties of our centralized construction that $\mathbf{w}(\mathcal{S}''(v)) = \mathcal{O}(1)\mathbf{w}(MST(\mathcal{N}^{\varepsilon/20}(v)))$. Assuming that $\mathcal{S}^*$ is an optimal spanner on the point set, we can observe that any $(1+\varepsilon)$-path (in $\mathcal{S}^*$) between any pair of vertices in $\mathcal{N}^{\varepsilon/20}(v)$ must be completely contained in a ball of radius $3\varepsilon/20$, otherwise the length of the path would be more than $(1+\varepsilon)\varepsilon/10$, the maximum allowed length for any

$(1+\varepsilon)$-path of any pair in the $\mathcal{N}^{\varepsilon/20}(v)$ neighborhood. Therefore, the induced sub-graph of $\mathcal{S}^*$ on $\mathcal{N}^{3\varepsilon/20}(v)$ has a connected component connecting the vertices of $\mathcal{N}^{\varepsilon/20}(v)$. Thus, its weight is at least equal to the weight of a minimum Steiner tree on $\mathcal{N}^{3\varepsilon/20}(v)$, with the required vertices being $\mathcal{N}^{\varepsilon/20}(v)$. This is at least equal to $\mathbf{w}(MST(\mathcal{N}^{\varepsilon/20}(v)))/2$. Therefore, the weight of $\mathcal{S}''(v)$ is bounded above by a constant factor of the weight of the induced sub-graph of $\mathcal{S}^*$ on $\mathcal{N}^{3\varepsilon/20}(v)$. Summing up these bounds for every v in every w would lead to at most a constant repetitions of every vertex and every edge (similar to Proposition 3.15) in $\mathcal{S}^*$, which shows that the total weight of $\mathcal{S}''(v)$ for different vertices of v would be bounded by a constant factor of the weight of the optimal spanner. □

Algorithm 7 Finding a spanner of the edges of length larger than 1/2 in $\mathcal{N}^2(w)$.

```
1: function Span-Long-Edges(vertex $u$)
2:     if $u \in I$ then
3:         Send a signal of type 3 to every $v \in \mathcal{N}^1(u)$.
4:         Wait for their maximal independent sets, $I_{\varepsilon/40}(v)$s.
5:         Calculate a maximal independent set of $\cup_{v\in\mathcal{N}^1(u)} I_{\varepsilon/40}(v)$ in $G_{<\varepsilon/40}$ greedily.
6:         Store this maximal independent set in $\mathcal{I}'(u)$.
7:         Send a signal of type 4 to every $v \in \mathcal{I}'(u)$.
8:         Calculate $\mathcal{S}'(u) \leftarrow$ Centralized-Spanner($\mathcal{I}'(u)$, $\varepsilon/5$)
9:         for $e = (a, b)$ in $\mathcal{S}'(u)$ do
10:            Send $e$ to $a$ and $b$
11:    if received type 3 signal from some $w$ then
12:        Calculate a maximal independent set of $\mathcal{N}^1(w)$ in $G_{\varepsilon/40}$ greedily.
13:        Send this maximal independent set to $w$.
14:    if received type 4 signal from some $w$ then
15:        Let $\mathcal{N}^{\varepsilon/20}(u)$ be the $\varepsilon/20$ neighborhood of $u$, i.e. the set of vertices that are at
       distance $\varepsilon/20$ or less from $u$.
16:        Calculate $\mathcal{S}''(u) \leftarrow$ Centralized-Spanner($\mathcal{N}^{\varepsilon/20}(u)$, $\varepsilon$)
17:        for $e = (a, b)$ in $\mathcal{S}''(u)$ do
18:            Send $e$ to $a$ and $b$
19:    Receive and store the edges sent by other centers
```

The union of the two spanners for the two partitions form a spanner for the 2-hop neighborhood of w, the goal we wanted to achieve in the CONGEST model. This completes our adjustments in this model.

Theorem 3.21 (CONGEST Spanner). *Given a weighted unit ball graph G with n vertices in a metric of bounded doubling dimension and a constant $\varepsilon > 0$, the algorithm* CONGEST-SPANNER*(G,ε) runs in $\mathcal{O}(\log^* n)$ rounds of communication in the CONGEST model of computation, and returns a $(1+\varepsilon)$-spanner of G that has constant bounds on its lightness and maximum degree. These constant bounds only depend on ε and the doubling dimension.*

Proof. The proof follows from Lemma 3.18 and Lemma 3.20. □

3.7 Low-Intersection Construction

The two dimensional Euclidean case of the unit ball graphs, also known as unit disk graphs, had gained a significant attention once it was introduced, due to its direct application in wireless ad-hoc networks. A huge amount of effort is still being made to improve the existing spanners in some aspects, or to introduce new constructions that possess good qualities, e.g. being fault tolerant, or having a low interference [78, 87, 20, 88].

In this section, we take the edge intersection as a simple representation of physical link-to-link interference, and we provide a distributed construction for a light-weight low-intersection bounded-degree spanner for unit disk graphs in the two dimensional Euclidean plane. To the best of our knowledge, this is the first distributed low-intersection construction with constant bounds on the degree and lightness.

We need to emphasize that after removing the edges of longer than 1 from the output of NAIVE-GREEDY on a unit disk graph G, the remaining graph would be a $(1+\varepsilon)$-spanner of G and it also has constant bounds on its lightness, maximum degree, and average number of edge intersections per node. Equivalently, we can stop the algorithm before reaching the pairs with distance greater than 1 and the resulting spanner would be the same. We present this centralized algorithm in the following form and we use it as a part of our distributed algorithm.

Algorithm 8 The centralized construction for a low-intersection spanner of UDG.
Input. A unit disk graph $G(V, E)$ in the two dimensional Euclidean plane.
Output. A light-weight low-intersection bounded-degree $(1+\varepsilon)$-spanner of G.

```
1: procedure Centralized-Euclidean-Spanner($G$, $\varepsilon$)
2:     Let $S$ be a graph with vertices $V$ and edges $E = \{\}$
3:     for each $(P, Q) \in V^2$ in increasing order of $\text{dist}(P, Q)$ do
4:         if $\text{dist}(P, Q) > 1$ then
5:             break
6:         if $\text{dist}_S(P, Q) > t \cdot \text{dist}(P, Q)$ then
7:             Add edge $PQ$ to $E$
       return S
```

Theorem 3.22 (Centralized Euclidean Spanner)**.** *Given a weighted unit disk graph G in the two dimensional Euclidean plane and a constant $\varepsilon > 0$, the spanner returned by* Centralized-Euclidean-Spanner*(G,ε) is a $(1+\varepsilon)$-spanner of G and has constant bounds on its lightness, maximum degree, and the average number of edge intersections per node. These constant bounds only depend on ε and the doubling dimension.*

Proof. The stretch-factor follows directly from the fact that every edge of the UDG can be approximated by a spanner path within a factor of t (otherwise the edge would have been added to the spanner). And since the spanner is a sub-graph of the greedy spanner on the complete weighted graph on V, we can deduce that its lightness, its maximum degree, and its average number of edge intersections per node are bounded by the lightness, maximum degree, and the average number of edge intersections per node of the greedy spanner, respectively. And these are all bounded by constants according to [51] and [45]. □

3.7.1 The algorithm

The main result of this section is the distributed construction. Our distributed algorithm (Algorithm 9) is based on the algorithm we presented in section 3.5, with a small change that we use the Centralized-Euclidean-Spanner procedure as a sub-routine instead of Centralized-Spanner. The pseudo-code is provided in Algorithm 9.

Algorithm 9 The distributed construction for a low-intersection spanner of UDG.

Input. A unit disk graph $G(V, E)$ in the two dimensional Euclidean plane.
Output. A light-weight bounded-degree $(1 + \varepsilon)$-spanner of G.

```
1: procedure Distributed-Euclidean-Spanner($G$, $\varepsilon$)
2:     Find a maximal independent set $I$ of $G$ using [70]
3:     Run Local-Greedy on the vertices of $G$
4: function Local-Greedy(vertex $w$)
5:     Retrieve $\mathcal{N}^2(w)$, the 2-hop neighborhood information of $w$
6:     if $w$ is in $I$ then
7:         $\mathcal{S}_w \leftarrow$ Centralized-Euclidean-Spanner($\mathcal{N}^2(w)$, $\varepsilon$)
8:         for $e = (u, v)$ in $\mathcal{S}_w$ do
9:             Send $e$ to $u$ and $v$
10:    Listen to incoming edges and store them
```

Similar to Algorithm 4 this algorithm runs in $\mathcal{O}(\log^* n)$ rounds of communications during which it builds a $(1 + \varepsilon)$-spanner of the unit disk graph G.

Lemma 3.23. *The spanner returned by* Distributed-Euclidean-Spanner *has a stretch-factor of* $1 + \varepsilon$*, a weight of* $\mathcal{O}(1)\mathbf{w}(MST)$*, and a maximum degree of* $\mathcal{O}(1)$*.*

Proof. The proof follows from Lemma 3.10, Proposition 3.15, and Lemma 3.11 which all hold for this construction as well. □

Now we prove the low-intersection property of our distributed construction.

Proposition 3.24. *The spanner returned by* Distributed-Euclidean-Spanner *has at most a linear number of edge intersections.*

Proof. We generalize the result of the previous chapter on the edge crossings of the greedy spanner, which states that an arbitrary edge AB of the greedy spanner intersects with at most a constant number of longer edges. The main observation is that the same statement is true when AB is an arbitrary segment on the plane, and the assumption of AB being an edge of the spanner could be eliminated from the proof of the theorem. The modified statement of the theorem for our case would be as follows,

Lemma 3.25. *Given an arbitrary segment $\|AB\| \leq 1$ in the plane, the number of edges $e \in S$ that intersect AB and $|e| \geq \|AB\|$ is bounded by a constant.*

Proof. We divide the proof into two cases:

1. Intersections of AB with significantly larger edges, and
2. Intersections of AB with edges of within a constant factor length difference.

For each case we prove a constant bound on the number of intersections of AB with those edges. First, we consider the intersections with significantly larger edges. It would be sufficient if we prove a constant bound on the number of θ-parallel large edges, where θ is a small constant. By θ-parallel (almost-parallel) we mean a set of edges that their pair-wise angle is at most θ. In Chapter 2 we defined an end-point ordering of the spanner edges over a baseline, which is selected arbitrarily from the set of almost parallel segments, and we proved the following lemma.

Lemma 3.26. *Let MN and PQ be two intersecting segments from a set of θ-parallel spanner segments. Also assume that $\theta < \frac{t-1}{2t}$ where t is the stretch factor of the spanner. Then MN and PQ are endpoint-ordered, i.e. the projection of one of the segments on the baseline of the set cannot be included in the projection of the other one.*

This lemma assumes that MN and PQ intersect each other on an interior point. We can generalize this lemma to a case that they both intersect an arbitrary segment AB on the plane.

Lemma 3.27. *Let MN and PQ be two segments chosen from a set of θ-parallel spanner segments that cross an arbitrary segment AB. Also assume that $\theta < \frac{t-1}{2(t+1)}$, and $\min(|MN|, |PQ|) \geq$*

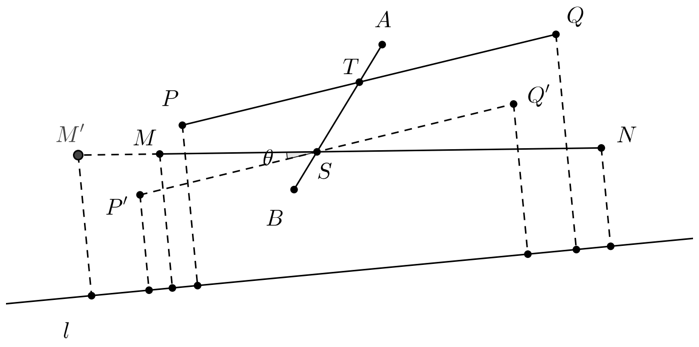


Figure 3.3: Proof of Lemma 3.27.

$\frac{3t(t+1)}{t-1}|AB|$, where t is the spanner parameter. Then MN and PQ are endpoint-ordered.

Proof. The proof goes by contradiction. Let l be an arbitrarily chosen baseline from our set of θ-parallel segments. Without loss of generality suppose that the projections of P and Q on the baseline are both between the projections of M and N. We use Lemma 3.26 to show that MN can be shortcut by PQ by a factor of t, i.e.

$$t \cdot |MP| + |PQ| + t \cdot |QN| \leq t \cdot |MN|$$

We move PQ by slightly with respect to its length, so that the new segment and MN intersect each other, and then we use Lemma 3.26 for these segments. Let MN and PQ intersect AB at S and T, respectively. We move PQ by $\overrightarrow{TS}$ so that it intersects MN. Let $P'Q'$ be the result of the movement. The projections of P' and Q' on the baseline may not be between M and N. We can extend MN on one side by $|\overrightarrow{TS}|$ to get a new segment $M'N'$, and the property would hold afterwards. Before the movement the projections of P and Q are both between the projections of M and N, so after movement at most one of the projections of P' or Q' can be outside of the projections of M and N. So extending on one side will be sufficient.

Now e can use Lemma 3.26 for $P'Q'$ and $M'N'$. By the assumption $\theta = \frac{t'-1}{2t'}$ where $t' =$

$(t+1)/2$, so Lemma 3.26 implies that,

$$t' \cdot |M'P'| + |P'Q'| + t' \cdot |Q'N'| \leq t' \cdot |M'N'| \tag{3.4}$$

By the triangle inequality after this movement MP and NQ each decrease by at most $|\overrightarrow{TS}| \leq |AB|$. So,

$$|M'P'| \geq |MP| - |AB|, \; |N'Q'| \geq |NQ| - |AB| \tag{3.5}$$

Also length of MN will increase by at most $|\overrightarrow{TS}| \leq |AB|$, so

$$|M'N'| \leq |MN| + |AB| \tag{3.6}$$

The length of PQ does not change. From equations 3.4, 3.5, and 3.6,

$$\begin{aligned} |PQ| = |P'Q'| &\leq t' \cdot (|M'N'| - |M'P'| - |N'Q'|) \\ &\leq \frac{t+1}{2} \cdot (|MN| - |MP| - |NQ| + 3|AB|) \\ &\leq \frac{t+1}{2} \cdot (|MN| - |MP| - |NQ|) + \frac{t+1}{2} \cdot \frac{t-1}{t(t+1)}|PQ| \end{aligned}$$

So

$$|PQ| \leq t \cdot (|MN| - |MP| - |NQ|)$$

□

This implies a total ordering of the set of almost-parallel edges on the given baseline.

Corollary 3.28. *Given an arbitrary segment AB one the plane, for a set of sufficiently large almost-parallel (with respect to AB) spanner edges that intersect AB, the endpoint-ordering is a total ordering.*

Also, by the endpoint-gap property between the edges of the greedy spanner, they proved the following lower bound on the endpoint distance of two greedy spanner edges,

Lemma 3.29. *Let MN and PQ be two θ-parallel spanner segments. The matching endpoints of these two segments cannot be closer than a constant fraction of the length of the smaller segment. More specifically,*

$$\min(|MP|, |NQ|) \geq \frac{t - 1 - 2\sin(\theta/2)}{2t} \min(|MN|, |PQ|)$$

Lemma 3.27 together with lemma 3.29 can be used to prove a constant bound on the number of such edges.

Lemma 3.30. *For small θ, the number of sufficiently large θ-parallel edges of a greedy t-spanner that intersect a arbitrary segment AB on the plane is bounded by a constant. More specifically, the length of the segments should be larger than $\frac{3t(t+1)}{t-1}|AB|$.*

This proof is similar to Theorem 2.14, except we use Lemma 3.30 instead of Proposition 2.13.

For edges whose lengths are between $|AB|$ and $\frac{3t(t+1)}{t-1}|AB|$ on the other hand, we only need the endpoint-gap property of the greedy spanner. Similar to Chapter 2 we can show that,

Lemma 3.31. *The number of spanner segments PQ that cross an arbitrary segment AB of the plane and that have length between $\alpha \cdot |AB|$ and $\beta \cdot |AB|$ is bounded by a constant.*

Proof. We partition the area around AB with squares of edge length $\frac{t-1}{2\sqrt{2}t} \cdot \alpha|AB|$ with edges parallel or perpendicular to AB. The area that an endpoint of a crossing segment can lie in is a rectangle of size $(2\beta + 1)|AB|$ by $2\beta|AB|$. The total number of cells in this area would be

$$\frac{2\beta(2\beta+1)}{\alpha^2} \cdot \frac{8t^2}{(t-1)^2}$$

And for each crossing segment the pair of cells that contain the two endpoints of the segment is unique. Otherwise two segments, e.g. MN and PQ, will have both endpoints at the same pair, which means

$$\begin{aligned}\max(|MP|, |NQ|) &< (\sqrt{2})(\frac{t-1}{2\sqrt{2}t} \cdot \alpha|AB|) = \frac{t-1}{2t} \cdot \alpha|AB| \\ &\leq \frac{t-1}{2t} \min(|MN|, |PQ|)\end{aligned}$$

which is impossible due to Proposition 2.13. So the total number of such edges would be at most,

$$\left[\frac{2\beta(2\beta+1)}{\alpha^2} \cdot \frac{8t^2}{(t-1)^2}\right]^2$$

□

Putting together Lemma 3.30 and Lemma 3.31 we can deduce Lemma 3.25. □

We use this lemma to bound the number of intersections of each edge of the spanner with the longer edges, which in turn proves the linear bound on the number of edge intersections in total.

Let $e = (u, v)$ be an edge of the final spanner and let $f = (u', v') \in S_w$ for some w be another edge that intersects e. By the triangle inequality, at least one of u' and v' needs to be adjacent to u in G. Thus $u \in \mathcal{N}^3(w)$, which means that $\|uw\| \leq 3$. So by the packing property there are at most $12^2 = 144$ different choices for w that $u \in \mathcal{N}^3(w)$. But for every such w, according to Lemma 3.25 for the segment $AB = e$, there are at most a constant number of intersections between e and the edges of length larger than $|e|$ in S_w. Since there are at most a constant number of choices for w and for each choice there are at most a constant number of intersections between e and the longer edges in S_w, we conclude that the number of intersections between e and longer edges in the final spanner would be bounded by a constant, which indeed proves the proposition. □

Thus the following theorem holds for our distributed spanner construction for the two dimensional Euclidean plane.

Theorem 3.32 (Distributed Euclidean Spanner)**.** *Given a weighted unit disk graph G with n vertices in the two dimensional Euclidean plane and a constant $\varepsilon > 0$, the algorithm* DISTRIBUTED-EUCLIDEAN*(G,ε) runs in $\mathcal{O}(\log^* n)$ rounds of communication and returns a bounded-degree $(1 + \varepsilon)$-spanner of G that has constant bounds on its lightness, maximum degree, and the average number of edge intersections per node. These constant bounds only depend on ε and the doubling dimension.*

Proof. Follows directly from Lemma 3.23 and Proposition 3.24. □

This low-intersection property also implies the existence of small separators for our spanner, which is stated in the following corollary.

Corollary 3.33. *The spanner returned by* DISTRIBUTED-EUCLIDEAN-SPANNER *has a separator of size $\mathcal{O}(\sqrt{n})$, where n is the number of vertices. Also, a separator hierarchy can be constructed from its planarization in linear time.*

Proof. Lemma 3.25 implies that the crossing graph of our spanner has a bounded degeneracy. Therefore, this corollary follows from the result of Eppstein and Gupta [44]. □

3.7.2 Higher dimensions

In higher dimensions of Euclidean spaces, it does not particularly make sense to talk about the edge intersections of the spanner, as the edges would not intersect for points in general locations. But we can generalize our separator result to higher dimensions of Euclidean spaces.

Recently, Li and Than [72] proved that any geometric graph with a property that they called τ-lanky has separators of size $\mathcal{O}(\tau n^{1-1/d})$, where d is the dimension of the space. They also proved that greedy spanners are $\mathcal{O}(1)$-lanky and therefore their k-vertex subgraphs have separators of size $\mathcal{O}(k^{1-1/d})$. We can take advantage of this result for higher dimensions and prove the existence of small separators for our construction in higher dimensions of Euclidean spaces.

Lemma 3.34. *Let S be the output of* Distributed-Euclidean-Spanner *on a set of points in the d-dimensional Euclidean space. Given an arbitrary ball of radius $R \leq 1$ in this space, the number of edges $e \in S$ that cut this ball would be bounded by a constant.*

Proof. A similar proof to the proof of Lemma 3.25 yields this result in higher dimensions as well. This result is also proven for the centralized greedy algorithm in [72], but the extension to the distributed setting needs some considerations, which are similar to the proof of Lemma 3.25, and not included to avoid repetition. □

According to Lemma 3.34 our construction in d-dimensional Euclidean space is $\mathcal{O}(1)$-lanky and therefore, it possesses separators of small size.

Corollary 3.35. *The spanner returned by* Distributed-Euclidean-Spanner *in the d-dimensional Euclidean space is $\mathcal{O}(1)$-lanky. Therefore, any k-vertex subgraph of this construction possesses separators of size $\mathcal{O}(k^{1-1/d})$. Also, a separator hierarchy can be built for this spanner in expected linear time.*

Proof. The $\mathcal{O}(1)$-lanky property follows from Lemma 3.34, and the existence of separators of size $\mathcal{O}(k^{1-1/d})$ and the expected time on finding a separator hierarchy follows from the result of [72]. $\square$

3.8 Experimental Results

In this section we provide empirical evidence for the efficiency of our distributed construction. While we have proven rigorous bounds on the lightness and sparsity of our spanner, it might be unclear how it performs in practice compared to an efficient centralized construction. We design an experiment in the two dimensional Euclidean plane that answers this question.

We run our distributed spanner algorithm (Algorithm 9) on a point set consisting of 100 points uniformly chosen at random from a 5 by 5 square in the two dimensional Euclidean plane. This point set, together with its unit disk graph, is drawn in Figure 3.4. In the first part of the experiment, we run our distributed algorithm for different values of t, the stretch-factor, and we compare the result of our algorithm with the output of the centralized greedy spanner on the same point set, with the same parameter t.

The results of this (Figure 3.5) shows a near-optimal performance from our distributed algorithm. As we mentioned earlier, the centralized greedy spanner is known for its near-optimality, and our construction is comparable with this near-optimal solution.

Next, we define the efficiency of a distributed algorithm with respect to a measure. Let $\mathcal{M}$ be a measure that we would like to minimize, e.g. maximum degree, size, or total weight. Then we define the *efficiency* of a distributed construction with respect to $\mathcal{M}$ to be the ratio

$$\frac{\mathcal{M}(Greedy)}{\mathcal{M}(ALG)}$$

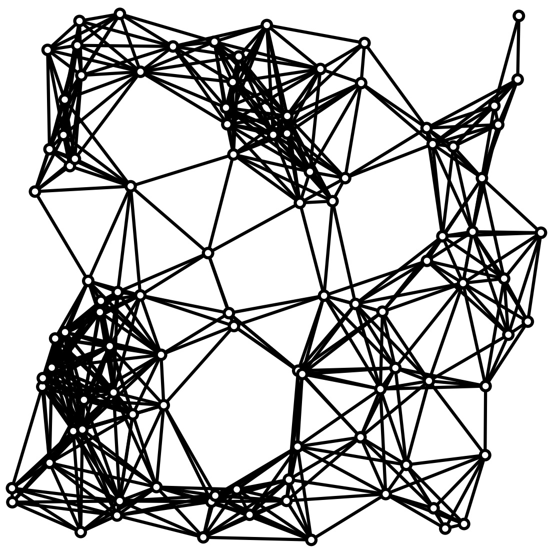
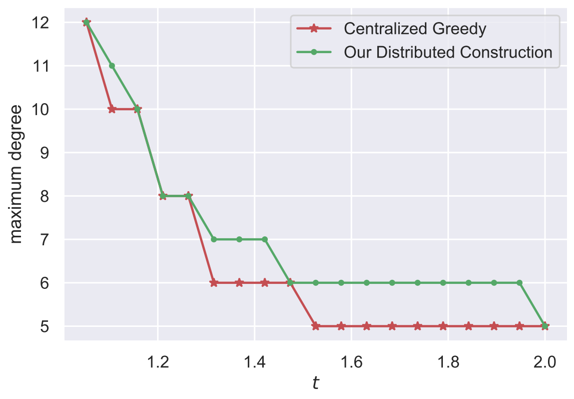


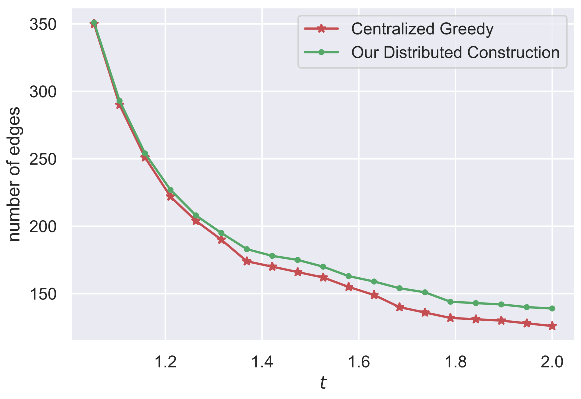
Figure 3.4: The random point set and its unit disk graph, from the first part of our experiment.

where *ALG* is the output of the distributed algorithm and *Greedy* is the output of the centralized greedy on the same point set. We choose the centralized greedy as our base of comparison because it is known to be near-optimal. In the second part of our experiments, we compare the efficiency of our distributed algorithm against the centralized greedy algorithm with respect to the maximum degree, size, and total weight, for 10 random point sets chosen the same way as in the first part. The average of these efficiencies for each measure is reported in Figure 3.6.

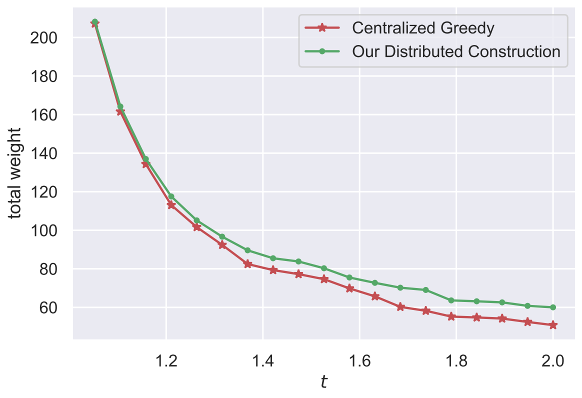
We again observe that our distributed algorithm is performing efficiently with respect to the size and total weight. We denote that when $t = 2$, the maximum degree of the greedy spanner is 5, so even a single extra edge around any vertex would cause an efficiency below 83%. Therefore, even for high values of t our algorithm is performing decently.



(a) Degree comparison



(b) Size comparison



(c) Weight comparison

Figure 3.5: Comparisons for a random instance of 100 points uniformly taken from a 5×5 square, for different values of the stretch parameter, t.

3.9 Conclusions

In this chapter we resolved an open question from 2006 and we proved the existence of light-weight bounded-degree $(1+\varepsilon)$-spanners for unit ball graphs in the spaces of bounded doubling dimension. Moreover, we provided a centralized construction and a distributed construction in the LOCAL model that finds a spanner with these properties. Our distributed construction runs in $\mathcal{O}(\log^* n)$ rounds, where n is the number of vertices in the graph. If a maximal independent set of the unit ball graph is known beforehand, our algorithm runs in constant number of rounds. Next, we showed how to adjust our distributed construction to work in the CONGEST model, without touching its asymptotic round complexity. In

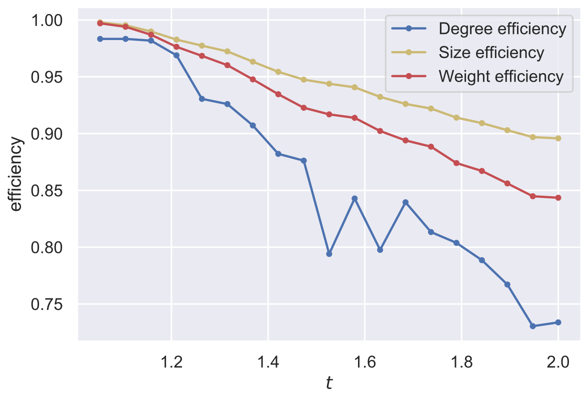


Figure 3.6: The average efficiencies with respect to maximum degree, size, and total weight, for 10 random instances of 100 points uniformly taken from a 5×5 square, plotted against different stretch parameters, t.

this way, we provided the first CONGEST algorithm for finding a light spanner of unit ball graphs.

In addition, we further adjusted these algorithms (in section 3.7) for the case of unit disk graphs in the two dimensional Euclidean plane, and we presented the first centralized and distributed constructions for a light-weight bounded-degree $(1+\varepsilon)$-spanner that also has a linear number of edge intersections in total. This can be useful for practical purposes if minimizing the number of edge intersections is a priority. We proved, based on this low-intersection property, that our spanner has sub-linear separators, and a separator hierarchy, and we were able to generalize this result to higher dimensions of Euclidean spaces.

Finally, we ran experiments (in section 3.8) on random point sets in the two dimensional Euclidean plane, to ensure that our theoretical bounds are also supported by enough empirical evidence. Our results show that our construction performs efficiently with respect to the maximum degree, size, and total weight.

Chapter 4

Fully Dynamic Spanners with Small Recourse

4.1 Background

In this chapter, we study the problem of maintaining $1+\varepsilon$-spanners under a dynamic model in which points are inserted and removed by an adversary and our goal is to minimize the recourse, which is the number of changes we make to the edge set of the spanner. The recourse should be distinguished from the time it takes us to calculate the changes we make, which might be larger; our use of recourse instead of update time is motivated by real-world networks, where making a physical change to the network is often more costly than the actual run-time of the algorithm that decides what changes need to be made.

We introduce a hierarchical structure that we update with minimal changes after each operation. We use this hierarchy as the basis of our sparse spanner. It is worth noting that using hierarchical structures to build sparse spanners was known in prior work, but our hierarchy is designed in a way that it suits our needs in this chapter. Then we turn our attention into light-weight spanners and we use novel concepts and ideas (such as the notion of stretch factor for subsets of edges and quantifying the impact of an edge update on other edges through a potential function) to iteratively lighten the weight of the spanner after point insertion and deletion. We also use well-known techniques in prior work such as bucketing and amortized analysis, which eventually lead us to our results on the amortized number of edge updates in each bucket. This was made possible through carefully crafting a potential function that decreases via our maintenance updates on the spanner.

We have covered many applications of geometric spanners so far. However, in all these applications the point set was known before running the algorithm, which is also known as the *offline* setting in algorithms. In some other applications, the points of an input set may repeatedly change as a spanner for them is used, and a static network would not accurately represent their distances. The dynamic model, detailed below, deal with these types of problems.

In the dynamic model, points are inserted or removed one at a time, and the algorithm has to maintain a t-spanner at all times. In this setting the algorithm is allowed to remove previous edges. For n points in d-dimensional Euclidean space, Arya, Mount, and Smid [7] designed a spanner construction with a linear number of edges and $\mathcal{O}(\log n)$ diameter under the assumption that a point to be deleted is chosen randomly from the point set, and a point to be inserted is chosen randomly from the new point set. Bose, Gudmundsson, and Morin [16] presented a semi-dynamic $(1+\varepsilon)$-spanner construction with $\mathcal{O}(\log n)$ maximum degree and diameter. Gao, Guibas, and Nguyen [55] designed the deformable spanner, a fully-dynamic construction with $\mathcal{O}(\log \Delta)$ maximum degree and $\mathcal{O}(\log \Delta)$ lightness, where Δ is the aspect ratio of the point set, defined as the ratio of the length of the largest edge divided by the length of the shortest edge.

In the spaces of bounded doubling dimension, Roditty [82] provided the first dynamic spanner construction whose update time (and therefore recourse) depended solely on the number of points ($\mathcal{O}(\log n)$ for point insertion and $\tilde{\mathcal{O}}(n^{1/3})$ for point removal). This was later improved by Gottlieb and Roditty [60], who extended this result in doubling metrics and provided a better update time as well as the bounded-degree property. The same authors further improved this construction to have an asymptotically optimal insertion time (and therefore recourse) of $\mathcal{O}(\log n)$ under the algebraic decision tree model [61] but logarithmic lightness.

It is worth to mention that none of the work mentioned above in the dynamic setting achieve a sub-logarithmic lightness bound on their output. The problem of maintaining a light spanner in this setting has remained open until now.

4.2 Overview

Light-weight fully-dynamic spanners have not been studied in the literature to the best of our knowledge. There are currently no known algorithms that provide a spanner with constant lightness except by rebuilding the whole spanner. We construct a fully-dynamic spanner that aims to minimize the recourse, defined by the number of edges updated after a point insertion or removal. Our spanner maintains, at all times, a lightness and a maximum degree that are bounded by constants. Our maintenance regime achieves amortized constant recourse per point insertion, and amortized $\mathcal{O}(\log \Delta)$ recourse per point deletion. We state and prove our bounds in theorem 4.24.

Theorem 4.24. Our fully-dynamic spanner construction in d-dimensional Euclidean spaces has a stretch-factor of $1 + \varepsilon$ and a lightness that is bounded by a constant. Furthermore, this construction performs an amortized $\mathcal{O}(1)$ edge updates following a point insertion, and an amortized $\mathcal{O}(\log \Delta)$ edge updates following a point deletion.

The hidden constants in our bounds only depend on ε and d. Our amortized bound for recourse after point insertion is optimal but for point deletion we do not claim optimality.

However, it is worth to mention that our recourse bound for deletion is not worse than the bounds achieved in prior work. In order to reach our bounds on recourse we introduce new techniques for iteratively improving the weight of the spanner without losing its other characteristics.

4.3 Preliminaries

In this section, we cover the notations as well as important definitions and facts that we use throughout the chapter. We also provide an overview of what to expect in the upcoming sections and the methods we use to reach our bounds on the recourse.

Notation. We denote the current point set by V and its aspect ratio (as defined earlier) by Δ. We use the notations $\|e\|$ and $\|P\|$ for the Euclidean length of an edge e and a path P, respectively. We also refer to the Euclidean distance of two points u and v by $\|uv\|$ or $d(u, v)$, interchangeably. The notation $|E|$ is used when we are referring to the size of a set E. Also, for a spanner S, the weight of S is shown by $w(S)$.

We build our spanners on top of a hierarchical clustering $(\mathcal{T}, R)$ of the point set that we maintain dynamically as the point set changes over time. The tree $\mathcal{T}$ represents the parent-child relationship between the clusters, and the constant R specifies how cluster radii magnify on higher levels. Each cluster $\mathcal{C} \in \mathcal{T}$ is specified by a pair $\mathcal{C} = (p, l)$ where $p \in \mathbb{R}^d$ is one of the given points at the center of the cluster and $l \in \mathbb{Z}$ is the level of the cluster. The level of a cluster determines its radius, R^l. It is possible for the same point to be the center of multiple clusters, at different levels of the hierarchy.

We maintain our hierarchy so that after a point insertion, a cluster is added centered at the new point, and after a point deletion, each cluster with the deleted point as its center is removed. Meanwhile, we maintain a *separation* property on the hierarchy to help us build a sparse spanner. Additional edges of our sparse spanner connect pairs of clusters of the same level. Each such edge ensures that pairs of descendants of its endpoints have the desired stretch-factor. These edges form a bounded-degree graph on the clusters at each level, but this property alone would not ensure bounded degree for our whole spanner, because of points that center multiple clusters. Instead, we redistribute the edges of large degree points to derive a bounded-degree spanner.

Maintaining bounded lightness on the other hand is done through an iterative pruning process. We start by removing certain edges to decrease the weight of the spanner, which in turn might cause some other pairs that previously used the removed edge in their shortest paths to not meet the stretch bound of $1 + \varepsilon$. We fix those pairs by adding an edge between them, which again increases the weight of the spanner. This causes a chain of updates that alternatively improve the stretch and worsen the weight of the spanner, or improve the weight and worsen the stretch of the spanner. We show that this sequence of updates, which we call *maintenance* updates, if performed properly and for the right pairs, will indeed not end in a loop, and even more strongly, will terminate after an amortized constant number of iterations. This will be covered in section 4.5.

The rest of this section includes the techniques we use for our light-weight spanner construction. We start with one of these techniques which is called the *bucketing* technique. Instead of enforcing the stretch bound and the lightness bound on the whole spanner, we partition its edges into a constant number of subsets and we enforce our criteria on these subsets. This partitioning is necessary for the purpose of our analysis.

Bucketing. We maintain a partition of the spanner edges into a constant number of subsets. As we mentioned before, our invariants are enforced on these subsets instead of the whole spanner. Let $C \gg c > 1$ be constants that we specify later. We partition the edges of the spanner into $k = \lceil \log_c C \rceil$ subsets, $S_0, S_1, \cdots, S_{k-1}$, so that for each set S_i and any pair of edges $e, f \in S_i$ such that $\|e\| \geq \|f\|$, one of the following two cases happen: (i) either

$\|e\|/\|f\| < c$ or (ii) $\|e\|/\|f\| \geq C$. In other words, the edge lengths in the same set are either very close, or very far from each other.

Such partitioning can be maintained easily by assigning an edge e to the set with index $\text{index}(e) = \lfloor \log_c \|e\| \rfloor \bmod k$. We refer to this as the *index* of the edge e. We also define the *size* of an edge e as $\text{size}(e) = \lfloor (\log_c \|e\|)/k \rfloor$. By definition, if $\text{index}(e) = i$ and $\text{size}(e) = j$, then $c^{kj+i} \leq \|e\| < c^{kj+i+1}$. We similarly define the index and the size for any pair (u, v) of vertices that are not necessarily connected in the spanner: $\text{index}(u, v) = \lfloor \log_c \|uv\| \rfloor \bmod k$, and $\text{size}(u, v) = \lfloor (\log_c \|uv\|)/k \rfloor$.

Invariants. In order to construct a light-weight spanner, we start from our sparse dynamic spanner construction. To distinguish the edges of our light spanner with the edges of our sparse spanner, we call the edges of our sparse spanner the *potential pairs*, since a carefully filtered set of those edges will make up our light-weight spanner. After bucketing the potential pairs, since we maintain the edges of each bucket separately, we must find per-bucket criteria that guarantee the the main properties we expect from our spanner: the stretch-factor and the lightness. We call these criteria the *invariants*. To make sure the union of the buckets meets the stretch bound, we generalize the notion of stretch factor to work on individual buckets and we call it Invariant 1.

- **Invariant 1.** For each pair of vertices $(u, v) \notin S_i$ with index i, there must exist a set of edges $e_1 = (x_1, y_1), e_2 = (x_2, y_2), \ldots, e_l = (x_l, y_l)$ in S_i such that

$$\sum_{i=1}^{l} \|e_i\| + (1+\varepsilon) \left(\|ux_1\| + \sum_{i=1}^{l-1} \|y_i x_{i+1}\| + \|y_l v\| \right) < (1+\varepsilon)\|uv\|.$$

  In other words, u must reach v by a path of cost at most $(1+\varepsilon)\|uv\|$ where the cost of every edge $e \in S_i$ is $\|e\|$ and the cost of every edge $e \notin S_i$ is $(1+\varepsilon)\|e\|$.

Lemma 4.1. *If Invariant 1 holds for all S_i, then $S = \bigcup_{i=0}^{k-1} S_i$ is a $(1+\varepsilon)$-spanner.*

Proof. Let (u, v) be a pair of vertices. We find a $(1 + \varepsilon)$-path between u and v using edges in S. Let $i = \text{index}(u, v)$. By Invariant 1 there exists a set of edges $e_1 = (x_1, y_1), e_2 = (x_2, y_2), \ldots, e_l = (x_l, y_l)$ in S_i such that

$$\sum_{i=1}^{l} \|e_i\| + (1 + \varepsilon) \left(\|ux_1\| + \sum_{i=1}^{l-1} \|y_i x_{i+1}\| + \|y_l v\| \right) < (1 + \varepsilon)\|uv\|.$$

Consider the path $P = ux_1y_1x_2y_2 \cdots x_ly_lv$ between u and v. We call this path the *replacement path* for (u, v). The edges $x_1y_1, x_2y_2, \ldots, x_ly_l$ are present in S_i (and therefore present in S) but the other edges of the replacement path are missing from S_i. A similar procedure can be performed on the missing pairs recursively to find and replace them with their corresponding replacement paths. This recursive procedure yields a $(1+\varepsilon)$-path for (u, v) and it terminates because the length of each missing edge in a replacement path is smaller than the length of the edge that is being replaced (otherwise Invariant 1 would not hold). □

Furthermore, we bound the weight of the spanner by ensuring the second invariant, which is the leapfrog property on S_i. [32]

- **Invariant 2.** Let $(u, v) \in S_i$. For every subset of edges $e_1 = (x_1, y_1), e_2 = (x_2, y_2), \ldots, e_l = (x_l, y_l)$ in S_i the inequality

  $$\sum_{i=1}^{l} \|e_i\| + (1 + \varepsilon) \left(\|ux_1\| + \sum_{i=1}^{l-1} \|y_i x_{i+1}\| + \|y_l v\| \right) > (1 + \varepsilon')\|uv\|$$

  holds, where $\varepsilon' < \varepsilon$ is a positive constant. In other words, u should not be able to reach v by a (short) path of cost $(1 + \varepsilon')\|uv\|$, where the edge costs are the same as in Invariant 1.

The leapfrog property leads to a constant upper bound on the lightness of S_i, for each $0 \leq i < k$. And since the weight of the minimum spanning tree on the end-points of each

S_i is at most a constant factor of the weight of the minimum spanning tree on the whole point set, this implies a constant upper bound on the lightness of the spanner $S = \bigcup_{i=0}^{k-1} S_i$. As well as the weight bound, we prove, in the following lemma, that Invariant 2 implies a similar result to the packing lemma, but for the number of edges on the same level.

Lemma 4.2 (Edge packing)**.** *Let E be a set of edges (segments) with the same index and the same level that is consistent with Invariant 2. Also, assume that E is contained in a ball of radius R, and the minimum edge size in E is r. Then*

$$|E| < C_1 (R/r)^{2d}$$

where $C_1 = (2(1+\varepsilon)/\varepsilon')^{2d} d^d$ is a constant.

Proof. A simple observation is that for any two segments (u, v) and (y, z) in E we must have

$$\max(\|uy\|, \|vz\|) > \frac{\varepsilon'}{2(1+\varepsilon)} \cdot r$$

because otherwise, assuming that $\|uv\| \geq \|yz\|$, for the pair (u, v) and the sequence $e_1 = (y, z)$, the left hand side of the inequality in Invariant 2 would be at most

$$2(1+\varepsilon) \cdot \frac{\varepsilon'}{2(1+\varepsilon)} \cdot r + \|yz\| \leq (1+\varepsilon')\|uv\|$$

contradicting the fact that E is consistent with Invariant 2. Thus, given a covering of a ball of radius R with M balls of radius $r' = \frac{\varepsilon'}{2(1+\varepsilon)} \cdot r$, every segment in E has its endpoints in a unique pair of balls, otherwise Invariant 2 will be compromised. Hence, $|E| \leq M^2$. A simple calculation yields a covering with $M < (2(1+\varepsilon)/\varepsilon')^d d^{d/2} (R/r)^d$ balls. □

We can simplify the two invariants by defining a distance function d_i^* over the pairs of vertices,

Definition 4.3. Let S_i^* be a complete weighted graph over the vertices such that the weight of an edge e in S_i^* is defined as

$$w(e) = \begin{cases} \|e\| & \text{if } e \in S_i \\ (1+\varepsilon)\|e\| & \text{if } e \notin S_i \end{cases}$$

We define an *extended path* between u and v in S_i as a path between u and v in S_i^* that only uses edges (y, z) where $\text{size}(y, z) < \text{size}(u, v)$. We also define the length of an extended path as the sum of its edge weights in S_i^*. Finally, we define $d_i^*(u, v)$ as the length of the shortest extended path between u and v.

Using this new distance function we can rephrase the two invariants as follows.

- **Invariant 1.** For every pair $(u, v) \notin S_i$ with $\text{index}(u, v) = i$, we have $d_i^*(u, v) < (1+\varepsilon)d(u, v)$.
- **Invariant 2.** For every pair $(u, v) \in S_i$, we have $d_i^*(u, v) > (1+\varepsilon')d(u, v)$.

It is worth noting that these forms are not exactly equivalent to the previous forms, as we are only considering paths of lower level edges in the definition of d_i^*, while a short path in the spanner could potentially contain an edge of the same level. This provides a stronger variation of Invariant 1, which still implies a $1+\varepsilon$ stretch for the spanner. However, this change weakens Invariant 2. But as we will see, a careful addition of the same-level edges can prevent any possible violations of Invariant 2 that could be caused by this new form.

Maintaining the invariants. The quality of our light-weight dynamic spanner depends on the two invariants we introduced above, and an update like a point insertion or removal could cause one of them to break, if not both. Therefore, we establish a procedure that addresses the inconsistencies and enforces the invariants to hold at all times.

The procedure for fixing a violation of Invariant 1 is straightforward: as long as there exists a pair (u, v) that violates Invariant 1 for its corresponding subset S_i, add an appropriate potential pair to S_i that connects an ancestor of u to an ancestor of v in the hierarchy $\mathcal{T}$. This resolves the inconsistency for (u, v) if the ancestors are chosen properly, but it might cause other pairs to violate Invariant 2 because of this edge addition. We will prove that if certain criteria are met, there would be no side effect on the same-level pairs and the addition can only result in a constant amortized number of inflicted updates on higher level pairs.

Fixing a violation of Invariant 2, on the other hand, is more tricky. After we remove the violating edge (u, v) from its subset S_i, the effect on higher level pairs would be similar to the previous case, but removing (u, v) might cause multiple updates on the same level, which in turn cascade to higher levels. We therefore analyze the removal of (u, v) together with the subsequent additions of same-level edges that aim to fix the incurred violations of Invariant 1, and we prove that a constant amortized bound on the number of inflicted updates on higher level pairs would still hold. We get to the details of our maintenance updates in section 4.5.3.

Amortized analysis. We analyze the effects of an update (edge addition and removal) on higher level pairs using a potential function, for each S_i separately. We define our potential function over the potential pairs in S_i. The change in the potential function shows how much a pair is close to violating one of the invariants. The higher the potential, the closer the pair is to violating the invariants. This enables us to assign a certain amount of credit to each update, that can be used to pay for the potential change of the updated pair and the affected pairs, which in turn results on an amortized upper bound on the number of edge updates in the future. Therefore, for a potential pair (u, v) with index i and following an update in S_i,

- if $(u, v) \in S_i$ and $d_i^*(u, v)$ decreases, or
- if $(u, v) \notin S_i$, and $d_i^*(u, v)$ increases,

we increase the potential of the pair (u, v) to account for its future violation of the invariants. More specifically, we define the potential function $p_i(u, v)$ of a potential pair (u, v) in S_i as

$$p_i(u,v) = \begin{cases} (1+\varepsilon) - d_i^*(u,v)/d(u,v) & \text{if } (u,v) \in S_i \\ C_\phi \cdot (d_i^*(u,v)/d(u,v) - (1+\varepsilon')) & \text{if } (u,v) \notin S_i \text{ and } \mathrm{index}(u,v) = i \end{cases}$$

where $C_\phi > 1$ is a positive constant coefficient that we specify later. This implies that if $p_i(u, v) < \varepsilon - \varepsilon'$, then both invariants would hold for the pair (u, v) (in S_i). Based on this observation, we define a potential function on S_i in the following way,

$$\Phi_i = \sum_{(u,v) \in \mathcal{P}_i \cup S_i} p_i(u,v)$$

where $\mathcal{P}_i$ is the set of potential pairs with index i. We simply define the potential of the whole spanner as

$$\Phi = \sum_i \Phi_i$$

We add another term to this potential function later in section 4.5 to account for future edges between the existing nodes.

$$\Phi^* = \Phi + \frac{p_{max}}{2} \cdot \sum_{i=1}^{n} (D_{max} - \deg_{\mathcal{S}_1}(v_i))$$

We first prove some bounds on Φ but we ultimately use the adjusted potential function Φ^* to prove our amortized bounds on the number of updates. In the remainder of this chapter, we specify our sparse and light-weight construction in more details, and we will provide our bounds on the recourse in each case separately.

4.4 Sparse spanner

In this section, we introduce our dynamic construction for a sparse spanner with constant amortized recourse per point insertion and $\mathcal{O}(\log \Delta)$ recourse per point deletion. We build our spanner on top of a hierarchical clustering that we design early in this section.

Krauthgamer and Lee [69] showed how to maintain such hierarchical structures in $\mathcal{O}(\log \Delta)$ update time by maintaining ε-nets. However, this hierarchy is not directly applicable to our case since a point can appear $\log \Delta$ times on its path to root, which would imply a $\mathcal{O}(\log \Delta)$ bound on the degree of the spanner instead of a constant bound. Cloe and Gottlieb [26] improved the update time of this hierarchy to $\mathcal{O}(\log n)$. Gottlieb and Roditty [61] later introduced a new hierarchical construction with the same update time for their fully-dynamic spanner, which also satisfied an extra close-containment property. Here, we introduce a simpler hierarchy that suits our needs and does not require the close-containment property. Our hierarchy performs constant cluster updates for a point insertion and $\mathcal{O}(\log \Delta)$ cluster updates for a point deletion.

Our hierarchy consists of a pair $(\mathcal{T}, R)$ where $\mathcal{T}$ is a rooted tree of clusters and $R > 0$ is a constant. Every cluster $\mathcal{C} \in \mathcal{T}$ is associated with a center $c(\mathcal{C}) \in V$ and a level $l(\mathcal{C}) \in \mathbb{Z}$. The level of a cluster specifies its radius; $\mathcal{C}$ *covers* a ball of radius $R^{l(\mathcal{C})}$ around $c(\mathcal{C})$. We denote the *parent* of $\mathcal{C}$ in $\mathcal{T}$ by $p(\mathcal{C})$. The root of $\mathcal{T}$, denoted by $\mathcal{T}.root$, is the only cluster without a parent. Furthermore, the level of a parent is one more than of the child, i.e. $l(p(\mathcal{C})) = 1 + l(\mathcal{C})$, for all $\mathcal{C} \in \mathcal{T}$ except the root. A parent must cover the centers of its children.

Besides these basic characteristics, we require our hierarchy to satisfy the separation property at all times. This property states that the clusters at the same level are separated by a distance proportional to their radii,

Definition 4.4 (Separation property)**.** For any pair of same-level clusters $\mathcal{C}_1, \mathcal{C}_2 \in \mathcal{T}$ on level j,

$$d(c(\mathcal{C}_1), c(\mathcal{C}_2)) > R^j$$

Each point at the time of insertion creates a single cluster centered at the inserted point, and during the future insertions, might have multiple clusters with different radii centered at it. In fact, each point could have clusters centered at it in at most $\mathcal{O}(\log \Delta)$ levels. At the time of deletion, any cluster that is centered at the deleted point will be removed.

Our clusters are of two types: explicit clusters and implicit clusters. Explicit clusters are the ones we create manually during our maintenance steps. Implicit clusters are the lower level copies of the explicit clusters that exist in the hierarchy even though we do not create them manually. Therefore, if a cluster $\mathcal{C} = (p, l)$ is created in the hierarchy at some point, we implicitly assume clusters (p, i) for $i < l$ exist in the hierarchy after this insertion, and they are included in their corresponding $\mathcal{T}_i$ as well. We maintain the separation property between all clusters, including the implicit ones. We use these implicit clusters for constructing our spanner.

4.4.1 Maintaining the hierarchy

We initially start from an empty tree $\mathcal{T}$ and a constant R that we specify later.

Point insertion. Let $\mathcal{T}_i$ be the set of clusters with level i, i.e. $\mathcal{T}_{\text{size}(\mathcal{T}.root)}$ only contains the root, $\mathcal{T}_{\text{size}(\mathcal{T}.root)-1}$ contains root's children, etc. Upon the insertion of a point p, we look for the lowest level (between explicit clusters) i that p is covered in $\mathcal{T}_i$. We insert $\mathcal{C} = (p, i-1)$ into the hierarchy. Since p is covered in T_i, we can find a cluster $\mathcal{C}' = (p, i)$ that covers p and assign it as the parent of $\mathcal{C}$ (algorithm 10).

In the case that p is not covered in any of the levels in $\mathcal{T}$, which we handle by replicating the root cluster from above until it covers the new point, then the insertion happens the same way as before.

Algorithm 10 Inserting a point to the hierarchy.

```
procedure INSERT-TO-HIERARCHY($\mathcal{T}$, $R$, $p$)
    if $|\mathcal{T}| = 0$ then
        Add a root cluster $\mathcal{C} = (p, 0)$ to $\mathcal{T}$.
        return $\mathcal{C}$
    Let $i$ be the lowest level in $\mathcal{T}$.
    while $\mathcal{T}_i$ does not cover $p$ do
        Increase $i$ by 1.
        if $i >$ size $\mathcal{T}.root$ then
            Create a new cluster $\mathcal{C} = (\mathcal{T}.root, \text{size}(\mathcal{T}.root) + 1)$.
            Make $\mathcal{C}$ the new root of the hierarchy.
            The old root becomes a child of $\mathcal{C}$.
    Let $\mathcal{C}'$ be a cluster in $\mathcal{T}_i$ that covers $p$.
    Create a cluster $\mathcal{C} = (p, \text{size}(\mathcal{C}') - 1)$ and add it as a child of $\mathcal{C}'$.
```

The basic characteristics of the hierarchy hold after an insertion. We now show that the separation property holds after the insertion of a new cluster $\mathcal{C} = (p, l)$. Assume, on the contrary, that there exists a cluster $\mathcal{C}' = (q, l)$ that $(\mathcal{C}, \mathcal{C}')$ violates the separation property. $\mathcal{C}$ is inserted on level l, thus p is not covered by $\mathcal{T}_l$. According to the assumption, $d(q, p) \leq R^l$, meaning that $\mathcal{C}'$ covers p. This contradicts the fact that $\mathcal{T}_l$ does not cover p since $\mathcal{C}' \in \mathcal{T}_l$. A similar argument shows that the separation property holds for the implicit copies of $\mathcal{C}$ as well.

Point deletion. Upon the deletion of a point p, we remove all the clusters centered at p in the hierarchy. The clusters centered at p create a chain in $\mathcal{T}$ that starts from the lowest level explicit copy of p and ends at the highest level copy. We remove this chain level by level, starting from the lowest level cluster $\mathcal{C} = (p, l)$ that is centered at p. Upon the removal of $\mathcal{C}$, we loop over children of $\mathcal{C}$ one by one, and we try to assign them to a new parent. If we find a cluster on level $l+1$ that covers them, then we assign them to that cluster, otherwise we replicate them on one level higher and we continue the process with the remaining children. After we are done with (p, l), we repeat the same process with $(p, l+1)$, until no copies of p exist in the hierarchy (algorithm 11).

Algorithm 11 Deleting a point from the hierarchy.

```
1: procedure DELETE-FROM-HIERARCHY($\mathcal{T}$, $R$, $p$)
2:     Let $\mathcal{C} = (p, l)$ be the lowest level (explicit) cluster centered at $p$.
3:     Delete $\mathcal{C}$ from $\mathcal{T}$ and mark its children.
4:     while there exists a marked cluster on level $l-1$ do
5:         Let $\mathcal{C}' = (q, l-1)$ be a marked cluster.
6:         Find a cluster $\mathcal{C}''$ on level $l$ that covers $q$.
7:         if such cluster exists then
8:             Assign $\mathcal{C}''$ as the parent of $\mathcal{C}'$ and unmark $\mathcal{C}'$.
9:         else
10:            Create $\mathcal{C}'' = (q, l)$ and make it the parent of $\mathcal{C}'$.
11:            Mark $\mathcal{C}''$ and unmark $\mathcal{C}'$.
12:    if there still exists a marked cluster in $\mathcal{T}$ then
13:        Increase $l$ by one and repeat the while loop above.
```

Again, the basic characteristics of the hierarchy hold after a deletion. We need to show that the separation property still holds. Immediately after removing the cluster (p, l) the separation property obviously holds. After re-assigning a marked child to another parent the property still holds since no cluster has changed in terms of their center or level. If a marked child is replicated on level $l+1$, it means that there was no cluster covering it on this level, otherwise it would have been assigned as its new parent. Therefore, the separation property holds after the replication on level $l+1$. We will prove more properties of our hierarchy later on when we define the spanner.

4.4.2 The initial spanner

Our initial spanner is a sparse spanner that is defined on the hierarchy $\mathcal{T}$ and it has bounded cluster degree but not bounded point degree. The reason that a bounded degree on the clusters would not imply a bounded degree on the point set is that every point could have multiple clusters centered at it, each of which have a constant number of edges connected to them. This would cause the degree of the point to get as large as $\Omega(\log \Delta)$. Later we will fix this issue by assigning edges connected to large degree points to other vertices.

The initial spanner consists of two types of edges. The first type that we already mentioned before, is the edges that go between clusters of the same level. These edges guarantee a short path between the descendants of the two clusters, similar to a spanner built on a well-separated pair decomposition. And the second type is the parent-child edges, that connect every node to its children. The edge weight between two clusters is the same as the distance between their centers.

We define the spanner formally as follows,

Definition 4.5 (Initial spanner)**.** Let $(\mathcal{T}, R)$ be a hierarchy that satisfies the separation property. We define our sparse spanner $\mathcal{S}_0$ to be the graph on the nodes of $\mathcal{T}$ that contains the following edges,

- Type I. Any pair of centers p and q whose clusters are located on the same level and $d(p,q) \leq \lambda \cdot R^l$ are connected together. Here, λ is a fixed constant.
- Type II. Any cluster center in $\mathcal{T}$ is connected to the centers of its children in $\mathcal{T}$.

Note that the implicit clusters are also included in this definition. Meaning that if two implicit same-level clusters are close to each other then there would be an edge of type I between them. We show that the spanner $\mathcal{S}_0$ has a bounded stretch.

Lemma 4.6 (Stretch-factor)**.** *For large enough $\lambda = \mathcal{O}(\varepsilon^{-1})$ the stretch-factor of $\mathcal{S}_0$ would be bounded from above by $1 + \varepsilon$.*

Proof. Let p and q be two points in the point set, and also let $\mathcal{C} = (p, l)$ and $\mathcal{C}' = (q, l')$

be the highest level clusters in $\mathcal{T}$ that are centered at p and q, respectively. By symmetry, assume $l \geq l'$. If $d(p,q) \leq \lambda \cdot R^{l'}$, then there is an edge between the (possibly implicit) cluster (p, l') and $\mathcal{C}'$. This edge connects p and q together, therefore the stretch would be equal to 1 for this pair. If $d(p,q) > \lambda \cdot R^{l'}$, we perform an iterative search for such shortcut edge. Start with $\mathcal{C} = (p, l')$ and $\mathcal{C}' = (q, l')$ and every time that the inequality $d(p,q) \leq \lambda \cdot R^{l'}$ is not satisfied set $\mathcal{C}$ and $\mathcal{C}'$ to their parents and set $l' = l' + 1$ and check for the inequality again. We show that the inequality eventually will be satisfied. Let p_i and q_i be the centers of $\mathcal{C}$ and $\mathcal{C}'$ on the i-th iteration of this iterative process ($i = 1, 2, \ldots$), and let l' have its initial value before any increments. We have $d(p_{i+1}, p_i) \leq R^{l'+i}$ and $d(q_{i+1}, q_i) \leq R^{l'+i}$. By the triangle inequality,

$$d(p_{i+1}, q_{i+1}) \leq d(p_{i+1}, p_i) + d(p_i, q_i) + d(q_{i+1}, q_i) \leq 2 \cdot R^{l'+i} + d(p_i, q_i)$$

Denote the ratio $d(p_i, q_i)/R^{l'+i-1}$ by x_i. We have,

$$x_{i+1} \leq 2 + \frac{x_i}{R}$$

Therefore, x_i is roughly being divided by R on every iteration and it stops when $x_i \leq \lambda$. We can easily see that the loop terminates and the value of x_i after the termination would be greater than λ/R. This particularly shows that the edge between $\mathcal{C}$ and $\mathcal{C}'$ is a long shortcut edge when λ is chosen large enough, since its length is more than λ/R times the radius of the centers it is connecting.

Now we show that this shortcut edge would be good enough to provide the $1 + \varepsilon$ stretch factor for the initial points, p and q. Note that because of the parent-child edges, p can find a path to q by traversing p_is in the proper order and using edge between p_i and q_i and traversing back to q. We show that the portion of the path from p to p_i (and similarly from q to q_i) is at most $\frac{R^{l'+i}-1}{R-1}$. We prove it only for p, the argument for q is similar. Note that if the termination level $l' + i \leq l$ then $p_i = p$ and this path length from p to p_i would be 0, confirming our claim for p. Therefore, we assume the termination level is above the level of p. The length of the path from p to p_i that only uses type II edges would be at most

$$R^{l+1} + \cdots + R^{l'+i} < \frac{R^{l'+i+1} - 1}{R - 1}$$

Thus the length of the path from p to q would be at most

$$2 \cdot \frac{R^{l'+i+1} - 1}{R - 1} + d(p_i, q_i)$$

On the other hand, by the triangle inequality,

$$d(p, q) \geq d(p_i, q_i) - 2 \cdot \frac{R^{l'+i+1} - 1}{R - 1}$$

Finally, the stretch-factor of this path would be at most

$$\frac{2 \cdot \frac{R^{l'+i+1}-1}{R-1} + d(p_i, q_i)}{d(p_i, q_i) - 2 \cdot \frac{R^{l'+i+1}-1}{R-1}}$$

A simple calculation yields that this fraction is less than $1 + \varepsilon$ when $\lambda = 2(2 + \varepsilon)\varepsilon^{-1}R = \mathcal{O}(\varepsilon^{-1})$. $\square$

Next, we show that the degree of every cluster in $\mathcal{S}_0$ is bounded by a constant. Note that this does not imply a bounded degree on every point, since a point could be the center of many clusters.

Lemma 4.7 (Degree bound)**.** *The degree of every cluster in $\mathcal{S}_0$ is bounded by $\mathcal{O}(\varepsilon^{-d})$.*

Proof. We first prove that the type I degree of every cluster $\mathcal{C} = (p, l)$ is bounded by a constant. Let $\mathcal{C}' = (q, l)$ be a cluster that has a type I edge to $\mathcal{C}$. This means that $d(p, q) \leq$

$\lambda \cdot R^l$. By the separation property, $d(p,q) > R^l$. Thus, by the packing lemma there are at most

$$d^{d/2}\lambda^d = \mathcal{O}(\varepsilon^{-d})$$

type I edges connected to $\mathcal{C}$. The last bound comes from the fact that a choice of $\lambda = \mathcal{O}(\varepsilon^{-1})$ would be enough to have a bounded stretch.

Now we only need to show that the parent-child edges also add at most a constant degree to every cluster, which is again achieved by the packing lemma. Because the children of this cluster are located in a ball of radius R^l around its center, p, and they are also pair-wise separated by a distance of at least R^{l-1}, we can conclude that the number of children of $\mathcal{C}$ would be upper bounded by $d^{d/2}R^d = \mathcal{O}(1)$. □

Representative assignment. So far we showed how to build a spanner that has a bounded degree on each cluster and the desired stretch-factor of $1+\varepsilon$. But this spanner does not have a degree bound on the actual point set and that is a property we are looking for. Here, we show how to reduce the load on high degree points and distribute the edges more evenly so that the bounded degree property holds for the point set as well.

The basic idea is that for every cluster $\mathcal{C}$ in the hierarchy, we pick one of lower level clusters, say $\mathcal{C}'$, to be its *representative* and play its role in the final spanner, meaning that all the spanner edges connecting $\mathcal{C}$ to other clusters will now connect $\mathcal{C}'$ to those clusters after the re-assignment. This re-assignment will be done for every cluster in the hierarchy until every cluster has a representative. Only then we can be certain that the spanner has a bounded degree on the current point set. Since by lemma 4.7 the degree of every center is bounded by a constant, we only need to make sure that every point is representing at most a constant number of clusters in the hierarchy.

First, we define the level of a point p, denoted by size(p) to be the level of the highest level cluster that has p as its center, i.e. $\text{size}(p) = \max_{(p,l)\in\mathcal{T}} l$.

Definition 4.8 (Representative assignment)**.** Let $\mathcal{T}$ be a hierarchy. We define the representative assignment of $\mathcal{T}$ to be a function $\mathcal{L}$ that maps every cluster $\mathcal{C} = (p, l)$ of $\mathcal{T}$ to a point q in the point set such that $l \geq \text{size}(q)$ and $d(p, q) \leq R^l$. We say $\mathcal{L}$ has bounded repetition b if $|\mathcal{L}^{-1}(q)| \leq b$ for every point q.

Connecting the edges between the representatives instead of the actual centers would give us our bounded-degree spanner.

Definition 4.9 (Bounded-degree spanner)**.** Define the spanner $\mathcal{S}_1$ to be the spanner connecting the pair $(\mathcal{L}(\mathcal{C}), \mathcal{L}(\mathcal{C}'))$ for every edge $(\mathcal{C}, \mathcal{C}') \in \mathcal{S}_0$.

Now we show that this re-assignment of the edges would not affect the stretch-factor and the degree bound significantly if the clusters are small enough, or equivalently, λ is chosen large enough.

Lemma 4.10 (Stretch-factor)**.** *For large enough $\lambda = \mathcal{O}(\varepsilon^{-1})$ and any representative assignment $\mathcal{L}$ the stretch-factor of $\mathcal{S}_1$ would be bounded from above by $1 + \varepsilon$.*

Proof. The proof works in a similar way to the proof of lemma 4.6. A shortcut edge would still provide a good path between two clusters even after its end points are replaced by their representatives. The path from a p to p_i will be doubled at most since a representative could be as far as a child from the center of a cluster. Therefore, the stretch-factor of the path between p and q will be

$$\frac{4 \cdot \frac{R^{l'+i+1}-1}{R-1} + d(p_i, q_i)}{d(p_i, q_i) - 4 \cdot \frac{R^{l'+i+1}-1}{R-1}}$$

Again, this fraction is less than $1 + \varepsilon$ when $\lambda = 4(2 + \varepsilon)\varepsilon^{-1}R = \mathcal{O}(\varepsilon^{-1})$. □

To construct a bounded-repetition representative assignment we pay attention to the neighbors of lower level copies of a cluster. Let $\mathcal{C} = (p, l)$ be a cluster that we want to find a representative for. As we mentioned before, (p, l') exists in the hierarchy for all $l' < l$. If l' is small enough, i.e. $l' < l - \log_R \lambda$, then the neighbors of (p, l') will be located within a distance $\lambda \cdot R^{l'} = R^l$ of p, making them good candidates to be a representative of $\mathcal{C}$. Therefore, having more neighbors on lower levels means having more (potential) representatives on higher levels. This is how we assign the representatives.

We define a chain to be a sequence of clusters with the same center that form a path in $\mathcal{T}$. We divide a chain into blocks of length $\log_R \lambda$. The best way to do this so that maintaining it dynamically is easy is to index the clusters in a chain according to their levels and gather the same indices in the same block. We define the block index of a cluster in a chain to be $\lfloor l / \log_R \lambda \rfloor$, where l is the level of the cluster. The clusters in a chain that have the same index form a block.

The first observation is that if we are given two non-consecutive blocks in the same chain, we can use the neighbors of the lower level block as representatives of the higher level block. This is the key idea to our representative assignment, which we call *next block assignment*. In this assignment, we aim to represent higher level points with lower level points. Let p be a point and $P_1, P_2, \ldots, P_k$ be all the blocks of the chain that is centered at p in $\mathcal{T}$, ordered from top to bottom (higher level blocks to lower level blocks). We say a block is empty if the clusters in the block have no neighbors in $\mathcal{T}$. We say the block is non-empty otherwise. We make a linked list $\mathcal{L}_0$ of all the even indexed non-empty blocks, and a separate linked list $\mathcal{L}_1$ for all the odd indexed non-empty blocks. For every element of $\mathcal{L}_0$ we pick an arbitrary neighbor cluster of its block in $\mathcal{L}_0$ (because the blocks are non-empty such neighbors exists), and we assign that neighbor to be the representative of the clusters in that element. More specifically, let B_i be a block in $\mathcal{L}_0$, and let B_{i+1} be the next block in $\mathcal{L}_0$. Let $\mathcal{C}$ be an arbitrary cluster in B_{i+1} that has a neighbor. This cluster exists, since B_{i+1} is a non-empty block. Let q be the center of a neighbor of $\mathcal{C}$. We assign $\mathcal{L}(\mathcal{C}') = q$ for all $\mathcal{C}' \in B_i$. The same approach works for $\mathcal{L}_1$. This assigns a representative to every block in the chain, except the last block in $\mathcal{L}_0$ and $\mathcal{L}_1$. We assign p itself to be the representative of the clusters in these blocks.

Now we show that this assignment has bounded repetition. First, we show that our assignment only assigns lower level points to be representatives of higher level points.

Lemma 4.11. *Let p and q be two points in the point set and let* $\text{size}(p) > \text{size}(q)$ *. In the next block assignment q would never be represented by p.*

Proof. Assume, on the contrary, that q is represented by p. Therefore, there exists two same-parity cluster blocks in the chain centered at q that a cluster centered at p is connected to the lower block. Let $\mathcal{C} = (p, l)$ and $\mathcal{C}' = (q, l')$ be the highest clusters centered at p and q, respectively. Since the connection between p and q is happening somewhere on the third block or lower on the chain centered at q, we can say that $d(p, q) < \lambda \cdot R^{l' - \log_R \lambda} = R^{l'}$. This means that the separation property does not hold for the lower level copy of $\mathcal{C}$, (p, l'), and $\mathcal{C}'$, which is a contradiction. □

Now that we proved that points can only represent higher level points in our assignment, we can show the bounded repetition property.

Lemma 4.12 (Bounded repetition)**.** *The next block assignment $\mathcal{L}$ described above has bounded repetition.*

Proof. We show that every point represents at most a constant number of clusters. First, note that the two bottom clusters of the two block linked lists have a constant number of clusters in them (to be exact, $2 \log_R \lambda$ clusters maximum). So we just need to show that the number of other clusters that are from other chains and assigned to the point are bounded by a constant. Let p be an arbitrary point and let $\mathcal{C} = (p, l)$ be the highest level cluster centered at p. According to the previous lemma, any point q that has a cluster $\mathcal{C}' = (q, l')$ that $\mathcal{L}(\mathcal{C}') = p$ must have a higher level than p. Therefore, there exists a lower level copy of q on level l. Also, the distance between p and q is bounded by $\lambda \cdot R^l$ since p and q are connected on a level no higher than l (remember that we only represent our clusters with their lower level neighbors). Now we can use the packing lemma, since all such points q have a cluster centered at them on level l and therefore separated by a distance of R^l. By the packing lemma, the number of such clusters would be bounded by $d^{d/2}\lambda^d$ such points. So the repetition is at most $b = d^{d/2}\lambda^d + 2$. □

Corollary 4.13. *The spanner $\mathcal{S}_1$ has bounded degree.*

4.4.3 Maintaining the spanner

So far we showed $\mathcal{S}_1$ has bounded stretch and bounded degree. Here we show that we can maintain $\mathcal{S}_1$ with $\mathcal{O}(1)$ amortized number of updates after a point insertion and $\mathcal{O}(\log \Delta)$ amortized number of updates after a point deletion. We know how to maintain the hierarchy from earlier in this section. Therefore, we just explain how to update the spanner, which includes maintaining our representative assignments dynamically.

Point insertion. We prove the amortized bound by assigning credits to each node, and using the credit in the future in the case of an expensive operation. Let D_{max} be the degree bound we proved for $\mathcal{S}_1$. When a new point is added to the spanner, we assign D_{max} credits to it.

We analyze the edge addition and removals that happen after the insertion of a point p in the spanner. Note that although only one explicit cluster is added to $\mathcal{T}$ after the insertion, there might be many new edges between the implicit (lower level) copies of the new cluster and other clusters that existed in $\mathcal{T}$ beforehand. We need to show that these new edges do not cause a lot of changes on the spanner after the representative assignment phase.

First, we analyze the effect of addition of p on points q that size$(p) >$ size(q). Similar to the proof of lemma 4.11, we can show that any edges between the chain centered at p and the chain centered at q will be connected to the top two cluster blocks of the chain centered at q. This means that these edges will have no effect on the assignment of other clusters in the chain centered at q, because each non-empty block is represented by some neighbor of the next non-empty same-parity block, and the first two blocks, whether they are empty or not, will not have any effect on the rest of the assignment. Therefore, no changes will occur on the representatives of q and therefore the edges that connect these representatives together will remain unchanged.

The addition of p as we mentioned, would cause the addition of some edges in the spanner $\mathcal{S}_1$, that we pay for using the constant amount of credit stored on the endpoints of those edges. Therefore, we are not spending more than constant amount of amortized update for this case.

Second, we analyze the effect of addition of p on points q that size$(p) \leq \text{size}\, q$. The outcome is different in this case. Similar to the previous case we can argue that any edge between the chain centered at p and the chain centered at q must be connected to the top two blocks of the chain centered at p, but they could be connected to anywhere relative to the highest cluster centered at q. This means that they could add a non-empty block in the middle of the chain centered at q. If this happens, then the assignment of the previous non-empty same-parity block changes and also the new non-empty block will have its own assignment. This translates into a constant number of changes (edge additions and removals) on the spanner $\mathcal{S}_1$ per such point q. We earlier in lemma 4.12 proved that there is at most a constant number of such clusters. This shows that there would be at most a constant number of changes on the spanner $\mathcal{S}_1$ from higher level points.

Finally, we can conclude that overall the amortized recourse for insertion is bounded by a constant, since in the first case we could pay for the changes using the existing credits, and in the second case we could pay for the changes from our pocket.

Point deletion. After a point deletion, all the clusters centered at that point will be removed from the hierarchy, and a set of replication to higher levels would happen to some clusters to fix the hierarchy after the removal. It is easy to see that the number of cluster changes (including removal and replication) would be bounded by a constant. Each cluster change would also cause a constant number of changes on the edges of the spanner $\mathcal{S}_0$. Note that a cluster removal can introduce an empty block to at most a constant number of higher level points and a cluster replication can also introduce an empty block to at most a constant number of higher level points. Therefore, the changes on the representative assignments would be bounded by a constant after a single cluster update. Since we have at most $\mathcal{O}(\log \Delta)$ levels in the hierarchy, each of which having at most a constant number of cluster updates, overall we would have at most $\mathcal{O}(\log \Delta)$ number of edge changes on $\mathcal{S}_1$. After the removal, we assign full D_{max} credit to any node that is impacted by the removal. This would make sure we have enough credits for the future additions.

4.5 Light spanner

So far we introduced our hierarchy and how to maintain it under point insertions and removals, and also how to create a spanner on top of the hierarchy and how to make it sparse with representative assignments. We also studied how our sparse spanner changes under point insertions and point removals and we bounded the amortized number of updates per insertion to a constant, and the bound for the number of updates per deletion to $\mathcal{O}(\log \Delta)$.

In this section, we introduce our techniques for maintaining a light spanner that has a constant lightness bound on top of all the properties we had so far. In our main result in this section we show that maintaining the lightness in our case is not particularly harder than maintaining the sparsity, meaning that it would not require asymptotically more changes than a sparse spanner would.

We ultimately want to select a subset of the edges of our sparse spanner that are light and preserve the bounded stretch to achieve a bounded degree light spanner. For this purpose, we introduce a set of maintenance updates that we perform after point insertion and removal. These maintenance updates aim to improve the weight of the spanner in iterations. In each iteration, we look at all the edge buckets of our spanner, and we search for an edge that does not satisfy the leapfrog property for certain constants. If no such edge is found in the buckets then the spanner's lightness is already bounded by a constant. If found, such an edge would be deleted from the bucket and removed from the spanner. The removal of this edge could cause the bounded stretch property to not hold for some other pairs. We take one such pair and we add the edge between the two points. Now the addition of the new edge could cause the appearance of some pairs that violate the leapfrog property and therefore increase the lightness. We then repeat this loop of removal and addition again until we reach a bounded lightness bounded stretch spanner. We show in this section that the number of iterations we need to reach a bounded degree bounded stretch spanner is proportional to the number of edges that we changed since our last bounded degree bounded stretch state. Therefore, if we only change an amortized constant number of edges to reach a state, then an amortized constant number of updates would be enough to make that state stable again.

We first analyze the effect of point insertion or deletion on the potential functions we defined earlier in section 4.3. Then we introduce our maintenance updates and we show our bounds on the recourse of a light spanner.

4.5.1 Bounding the potential function

In this section we analyze the behavior of our potential functions, after a point insertion and a point deletion. These bounds will later help us prove the amortized bounds on the recourse. As we defined in section 4.3, the potential function $p_i(u, v)$ on a potential pair

(u, v) in a bucket S_i is equal to

$$p_i(u,v) = \begin{cases} (1+\varepsilon) - d_i^*(u,v)/d(u,v) & \text{if } (u,v) \in S_i \\ C_\phi \cdot (d_i^*(u,v)/d(u,v) - (1+\varepsilon')) & \text{if } (u,v) \notin S_i \text{ and } \text{index}(u,v) = i \end{cases}$$

And the overall potential function on a bucket is defined as

$$\Phi_i = \sum_{(u,v) \in \mathcal{P}_i \cup S_i} p_i(u,v)$$

where $\mathcal{P}_i$ is the set of potential pairs with index i. And we defined a potential function on the whole spanner as

$$\Phi = \sum_i \Phi_i$$

Single edge update. We start with a simple case of bounding the potential function after a single edge insertion, then we consider a single edge deletion, and finally we extend our results to point insertions and deletions. We assume the pair that we insert to or delete from the spanner is an arbitrary pair from the set of potential pairs, because we only deal with potential pairs in our light spanner.

First, we consider a single edge insertion. We divide the analysis into two parts: the effect of the insertion of the potential pair onto the same level potential pairs, and the effect of the insertion onto higher level potential pairs. Recall that the level of a pair was defined in section 4.3.

We show that the edges of the same level satisfy a separation property, meaning that two edges in the same bucket cannot have both their endpoints close to each other.

Lemma 4.14 (Edge separation)**.** *Let (u, w) and (y, z) be two potential pairs in the same bucket. Assuming that (u, w) and (y, z) are not representing clusters from the same pair of* *chains in $\mathcal{T}$,*

$$\max\{d(u,y), d(w,z)\} > \frac{1}{\lambda^2 \cdot c} \max\{d(u,w), d(y,z)\}$$

Proof. Note that the constraint on not connecting the same pair of chains in the lemma is necessary, because in our sparse spanner construction, it is possible that two points are connected on two different levels on two different pairs of clusters. These two edges could potentially go into different non-empty blocks and get assigned different representatives and cause two parallel edges between two neighborhoods. While this is fine with sparsity purposes as long as there is at most a constant number of such parallel edges, we do not want to have them in our light spanner since they will make the analysis harder. Therefore, we assume that the edges are not connecting clusters centered at the same pair of points.

Next we show that these two pairs are from two cluster levels that are not far from each other. Let (u, w) be an edge on level l of the hierarchy and (y, z) be an edge on level l' of the hierarchy. Without loss of generality, assume that $l \geq l'$. We know that the potential pairs connect same level clusters together. Therefore, the length of (u, w) could vary between R^l and $\lambda \cdot R^l$. A similar inequality holds for (y, z). Thus the ratio of the length of the two would be at least $\lambda^{-1} R^{l-l'}$. Also, if C is chosen large enough it is clear that the two edges must have the same index as well, otherwise the length ratio of C between the two edges would make their endpoints very far from each other. Thus, the edges belong to the same bucket and index, meaning that the length of their ratio is at most c. So,

$$\lambda^{-1} R^{l-l'} < c$$

Now, the separation property on level l' between the clusters that these two edges are connecting to each other states that

$$\max\{d(u,y), d(w,z)\} \geq R^{l'} > \frac{R^l}{\lambda \cdot c}$$

Also according to earlier in this proof, $R^l \geq d(u,w)/\lambda$. Thus,

$$\max\{d(u,y), d(w,z)\} > \frac{d(u,w)}{\lambda^2 \cdot c} = \frac{1}{\lambda^2 \cdot c} \max\{d(u,w), d(y,z)\}$$

□

Now using this lemma we show that the insertion of a potential pair will not cause any violations of Invariant 2 on the same level.

Lemma 4.15. *Let (u,w) be a potential pair that is inserted to S_i where $i = \text{index}(u,w)$. If $d_i^*(u,w) > (1+\varepsilon')d(u,w)$, then the insertion of (u,w) results in no violations of Invariant 2 on same or lower level edges, assuming that $c^{-1}(1+\lambda^{-2}) \geq 1+\varepsilon'$.*

Proof. It is clear that (u,w) cannot participate in a shortest-path (in S_i^*) for any of the lower level pairs, so adding it does not affect any of those pairs. Also adding (u,w) would not violate Invariant 2 for the pair itself because of the assumption $d_i^*(u,w) > (1+\varepsilon')d(u,w)$. Thus we only need to analyze the other same level edges.

So let (y,z) be a same-level edge in S_i. If one of (u,w) or (y,z) use the other one in its shortest extended path (in S_i^*), then by lemma 4.14, the length of the path would be at least

$$\min\{d(u,w), d(y,z)\} + \max\{d(u,y), d(w,z)\} > \min\{d(u,w), d(y,z)\} + \frac{1}{\lambda^2 \cdot c} \max\{d(u,w), d(y,z)\}$$

We also know, from the assumption, that (u,w) and (y,z) are same-level edges in S_i, so $c^{-1} < d(u,w)/d(y,z) < c$. Therefore, the stretch of the path would be at least

$$\frac{\min\{d(u,w), d(y,z)\} + \max\{d(u,y), d(w,z)\}}{\max\{d(u,w), d(y,z)\}} > c^{-1}(1+\lambda^{-2}) \geq 1+\varepsilon'$$

Thus the stretch of the path is more than $1+\varepsilon'$, which shows that this addition would not violate Invariant 2 for any of the two pairs, even though the paths of same level edges are excluded in $d_i^*(u, w)$. □

Note that satisfying the condition in lemma 4.15 is easy. We first choose large enough λ to have a fine hierarchy, then we choose c small enough that $c < 1 + \lambda^{-2}$, then we choose $\varepsilon' = c^{-1}(1+\lambda^{-2}) - 1$. Now we show that the potential change on higher level potential pairs would be bounded by a constant after the insertion of (u, w).

Lemma 4.16. *Let (u, w) be a potential pair that is inserted to S_i where $i = \operatorname{index}(u, w)$. The insertion of (u, w) results in at most*

$$\frac{C_3}{c^k - 1}$$

potential increase on higher level potential pairs in S_i, where

$$C_3 = \varepsilon(1+\varepsilon)^d c^{d+1} C_1$$

is a constant (and k is the number of buckets).

Proof. Let (y, z) be an edge of level $j' > j$ in S_i whose d_i^* is decreased by the addition of (u, w). Thus the shortest extended path between y and z in S_i^* passes through (u, w). Denote this path by $P_i^*(y, z)$. Before the addition of (u, w), the length of the same path in S_i^* was at most $\|P_i^*(y, z)\| + \varepsilon d(u, w)$. Hence, $\Delta d_i^*(y, z) \geq -\varepsilon d(u, w)$, and the potential change of this edge would be

$$\Delta p_i(y, z) = \frac{-\Delta d_i^*(y, z)}{d(y, z)} \leq \frac{\varepsilon d(u, w)}{d(y, z)} \leq \varepsilon c^{k(j-j')+1}$$

In the next step, we bound the number of such (y, z) pairs. Let r be the minimum length of such edge in level j'. Both y and z must be within $(1+\varepsilon)cr$ Euclidean distance of u (and w), otherwise the edge (u, w) would be useless in (y, z)'s shortest path in S_i^*. Thus, all such pairs are located in a ball $B(u,(1+\varepsilon)cr)$, and according to lemma 4.2, there would be at most

$$C_2 = (1+\varepsilon)^d c^d C_1$$

number of them.

Thus, the overall potential change on level j' would be upper bounded by $C_2 \varepsilon c^{k(j-j')+1}$. Summing this up over $j' > j$, the overall potential change on higher level pairs would be at most

$$\Delta\Phi_i < \sum_{j'>j} \varepsilon C_2 c^{k(j-j')+1} = \frac{C_3}{c^k - 1}$$

where $C_3 = \varepsilon C_2 c$. □

Now we analyze the removal of a potential pair from a bucket. The difference with the removal is that it could cause violations of Invariant 1 on its level. Therefore, we analyze a removal, together with some subsequent edge insertions that fix any violations of Invariant 1 on the same level.

Definition 4.17 (Edge removal process)**.** Let (u,w) be a potential pair that is located in S_i where $i = \text{index}(u,w)$. We define the single edge removal process on (u,w) to be the process that deletes (u,w) from S_i and fixes the subsequent violations of Invariant 1 on the same level by greedily picking a violating pair, and connecting its endpoints in S_i, until no violating pair for Invariant 1 is left.

We analyze the effect of the edge removal process in the following two lemmas,

Lemma 4.18. *Let (u,w) be a potential pair that does not violate Invariant 1 ($d_i^*(u,w) < (1+\varepsilon)d(u,w)$) and is deleted from S_i ($i = \text{index}(u,w)$), using the edge removal process. The deletion of (u,w) together with these subsequent insertions results in no violations of Invariant 1 or Invariant 2 on same or lower level edges, assuming that $c^{-1}(1+\lambda^{-2}) \geq 1+\varepsilon'$.*

Proof. It is clear that (u, w) cannot participate in a shortest-path (in S_i^*) for any of the lower level pairs, so deleting it does not affect any of those pairs. Also, every same level pair that violates Invariant 1 is fixed after the insertion of subsequent edges. Therefore, we just need to show there are no violations of Invariant 2 after these changes. This is also clear by lemma 4.15, because we are only inserting edges (y, z) that that violate Invariant 1, i.e. $d_i^*(y, z) > (1 + \varepsilon)d(y, z) > (1 + \varepsilon')d(y, z)$, meaning that the assumption of the lemma holds in this insertion. □

We show a similar bound as edge insertion on the effect of the edge removal process on higher level pairs.

Lemma 4.19. *Let (u, w) be a potential pair that is deleted from to S_i where $i = \text{index}(u, w)$. The edge removal process on (u, w) results in at most*

$$\frac{C_5}{c^k - 1}$$

potential increase on higher level potential pairs in S_i, for some constant C_5 that depends on ε, ε', and c. is a constant.

Proof. The edge removal process can be divided into two phases. The deletion of (u, w), and the insertion of the subsequent pairs. First, we show that the potential increase after the edge deletion is bounded. Let (y, z) be an edge of level $j' > j$ in S_i whose d_i^* is increase by the deletion of (u, w). Thus the shortest extended path between y and z in S_i^* passes through (u, w). Denote this path by $P_i^*(y, z)$. After the removal of (u, w), the length of the same path in S_i^* is at most $\|P_i^*(y, z)\| + \varepsilon d(u, w)$. Hence, $\Delta d_i^*(y, z) \leq \varepsilon d(u, w)$, and the potential change of this edge would be

$$\Delta p_i(y, z) = \frac{\Delta d_i^*(y, z)}{d(y, z)} \leq \frac{\varepsilon d(u, w)}{d(y, z)} \leq \varepsilon c^{k(j-j')+1}$$

Again, the number of such (y, z) pairs is bounded by

$$C_2 = (1+\varepsilon)^d c^d C_1$$

according to lemma 4.2. Thus, the overall potential change on level j' would be upper bounded by $C_2 \varepsilon c^{k(j-j')+1}$. Summing this up over $j' > j$, the overall potential change on higher level pairs would be at most

$$\Delta\Phi_i < \sum_{j'>j} \varepsilon C_2 c^{k(j-j')+1} = \frac{C_3}{c^k - 1}$$

where $C_3 = \varepsilon C_2 c$.

Now, the number of subsequent edge insertions would also bounded by a constant. Because in order for an inserted pair (y, z) to violate Invariant 1 after the deletion of (u, w), u and w must be within a distance $c(1+\varepsilon)d(u, w)$, otherwise the edge (u, w) would be useless in their shortest-path. Also since they satisfy Invariant 2, we conclude from lemma 4.2 that the number of such pairs is bounded by a constant. Denote this bound by C_4. Then the potential on higher level pairs from the insertions of C_4 pairs on the same level would be at most $C_3 C_4/(c^k - 1)$.

Overall, the potential increase on higher level pairs from the edge removal process will be $C_5/(c^k - 1)$ where $C_5 = C_3(C_4 + 1)$. □

Adjusted potential function. We have one last step before analyzing the potential function after a point insertion and a point deletion. We need to slightly adjust the potential function to take into account future edges that might be added between the existing points because of a new point. As we saw in section 4.4, a new point can have a large degree in $\mathcal{S}_0$ due to its implicit clusters in multiple levels of the hierarchy. We handled this by assigning these edges to nearby representatives and we proved a constant degree bound on $\mathcal{S}_1$. But this still would mean adding a point could increase the potential function by $\Omega(\log \Delta)$ since logarithmic number of edges could be added to the sparse spanner. We fix this issue in our potential function by taking into account all the future edges that can be incident to a point. Our adjusted potential function on the whole spanner, denoted by Φ^*, has an extra term compared to the previous potential function Φ,

$$\Phi^* = \Phi + \frac{p_{max}}{2} \cdot \sum_{i=1}^{n} (D_{max} - \deg_{\mathcal{S}_1}(v_i))$$

$\deg_{\mathcal{S}_1}(v_i)$ is the degree of the i-th point (in any fixed order, e.g. insertion order) in the sparse bounded degree spanner $\mathcal{S}_1$, and

$$p_{max} = \max\{1 + \varepsilon, C_\phi(\varepsilon - \varepsilon')\}$$

is the maximum potential value a potential pair can have in its own bucket given the fact that it does not violate Invariant 1. Note that the first term is the maximum of the potential of any pair if its edge is present in the bucket and the second term is the maximum potential of the pair if its edge is absent from the bucket and it is not violating Invariant 1. We will later see why the assumption that Invariant 1 holds for such pairs is fine. But this extra term in the potential function will be used to cover the potential p_i of the extra potential pairs added by the new point.

4.5.2 Maintaining the light spanner

We are finally ready to introduce our techniques for maintaining a light spanner under a dynamic point set. For point insertion, we select a subset of edges added in the sparse spanner to be present in the light spanner. We show that the potential increase on Φ^* after inserting the new point would be bounded by a constant. Then we perform the same analysis for point deletion and we show that the potential increase is bounded by $\mathcal{O}(\log \Delta)$. In the last part of this section we introduce our methods for iteratively improving the weight of the spanner by showing an algorithm that decreases the potential function by a constant value in each iteration. This concludes our results on the recourse for point insertion and point deletion.

Point insertion. Following a point insertion for a point p, we insert p into the hierarchy and we update our sparse spanner $\mathcal{S}_1$. There are at most a constant number of pairs whose representative assignment has changed, we update these pairs in the light spanner as well. Meaning that if they were present in the light spanner, we keep them present but with the new endpoints, and if they were absent, we keep them absent. Besides the re-assignments, there could be some (even more than a constant) edge insertions into the sparse spanner, but the degree bound of D_{max} would still hold on every point. We greedily pick one new edge at a time that its endpoints violate Invariant 1 in the light spanner, and we add that edge to the light spanner. (algorithm 12)

Algorithm 12 Inserting a point to the light spanner.

```
1: procedure INSERT-TO-LIGHT-SPANNER($p$)
2:     Insert $p$ into the hierarchy $\mathcal{T}$.
3:     Make the required changes on the sparse bounded degree spanner $\mathcal{S}_1$.
4:     for any pair $(u, w)$ with updated representative assignment do
5:         Update the endpoints of the edge in the light spanner.
6:     for any edge $(u, w)$ added to the sparse spanner do
7:         if Invariant 1 is violated for this pair on the light spanner then
8:             Add $(u, w)$ to the light spanner (to its own bucket).
```

We now analyze the change in the potential function after performing this function following a point insertion.

Lemma 4.20. *The procedure* Insert-to-Light-Spanner *adds at most a constant amount to* Φ^*.

Proof. Note that at most a constant number of edges will go through a representative assignment change. Each representative change can be divided into removing the old pair and adding the new one. Each removal will increase the potential of at most a constant number of pairs on any same or higher level pairs. This would sum up to a constant amount as we saw earlier in lemma 4.18 and lemma 4.19. Also, inserting the updated pairs would also sum up to a constant amount of increase in the potential function as we saw in lemma 4.15 and lemma 4.16.

For the edge insertions however, we will get help from the extra term in our potential function. Note that any extra edge that is added between any two points that existed before the new point will increase both of their degrees by 1 and therefore, decrease the term

$$p_{max} \cdot \sum_{i=1}^{n} (D_{max} - \deg_{\mathcal{S}_1}(v_i))$$

by p_{max}. On the other hand, the new pair will either be added to the light spanner or will satisfy Invariant 1 if not added. Thus, its potential will be at most $1 + \varepsilon$ in the first case, and at most $C_\phi(\varepsilon - \varepsilon')$ in the second case. In any case, the potential of the new pair is not more than p_{max}, and hence Φ^* will not increase due to the addition of the new pair.

Lastly, the new point will introduce a new term $p_{max} \cdot (D_{max} - \deg_{\mathcal{S}_1}(v_{n+1}))$ in Φ^* which would also be bounded by a constant. Overall, the increase in Φ^* will be bounded by a constant. $\square$

This lemma by itself does not provide an upper bound on the number of inflicted updates. However, later in section 4.5.3, when we analyze our maintenance updates, we use this lemma to prove that the amortized number of edge updates would be bounded by a constant.

Point deletion. Following a point deletion, we perform the deletion on the hierarchy and update the sparse spanner accordingly. This would cause at most $\mathcal{O}(\log \Delta)$ potential pairs to be deleted from or inserted into the spanner. The procedure on the light spanner is simple in this case. We add all the inserted pairs to the light spanner, and we remove the removed pairs from the light spanner if they are present.

Algorithm 13 Deleting a point from the light spanner.

```
1: procedure Delete-from-Light-Spanner($p$)
2:     Delete $p$ from the hierarchy $\mathcal{T}$.
3:     Make the required changes on the sparse bounded degree spanner $\mathcal{S}_1$.
4:     for any pair $(u, w)$ removed from the sparse spanner do
5:         Remove $(u, w)$ from the light spanner if present.
6:     for any pair $(u, w)$ added to the sparse spanner do
7:         Add $(u, w)$ to the light spanner.
8:     for any pair $(u, w)$ with updated representative assignment do
9:         Update $(u, w)$ in the light spanner as well.
```

Lemma 4.21. *The procedure* Delete-from-Light-Spanner *adds at most $\mathcal{O}(\log \Delta)$ to Φ^*.*

Proof. The number of edges updated on every level of hierarchy after a point removal is bounded by a constant. Therefore, the total number of changes would be bounded by $\mathcal{O}(\log \Delta)$. Each change would cause Φ^* to increase by at most p_{max}. Thus, the total increase is bounded by $\mathcal{O}(\log \Delta)$. □

4.5.3 Maintenance updates

As we saw earlier in this section, following a point insertion and removal many edge updates happen on the light spanner, and we did not check for the invariants to hold after these changes. Maintaining Invariant 1 and Invariant 2 is crucial for the quality of our spanner. Here, we show how we can maintain both invariants following a point insertion and deletion. We also complete our amortized bounds on the number of updates required to make the spanner consistent with the two invariants.

Our maintenance updates are of two different types, each designed to fix the violations of one invariant. Whenever a violation of Invariant 1 occurs for a potential pair (v, w), we fix the violation by simply adding the edge between v and w to its corresponding S_i. This fixes the violation for this pair, and all pairs of descendants of the two clusters that v and w represent. We will show that this change will decrease the value of the potential function by a constant amount.

Fixing a violation of Invariant 2 on the other hand is not that simple. Removing the edge between v and w might cause multiple violations of Invariant 1 on the same level. As we discussed before, we address this issue by fixing the same-level violations of Invariant 1 first through adding edges between v's neighborhood and w's neighborhood. Then we show that the removal of (v, w) together with these additions would lower the value of the potential function by a constant amount.

Our maintenance approach is simple, as long as there exists a potential pair on any S_i that violates either of the two invariants, we perform the corresponding procedure to enforce that invariant for that pair. The fact that the potential function decreases by a constant amount after each fix is the key to our amortized analysis on the number of maintenance updates to reach a spanner with bounded degree and bounded lightness.

Fixing a violation of Invariant 1. In our first lemma in this section, we show that fixing a violation of Invariant 1 in the way that we mentioned above, would decrease the value of the potential function on each S_i.

Lemma 4.22. *Let (v, w) be a potential pair with* $\text{index}(v, w) = i$ *that violates Invariant 1, i.e. $d_i^*(v, w)/d(v, w) > 1 + \varepsilon$. Also, assume that*

$$k \geq \log_c \left(1 + \frac{C_3}{(C_\phi - 1)(\varepsilon - \varepsilon')} \right)$$

Then adding the edge (v, w) to S_i decreases the overall potential Φ_i of S_i by at least $(\varepsilon - \varepsilon')$.

Proof. Note that adding (v, w) would have no effect on the potential of the lower level or same level potential pairs, due to the definition of d_i^*. We know from lemma 4.16 that adding (v, w) to S_i would increase the potential on higher level pairs by at most $C_3/(c^k - 1)$. Also, the potential of the pair itself before the addition is

$$p_i(v, w) = C_\phi \cdot \left(\frac{d_i^*(v, w)}{d(v, w)} - (1 + \varepsilon') \right)$$

On the other hand, after the addition,

$$p_i(v, w) = (1 + \varepsilon) - \frac{d_i^*(v, w)}{d(v, w)}$$

Therefore,

$$\Delta p_i(v, w) = (\varepsilon - \varepsilon') + (C_\phi + 1) \left(1 + \varepsilon' - \frac{d_i^*(v, w)}{d(v, w)} \right)$$

We know by the assumption that the stretch of the shortest extended path between v and w would be more than $1 + \varepsilon$, since (v, w) is violating Invariant 1. Therefore,

$$1 + \varepsilon' - \frac{d_i^*(v, w)}{d(v, w)} < -(\varepsilon - \varepsilon')$$

Thus,

$$\Delta p_i(v, w) < (\varepsilon - \varepsilon') - (C_\phi + 1)(\varepsilon - \varepsilon') = -C_\phi(\varepsilon - \varepsilon')$$

According to this and what we mentioned earlier in the proof,

$$\Delta \Phi_i \leq -C_\phi(\varepsilon - \varepsilon') + \frac{C_3}{c^k - 1}$$

and if

$$k \geq \log_c \left(1 + \frac{C_3}{(C_\phi - 1)(\varepsilon - \varepsilon')} \right)$$

then $\Delta \Phi_i \leq -(\varepsilon - \varepsilon')$, which is a negative constant. □

Fixing a violation of Invariant 2. Next, we consider the second type of maintenance updates, which is to fix the violations of Invariant 2. Whenever a pair (v, w) that violates Invariant 2 is found, the first step is to remove the corresponding edge from its subset S_i. Afterwards, we address the same-level violations of Invariant 1 by greedily adding a pair that violates Invariant 1, until none is left. This is the same as performing the edge removal process on the violating pair.

Lemma 4.23. *Let $(v, w) \in S_i$ be an edge that violates Invariant 2, i.e. $d_i^*(v, w)/d(v, w) \leq 1 + \varepsilon'$. Also assume that*

$$k \geq \log_c \left(1 + \frac{2C_5}{\varepsilon - \varepsilon'}\right)$$

Then performing the edge removal process on (v, w) decreases the overall potential Φ_i of S_i by at least $(\varepsilon - \varepsilon')$.

Proof. Since all the additions and removals in the edge removal process are happening on the same level and also due to the definition of d_i^*, there would be no potential change on any of the same or lower level pairs. We know from lemma 4.19 that deleting (v, w) from S_i would increase the potential on higher level pairs by at most $C_5/(c^k - 1)$. The potential of the pair itself before the deletion is

$$p_i(v, w) = (1 + \varepsilon) - \frac{d_i^*(v, w)}{d(v, w)}$$

After the deletion,

$$p_i(v, w) = C_\phi \cdot \left(\frac{d_i^*(v, w)}{d(v, w)} - (1 + \varepsilon')\right)$$

Therefore,

$$\Delta p_i(v, w) = -(\varepsilon - \varepsilon') - (C_\phi + 1)\left(1 + \varepsilon' - \frac{d_i^*(v, w)}{d(v, w)}\right)$$

We know by the assumption that the stretch of the shortest extended path between v and

w would be less than $1 + \varepsilon'$, since (v, w) is violating Invariant 2. Therefore,

$$1 + \varepsilon' - \frac{d_i^*(v, w)}{d(v, w)} > 0$$

Thus,

$$\Delta p_i(v, w) < (\varepsilon - \varepsilon')$$

According to this and what we mentioned earlier in the proof,

$$\Delta \Phi_i \leq -(\varepsilon - \varepsilon') + \frac{C_5}{c^k - 1}$$

and if

$$k \geq \log_c \left(1 + \frac{2C_5}{\varepsilon - \varepsilon'}\right)$$

then $\Delta \Phi_i \leq -(\varepsilon - \varepsilon')/2$, which is a negative constant. □

Bounding the number of updates. Now that we introduced our maintenance updates and we analyzed the change in the potential functions after each of these updates, we can finally prove our amortized bounds. We prove that the amortized number of edge updates in our algorithm after a point insertion is $\mathcal{O}(1)$, while the amortized number of edge updates after a point deletion is $\mathcal{O}(\log \Delta)$.

Theorem 4.24. *Our fully-dynamic spanner construction in d-dimensional Euclidean spaces has a stretch-factor of $1 + \varepsilon$ and a lightness that is bounded by a constant. Furthermore, this construction performs an amortized $\mathcal{O}(1)$ edge updates following a point insertion, and an amortized $\mathcal{O}(\log \Delta)$ edge updates following a point deletion.*

Proof. The stretch factor and the lightness immediately follow from the fact that our spanner always satisfies the two invariants, and according to lemma 4.1 and the leapfrog property, that would be enough for a $1 + \varepsilon$ stretch factor and constant lightness.

In order to prove the amortized bounds on the number of edge updates after each operation, we recall that by lemma 4.20, the potential change $\Delta\Phi^*$ after a point insertion is bounded by a constant, and by lemma 4.21, the potential change after a point deletion is bounded by $\mathcal{O}(\log \Delta)$. On the other hand, by lemma 4.22 and lemma 4.23, each maintenance update reduces the potential Φ^* by at least $(\varepsilon - \varepsilon')/2$, since the impacted Φ_i reduces after the maintenance update, Φ_j for $j \neq i$ will remain unchanged, and the extra term $\frac{p_{max}}{2} \cdot \sum_{i=1}^{n}(D_{max} - \deg_{\mathcal{S}_1}(v_i))$ will also remain unchanged since the sparse spanner is not affected by the maintenance updates. Therefore, the amortized number of maintenance updates required after each point insertion is $\mathcal{O}(1)$ while this number after a point deletion is $\mathcal{O}(\log \Delta)$. Also, the number of edge updates before the maintenance updates would be bounded by the same amortized bounds. Thus, we can finally conclude that the amortized number of edge updates following a point insertion is $\mathcal{O}(1)$, while for a point deletion it is $\mathcal{O}(\log \Delta)$. □

4.6 Conclusions

In this chapter, we presented the first fully-dynamic lightweight construction for $(1 + \varepsilon)$-spanners in the d-dimensional Euclidean space. In our construction, the amortized number of edge updates following a points insertion is bounded by a constant, and the amortized number of edge updates following a point deletion is bounded by $\mathcal{O}(\log \Delta)$. To achieve this, we defined a set of maintenance updates that could reduce the weight of an existing spanner iteratively, leading to a bounded lightness spanner. We also defined a potential function that could be used to provide an amortized bound on the number of such updates. This framework can be used to find lightweight Euclidean spanners under circumstances other than the fully-dynamic setting, e.g. semi-dynamic or online setting with recourse. It would be interesting to explore other applications of this framework. Since our construction, like the celebrated greedy spanner construction, takes advantage of shortest path queries, it is not necessarily efficient in terms of the running time, and it is suitable when the problem-dependent update cost for a single edge is high. Although we did not focus on optimizing the running time, it would be interesting to look at our construction from that perspective, and find ways to improve its efficiency. We also leave as an open problem whether the amortized bound on the number of edge updates following a point deletion can be improved to $\mathcal{O}(1)$.

Bibliography

[1] M. A. Abam, M. De Berg, M. Farshi, and J. Gudmundsson. Region-fault tolerant geometric spanners. *Discrete & Computational Geometry*, 41(4):556–582, 2009.

[2] M. A. Abam and S. Har-Peled. New constructions of sspds and their applications. *Computational Geometry*, 45(5-6):200–214, 2012.

[3] I. Althöfer, G. Das, D. Dobkin, and D. Joseph. Generating sparse spanners for weighted graphs. In J. R. Gilbert and R. Karlsson, editors, *Proceedings of the 2nd Scandinavian Workshop on Algorithm Theory (SWAT)*, volume 447 of *Lecture Notes in Computer Science*, pages 26–37. Springer, 1990.

[4] I. Althöfer, G. Das, D. Dobkin, D. Joseph, and J. Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9(1):81–100, 1993.

[5] K. Alzoubi, X.-Y. Li, Y. Wang, P.-J. Wan, and O. Frieder. Geometric spanners for wireless ad hoc networks. *IEEE Transactions on Parallel and Distributed Systems*, 14(4):408–421, 2003.

[6] S. Arikati, D. Z. Chen, L. P. Chew, G. Das, M. Smid, and C. D. Zaroliagis. Planar spanners and approximate shortest path queries among obstacles in the plane. In J. Diaz and M. Serna, editors, *Proceedings of the 4th European Symposium on Algorithms (ESA)*, volume 1136 of *Lecture Notes in Computer Science*, pages 514–528. Springer, 1996.

[7] S. Arya, D. M. Mount, and M. Smid. Randomized and deterministic algorithms for geometric spanners of small diameter. In *Proc. 35th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 703–712, 1994.

[8] B. Awerbuch, B. Berger, L. Cowen, and D. Peleg. Near-linear time construction of sparse neighborhood covers. *SIAM Journal on Computing*, 28(1):263–277, 1998.

[9] S. W. Bae, J.-F. Baffier, J. Chun, P. Eades, K. Eickmeyer, L. Grilli, S.-H. Hong, M. Korman, F. Montecchiani, I. Rutter, and C. D. Tóth. Gap-planar graphs. *Theoretical Computer Science*, 745:36–52, 2018.

[10] S. Baswana, T. Kavitha, K. Mehlhorn, and S. Pettie. Additive spanners and (α, β)-spanners. *ACM Transactions on Algorithms*, 7(1):5, 2010.

[11] S. Bhore, A. Filtser, H. Khodabandeh, and C. D. Tóth. Online spanners in metric spaces. In *30th Annual European Symposium on Algorithms (ESA 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

[12] S. Bhore, A. Filtser, H. Khodabandeh, and C. D. Tóth. Online spanners in metric spaces. *SIAM Journal on Discrete Mathematics*, 38(1):1030–1056, 2024.

[13] S. Bhore and C. D. Tóth. Online Euclidean spanners. In P. Mutzel, R. Pagh, and G. Herman, editors, *29th Annual European Symposium on Algorithms, ESA 2021, September 6–8, 2021, Lisbon, Portugal (Virtual Conference)*, volume 204 of *LIPIcs*, pages 16:1–16:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

[14] G. Borradaile, H. Le, and C. Wulff-Nilsen. Greedy spanners are optimal in doubling metrics. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2371–2379, 2019.

[15] P. Bose, P. Carmi, M. Farshi, A. Maheshwari, and M. Smid. Computing the greedy spanner in near-quadratic time. *Algorithmica*, 58(3):711–729, 2010.

[16] P. Bose, J. Gudmundsson, and P. Morin. Ordered theta graphs. *Computational Geometry*, 28(1):11–18, 2004.

[17] R. Braynard, D. Kostic, A. Rodriguez, J. Chase, and A. Vahdat. Opus: an overlay peer utility service. In *Proceedings of the 5th IEEE Conference on Open Architectures and Network Programming (OPENARCH)*, pages 167–178. IEEE, 2002.

[18] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Journal of the ACM (JACM)*, 42(1):67–90, 1995.

[19] T.-H. H. Chan and A. Gupta. Small hop-diameter sparse spanners for doubling metrics. *Discrete & Computational Geometry*, 41(1):28–44, 2009.

[20] T. M. Chan and D. Skrepetos. Approximate shortest paths and distance oracles in weighted unit-disk graphs. *Journal of Computational Geometry (Old Web Site)*, 10(2):3–20, 2019.

[21] S. Chechik, M. Langberg, D. Peleg, and L. Roditty. Fault tolerant spanners for general graphs. *SIAM Journal on Computing*, 39(7):3403–3423, 2010.

[22] P. Chew. There is a planar graph almost as good as the complete graph. In *Proceedings of the second annual symposium on Computational geometry*, pages 169–177, 1986.

[23] P. Chew. There are planar graphs almost as good as the complete graph. *Journal of Computer and System Sciences*, 39(2):205–219, 1989.

[24] P. Choudhary, M. T. Goodrich, S. Gupta, H. Khodabandeh, P. Matias, and V. Raman. Improved kernels for tracking paths. *Information Processing Letters*, 181:106360, 2023.

[25] E. Cohen. Fast algorithms for constructing t-spanners and paths with stretch t. *SIAM Journal on Computing*, 28(1):210–236, 1998.

[26] R. Cole and L. Gottlieb. Searching dynamic point sets in spaces with bounded doubling dimension. In J. M. Kleinberg, editor, *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21–23, 2006*, pages 574–583. ACM, 2006.

[27] M. Damian, S. Pandit, and S. Pemmaraju. Distributed spanner construction in doubling metric spaces. In *International Conference on Principles of Distributed Systems*, pages 157–171. Springer, 2006.

[28] M. Damian, S. Pandit, and S. Pemmaraju. Local approximation schemes for topology control. In *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 208–217, 2006.

[29] G. Das. The visibility graph contains a bounded-degree spanner. In *Proceedings of the 9th Canadian Conference on Computational Geometry (CCCG)*, pages 70–75, 1997.

[30] G. Das and D. Joseph. Which triangulations approximate the complete graph? In *Proc. International Symposium on Optimal Algorithms*, pages 168–192. Springer, 1989.

[31] G. Das and G. Narasimhan. A fast algorithm for constructing sparse Euclidean spanners. *International Journal of Computational Geometry & Applications*, 7(04):297–315, 1997.

[32] G. Das, G. Narasimhan, and J. Salowe. A new way to weigh malnourished euclidean graphs. In *proceedings of the sixth annual ACM-SIAM symposium on discrete algorithms*, pages 215–222, 1995.

[33] A. Dobson and K. E. Bekris. Sparse roadmap spanners for asymptotically near-optimal motion planning. *International Journal of Robotics Research*, 33(1):18–47, 2014.

[34] V. Dujmović, D. Eppstein, and D. R. Wood. Structure of graphs with locally restricted crossings. *SIAM Journal on Discrete Mathematics*, 31(2):805–824, 2017.

[35] Z. Dvorak and S. Norin. Strongly sublinear separators and polynomial expansion. *SIAM Journal on Discrete Mathematics*, 30(2):1095–1101, 2016.

[36] M. Elkin. Computing almost shortest paths. *ACM Transactions on Algorithms*, 1(2):283–323, 2005.

[37] M. Elkin, A. Filtser, and O. Neiman. Distributed construction of light networks. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, pages 483–492, 2020.

[38] M. Elkin and D. Peleg. $(1+\varepsilon,\beta)$-spanner constructions for general graphs. *SIAM Journal on Computing*, 33(3):608–631, 2004.

[39] M. Elkin and J. Zhang. Efficient algorithms for constructing $(1+\varepsilon,\beta)$-spanners in the distributed and streaming models. *Distributed Computing*, 18(5):375–385, 2006.

[40] D. Eppstein. Spanning trees and spanners. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 425–461. North-Holland, 2000.

[41] D. Eppstein et al. Spanning trees and spanners., 2000.

[42] D. Eppstein and M. T. Goodrich. Studying (non-planar) road networks through an algorithmic lens. In *Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages A16:1–A16:10. ACM, 2008.

[43] D. Eppstein, M. T. Goodrich, and D. Strash. Linear-time algorithms for geometric graphs with sublinearly many edge crossings. *SIAM Journal on Computing*, 39(8):3814–3829, 2010.

[44] D. Eppstein and S. Gupta. Crossing patterns in nonplanar road networks. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages A40:1–A40:9. ACM, 2017.

[45] D. Eppstein and H. Khodabandeh. On the edge crossings of the greedy spanner. In *37th International Symposium on Computational Geometry*, volume 12, page 37, 2021.

[46] D. Eppstein and H. Khodabandeh. Brief announcement: Distributed lightweight spanner construction for unit ball graphs in doubling metrics. In *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 57–59, 2022.

[47] D. Eppstein and H. Khodabandeh. Distributed construction of lightweight spanners for unit ball graphs. In *36th International Symposium on Distributed Computing*, 2022.

[48] D. Eppstein and H. Khodabandeh. Maintaining light spanners via minimal updates. *36th Canadian Conference on Computational Geometry (CCCG 2024)*, 2024.

[49] M. Farshi and J. Gudmundsson. Experimental study of geometric t-spanners. *ACM Journal of Experimental Algorithmics*, 14:1.3:1–1.3:29, 2009.

[50] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. Graph distances in the streaming model: the value of space. In *Proceedings of the 16th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 745–754. Society for Industrial and Applied Mathematics, 2005.

[51] A. Filtser and S. Solomon. The greedy spanner is existentially optimal. In *Proceedings of the 35th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, pages 9–17. ACM, 2016.

[52] A. M. Frieze, G. L. Miller, and S.-H. Teng. Separator based parallel divide and conquer in computational geometry. In *Proceedings of the 4th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, volume 92, pages 420–429, 1992.

[53] M. Fürer and S. P. Kasiviswanathan. Spanners for geometric intersection graphs. In *Workshop on Algorithms and Data Structures*, pages 312–324. Springer, 2007.

[54] J. Gao, L. J. Guibas, J. Hershberger, L. Zhang, and A. Zhu. Geometric spanners for routing in mobile networks. *IEEE journal on selected areas in communications*, 23(1):174–185, 2005.

[55] J. Gao, L. J. Guibas, and A. Nguyen. Deformable spanners and applications. *Comput. Geom.*, 35(1-2):2–19, 2006.

[56] M. J. Golin, H. Khodabandeh, and B. Qin. Non-approximability and polylogarithmic approximations of the single-sink unsplittable and confluent dynamic flow problems. In *28th International Symposium on Algorithms and Computation (ISAAC 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[57] M. T. Goodrich. Planar separators and parallel polygon triangulation. *Journal of Computer and System Sciences*, 51(3):374–389, 1995.

[58] M. T. Goodrich, S. Gupta, H. Khodabandeh, and P. Matias. How to catch marathon cheaters: New approximation algorithms for tracking paths. In *Workshop on Algorithms and Data Structures*, pages 442–456. Springer, 2021.

[59] L.-A. Gottlieb. A light metric spanner. In *Proc. 56th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 759–772, 2015.

[60] L.-A. Gottlieb and L. Roditty. Improved algorithms for fully dynamic geometric spanners and geometric routing. In S.-H. Teng, editor, *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20–22, 2008*, pages 591–600. SIAM, 2008.

[61] L.-A. Gottlieb and L. Roditty. An optimal dynamic spanner for doubling metric spaces. In *Proc. 16th European Symposium on Algorithms (ESA)*, volume 5193 of *LNCS*, pages 478–489. Springer, 2008.

[62] J. Gudmundsson, C. Levcopoulos, and G. Narasimhan. Fast greedy algorithms for constructing sparse geometric spanners. *SIAM Journal on Computing*, 31(5):1479–1500, 2002.

[63] J. P. Jenkins, I. A. Kanj, G. Xia, and F. Zhang. Local construction of spanners in the 3d space. *IEEE Transactions on Mobile Computing*, 11(7):1140–1150, 2012.

[64] L. Jia, R. Rajaraman, and C. Scheideler. On local algorithms for topology control and routing in ad hoc networks. In *Proceedings of the 15th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 220–229. ACM, 2003.

[65] I. A. Kanj, L. Perković, and G. Xia. Computing lightweight spanners locally. In *International Symposium on Distributed Computing*, pages 365–378. Springer, 2008.

[66] J. M. Keil. Approximating the complete euclidean graph. In *SWAT 88: 1st Scandinavian Workshop on Algorithm Theory Halmstad, Sweden, July 5–8, 1988 Proceedings 1*, pages 208–213. Springer, 1988.

[67] D. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12(1):28–35, 1983.

[68] P. N. Klein, S. Mozes, and C. Sommer. Structured recursive separator decompositions for planar graphs in linear time. In *Proceedings of the 45th ACM Symposium on Theory of Computing (STOC)*, pages 505–514. ACM, 2013.

[69] R. Krauthgamer and J. R. Lee. Navigating nets: simple algorithms for proximity search. In J. I. Munro, editor, *Proceedings of the Fifteenth Annual ACM–SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11–14, 2004*, pages 798–807. SIAM, 2004.

[70] F. Kuhn, T. Moscibroda, and R. Wattenhofer. On the locality of bounded growth. In *Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, pages 60–68, 2005.

[71] H. Le and S. Solomon. Truly optimal Euclidean spanners. In *Proc. 60th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 1078–1100, 2019.

[72] H. Le and C. Than. Greedy spanners in euclidean spaces admit sublinear separators. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3287–3310. SIAM, 2022.

[73] N. Li, J. C. Hou, and L. Sha. Design and analysis of an mst-based topology control algorithm. *IEEE Transactions on Wireless Communications*, 4(3):1195–1206, 2005.

[74] X.-Y. Li, G. Calinescu, P.-J. Wan, and Y. Wang. Localized delaunay triangulation with application in ad hoc wireless networks. *IEEE Transactions on Parallel and Distributed Systems*, 14(10):1035–1047, 2003.

[75] X.-Y. Li, P.-J. Wan, and Y. Wang. Power efficient and sparse spanner for wireless ad hoc networks. In *Proceedings of the 10th International Conference on Computer Communications and Networks (ICCCN)*, pages 564–567. IEEE, 2001.

[76] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

[77] J. D. Marble and K. E. Bekris. Asymptotically near-optimal planning with probabilistic roadmap spanners. *IEEE Transactions on Robotics*, 29(2):432–444, 2013.

[78] W. Mulzer and M. Willert. Compact routing in unit disk graphs. In *31st International Symposium on Algorithms and Computation (ISAAC 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

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

[80] D. Peleg and L. Roditty. Localized spanner construction for ad hoc networks with variable transmission range. *ACM Transactions on Sensor Networks (TOSN)*, 7(3):1–14, 2010.

[81] D. Peleg and A. A. Schäffer. Graph spanners. *Journal of Graph Theory*, 13(1):99–116, 1989.

[82] L. Roditty. Fully dynamic geometric spanners. *Algorithmica*, 62(3):1073–1087, 2012.

[83] L. Roditty and U. Zwick. On dynamic shortest paths problems. *Algorithmica*, 61(2):389–401, 2011.

[84] J. Schneider and R. Wattenhofer. A log-star distributed maximal independent set algorithm for growth-bounded graphs. In *Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*, pages 35–44, 2008.

[85] M. Smid. The weak gap property in metric spaces of bounded doubling dimension. In *Efficient Algorithms*, pages 275–289. Springer, 2009.

[86] W. Wang, C. Jin, and S. Jamin. Network overlay construction under limited end-to-end reachability. In *Proceedings of the 24th Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, volume 3, pages 2124–2134. IEEE, 2005.

[87] Y. Wu, Y. Hu, Y. Su, N. Yu, and R. Feng. Topology control for minimizing interference with delay constraints in an ad hoc network. *Journal of Parallel and Distributed Computing*, 113:63–76, 2018.

[88] D. Yu, L. Ning, Y. Zou, J. Yu, X. Cheng, and F. C. Lau. Distributed spanner construction with physical interference: constant stretch and linear sparseness. *IEEE/ACM Transactions on Networking*, 25(4):2138–2151, 2017.

[89] A. H. Zargari, S. A. Aqajari, H. Khodabandeh, A. M. Rahmani, and F. Kurdahi. An accurate non-accelerometer-based ppg motion artifact removal technique using cyclegan. *ACM Transactions on Computing for Healthcare*, 2022.