Disk Intersection Graphs: Models, Data Structures, and Algorithms

Dissertation zur Erlangung des Doktorgrades

vorgelegt am

Fachbereich Mathematik und Informatik
der Freien Universität Berlin

von

Paul Seiferth

2016

Erstgutachter: Prof. Dr. Wolfgang Mulzer
Zweitgutachter: Prof. Dr. Christian Knauer

Tag der Disputation: 19.08.2016

Abstract

Let $P \subset \mathbb{R}^2$ be a set of n point sites. The *unit disk graph* $UD(P)$ on P has vertex set P and an edge between two sites $p, q \in P$ if and only if p and q have Euclidean distance $|pq| \leqslant 1$. If we interpret P as centers of disks with diameter 1, then $UD(P)$ is the intersection graph of these disks, i.e., two sites p and q form an edge if and only if their corresponding unit disks intersect. Two natural generalizations of unit disk graphs appear when we assign to each point $p \in P$ an *associated radius* $r_p > 0$. The first one is the *disk graph* $D(P)$, where we put an edge between p and q if and only if $|pq| \leqslant r_p + r_q$, meaning that the disks with centers p and q and radii r_p and r_q intersect. The second one yields a *directed* graph on P, called the *transmission graph* of P. We obtain it by putting a directed edge from p to q if and only if $|pq| \leqslant r_p$, meaning that q lies in the disk with center p and radius r_p. For disk and transmission graphs we define the *radius ratio* Ψ to be the ratio of the largest and the smallest radius that is assigned to a site in P. It turns out that the radius ratio is an important measure of the complexity of the graphs and some of our results will depend on it.

For these three classes of disk intersection graphs we present data structures and algorithms that solve four types of graph theoretic problems: dynamic connectivity, routing, spanner construction, and reachability oracles; see below for details. For disk and unit disk graphs, we improve upon the currently best known results, while the problems we consider for transmission graphs abstain non-trivial solutions so far.

Dynamic Connectivity. First, we present a data structure that maintains the connected components of a unit disk graph $UD(P)$ when P changes dynamically. It takes $O(\log^2 n)$ amortized time to insert or delete a site in P and $O(\log n / \log\log n)$ worst-case time to determine if two sites are in the same connected component. Here, n is the maximum size of P at any time. A simple variant improves the amortized update time to $O(\log n \log\log n)$ at the cost of a slightly increased worst-case query time of $O(\log n)$.

Using more advanced data structures, we can extend our approach to disk graphs. While the worst-case query time remains $O(\log n / \log\log n)$, an update now requires $O(\Psi^2 2^{\alpha(n)} \log^{10} n)$ amortized expected time, where Ψ is the radius ratio of the disk graph and $\alpha(n)$ is the inverse Ackermann function.

Routing. As the second problem, we consider routing in unit disk graphs. A *routing scheme* R for $UD(P)$ assigns to each site $s \in P$ a *label* $\ell(s)$ and a *routing table* $\rho(s)$. For any two sites $s, t \in P$, the scheme R must be able to route a packet from s to t in the following way: given a current site r (initially, $r = s$), a header h (initially empty), and the target label $\ell(t)$, the scheme R may consult the current routing table $\rho(r)$ to compute

a new site r' and a new header h', where r' is a neighbor of r in $UD(P)$. The packet is then routed to r', and the process is repeated until the packet reaches t. The resulting sequence of sites is called the *routing path*. The *stretch* of R is the maximum ratio of the (Euclidean) length of the routing path produced by R and the shortest path in $UD(P)$, over all pairs of distinct sites in P.

For any given $\varepsilon > 0$, we show how to construct a routing scheme for $UD(P)$ with stretch $1+\varepsilon$ using labels of $O(\log n)$ bits and routing tables of $O(\varepsilon^{-5} \log^2 n \log^2 D)$ bits, where D is the (Euclidean) diameter of $UD(P)$. The header size is $O(\log n \log D)$ bits.

Spanners. Next, we construct sparse approximations of transmission and disk graphs. Let G be a transmission graph. A t-*spanner* for G is a subgraph $H \subseteq G$ with vertex set P so that for any two sites $p, q \in P$, we have $d_H(p, q) \leqslant t d_G(p, q)$, where d_H and d_G denote the shortest path distance in H and G (with Euclidean edge lengths). We show how to compute a t-spanner for G with $O(n)$ edges in $O(n(\log n + \log \Psi))$ time, where Ψ is the radius ratio of P. Utilizing advanced data structures, we obtain a construction that runs in $O(n \log^5 n)$ time, independent of Ψ. This construction can be adapted to disk graphs and gives a t-spanner for $D(P)$ in expected time $O(n 2^{\alpha(n)} \log^{10} n)$, where $\alpha(n)$ is the inverse Ackermann function.

As an application we show that our t-spanner can be used to find a BFS tree in a transmission or disk graph for any given start vertex in $O(n \log n)$ additional time.

Reachability Oracles. Finally, we compute *reachability oracles* for transmission graphs. These are data structures that answer *reachability queries*: given two sites p and q, is there a directed path between them? The quality of an oracle is measured by the space $S(n)$, the query time $Q(n)$, and the preproccesing time. We present three reachability oracles whose quality depends on the radius ratio Ψ: the first one works only for $\Psi < \sqrt{3}$ and achieves $Q(n) = O(1)$ with $S(n) = O(n)$ and preprocessing time $O(n \log n)$; the second data structure gives $Q(n) = O(\Psi^3 \sqrt{n})$ and $S(n) = O(\Psi^3 n^{3/2})$; the third data structure is randomized with $Q(n) = O(n^{2/3}(\log n + \log \Psi))$ and $S(n) = O(n^{5/3}(\log n + \log \Psi))$ and answers queries correctly with high probability.

As a second application for our spanners, we employ them to achieve a fast preprocessing time for our reachability oracles.

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich alle Hilfsmittel und Hilfen angegeben habe und versichere, auf dieser Grundlage die Arbeit selbständig verfasst zu haben. Die Arbeit habe ich nicht in einem frühren Promotionsverfahren eingereicht.

Berlin, den 17. Mai 2016

Acknowledgements

First of all I would like to thank my advisor Wolfgang Mulzer. He literally made my PhD a long journey and on the way he never got tired in teaching me geometry with endless passion, at any time and any place. Beyond this, he also gave me the opportunity to visit numerous exciting places around the world and to experience foreign cultures. Thanks for the wonderful and unique time we spend together, I really appreciate it and it won't be forgotten.

Furthermore, I am very grateful for my second referee Christian Knauer who, without any hesitation, agreed to review my thesis, even after learning about my tight time constraints.

I would like to thank all my coauthors, it has been a pleasure to work with so many brilliant people. Especially, I need thank Liam Roditty and Haim Kaplan. They never run out of challenging and interesting problems to attack and they encouraged me to never stop hunting for more simple and more intuitive arguments. Also their hospitality at our regular visits in Israel was outstanding and it made me feel very welcome. Furthermore, I like to thank Yannik Stein, who was a worthy companion on all the trips we made together and with whom I had a lot of fun at and after work. He participated in all our common projects very diligently and I always enjoyed to work with him, no matter of time and date.

I am much obliged to every single member of the AG Theoretische Informatik. All of them offered me assistance and guidance in any situation of my PhD-life and my life in general. They always managed to cheer me up, especially in the more cumbersome beginning of my studies. I also have to thank them for nurturing my common knowledge day by day with some of the, sometimes, uttermost strange discussions during the coffee round.

Finally, and most of all, I like to thank my family and my friends for their support and for showing interest in what I am doing, even though it is tough to understand what all this is about. In particular, I thank my cohabitant Nadja Scharf. She was one of the major discoveries during my PhD and she managed keep me motivated and focused during the last weeks.

Contents

Chapter 1

Introduction

Since the introduction of wireless networking and the development of the first Wi-Fi devices in the early 1990's, a rapid growth of the wireless industry took place. Nowadays, almost every laptop or cell phone has Wi-Fi capabilities and this trend expands to other devices including printers, television sets, video game consoles, smartwatches [HR16, Jur15, Phi16, Son15] or appliances like personal scales, washing machines, refrigerators, and home brewing machines for beer [Inc16, Kan15, Sch13, Wil16]. This results in a dramatically increasing number of wireless networking devices that have been deployed over the last years. By the end of 2013 the number of network devices even exceeded the number of people on earth [Afs14]. To satisfy the high demand, the industry has developed large-scale manufacturing processes that led to a dramatic decreases in the price of wireless networking devices. This and other technological advances drove the development of small-sized computing nodes with Wi-Fi capabilities. Their extremely low acquisition costs soon inspired researchers to explore their potential outside the consumer area. Connecting a multitude of these nodes via wireless links forms what we call a *sensor network* [Bou08, Sto05]. Typical applications of sensor networks occur in situations when hardly accessible and sensitive environments need to be monitored. Each computing node is equipped with a sensor that can measure environmental parameters such as temperature, humidity, sound, chemicals, or the presence of certain objects [Sto05]. The measured data is sent to a central data gathering station that collects and processes the information. This is done either via direct links to the gathering station, or more often, in a multi-hop fashion using the underlying sensor network. Some domains where sensor networks are used successfully include military scenarios, environmental applications, and health monitoring [HS95, Sto05]. More concrete examples and further applications can be found in the Handbook of Sensor Networks [Sto05].

In order to keep sensor nodes cost-effective, their hardware is usually very limited. Furthermore sensor nodes are supposed to work as stand-alone units and thus their power supply is often battery based. As a consequence, sensor networks face several design challenges, for example

Unstable Topology It is likely that sensor nodes fail. In the simplest case they run out of battery, but other reasons like hardware failure or intermittent interference are also conceivable. On the other hand, sensor nodes are cheap and thus there is no obstacle to extend the network whenever necessary. Both issues cause frequent topology changes that need to be handled.

Energy Efficiency Since sensor nodes are battery powered, there is a great need for energy efficiency. This is achieved by avoiding expensive local computations and by limiting the transmission range of the wireless adapters.

Limited Computational Power and Memory In most cases, sensor network nodes are equipped with cheap and energy efficient processors that suffer from a lower performance. For the same reasons, the installed memory is kept as low as possible.

In this thesis, we try to tackle these difficulties by studying sensor networks from a *theoretical* point of view. We develop several data structures and algorithms in the context of sensor networks that meet some of their requirements. Our first data structure is dynamic and can maintain the topology of a sensor network when nodes are added or removed. Then, we show how to route a packet between two arbitrary nodes in the network. Our routing scheme requires a very small amount of storage at each node and nevertheless each packet will be sent along an approximately shortest path to its destination. For a node to make a routing decision, only a quite simple and efficient local computation must be done. Our next result is a way to approximate the whole topology of a sensor network in a space efficient manner while preserving the distances in the best possible way. This yields a succinct representation of the network that requires only little storage compared to storing the whole topology. Our last data structure can answer for any two sensor nodes if they are connected by a path through the network or not. It can be used to verify the existence of a path to the destination before sending the packet. This avoids an expensive and energy consuming usage of the network, just to send packets that cannot arrive at its destination in the first place.

The data structures and algorithms we describe will be judged by all quality standards of theoretical computer science: we will prove their correctness and analyze their running times. In order to do so, we need an abstract, mathematical model that describes a sensor network accurately. We will always represent the network itself as a geometric graph, but in what follows we will describe several models that determine which edges are present in this graph. All of these models will have some geometric flavor and there will be an focus on three models that are based on intersection patterns of disks. All our data structures and algorithm will be developed and analyzed for these three disk based models.

1.1 Modeling Sensor Networks

Even though physical sensor networks operate in three-dimensional environments, we use two-dimensional models to describe them. As we will see throughout this thesis, already

the two-dimensional cases capture most of the difficulties that arise when studying sensor networks. Furthermore, in most of the practical applications the sensor nodes are placed on an (almost) flat surface and thus two-dimensional models are sufficient.

For any concrete model we think of each sensor node as a single point in the Euclidean plane. We denote by P the set of points corresponding to the sensor nodes and we call these points *sites*. Around each site $p \in P$ there is an area where p the signal of p can reach to, i.e., the other sensors p can communicate with must lie in this area. Since radio signals emitted from isotropic antennas spread radially from the source, we often think of these areas as disks with a certain radius. It is known that this is a simplification that might not reflect the behavior of the network accurately enough in some situations.

Next we introduce the three disk-based models that are considered in this thesis and below we give further models that are intended to overcome the shortcomings of disk-based models.

Three Models Based on Disk Intersection Graphs

One of the most basic models for sensor networks is the *unit disk graph*. For this we assume that all sensors are equivalent and that each sensor site $p \in P$ has a disk around it with unit diameter. We draw an edge between two sites if and only if the corresponding disks intersect. An example of a unit disk graph is shown in Figure 1.1 (left). Equivalently, we can say that there is an edge between two sites if and only if their Euclidean distance is at most 1, which is the reason why unit disk graphs are also known as *unit distance graphs*. Unit disk graphs have been widely studied in the theoretical computer science community [AW05, BBK16, BK98, CCJ90, CJ15, KM12, KMRS16a, Mar06].

In some scenarios, the assumption that all sensor nodes are equivalent can be too restrictive. For instance, one can consider the following tree-like topology: we have groups of cheap devices that measure some environmental data. All devices within one group report their measurements to a central gateway node of the group. The gateway node is equipped with more expensive hardware that allows it to send data over longer distances, e.g., to a base station that is far away. To cover such cases, we extend the model of unit disk graphs and we assign to each site $p \in P$ a *transmission radius* $r_p > 0$. Instead of a unit diameter disk, the transmission area of a site p is the disk with radius r_p and with center p. Two sites are connected by an edge if and only if their corresponding disk intersect, see Figure 1.1 (center). This second model is called *disk graph* and unit-disk graphs are a special kind of disk graphs where each transmission radius is $1/2$.

A huge drawback of disk graphs is that all connections are symmetric while in some situations an asymmetric connection is more realistic. Consider for instance the leftmost site in the disk graph in Figure 1.1 (center). Even though its transmission radius is very small, it can still send data to the distant site with the largest disk. In other words, with disk graphs a site can transmit its signal over arbitrarily long distances, as long as the transmission radius of the *target* site is large enough. To avoid such unrealistic behavior, we introduce *transmission graphs* as third model that leads to directed graphs. To obtain a transmission graph, again each site p has its transmission radius r_p. This

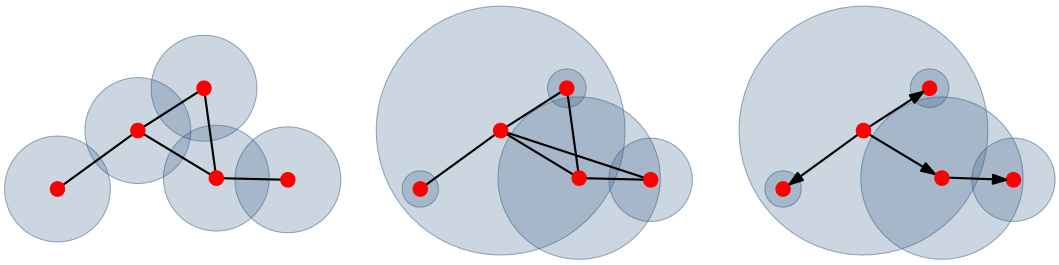


Figure 1.1: The three models of disk intersection graphs: unit disk graphs (left), disk graphs (center), and transmission graphs (right).

time we have a *directed* edge from p to another site q if and only if q lies in the disk around p with radius r_p, see Figure 1.1 (right). Note that setting each radius to 1 yields the same graph as the unit disk graph and thus transmission graphs constitute another generalization of unit disk graphs.

Further Models

In some real world scenarios disk graphs tend to be too unrealistic. The oversimplification in the models might not reflect the physical behavior of the network accurately enough and might not be able to represent more complex topologies. For these reasons, there are more involved models which turn out to be more realistic in some situations. The following examples are intended to give an idea of how such models can be defined. A more exhaustive list can be found in the book of Boukerche [Bou08].

Since most of the networking protocols, like TCP/IP, require a bidirectional communication already to establish a connection, the unidirectional links created by transmission graphs are of no use when such protocols are used. For such cases there is a version of transmission graphs called *undirected transmission graphs* where we keep only the edges between pairs of sites where both directed edges exist, i.e., we have an *undirected* edge between p and q if and only p lies in the disk around q with radius r_q and q lies in the disk around p with radius r_p.

Overcoming long distances with radio signals emitted from isotropic antennas requires a high power consumption at the sender. To increase the energy efficiency in these situations, one typically uses dipole antennas that bundle the signal in one direction. To model dipole antennas we use the *directed antenna model*: we fix a cone C that will define the area a site can send signals to. At each site p we attach a translated copy C_p of C such that the apex of C_p is p. Furthermore, each site p has an assigned angle α_p that determines the direction to which C_p points. Then we have a directed edge from a site p to another site q if and only if q lies in C_p.

The last type of models emphasizes the fact that wireless networks in real world environments are disturbed by different noises and interference between the senders. It it based on the *signal-to-interference-plus-noise ratio* (SINR) that measures for a receiving site $q \in P$ the quality of the signal received by a sending site $p \in P$ using the

following formula:

$$\mathsf{SINR}(p,q) = \frac{\frac{r_p}{|pq|^\alpha}}{\sum_{t \in P \setminus \{p,q\}} \frac{r_t}{|tq|^\alpha} + N},$$

where $\alpha, N > 0$ are given constants that represent hardware specific antenna parameters and the background noise, respectively. The numerator accounts for the fact that quality of radio signals drops with increasing distance and the sum in the denominator measures the interference caused by other senders. In the SINR-model we have a directed edge from a site p to a site q if and only $\mathsf{SINR}(p,q) > \beta$ for some fixed threshold $\beta > 0$. A more practical version is the k-SINR model where only the k most significant senders are considered to compute the SINR value of a pair p, q. The adapted formula is

$$\mathsf{SINR}_k(p,q) = \frac{\frac{r_p}{|pq|^\alpha}}{\max_{Q \subseteq P \setminus \{p,q\}, |Q|=k} \sum_{t \in Q} \frac{r_t}{|tq|^\alpha} + N}$$

and we again have a directed edge from p to q if and only if $\mathsf{SINR}_k(p,q)$ exceeds some fixed threshold.

1.2 The Problems Considered

In this thesis we consider several classic algorithmic problems on graphs and consider their applicability to unit disk graphs and transmission graphs. Our hope is that the geometry of these graphs will reveal a deeper structure that helps to develop efficient solutions. For the following problems we present data structures and algorithms that will be faster and often even simpler than the state of the art solutions for general graphs.

Dynamic Connectivity for Unit Disk Graphs. Let $P \subset \mathbb{R}^2$ be a set of sites and let $\mathrm{UD}(P)$ be the corresponding unit disk graph. We want a data structure that maintains the connectivity of $\mathrm{UD}(P)$ when the set P changes dynamically. More precisely, our data structure must support insertions and deletions of sites in P and must be able to answer *connectivity queries*: given two sites $s, t \in P$, are s and t in the same connected component of $\mathrm{UD}(P)$? The quality of such a data structure is measured by the update and query time and by the required storage.

Routing in Unit Disk Graphs. Next, we want to route a packet through a connected unit disk graph, using only a small amount of *local* information at each routing step. To measure the distance the packet travels during the routing, we weight each edge in $\mathrm{UD}(P)$ by its Euclidean length and we define the length of a path π in $\mathrm{UD}(P)$ as the sum of the edge weights along π.

Our goal is to preprocess $\mathrm{UD}(P)$ into a *routing scheme* R. The scheme assigns to each site $s \in P$ a *label* $\ell(s)$ and a *routing table* $\rho(s)$. Furthermore, the scheme R must be able to route a packet from any given start site $s \in P$ to any target site $t \in P$ in the following way: given a current site r (initially, $r = s$), a *header* h (initially empty), and a *target*

label $\ell(t)$, the scheme R can use $\rho(r)$, h, and $\ell(t)$ to compute a new site r' and a new header h', where r' must be a neighbor of r in $UD(P)$. The packet is then routed to r' and the process is repeated. The scheme R must guarantee that the packet eventually arrives at t. The sequence of sites that are visited during the routing process is called the *routing path*. The *stretch* of R is the maximum ratio of the length of the routing path produced by R and the shortest path in $UD(P)$, over all pairs of sites $s, t \in P$, $s \neq t$. In other words, the stretch describes the length of the maximum detour the packet can take when using the routing path instead of the shortest path.

The quality of a routing scheme R is measured by the size of the routing tables and labels produced by R, the maximum header size that is required during the routing, and the stretch of R.

Spanner Construction. Let $P \subset \mathbb{R}^2$ be a set of sites, each site $p \in P$ having an assigned radius and let G be the transmission graph induced by P. The graph G can be quite dense and in the worst-case it can even be the complete graph. Thus, it is desirable to have a sparse approximation for G that preserves the distances in G. Such an approximation is called a *spanner*. Fix a parameter $t > 1$. A subgraph H of G is a t-*spanner* for G if

(i) H is sparse, i.e., has a linear number of edges, and

(ii) for any pair of sites $p, q \in G$ the shortest p-q-path in H is a most t times as long as the shortest p-q-path in G.

To measure the length of the path we weight each edge in G and H by its Euclidean length. We consider two major questions related to spanners for transmission graphs: is there always a t-spanner for every transmission graph and every value of t? And if yes, how fast can it be computed?

Reachability Oracles. Similar to the dynamic connectivity structure problem for unit disk graphs, we also want to store the connectivity of a transmission graph in a space efficient manner. For transmission graphs already the static case turns out to be a challenging problem.

Let G be the transmission graph of a set of sites $P \subset \mathbb{R}^2$. We say that a site $p \in P$ can *reach* a site $q \in P$ if there is a directed path from p to q in G. A *reachability oracle* for G is a data structure that can decide for any two given query sites $p, q \in P$ if p can reach q. The quality of a reachability oracle is determined by the query time, the used space, and the time required to compute it.

The geometry of transmission graphs allows for a more general type of query where the target is not necessarily a site in P but it can be an arbitrary point in $\mathbb{R}^2$. We say that a site $p \in P$ can reach a *point* $t \in \mathbb{R}^2$ if there is a site q such that p can reach q and such that t lies in the disk around q with radius r_q (cf. Figure 1.2). A reachability oracle that can answer this type of queries is called a *geometric* reachability oracle. Ideally we would like our oracles to be geometric.

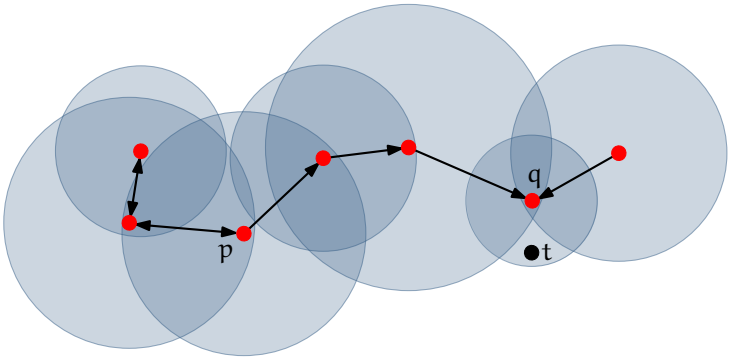


Figure 1.2: The site p can reach the site q by following the directed path. Also p can reach the black *point* t since t is in the disk of q.

1.3 Further Problems on Disk Intersection Graphs

Unit disk graphs have been well-studied in a multitude of different algorithmic contexts. A recent example is due to Cabello and Jejčič, who gave fast algorithms to compute BFS trees and shortest path trees in unit disk graphs [CJ15]. Furthermore, most classic NP-complete graph problems turned out to be NP-complete for unit disk graphs as well. This list includes computing the chromatic number, a maximum independent set, a maximum connected dominating set, finding hamiltonian cycles and paths, and finding steiner trees. For a more detailed overview, see Clark et al. [CCJ90] and the references therein. The dominating set and connected dominating set problems are even W[1]-hard for unit disk graphs when parametrized by the size of the dominating set [Mar06,BBK16]. A notable exception from this list is the maximum clique problem. Already Clark et al. gave a simple polynomial time algorithm to compute the maximum clique in unit disk graphs [CCJ90]. More interestingly, for disk graphs it is not known if the maximum clique problem is NP-hard or if there is a polynomial time algorithm. The best known result is a 1/2-approximation algorithm by Ambühl and Wagner [AW05].

All the problems described so far in this and in the previous section have a common flavor: our input is a unit disk or transmission graph and we want to find or compute a certain structure. However, there are a lot of problems where it is reversely and we want to find a graph that fulfills a given set of requirements. One problem of this kind is *realizability* of unit disk graphs: given an abstract graph G, is there a set $P \subset \mathbb{R}^2$ of sites such that the unit disk graph $UD(P)$ is isomorphic to G? Breu and Kirckpatrick showed that this problem is NP-hard but it is still unknown whether it is in NP [BK98]. Kang and Müller found (abstract) graphs such that each realization as an unit disk graph requires exponentially many bits to describe it and thus showing containment in NP would require some non-trivial insights in the structure of the problem. Recently, a complexity class called *existential theory of the reals* ($\exists\mathbb{R}$ for short) received more and more attention in

the graph drawing community, see the survey paper of Matoušek for a comprehensive overview of the class $\exists\mathbb{R}$ [Mat14]. This complexity class is a superclass of NP and a subclass of PSPACE. It seems that $\exists\mathbb{R}$ is well suited to capture hardness of graph drawing and realizability problems since containment in $\exists\mathbb{R}$ is often almost trivial, as it is the case for unit disk graphs. Furthermore, using polynomial time reductions, we can show that unit disk graph realizability is even $\exists\mathbb{R}$-complete: Matoušek showed that stretchability of pseudoline arrangements is $\exists\mathbb{R}$-hard [Mat14] and Kang and Müller reduced pseudoline stretchablity to realizability of unit disk graphs [KM12]. The reduction of Kang and Müller can seems not to be limited to unit disk and could most likely be extended to transmission graphs with only a few modifications. Thus, we conjecture that realizability of transmission graphs is $\exists\mathbb{R}$-complete as well.

Transmission graphs were also considered in the context of *range assignment* problems [FWZ04]. Here, we are given a set of sites $P \subset \mathbb{R}^2$ and the goal is to pick a radius r_p for each $p \in P$ such that the resulting transmission graph fulfils certain properties (usually related to connectivity) while minimizing some function that depends on the radii. One concrete instance is to minimize the sum of radii $\sum_{p\in P} r_p$, while having a transmission graph that has diameter h, i.e., for any two sites $p, q \in P$ there is a directed path from p to q that uses at most h edges. Clementi et al. [CPS04] proved that the problem is NP-hard for the special case $h = n - 1$, where n is the size of P. The complexity of the problem for other values of h is still unknown. Nevertheless, the $h = n - 1$ case has been well studied: Kirousis et al. [KKKP00] gave a polynomial time 2-approximation algorithm that later was improved to a $(1.5-\varepsilon)$-approximation for a small constant $\varepsilon > 0$ by Carmi and Chaitman-Yerushalmi [CC15]. The one-dimensional problem, where all sites in P are collinear, has also been studied and can be solved optimally in polynomial time [DGN07, KKKP00] where the best known algorithm runs in time $O(n^2)$ [CC15]. When the parameter h is constant, Carni et al. gave a $(6 + \varepsilon)$-approximation for the planar case and a 1.5-approximation in 1D [CCT15].

Other reasonable objective functions to study are related to the the maximum indegree of a site $p \in P$, since the indegree translates to the interference at the sensor node corresponding to p. Brise et al. [BBE+14] showed that minimizing the maximum indegree is NP-complete for the case $h = n - 1$. For the one-dimensional case they gave an algorithm with quasipolynomial running time $2^{O(\log^2 n)}$. As with previous range assignment problems, the complexity for the planar case for other values of h is not known and it constitutes an interesting question for further research.

1.4 Results and Organization of the Thesis

We now explain our results and the techniques we use. At the same time, we give an overview of the thesis and how the different parts are connected.

The next chapter contains important preliminary notations and definitions that are used throughout the thesis. Furthermore, we review some well-known geometric tools including well-separated pair decompositions and dynamic data structures for Euclidean and additively weighted nearest neighbor search, as we are going to use them for some

of our results. After this preliminaries chapter, the thesis is divided into two parts: the first part contains our results for unit disk graphs and the second part is devoted to transmission graphs. We will see that some of our solutions can be extended to disk graphs.

In Chapter 3 in the first part of the thesis we present a data structure for dynamic connectivity of unit disk graphs. We obtain this data structure by a delicate combination of known data structures with several geometric observations. In its most basic form our structure can insert or delete a disk in $O(\log^2 n)$ amortized time and a connectivity query can be answered in $O(\log n / \log\log n)$ worst-case time (Theorem 3.2). Here n is the maximum number of disk that is stored in the data structure at any time. Then we use a planarization argument for unit disk graphs (see Lemma 3.8 and Lemma 3.9) that allows us to work on a planar graph and substitute the data structure responsible for the $O(\log^2 n)$ bottleneck by its planar counterpart. This yields a dynamic connectivity structure with amortized update time $O(\log n \log\log n)$ but with a slightly increased worst-case query time of $O(\log n)$ (Theorem 3.7). Our techniques extend to disk graphs as long as the *radius ratio* Ψ is bounded, meaning that all radii are in the interval $[1, \Psi]$. In this case, we can achieve an amortized expected update time of $O(\Psi^2 2^{\alpha(n)} \log^{10} n)$ with $O(\log n / \log\log n)$ worst-case query time, where $\alpha(n)$ is the inverse Ackermann function (Theorem 3.11). The rather high exponent of the logarithm and the $2^{\alpha(n)}$ term come from the fact that this extension requires a dynamic additively weighted nearest neighbor structure as given by Corollary 2.7.

In Chapter 4 we present a routing scheme for unit disk graphs. For any $\varepsilon > 0$ we can preprocess a unit disk graph with n sites into a routing scheme with stretch $1 + \varepsilon$. For this we need labels with $O(\log n)$ bits, routing tables with $O(\varepsilon^{-5} \log^2 n \log^2 D)$ bits, and the maximum header size during the routing process is $O(\log n \log D)$ bits, where D is the diameter of the unit disk graph when edges are weighted by their Euclidean lengths (Theorem 4.13). The main tool we use is the compact well-separated pair decomposition for the unit disk metric from Gao and Zhang [GZ05].

In the second part of the thesis we consider transmission graphs. In Chapter 5 we provide an efficient way to construct spanners for transmission graphs. For this, we define a decomposition of the graph that is heavily inspired by the way well-separated pair decompositions work (Definition 5.2). From this decomposition we can easily read off a set of edges that, for any given $t > 1$, yield a t-spanner for the transmission graph (Lemma 5.8). Then we show how to find this decomposition and the required edges efficiently in the restricted case that the radius ratio Ψ is bounded. The running time will be $O(n(\log n + \log \Psi))$ (Theorem 5.12). In the next step we make this construction independent of Ψ. To achieve this we first use the classic well-separated pair decomposition algorithm to compute our decomposition of the transmission graph. Then we employ a dynamic Euclidean nearest neighbor structure as given by Corollary 2.5 to find the edges quickly. The total running time is $O(n \log^5 n)$. This spanner construction extends to disk graphs using an additively weighted nearest neighbor structure (as in Corollary 2.7) instead of the Euclidean one. The time required to construct a t-spanner for a disk graph is $O(n 2^{\alpha(n)} \log^{10} n)$. We also give two applications of our spanners: first we use

the spanner to find a BFS tree from any given start site s in additional time $O(n \log n)$, once the spanner is computed (Theorem 5.23). Second, we show that with the help of the spanner, we can transform any reachability oracle for a transmission graph into a geometric reachability oracle (Theorem 5.25). This allows us in the last chapter to focus on constructing oracles for reachability queries while ignoring the geometric queries.

In the last chapter we will see three different geometric reachability oracles, each performing best for a certain radius ratio. The first reachability oracle works only for $\Psi < \sqrt{3}$. In this case, we can use a similar planarization argument as we did for unit disk graphs, though the proof is much more involved (see Lemma 6.5 and Lemma 6.6). This yields a planar graph on which we can use known results for planar reachability [HRT14] to obtain a geometric reachability oracle with $O(n)$ space and $O(\log n)$ worst-case query time, or $O(1)$ worst-case query time for *non-geometric* queries (Theorem 6.2). The second oracle has polynomial dependence of Ψ: it needs $O(\Psi^3 n^{3/2})$ space and has worst-case query time $O(\Psi^3 \sqrt{n})$ (Theorem 6.8). For this we use a separator theorem for disks by Alber and Fiala [AF04] that is based on the planar separator theorem [LT80]. The last oracle decreases the dependence on Ψ to be logarithmic while slightly increasing the dependence on n. It needs $O(n^{5/3} \log^{1/3} \Psi \log^{2/3} n)$ space and can answer a query correctly with high probability in time $O(n^{2/3} \log^{1/3} \Psi \log^{2/3} n)$. The randomized queries come from random sampling techniques that are used to find a small hitting set for all long paths in the graph (Lemma 6.12).

Preliminary versions of some of these results were already published as parts in the following papers:

[KMRS15] Haim Kaplan, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. Spanners and reachability oracles for directed transmission graphs. In *Proc. 31st Int. Sympos. Comput. Geom. (SoCG)*, pages 156–170, 2015.

[KMRS16a] Haim Kaplan, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. Dynamic connectivity for unit disk graphs. In *Proc. 32nd European Workshop on Comput. Geom. (EWCG)*, 2016.

[KMRS16b] Haim Kaplan, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. Routing in unit disk graphs. In *Proc. 12th Latin American Theoretical Informatics Symposium (LATIN)*, pages 536–548, 2016.

[KMR+16] Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. Dynamic planar Voronoi diagrams for general distance functions and their algorithmic applications. *arXiv:1604.03654 [cs.CG]*, 2016.

Chapter 2

Preliminaries

In this chapter we introduce basic notations and concepts that we are going to use throughout the thesis. In particular, we define planar grids as we frequently use them in later chapters to approximate and structure the intersection graphs. Furthermore, we define several kinds of well-separated pair decompositions, as they are required in Chapters 4 & 5.

Finally, we review data structures for the dynamic maintenance of lower envelopes and their connection to various versions of the dynamic nearest neighbor problem. These data structures will play a crucial role in Chapter 3 and Chapter 5.

2.1 Notation and Planar Grids

Unless otherwise stated, we let $P \subset \mathbb{R}^2$ denote a planar n-point set, and we assume that for each point $p \in P$ we have an *associated radius* $r_p > 0$. Furthermore, we usually assume that the input is scaled so that the smallest radius in P is 1. The elements in P are called *sites* or *vertices*. Given a point $p \in \mathbb{R}^2$ and a radius r, we denote by $D(p, r)$ the closed disk with center p and radius r. If $p \in P$, we use $D(p)$ as a shorthand for $D(p, r_p)$. We write $C(p, r)$ for the boundary circle of $D(p, r)$.

We define three graphs with P as vertex set: the *unit disk graph* $UD(P)$ has an edge between $p, q \in P$ if and only if they have Euclidean distance $|pq| \leqslant 1$; the *disk graph* $D(P)$ has an edge between $p, q \in P$ if and only if $|pq| \leqslant r_p + r_q$ (i.e., $D(p) \cap D(q) \neq \emptyset$); and the *transmission graph* has a *directed* edge from p to q if and only if $|pq| \leqslant r_p$ (i.e., $q \in D(p)$). We consider these graphs as geometric graphs embedded in the plane such that each vertex lies on its corresponding point in P and such that the edges are drawn as straight line segments. Furthermore, we assign to each edge pq its Euclidean length $|pq|$ as weight. For a weighted graph G and two vertices p, q of G we denote by $d_G(p, q)$ the length of a shortest path from p to q in G. If p and q are not connected in G we set $d_G(p, q) = \infty$. Let $t > 1$ be a parameter. A subgraph $H \subseteq G$ is called a t-*spanner* for G if for each pair of vertices $p, q \in G$ we have $d_H(p, q) \leqslant t \cdot d_G(p, q)$.

The *spread* Φ of P is defined as $\Phi = \max_{p,q\in P} |pq| / \min_{p\neq q\in P} |pq|$, and the *radius ratio* Ψ of P is defined as $\Psi = \max_{p,q\in P} r_p/r_q$. The next observation shows that for transmission graphs we can always assume $\Psi \leqslant 2\Phi$.

Observation 2.1. *Let $P \subset \mathbb{R}^2$ be a set of sites with spread Φ and let G be the transmission graph of P. We can assign to each site $p \in P$ a new radius such that G does not change and such that $\Psi \leqslant 2\Phi$.*

Proof. Let d_{min} be the smallest and d_{max} be the largest distance between any two sites in P, i.e., $\Phi = d_{max}/d_{min}$. All sites $p \in P$ with $r_p < d_{min}$ cannot have an outgoing edge in G. Thus we can set these radii to $d_{min}/2$.

Let $p \in P$ be a site with $r_p \geqslant d_{max}$. Then we have $P \subseteq D(p)$, i.e., p has an outgoing edge to all other sites in P. If we set $r_p = d_{max}$ we still have $P \subseteq D(p)$ and G remains unchanged. If we do this for all sites with radius larger than d_{max}, we have $\Psi \leqslant 2d_{max}/d_{min} = 2\Phi$. □

Our constructions make extensive use of planar grids. For $i \in \{0, 1, \dots\}$, we define $\mathcal{G}_i$ to be the *grid at level* i. It consists of axis-parallel squares with diameter 2^i that partition the plane in grid-like fashion (the *cells*). We write $\mathrm{diam}(\sigma)$ for the diameter of a grid cell σ. Each grid $\mathcal{G}_i$ is aligned so that the origin lies at the corner of a cell. The *distance* $d(\sigma, \tau)$ between two grid cells σ, τ is the smallest distance between any pair of points in $\sigma \times \tau$, see Figure 2.1. We assume that our model of computation allows us to find in constant time for any given point the grid cell that contains it.

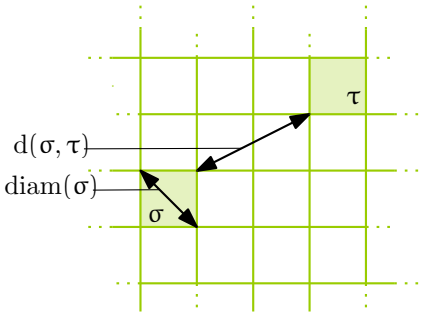


Figure 2.1: The grid (green) and two cells σ and τ.

2.2 Well Separated Pair Decompositions

Let $P \subset \mathbb{R}^2$ be an n-point set and let $c > 1$ be a parameter. We call two subsets $A, B \subseteq P$ c-*well-separated* if $c \max\{\mathrm{diam}(A), \mathrm{diam}(B)\} \leqslant d(A, B)$, where $\mathrm{diam}(\cdot)$ denotes the diameter of a set and $d(A, B)$ is the minimum distance between any pair a, b with $a \in A$ and $b \in B$. If A and B are well-separated, then any pair a, b is an approximation of the $|A| \cdot |B|$ distances in $A \times B$, see Figure 2.2. A c-*well-separated pair decomposition* (c-WSPD) is a set of pairs $(A_1, B_1), \dots, (A_m, B_m)$ such that for all $1 \leqslant i \leqslant m$ we have that (i) $A_i, B_i \subseteq P$, (ii) the pair (A_i, B_i) is c-well separated, and (iii) the sets $A_i \times B_i$ form a partition of $P \times P$.

WSPDs have been introduced to the computational geometry community by Callahan and Kosaraju, and since then they have proven to be a useful tool for the design of efficient geometric algorithms and data structures [CK95a]. Callahan and Kosaraju

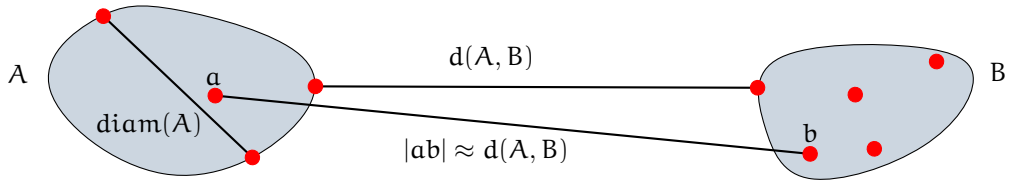


Figure 2.2: The sets A and B are c-well-separated.

show that for any $c > 1$ there is always a c-WSPD with $m = O(c^2 n)$ pairs and that it can be computed efficiently. Thus, we can consider WSPDs as a succinct approximation of the $\binom{n}{2}$ distances between sites in P. Note that even though we need only $O(c^2 n)$ pairs, the total size (number of elements in the sets) of the WSPD may be quadratic. Therefore, we need to represent the sets *implicitly*, as we will see, e.g., in Section 4.2 or in Section 5.2.3.

Some prominent problems solvable with WSPDs include distance selection [AMS94], dynamic closest pair [CK95b], and computing (high dimensional) approximate Euclidean minimum spanning trees [CK93] as well as several other geometric structures (see, e.g., Löffler and Mulzer [LM12] and the references therein).

For this thesis it is more relevant to note that there are spanner constructions that rely on WSPDs. In fact, the book on geometric spanners by Narasimhan and Smid dedicates more than two chapters to such constructions [NS07]. We now review the WSPD spanner, the most elementary spanner construction that directly uses the WSPD. It illustrates some of the ideas that will reappear in the routing scheme in Chapter 4. Furthermore, the general approach serves as the starting point for our c-separated annulus decomposition (Definition 5.2) which is the cornerstone in all spanner constructions given in Chapter 5.

2.2.1 The WSPD Spanner

Let $P \subset \mathbb{R}^2$ be an n-point set and let $t > 1$ be a parameter. Our goal is to find a t-spanner for the complete Euclidean graph G on P. For this, we choose a large enough parameter $c = c(t)$, depending on t, and we compute a c-WSPD $\Xi = \{(A_1, B_1), \ldots, (A_m, B_m)\}$ for P with $m = O(c^2 n)$ pairs. We construct a subgraph H of G with vertex set P and the following edges. For each pair $(A_i, B_i) \in \Xi$, we take two arbitrary points $a \in A_i$ and $b \in B_i$ and we add the edge ab to H. We have $O(c^2 n)$ edges in total and we now show that H is a t-spanner for G.

Lemma 2.2. *Let $c = (2t + 2)/(t - 1)$. For any two sites $p, q \in P$, we have $d_H(p, q) \leqslant t d_G(p, q) = t|pq|$.*

Proof. We use induction on the rank of the distance $|pq|$ in the sorted list of all distances in $P \times P$. For $|pq| = 0$, the claim is trivially true.

Now, assume that for all sites $x, y \in P$ with $|xy| < |pq|$ we have an x-y-path in H of length at most $t|xy|$. Let $(A, B) \in \Xi$ be the pair with $p \in A$ and $q \in B$. Then there is an

edge ab in H with $a \in A$ and $b \in B$ and the distance from p to q in H is bounded by

$$d_H(p,q) \leqslant d_H(p,a) + |ab| + d_H(b,q). \tag{2.1}$$

Since A, B are c-well-separated, $p, a \in A$, $b, q \in B$, and since $c > 1$, we get

$$|pq| \geqslant d(A,B) \geqslant c \max\{\operatorname{diam}(A), \operatorname{diam}(B)\} > \max\{|pa|, |bq|\} \tag{2.2}$$

and thus $d_H(p,a) \leqslant t|pa|$ and $d_H(b,q) \leqslant t|bq|$ by induction. Plugging these bounds into (2.1) and using the triangle inequality yields

$$\begin{aligned} d_H(p,q) &\leqslant t|pa| + |ab| + t|bq| \\ &\leqslant t|pa| + |pa| + |pq| + |bq| + t|bq| \\ &\leqslant (2t+2)\max\{\operatorname{diam}(A), \operatorname{diam}(B)\} + |pq| \\ &\leqslant \big(1 + (2t+2)/c\big)|pq| = t|pq| \qquad \text{(by 2.2)}, \end{aligned}$$

by our assumption that $c = (2t+2)/(t-1)$. □

2.2.2 WSPDs for General Metric Spaces

Even though WSPDs are usually used for points in $\mathbb{R}^d$ with the Euclidean metric, it is natural to extend the concept to more general settings. A *finite metric space* is a pair (S, δ) where S is a finite set and δ is a function $\delta : S \times S \to \mathbb{R}$, called a *metric*, such that for any $p, q, r \in S$ the following holds:

identity of indiscernibles $\delta(p,q) = 0 \Leftrightarrow p = q$
symmetry $\delta(p,q) = \delta(q,p)$
triangle inequality $\delta(p,r) \leqslant \delta(p,q) + \delta(q,r)$

As in the Euclidean case, for two non-empty subsets $A, B \subseteq S$ we define $\operatorname{diam}(A) = \max_{p,q \in A} \delta(p,q)$ to be the diameter of A and $\delta(A,B) = \min_{p \in A, q \in B} \delta(p,q)$ to be the distance between A and B. The definition of two sets being c-well-separated then extends naturally to metric spaces. Unfortunately, unlike in the Euclidean case, not all metric spaces admit small-sized WSPDs. For example consider a finite set S with the following distance function:

$$\delta(p,q) = \begin{cases} 0, & \text{if } p = q \\ 1, & \text{if } p \neq q \end{cases}$$

Observe that δ is a metric for S and that for *any* $c > 1$ only singleton sets $A, B \subseteq S$ can be c-well-separated. Thus, since the WSPD pairs in Ξ need to partition $S \times S$, each of the $|S|^2$ tuples $(p,q) \in S \times S$ need to form a pair $(\{p\},\{q\}) \in \Xi$. However, for certain interesting metrics small WSPDs exist. The first example relevant to this thesis is the *unit disk metric*. Let $P \subset \mathbb{R}^2$ be a set of n sites in the plane and let $G = \mathrm{UD}(P)$. For $p, q \in P$, we let $\delta(p,q)$ be the length of a shortest path between p and q in G, i.e., $\delta = d_G$. Gao and Zhang showed that if $\mathrm{UD}(P)$ is connected, then the metric δ for

$UD(P)$ admits a c-WSPD with $O(c^2 n \log n)$ pairs, and these pairs can be computed in $O(c^2 n \log n)$ time [GZ05].

The second interesting example are *doubling* spaces. Let (S, δ) be a metric space. For $p \in S$ and $r \in \mathbb{R}$, let $B(p, r) = \{q \in S \mid \delta(p, q) \leqslant r\}$ be the *ball with radius* r around p. The *doubling dimension* of (S, δ) is the smallest positive integer D such that for any $p \in S$ and any $r \in \mathbb{R}$, we can find a set of 2^D balls of radius $r/2$ whose union contains $B(p, r)$. Har-Peled and Mendel [HM06] showed that spaces with doubling dimension D admit always a c-WSPD with $c^{O(D)} n$ pairs that can be computed in $2^{O(D)} n \log n + c^{O(D)} n$ expected time. Since S is finite, the approach from Section 2.2.1 also leads to a spanner for the complete graph on S when edges are weighted according to δ [HM06].

2.3 Dynamic Lower Envelopes

A lot of effort in computational geometry has been invested in the study of lower envelopes of (linear) functions in the last decades [SA96]. Two reasons for this are the well-known duality between lower envelopes of hyperplanes and convex hulls of point sets and the connection between d-dimensional Voronoi diagrams and lower envelopes of $(d+1)$-dimensional hyperplane arrangements [BCvKO08]. In this section we review several data structures for dynamically maintaining lower envelopes of (pseudo-)line arrangements in the plane and of arrangements of planes and more general functions in $\mathbb{R}^3$. These data structures will find applications in Chapter 3 and Chapter 5.

2.3.1 The Planar Case

Let L be a set of non-vertical lines in the plane. The *lower envelope* of L is the pointwise minimum of the functions whose graphs constitute the lines in L. A dynamic data structure that maintains the lower envelope of L supports the following operations: inserting a line into L, deleting a line from L, and performing a *vertical ray shooting query*. For a vertical ray shooting query we are given an x-coordinate and we want to determine the line $\ell \in L$ whose function attains the minimum at x, i.e., ℓ is the first line we hit when shooting an upward vertical ray from $(x, -\infty)$.

The first efficient such data structure dates back to 1980 when Overmars and van Leeuwen solved the problem with $O(\log^2 n)$ worst-case time per operation [OvL80, OvL81]. More than 20 years later, Chan improved this to amortized time $O(\log^{1+\varepsilon} n)$ for updates and queries [Cha01], where $\varepsilon > 0$ can be made arbitrarily small. Kaplan et al. approached the problem in a purely combinatorial way, without insisting on the geometry of the lines [KTT01]. They called the resulting data structure *parametric heap* and it solves the dynamic lower envelope problem for lines with worst-case query time $O(\log n)$ and amortized update time $O(\log n \log\log n)$. Independently, Brodal and Jacob also achieved the same time bound [BJ00] before they eventually settled the problem by achieving the optimal amortized update time of $O(\log n)$ with $O(\log n)$ worst-case query time [BJ02]. Their solution is fairly involved and it is an interesting task to substantially

simplify it. [1]

Instead of lines, in Chapter 3 we will need to maintain the lower envelope of *pseudolines*. A set L of continuous x-monotone curves in the plane is called a set of *pseudolines* if any two curves in L properly intersect in exactly one point, and there are no other intersections. Except for the last result by Brodal and Jacob [BJ02], one can verify easily that all the previous approaches also work with pseudolines. They only depend on a total ordering of the elements in L along the lower envelope. We state this fact in the following lemma.

Lemma 2.3. *Let L be a dynamic set of pseudolines. We can maintain the lower envelope of L with $O(\log n \log\log n)$ amortized update time and $O(\log n)$ worst-case query time, where n is the maximum size of L at any time. The data structure requires $O(n)$ space.*

Remark. The applicability of the result by Brodal and Jacob [BJ02] is not clear to us, and poses an interesting challenge for further investigation.

2.3.2 Dynamic Lower Envelopes in 3-space

The problem of dynamically maintaining the lower envelope of lines naturally extends to higher dimensions, and we now focus on the three dimensional case. For this, let H be a dynamic set of non-vertical planes in $\mathbb{R}^3$. The first major step towards a data structure with performance guarantees close to the planar case was due to Agarwal and Matoušek [AM95]. They gave two variants: for any constant $\varepsilon > 0$, either one can have a data structure with $O(n^\varepsilon)$ amortized update time and a vertical ray shooting query that takes $O(\log n)$ worst-case time, or one can achieve $O(\log^2 n)$ worst-case update time with $O(n^\varepsilon)$ amortized query time. It took over a decade before Chan managed to get polylogarithmic bounds for both, the update and query time [Cha06, Cha10]. His solution is randomized and has $O(\log^3 n)$ expected amortized insertion time, $O(\log^6 n)$ expected amortized deletion time, and it can answer vertical ray shooting queries in worst-case time $O(\log^2 n)$. This data structure underwent several improvements that we will now briefly review.

The data structure as described by Chan requires $O(n \log n)$ space, but already in the journal version of the paper it was observed how to reduce the space to $O(n \log\log n)$ using a three-dimensional halfspace range reporting data structure [Cha10]. Later, Afshani and Chan gave a randomized data structure for halfspace range reporting that only requires *linear* space [AC09]. This automatically decreased the space usage for Chan's dynamic convex hull structure to be linear as well. The geometric key ingredient in both data structures are so called *shallow-cuttings* for planes in three dimensions [Mat92]. In particular, the only reason for both data structures to be randomized is the use of Ramos's algorithm to construct these shallow cuttings [Ram99]. In 2015, Chan and Tsakalidis presented a deterministic algorithm for this task [CT15]. This now implies that there are deterministic versions of both, the halfspace range reporting structure of

[1] The current draft of the journal version already has 100 pages. It can be found here: http://pwgrp1.inf.ethz.ch/Current/DPCH/Journal/topdown.pdf

Afshani and Chan and the dynamic lower envelope structure of Chan. Finally, Kaplan et al. managed to improve the amortized deletion time to $O(\log^5 n)$ by carefully tweaking the parameters of Chan's data structure [KMR+16]. We summarize the properties that follow from this sequence of papers in the next theorem. A more detailed description is given by Kaplan et al. [KMR+16].

Theorem 2.4. *There is a dynamic data structure that maintains a set* H *of planes in* $\mathbb{R}^3$ *such that*

(i) we can insert a plane into H *in* $O(\log^3 n)$ *amortized time;*

(ii) we can delete a plane from H *in* $O(\log^5 n)$ *amortized time; and*

(iii) given a query point $q \in \mathbb{R}^2$*, we can perform a vertical ray shooting query with* q *in* $O(\log^2 n)$ *worst-case time,*

where n *is the maximum size of* H *at any time. The space requirement is* $O(n)$*.*

In Chapter 5 we need a dynamic data structure for Euclidean nearest neighbor queries in the plane. This data structure should maintain a set P of point sites such that we can insert points, delete points, and for every query point $q \in \mathbb{R}^2$ we can find the point in P that is closest to q according to the Euclidean distance. This can be done easily by using Theorem 2.4 together with the standard mapping from points in $\mathbb{R}^2$ to tangent planes of the three dimensional unit paraboloid [BCvKO08], i.e, there is a bijective mapping from a planar point set P to a set of hyperplanes H in $\mathbb{R}^3$ such that distances between points P can be represented by vertical distances in H, see Figure 2.3(left). In particular, a nearest neighbor query for P can be done by performing a vertical ray shooting query on the upper envelope of the arrangement of the planes in H [BCvKO08]. This gives the following corollary of Theorem 2.4.

Corollary 2.5. *There is a dynamic data structure that maintains a planar point set* P *such that*

(i) we can insert a point into P *in* $O(\log^3 n)$ *amortized time;*

(ii) we can delete a point from P *in* $O(\log^5 n)$ *amortized time; and*

(iii) given a query point q*, we can find the nearest neighbor of* q *in* P *in* $O(\log^2 n)$ *worst-case time,*

where n *is the maximum size of* P *at any time. The space requirement is* $O(n)$*.*

When generalizing some of our results for unit disk and for transmission graphs to disk graphs (Section 3.4 and Section 5.3.1), we face the task of maintaining a dynamic nearest neighbor structure for more general distance functions than the Euclidean metric. For this, we again use a mapping from two-dimensional point sites to three-dimensional objects, but this time instead of planes we end up with *surfaces*. Suppose we have for each site $p \in P$ a continuous *distance function* $\delta_p : \mathbb{R}^2 \to \mathbb{R}$ that assigns to each

point $q = (q_x, q_y)$ in the plane a distance from p to q. Then p defines a bivariate function $f_p(q_x, q_y) = \delta(q)$, $q \in \mathbb{R}^2$, whose graph is an x-y-monotone surface in three dimensions. For the case when δ_p is the Euclidean distance, the corresponding surface is the boundary of the cone with apex p, opening angle $\pi/2$, and middle axis that is parallel to the z-axis, as shown in Figure 2.3 (right). Now, let $F = \{f_p \mid p \in P\}$ be the set of functions corresponding to sites in P and let $x \in \mathbb{R}^2$ be a point. To find the site $p \in P$ that minimizes $\delta_p(x)$, i.e., the nearest neighbor of x in P according the distances δ_p, it suffices to find the function in P whose surface is the lowest at q. In other words, we need to do vertical ray shooting on the lower envelope of F.

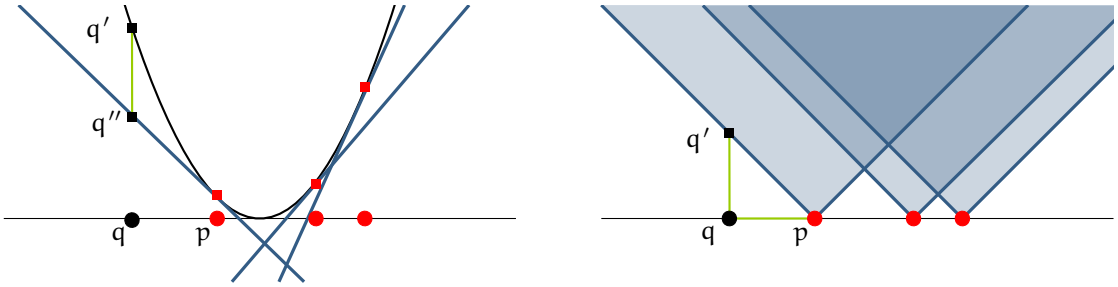


Figure 2.3: The lifting from points to tangent planes (right) and to cones (left). The distance $|pq|$ is either $\sqrt{|q''q'|}$ (left) or $|qq'|$ (right).

This fact motivated researchers to study dynamic data structures to maintain the lower envelope of the surfaces corresponding to the functions in F. The history is quite similar to the case of planes. First there was a data structure by Agarwal, Efrat, and Sharir that achieved update time $O(n^\varepsilon)$ and query time $O(\log n)$, for any fixed $\varepsilon > 0$ [AES99] and recently Kaplan et al. improved this to get polylogarithmic bounds for the update and for the query time [KMR+16]. The second result of Kaplan et al. requires that the complexity of the lower envelope is linear in the size of F and that each function in F is reasonably simple. In particular each function in F must have *bounded description complexity* which formally means that the graph of each function is a semialgebraic set that is defined by a constant number of polynomial equalities and inequalities of constant maximum degree.

The concrete bounds of Kaplan et al. make use of the function $\lambda_{s+2}(\cdot)$ which is the well-known bound on the maximum length of a Davenport-Schinzel sequence [SA96] of order $s + 2$. Here, s is a constant that depends on the complexity of the functions in F as follows: let $f_1, f_2, f_3, f_4 \in F$ be a quadruple of functions. We denote by $s(f_1, f_2, f_3, f_4)$ the number of co-vertical points $p \in f_1 \cap f_2$ and $q \in f_3 \cap f_4$. Then s is the maximum value of $s(f_1, f_2, f_3, f_4)$ over all quadruples of functions in F. If F has bounded description complexity, then s will be constant. Now we can state the theorem of Kaplan et al.

Theorem 2.6 (Theorem 8.2 in [KMR+16])**.** *Let $\mathcal{F}$ be a family of bivariate functions with bounded description complexity such that for any finite set $F \subset \mathcal{F}$ the complexity of the lower envelope of the graphs of the functions in F is $O(|F|)$. Then there is a dynamic data structure to maintain a finite set F of functions from $\mathcal{F}$ such that*

(i) we can insert a function into F *in* $O(\log^5 n \lambda_{s+2}(\log n))$ *expected amortized time;*

(ii) we can delete a function from F *in* $O(\log^9 n \lambda_{s+2}(\log n))$ *expected amortized time; and*

(iii) given a query point $q \in \mathbb{R}^2$*, we can perform a vertical ray shooting query with* q *in* $O(\log^2 n)$ *worst-case time,*

where n *is the maximum size of* F *at any time. The expected space requirement is* $O(n \log^3 n)$.

We employ Theorem 2.6 to obtain a dynamic data structure for *additively weighted Euclidean* nearest neighbor search. Let $P \subset \mathbb{R}^2$ be a set of sites such that each $p \in P$ has an assigned radius $r_p > 0$. We define the *additively weighted Euclidean distance* from a point $x \in \mathbb{R}^2$ to a site $p \in P$ to be $\delta_p(x) = |px| - r_p$. The *additively weighted Voronoi diagram*[2] (AWVD) of P is a subdivision of the plane such that all points in one region have a common nearest neighbor in P according to the additively weighted Euclidean distance, see Figure 2.4. Now, consider the set F of bivariate, three-dimensional functions that is induced by the Euclidean additively weighted distance functions δ_p, for $p \in P$. The surfaces of these functions are the same cones as for ordinary Voronoi diagrams for points, but the apex of a cone corresponding to the point p is shifted by $-r_p$ along its middle axis in z-direction, see Figure 2.4. To apply Theorem 2.6 on F, we need to argue that the lower

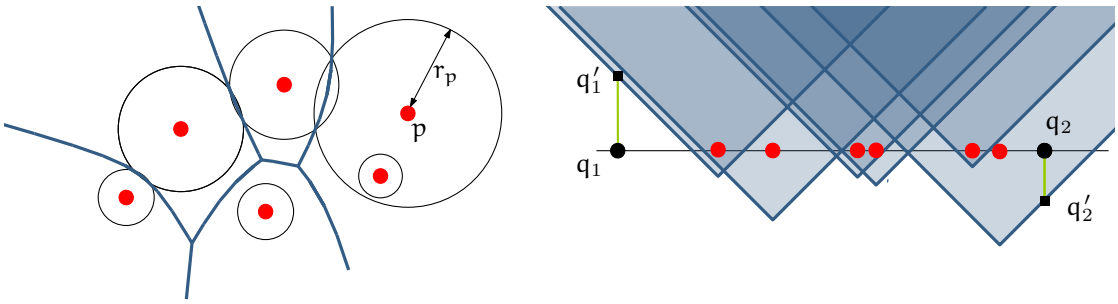


Figure 2.4: The additively weighted Voronoi diagram of six weighted sites (left) and the two-dimensional representation of the corresponding surfaces F (right). The additively weighted Euclidean distance can be positive (q_1) or negative (q_2), depending on whether the query lies inside the disk of its nearest neighbor.

envelope of the surfaces of F has linear complexity. It is well known that the projection of this lower envelope to the x-y-plane is exactly the AWVD of P [AKL13]. Furthermore, Sharir shows that the AWVD of n sites has complexity $O(n)$ [Sha85]. To obtain more precise bounds, we bound the parameter s in Theorem 2.6 when the surfaces are induced by additively weighted Euclidean distances. Let $p_1, p_2, p_3, p_4 \in P$ be a quadruple of sites and let f_1, f_2, f_3, f_4 the corresponding three-dimensional distance functions. The value $s(f_1, f_2, f_3, f_4)$ now has a more natural two-dimensional interpretation: the xy-projection

[2] Also known as the Apollonius diagram.

of an intersection curve of two functions, say $f_1 \cap f_2$, is the bisector of the sites p_1 and p_2 according to the additively weighted Euclidean distance. Then s is just the maximum number of intersections between two such bisectors. As shown by Sharir, the bisectors in an AWVD are either lines, halflines, or hyperbolas [Sha85], i.e., in any case they can be described by a polynomial function of maximum degree 2. Thus, two bisectors can intersect at most twice and we have $s = 2$.

The best bounds on $\lambda_s(n)$ for $s \geqslant 4$ are due to Nivasch [Niv10]. Depending on the parity of s they are

$$\lambda_s(n) = \begin{cases} n \cdot 2^{\frac{1}{t!}\alpha(n)^t(1+o(1))} & \text{if } n \text{ is even} \\ n \cdot 2^{\frac{1}{t!}\alpha(n)^t \log(\alpha(n))(1+o(1))} & \text{if } n \text{ is odd,} \end{cases}$$

where $t = (s-2)/2$ and $\alpha(n)$ is the inverse Ackermann function. For even s this bound is known to be tight up to lower order terms in the exponent [Niv10, SA96]. For the bound in Theorem 2.4 we are interested in the value of $\lambda_4(\log n)$. In this case $t = 1$ and since $\alpha(\log n) = \alpha(n) + O(1)$, we get $\lambda_4(\log n) = O(2^{\alpha(n)} \log n)$. This gives the following corollary of Theorem 2.4.

Corollary 2.7. *There is a dynamic data structure that maintains a set P of point sites in the plane, each $p \in P$ having an assigned radius r_p, such that*

(i) we can insert a point into P in $O(2^{\alpha(n)} \log^6 n)$ expected amortized time;

(ii) we can delete a point from P in $O(2^{\alpha(n)} \log^{10} n)$ expected amortized time; and

(iii) given a query point $q \in \mathbb{R}^2$, we can find the additively weighted nearest neighbor of q in P in $O(\log^2 n)$ worst-case time,

where n is the maximum size of P at any time and $\alpha(n)$ is the inverse Ackermann function. The expected space requirement is $O(n \log^3 n)$.

In Chapter 3 and Chapter 5 we will use Corollary 2.7 to solve the following problem: given a dynamic set of sites P and another site q with an assigned radius r_q, find a neighbor of q in the disk graph $D(P \cup \{q\})$. For this, we maintain P using the data structure from Corollary 2.7. Then we perform a nearest neighbor query to find the additively weighted nearest neighbor of q in P. It now suffices to check if the disks of q and the nearest neighbor intersect. We formalize this observation with the next lemma.

Lemma 2.8. *Let $P \subset \mathbb{R}^2$ be a set of sites with assigned radii and let $q \notin P$ be another site with radius r_q. Let $p \in P$ be the site that minimizes $\delta_p(q) = |pq| - r_p$. Then there is an edge pq in $D(P \cup \{q\})$ if and only if there is an edge between q and some $p' \in P$ in $D(P \cup \{q\})$.*

Proof. Let pq be an edge in $D(P \cup \{q\})$. Since p is a site in P, we can use $p' = p$ to get the desired edge $p'q$.

For the other direction, let $p' \in P$ be a site that forms an edge with q in $D(P \cup \{p\})$. Then $|p'q| \leqslant r_{p'} + r_q$. Since p minimizes $\delta_p(q) = |pq| - r_p$, we have $|pq| - r_p \leqslant |p'q| - r_{p'} \leqslant r_q$ and thus pq is an edge in $D(P \cup \{q\})$. □

I

Unit Disk Graphs

Chapter 3

Dynamic Unit Disk Graph Connectivity

Computing the connected components of a graph G is one of the most fundamental problems in algorithmic graph theory. When G is static, several classic solutions exist, e.g., breadth first search (BFS) or depth first search (DFS) [CLRS09]. However, if G can change dynamically, the problem becomes much more challenging. In this case, we would like a data structure that can answer *connectivity queries*: given two vertices s and t, are s and t in the same connected component of G? Additionally, we would like to be able to insert and delete edges or singleton vertices. For general graphs several solutions exist, and the following result due to Holm et al. [HdLT01] has the currently fastest amortized update and query time.

Theorem 3.1 (Holm et al., Theorem 3)**.** *Let G be a graph with n vertices. There is a deterministic data structure such that edge insertions or deletions in G take amortized time $O(\log^2 n)$, and connectivity queries take worst-case time $O(\log n / \log\log n)$.*

Even though Theorem 3.1 assumes n to be fixed, we can use a standard rebuilding method to support vertex insertion and deletion within the same amortized time bounds, by rebuilding the data structure whenever the number of vertices changes by a factor of two [Meh84]. For planar graphs, Eppstein et al. achieved $O(\log n)$ amortized time for updates and $O(\log n)$ worst-case time for queries [EIT+92].

However, the model of edge insertions and deletions may be too restrictive. For example, a natural situation where more powerful operations are needed occurs in unit disk graphs and, more generally, in disk graphs. Let $P \subset \mathbb{R}^2$ be a set of point sites, each having an associated radius $r_p > 0$. We denote by $UD(P)$ the unit disk graph of P and by $D(P)$ the disk graph of P. We want to maintain the connected components of $UD(P)$ and $D(P)$ as the *vertex set* P changes dynamically. In this case, a single update of P may change the graph quite dramatically, since one site may have many incident edges.

Nevertheless Chan, Pătraşcu, and Roditty managed to tackle this difficulty, even in the more general case of disk graphs, and gave a data structure with sublinear update and query time [CPR11]. To maintain a disk graph with at most n sites, their data structure needs $O^*(n^{20/21})$ amortized time for an update and $O^*(n^{1/7})$ worst-case time to answer a connectivity query, where the $O^*(n)$ notation hides polylogarithmic factors in n. In the same paper Chan et al. [CPR11] observed that the special case of unit disk graphs admits a solution with only *polylogarithmic* update and query time. The approach is basically a combination of known results and yields a data structure with update time $O(\log^9 n)$ and query time $O(\log n / \log\log n)$. The construction for a unit disk graph $UD(P)$ works as follows (see Figure 3.1): ① let T be the Euclidean minimum spanning

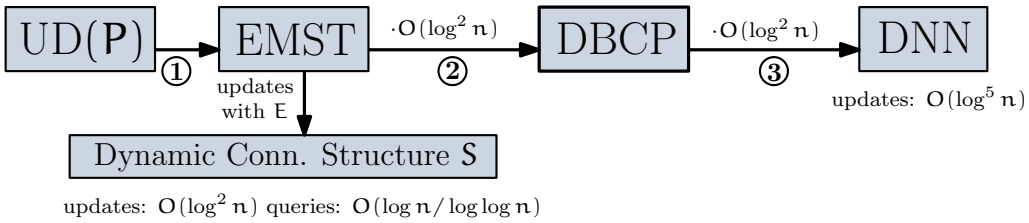


Figure 3.1: A solution with $O(\log^9 n)$ update time.

tree (EMST) of P. If we remove all edges with length larger than 1 from T, the resulting forest F is a spanning forest for $UD(P)$. Thus, to maintain the components of $UD(P)$, it suffices to maintain the components of F. We create the data structure S of Holm et al. to maintain F. Since the EMST has maximum degree 6 [BCvKO08], inserting or deleting a site from P changes $O(1)$ edges in T. Suppose we can efficiently find the set E of edges that change due to an update of P. Then, we can update the components in F through $O(1)$ edge updates in S, using all edges in E of length at most 1. Each edge update of S needs $O(\log^2 n)$ amortized time by Theorem 3.1. ② To find E, we need to dynamically maintain the adjacency lists of the EMST T when P changes. This can be done using a technique of Agarwal et al. that reduces the problem to several instances of the *dynamic bichromatic closest pair problem* (DBCP), with a multiplicative overhead of $O(\log^2 n)$ in the update time [AESW91]. ③ Eppstein showed that the DBCP problem can in turn be solved through a reduction to several instances of the dynamic nearest neighbor problem (DNN) for points in the plane [Epp95]. Again, we incur another $O(\log^2 n)$ factor as multiplicative overhead in the update time. Using Chan's dynamic nearest neighbor structure [Cha10] with all the improvements as described in Section 2.3.2, we get an amortized update time of $O(\log^5 n)$. Thus, the total amortized update time is $O(\log^9 n)$ to find the edges E plus $O(\log^2 n)$ to perform the updates in S with the $O(1)$ edges in E. We can use S to answer queries in $O(\log n / \log\log n)$ time.

3.1 Our Results

First we improve the previous result for unit disk graphs by following a similar approach, but in every step we use a method more specifically tailored to unit disks. Instead of

the EMST in ①, we use a much simpler graph on grid cells that also captures the connectivity of UD(P). Then we can avoid the $O(\log^2 n)$ overhead in ② and ③, and we can substitute the DNN data structure by a *dynamic lower envelope* (DLE) structure for pseudolines in $\mathbb{R}^2$. In particular, instead of reducing the problem to dynamically maintaining hyperplanes in $\mathbb{R}^3$ (or equivalently to dynamic nearest neighbors in $\mathbb{R}^2$, cf. Section 2.3.2), we show that it suffices to dynamically maintain lower envelopes of pseudolines in one dimension lower. In Section 3.2 we describe these reductions in full detail and this will give our first main theorem for this chapter:

Theorem 3.2. *There is a dynamic connectivity structure for unit disk graphs such that the insertion or deletion of a site takes amortized time* $O(\log^2 n)$ *and a connectivity query takes worst-case time* $O(\log n / \log\log n)$*, where* n *is the maximum number of sites at any time. The data structure requires* $O(n)$ *space.*

In Section 3.3, we use a grid-based *planar* graph to represent the connectivity of a unit disk graph. Then we can replace Theorem 3.1 by the result for planar graphs by Eppstein et al. [EIT+92]. Updates now take $O(\log n \log\log n)$ amortized time, but the worst-case query time slightly increases to $O(\log n)$.

Finally, we extend our data structure and make it work with disk graphs whose radii are in the interval $[1, \Psi]$. This extension requires the additively weighted Euclidean nearest neighbor structure from Corollary 2.7 and the update time will increase to $O(\Psi^2 2^{\alpha(n)} \log^{10} n)$. It can be found in Section 3.4.

3.2 The Data Structure for Unit Disks

Let $P \subset \mathbb{R}^2$ be a set of sites and let UD(P) be the unit disk graph for P. We use the following approach to obtain a dynamic connectivity structure for UD(P) with amortized update time $O(\log^2 n)$ and worst-case query time $O(\log n / \log\log n)$: first we define an *auxiliary graph* G that represents the connectivity of UD(P) and that has bounded degree, see Section 3.2.1. We create a data structure S as in Theorem 3.1 to maintain G. The graph G will substitute the EMST used in the previous approach (cf. ① in Figure 3.1 and Figure 3.2). The vertices of G are cells of a grid and to see if two cells form an edge, we maintain a maximal bichromatic matching (MBM) of the sites in the grid cells (see Section 3.2.2). This allows us to find the $O(1)$ edges E of G that change when P is updated. This time, we only have an $O(1)$ multiplicative overhead (② in Figure 3.2). Finally, we show that maintaining a bichromatic matching can be done with the help of two dynamic lower envelope (DLE) data structures for pseudolines (③ in Figure 3.2). This reduction also has an $O(1)$ multiplicative overhead and is described in Section 3.2.3.

The total amortized update time will be $O(\log n \log\log n)$ for updating the dynamic lower envelope structures plus $O(\log^2 n)$ for updating S with the edges in E. Details follow.

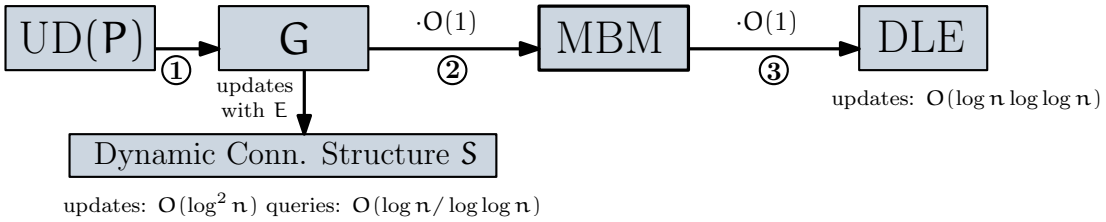


Figure 3.2: Our improved solution with an update time of $O(\log^2 n)$.

3.2.1 The Grid Graph (new ①).

Let $\mathcal{G} = \mathcal{G}_0$ be the planar grid whose cells have diameter 1. For any grid cell $\sigma \in \mathcal{G}$, the sites $\sigma \cap P$ induce a clique in UD(P). For $P \subset \mathbb{R}^2$, we define a graph G whose vertices are the *non-empty* cells $\sigma \in \mathcal{G}$, i.e., the cells with $\sigma \cap P \neq \emptyset$. The *neighborhood* $N(\sigma)$ of a cell $\sigma \in \mathcal{G}$ is the 5×5 block of cells in $\mathcal{G}$ with σ in the center. We call two cells *neighboring* if they are in each other's neighborhood. The endpoints of any edge in UD(P) must lie in neighboring cells. To obtain the edges of G, we connect every pair of distinct neighboring grid cells that contain the endpoints of an edge in UD(P). By construction, and since the sites inside each cell form a clique, the connectivity between two sites s, t in UD(P) is the same as for the corresponding cells in G.

Lemma 3.3. *Let $s, t \in P$ be two sites and let σ and τ be the cells in $\mathcal{G}$ that contain s and t, respectively. There is an s-t path in UD(P) if and only if there is a σ-τ path in G.*

Proof. Let $s = s_1, s_2 \ldots, s_k = p$ be an s-t-path in UD(P). For $1 \leqslant i \leqslant k$, let $\sigma_i \in \mathcal{G}$ be the cell that contains the site s_i. Either $\sigma_i = \sigma_{i+1}$ or there is an edge $s_i s_{i+1}$ in UD(P). In the latter case, by definition of G, $\sigma_i \sigma_{i+1}$ forms an edge in G. Thus, there is a σ_1-σ_k path in G.

Now let $\sigma, \tau \in \mathcal{G}$ be the two cells containing p and q. Let $\sigma = \sigma_1, \ldots, \sigma_k = \tau$ be a σ-τ path in G. Then, for $1 \leqslant i \leqslant k-1$, there is an edge $s_i t_{i+1}$ in UD(P). Since each cell σ_i has diameter 1, also $t_i s_i$ is an edge in UD(P), for $2 \leqslant i \leqslant k-1$ (unless $t_i = s_i$). For the same reasons, UD(P) contains the edges $s s_1$ and $t_k t$ and hence the sequence $s, s_1, t_2, s_2, \ldots, t_{k-1}, s_{k-1}, t_k, t$ yields an s-t path in UD(P). □

We build the data structure from Theorem 3.1 for G. When a site s is inserted into or deleted from P, only $O(1)$ edges in G change, since only the neighborhood of the cell of s is affected. Thus, once the set E of changing edges is determined, we can maintain the connectivity of G in time $O(\log^2 n)$ per update, by Theorem 3.1.

3.2.2 Maintaining the Adjacency Lists of G (new ②).

Our next step is to *dynamically maintain the adjacency lists* of G, i.e., we want a data structure for P that, whenever we update P, returns the set of edges E that change in G. For this, we maintain for each pair of non-empty neighboring cells a *maximal bichromatic matching* (MBM) between their sites, similar to Eppstein's method [Epp95]. Let $R \subseteq P$

and $B \subseteq P$ be two sets of sites. An MBM between R and B is a maximal set of pairwise vertex-disjoint edges in the bipartite graph on $R \cup B$ consisting of all edges of $UD(P)$ with one endpoint in R and one endpoint in B.

For each pair $\{\sigma, \tau\}$ of neighboring cells in $\mathcal{G}$, we build an MBM $M_{\{\sigma,\tau\}}$ for $R = \sigma \cap P$ and $B = \tau \cap P$. By definition, there is an edge between σ and τ in G if and only if $M_{\{\sigma,\tau\}}$ is not empty. When inserting or deleting a site p from P, we proceed as follows: let $\sigma \in \mathcal{G}$ be the cell with $p \in \sigma$. We go through all cells $\tau \in N(\sigma)$ and update the MBM $M_{\{\sigma,\tau\}}$ (by inserting or deleting p from the relevant set). If $M_{\{\sigma,\tau\}}$ becomes non-empty during an insertion or becomes empty during a deletion, we add the edge $\sigma\tau$ to E and mark it for insertion or deletion, respectively. Since $|N(\sigma)| = O(1)$, we have $O(1)$ edges in E and we can find these edges by updating $O(1)$ MBMs. This construction is summarized in the following lemma.

Lemma 3.4. *Suppose we can maintain an MBM for each pair of non-empty neighboring cells with update time $O(U(n))$, where n is the maximum number of sites. Then we can dynamically maintain the adjacency lists of G with update time $O(U(n))$.*

3.2.3 Dynamically Maintaining an MBM (new ③).

Let $\sigma \neq \tau$ be two neighboring cells of $\mathcal{G}$, and let $R = \sigma \cap P$ and $B = \tau \cap P$. We show that an MBM between R and B can be efficiently maintained using two dynamic lower envelope structures for pseudolines as in Lemma 2.3. We fix a line ℓ that separates R and B. Since R and B are in two distinct grid cells, we can take a supporting line of one of the four boundaries of σ. We now prove the following lemma.

Lemma 3.5. *Let $R, B \subseteq P$ be two sets with a total of at most n sites, separated by a line ℓ. There is a dynamic data structure that maintains an MBM for R and B with $O(\log n \log\log n)$ amortized update time. The data structure requires $O(n)$ space.*

Proof. We rotate and translate everything such that ℓ is the x-axis and all sites in R have positive x-coordinate. To simplify some of the arguments, we assume general position, meaning that any two sites in R have different x-coordinates and that no two sites have x-coordinates whose difference is 1.

We consider the set D_R of radius 1 disks with centers in R (see Figure 3.3). Then a site in B forms an edge with *some* site in R if and only if it is contained in the union of the disks in D_R. To detect this, we maintain the lower envelope of D_R. More precisely, consider the following set L_R of curves: for each disk in D_R, take the arc that defines the lower envelope of the disk and extend both ends straight upward to ∞, as shown in Figure 3.3. By our general position assumption, L_R is a set of pseudolines. We build a data structure S_R for L_R according to Lemma 2.3. It has worst-case query time $O(\log n)$, amortized update time $O(\log n \log\log n)$, and it needs $O(n)$ space. Analogously, we define a set of pseudolines L_B and a dynamic envelope structure S_B for B. Since the lower envelope of the pseudolines L_R corresponds to the lower envelope of D_R below ℓ, and since these lower envelopes are x-monotone, we can for every site $b \in B$ detect if b is contained in the union of the disks in D_R by shooting a ray from $-\infty$ at the x-coordinate of b.

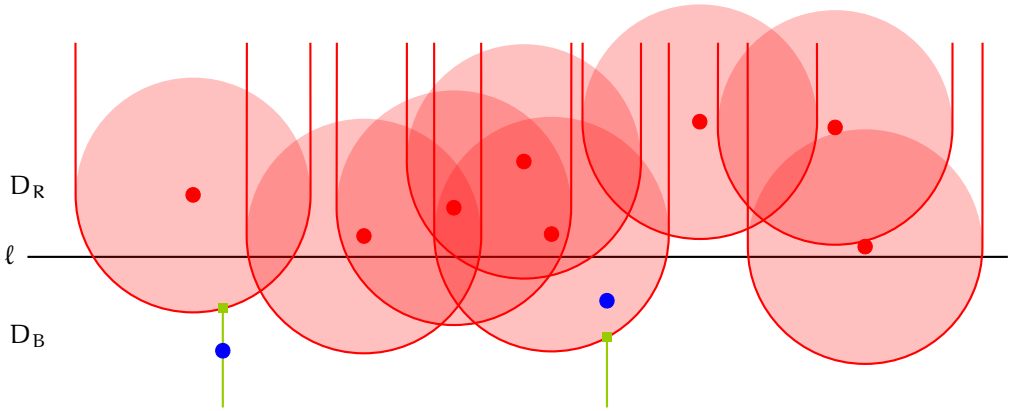


Figure 3.3: The set L_R induced by R.

This can be done using S_R, see Figure 3.3. Likewise we can detect for every site $r \in R$ whether it is in the union of the disk corresponding to the sites in B.

To maintain the MBM M, we store in S_R the pseudolines of the currently unmatched sites of R, and in S_B the pseudolines of the unmatched sites of B. By construction, there cannot be an edge between unmatched sites in R and unmatched sites in B, and our invariant will maintain this property. When inserting a site r into R, we perform a vertical ray shooting query in S_B with r to get a pseudoline of L_B. Let $b \in B$ be the site for that pseudoline. If $|rb| \leqslant 1$, we add the edge rb to M, and delete the pseudoline of b from S_B. Otherwise, r will be an unmatched site and we insert the pseudoline of r into S_R. Since the lower envelope of L_B is x-monotone inside the halfplane bounded by ℓ where r lies in, there is an edge between r and an unmatched site in B if and only if there is an edge between r and b. Hence, the insertion procedure correctly maintains the invariant on the unmatched sites.

Now suppose we want to delete a site r from R. If r is unmatched, we delete the pseudoline corresponding to r from S_R. This maintains our invariant, since we only delete a site. Otherwise, we remove the edge rb from M, and we reinsert b as above, looking for a new unmatched site in R for b. As argued above, inserting sites also maintains the invariant. Updating B is analogous.

Inserting and deleting a site requires $O(1)$ insertions, deletions, or queries in S_R or S_B, so the lemma follows. □

To obtain Theorem 3.2 we can combine our previous results as follows (cf. Figure 3.2 in the beginning of this section): we start with the grid graph G that, by Lemma 3.3, has the same connectivity as $UD(P)$. To maintain the connectivity of G, we build a dynamic connectivity structure S for G as in Theorem 3.1. Whenever we update P, we need to find the $O(1)$ edges E that change in G. By Lemma 3.4 we can find E by maintaining an MBM between any pair of non-empty neighboring gridcells in $\mathcal{G}$. In Lemma 3.5 we proved that maintaining an MBM can be done with an amortized update time $O(\log n \log\log n)$ and with space $O(n)$ by using two dynamic lower envelope (DLE)

structures for pseudolines. Thus, with an update time of $O(\log n \log\log n)$, we can find the edges E. To maintain the connectivity of G, we update S with the $O(1)$ edges in E. This needs $O(\log^2 n)$ time by Theorem 3.1. We can use S to answer connectivity queries in worst-case time $O(\log n/\log\log n)$. By construction, each site of P is in $O(1)$ dynamic lower envelope structures and thus we need $O(n)$ space in total.

3.3 Improving the Update Time by Using a Planar Graph

The bottleneck for the update time in Section 3.2 lies in the use of Theorem 3.1: we need to update the data structure S with the edges in E, which takes $O(\log^2 n)$ amortized time. We now define a planar graph G_p that is similar to the grid graph G. It represents the connectivity of $UD(P)$ and an update of P changes $O(1)$ vertices and edges in G_p. These vertices and edges can be found in $O(1)$ time. Since G_p is planar, we can use the following data structure of Eppstein et al. [EIT+92] instead of Theorem 3.1 to maintain the connectivity of G_p:

Theorem 3.6 (Eppstein et al., Theorem 1)**.** *Let G be a planar graph with n vertices. There is a deterministic data structure such that edge insertions or deletions in G take amortized time $O(\log n)$, insertions or deletions of single vertices take $O(\log n)$ amortized time, and connectivity queries take worst-case time $O(\log n/\log\log n)$.*

This gives the next theorem that we are going to prove in this chapter.

Theorem 3.7. *There is a dynamic connectivity structure for unit disk graphs such that insertion or deletion of a site takes amortized time $O(\log n \log\log n)$ and a connectivity query takes worst-case time $O(\log n)$, where n is the maximum number of sites at any time. The data structure requires $O(n)$ space.*

The Planar Graph

Let $P \subset \mathbb{R}^2$ be a set of sites. For any pair of non-empty grid cells σ, τ let $M_{\{\sigma,\tau\}}$ be the MBM as defined in Section 3.2.2. For any non-empty MBM $M_{\{\sigma,\tau\}}$, we pick an arbitrary edge $rb \in M_{\{\sigma,\tau\}}$ with $r \in \sigma$ and $b \in \tau$ as the *representative edge*. Let $T \subseteq P$ be the set of sites incident to a representative edge. We use the unit disk graph $UD(T)$ as basis for our planar graph G_p. If we contract in each grid cell σ the subgraph of $UD(T)$ induced by $T \cap \sigma$ to a single vertex, we get the graph G from Section 3.2. Hence, by Lemma 3.3, $UD(T)$ represents the connectivity of $UD(P)$.

To get G_p from $UD(T)$, we consider the straight line drawing of $UD(T)$. For a crossing of two edges ab and uv in $UD(T)$, we add a new site x at the intersection and call x a *crossing site*. We remove ab and uv and we add the four new edges ax, xb, ux, and xv, see Figure 3.4. We repeat this operation until there are no more crossings in $UD(T)$. This is a standard method for making unit disk graphs planar. The next lemma, due to

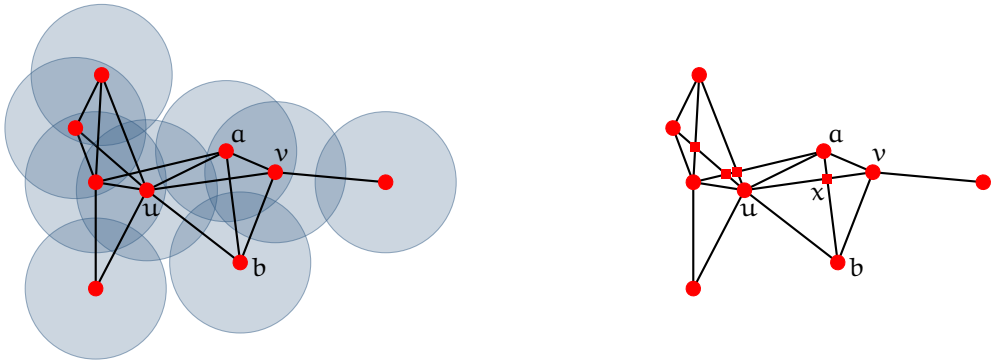


Figure 3.4: A unit disk graph (left) and its planarization (right). Red squares are crossing sites.

Yan et al. [YXD12], shows that it does not add new connections between the four sites participating in a crossing.

Lemma 3.8. *Let* ab *and* uv *be edges in* $\mathrm{UD}(T)$ *that cross. Then* a, b, u, *and* v *are in the same connected component of* $\mathrm{UD}(\{a, b, u, v\})$.

Proof. Let x be the intersection point of the edges ab and uv. We know that $|ab| \leqslant 1$ and $|uv| \leqslant 1$. By the triangle inequality, $|ax| + |xu| \geqslant |au|$ and $|bx| + |xv| \geqslant |bv|$. Combining these inequalities, we get

$$2 \geqslant |ab| + |uv| = |ax| + |bx| + |xu| + |xv| \geqslant |au| + |bv|.$$

Thus, we have $|au| \leqslant 1$ or $|bv| \leqslant 1$ and $\mathrm{UD}(\{a, b, u, v\})$ contains at least one of the edges au, bv. Hence, a, b, u, v are in the same connected component of $\mathrm{UD}(\{a, b, u, v\})$. □

Using Lemma 3.8 we now show that G_p has the same connectivity as $\mathrm{UD}(T)$. Thus, by Lemma 3.3, G_p also represents the connectivity of $\mathrm{UD}(P)$.

Lemma 3.9. *Let* $s, t \in T$ *be two sites. Then* s *and* t *are connected in* $\mathrm{UD}(T)$ *if and only if they are connected in* G_p.

Proof. Since going from $\mathrm{UD}(T)$ to G_p only increases the connectivity, all sites s and t that are connected in $\mathrm{UD}(T)$ are also connected in G_p.

For the other direction, let $s = p_1, \ldots, p_k = t$ be a path in G_p between $s, t \in T$. For each p_i, we define a set $V_i \subseteq T$ as follows: if p_i is a site in T, we set $V_i = \{p_i\}$. Otherwise, p_i is a crossing site, created by a crossing of two edges uv and ab in $\mathrm{UD}(T)$. We set $V_i = \{a, b, u, v\}$. By Lemma 3.8, the sites a, b, u, v are in the same connected component of $\mathrm{UD}(T)$. Furthermore, if p_i is a crossing site, we have $V_{i-1} \cap V_i \neq \emptyset$: the edge $p_{i-1}p_i$ is a proper subsegment of an edge e in $\mathrm{UD}(T)$, and at least one endpoint of e lies in V_{i-1}.

We prove by induction that all sites in $\bigcup_{i=1}^{j} V_i$ lie in the same connected component of $\mathrm{UD}(T)$, for $j = 1, \ldots, k$. For $j = 1$, this is clear. Now, consider V_j. If $V_{j-1} \cap V_j \neq \emptyset$,

then the claim follows by induction, since all sites in V_j are in the same component. Otherwise, $V_j = \{p_j\}$, p_j is a site in T, and there is an edge in $UD(T)$ between p_j and $\bigcup_{i=1}^{j-1} V_i$, implying the claim. By setting $j = k$, we now have that s and t are connected in $UD(T)$. □

Maintaining the Planar Graph G_p

We maintain G_p as a doubly-connected edge list (DCEL) and we build a planar dynamic connectivity data structure S for G_p, as in Theorem 3.6. Furthermore, we maintain an MBM between any two neighboring non-empty grid cells and we pick one representative edge for each MBM. Let s be a site we want to insert or delete from P. Let σ be the grid cell containing s. We update for all cells $\tau \in N(\sigma)$ the MBM $M_{\{\sigma,\tau\}}$, and we collect the sites of all representative edges that need to be inserted or deleted in two sets I and D: if $M_{\{\sigma,\tau\}}$ changes from empty to non-empty, we pick a representative edge for $M_{\{\sigma,\tau\}}$ and put its two endpoints into I. If we delete the representative edge of $M_{\{\sigma,\tau\}}$, we put its two endpoints into D, and, if possible, we pick a new representative edge for $M_{\{\sigma,\tau\}}$ and put its endpoints into I. Since $|N(\sigma)| = O(1)$, the sets I and D contain $O(1)$ sites that need to be added or deleted from G_p.

Next, we show how to update G_p with a site s in I or D. First we insert or delete s in $UD(T)$ and determine which edges change in $UD(T)$. Each such edge may create or delete several edges and sites in G_p that need to be handled. The next lemma shows that s can create or delete $O(1)$ edges and vertices in G_p and that these edges can be found in $O(1)$ time. Once we know them, we can simply update G_p and S accordingly. This finishes the proof of Theorem 3.7.

Lemma 3.10. *Let s be a site in I or D. Updating G_p with s changes $O(1)$ edges and vertices. They can be found in $O(1)$ time.*

Proof. Suppose that $s \in I$, i.e., we want to insert s. Let σ be the cell containing s. We add s to T and collect all edges in $UD(T)$ incident to s in a set E as follows: we start with $E = \emptyset$. First, for each $t \in T \cap \sigma$ we add the edge st to E. Since σ has diameter 1, all these edges are valid edges in $UD(T)$. Next, we go through all cells $\tau \in N(\sigma)$. We check for all sites $t \in \tau \cap T$ if $|st| \leqslant 1$. If so, we add the edge st to E.

To update G_p, we find all edges E_c in G_p crossed by edges in E. Since all edges in E and in G_p cross $O(1)$ grid cells, and since each grid cell contains $O(1)$ sites and crossing sites, this can be done in $O(1)$ time. We add all edges in E_c to E and we perform the planarization procedure as in Lemma 3.8 on E. To update G_p, we delete all edges in E_c from G_p and we add all edges and vertices in E to G_p. This gives all edges and vertices in G_p that need to be changed, in $O(1)$ time.

Deleting a site is done in a similar manner. □

3.4 Extension to Disk Graphs

Next, we describe how to extend our data structure to make it work with disk graphs. Let P be a dynamic set of sites such that each site has an associated radius r_p. We assume that all radii in P are in the interval $[1, \Psi]$ for some $\Psi \geqslant 1$. We will prove the following theorem.

Theorem 3.11. *Let $\Psi \geqslant 1$ and let P be a set of sites with radii in $[1, \Psi]$. There is a dynamic connectivity structure for $D(P)$ such that the insertion or deletion of a site takes amortized expected time $O(\Psi^2 2^{\alpha(n)} \log^{10} n)$ and a connectivity query takes worst-case time $O(\log n / \log\log n)$, where n is the maximum number of sites at any time and $\alpha(n)$ is the inverse Ackermann function. The data structure requires $O(\Psi^2 n \log^3 n)$ expected space.*

To maintain the connectivity of $D(P)$, we use the same approach as in Section 3.2: we consider the grid $\mathcal{G} = \mathcal{G}_0$ with diameter 1 and we use a grid graph G whose vertices are the non-empty grid cells. For a grid cell $\sigma \in \mathcal{G}$ we define the neighborhood $N(\sigma)$ as the $\big(2\lceil 2\sqrt{2}\Psi\rceil + 1\big) \times \big(2\lceil 2\sqrt{2}\Psi\rceil + 1\big)$ block of cells centered at σ. The increased neighborhood guarantees that every edge of $D(P)$ must have its endpoints in neighboring grid cells. The maximum degree in G is now $|N(\sigma)| = O(\Psi^2)$ and thus an update of P changes $O(\Psi^2)$ edges in G. We build a data structure S according to Theorem 3.1 to maintain G. To detect the edges that change when P is updated, we maintain an MBM $M_{\{\sigma,\tau\}}$ between any pair $\sigma, \tau \in \mathcal{G}$ of non-empty neighboring grid cells. Then we have an edge $\sigma\tau$ in G if and only if $M_{\{\sigma,\tau\}}$ is non-empty. When inserting or deleting a site s, we determine the cell σ of s and we update for all $\tau \in N(\sigma)$ the MBM $M_{\{\sigma,\tau\}}$. For all MBMs that change from empty to non-empty or vice versa, we update the corresponding edges in S accordingly. Thus, there are at most $O(\Psi^2)$ updates in S. Since each site participates in $O(\Psi^2)$ MBMs at any time, we can reduce the dynamic maintenance of $D(P)$ to dynamically maintaining an MBM. We summarize this in the following lemma, which is the analog to Lemma 3.4.

Lemma 3.12. *Suppose we can maintain an MBM for each pair of non-empty neighboring cells with update time $U(n)$ and space $S(n)$, where n is the maximum number of sites. Then we can maintain G with update time $O(\Psi^2 U(n))$ and space $O(\Psi^2 S(n))$.*

Maintaining an MBM for Disk Graphs

Let $R, B \subseteq P$ be two sets of at most n sites. We show how to maintain a maximal bichromatic matching M between R and B with respect to $D(R \cup B)$. The amortized expected update time will be $O(2^{\alpha(n)} \log^{10} n)$, where $\alpha(n)$ is the inverse Ackermann function and we will require $O(n \log^3 n)$ expected space. Using Lemma 3.12, Theorem 3.11 then is immediate.

To maintain M, we build two additively weighted Euclidean nearest neighbor structures as in Corollary 2.7, one for R and one for B. The distance from a site $p \in R \cup B$ to any point $x \in \mathbb{R}^2$ is $\delta(p, x) = |px| - r_p$, i.e., the weight of the site p is its radius.

We denote by S_R the structure for R and by S_B the structure for B. We store in S_R the currently unmatched sites in R, and in S_B the currently unmatched sites in B. When inserting a site r into R, we query S_B with r to get an ummatched site $b \in B$ that minimizes $|rb| - r_b$. By Lemma 2.8, if there is an edge from r to some site in B in $D(R \cup B)$, then also rb must be an edge in $D(R \cup B)$. Thus, if $|rb| \leqslant r_r + r_b$, we add the edge rb to M, and we delete b from S_B. Otherwise, there is no edge between r and B and we insert r into S_R. By Lemma 2.8, the insertion procedure correctly maintains an MBM. Now suppose we want to delete a site r from R. If r is unmatched, we simply delete r from S_R. Otherwise, we remove the edge rb from M, and we reinsert b as above, looking for a new unmatched site in R for b. The procedures for updating B are analogous.

Since inserting and deleting sites requires $O(1)$ insert, delete or query operations in S_R or S_B, and since a nearest neighbor structure with n sites needs $O(n \log^3 n)$ expected space, by Corollary 2.7 we get the following lemma.

Lemma 3.13. *Let $R, B \subseteq P$ be two sets with a total of at most n sites. There exists a dynamic data structure that maintains an MBM for R and B with respect to $D(R \cup B)$ that has expected amortized update time $O(2^{\alpha(n)} \log^{10} n)$ and that requires expected space $O(n \log^3 n)$.*

Theorem 3.11 now follows by combining Lemma 3.12 and Lemma 3.13.

3.5 Conclusion

We gave an improved data structure for dynamic connectivity in unit disk graphs. It has an amortized update time of $O(\log n \log\log n)$ and it can answer connectivity queries in $O(\log n)$ worst-case time.

The bottleneck in our query time is the query time of the dynamic connectivity structure for planar graphs of Eppstein et al. that we use in Section 3.3. It is notable that the data structure from Theorem 3.1 for general graphs has a better query time of $O(\log n / \log\log n)$ (even though the update time is worse). Since the result of Eppstein et al. is quite old compared to the one by Holm et al., it is likely that a similar improvement is possible for planar graphs (without affecting the update time) by using more recent data structure techniques. This would immediately improve our query time as well.

The bottleneck for the update time is the update time of the dynamic lower envelope structure that we use in Section 3.2.3. In Section 2.3.1 we argued that there are several dynamic data structures for *lines* and most of them also work with pseudolines. However, for the fastest such structure, due to Brodal and Jacob [BJ02], we do not know if it is applicable to pseudolines. If true, this would imply an amortized update time of $O(\log n)$ for our data structure, and thus understanding and extending the result of Brodal and Jacob constitutes an interesting and challenging task for future work.

Furthermore, we extended our data structure to disk graphs with radii in the interval $[1, \Psi]$. This extension requires quite heavy machinery, namely a dynamic nearest

neighbor structure for additively weighted Euclidean distances. This has two major drawbacks: the update and query time increase by several log factors and the complete data structure is more involved and hence gets harder to implement. Thus, we would like to know if there is an easier solution for disk graphs of bounded radii, without using additively weighted nearest neighbors. Furthermore, it is still an open question whether a data structure with polylogarithmic update time exists for disk graphs with arbitrarily large radii.

Chapter 4

Routing in Unit Disk Graphs

Routing data through graphs constitutes a fundamental problem in distributed graph algorithms [GS04, PU89]. Given a graph G, we would like to be able to route a packet from any node in G to any other node. The routing algorithm should be *local*, meaning that it uses only information stored with the packet and with the current node, and it should be *efficient*, meaning that the packet does not travel much further than necessary. There is an obvious solution to this problem: with each node s of G, we store the shortest path tree for s. Then it is easy to route a packet along the shortest path to its destination. However, this solution is very inefficient: we need to store the complete topology of G with each node, leading to quadratic space usage. Thus, the goal of a routing scheme is to store as little information as possible with each node of the graph, while still guaranteeing a routing path that is not too far from optimal.

For general graphs a plethora of results is available, reflecting the work of almost three decades (see, e.g., [Che13, RT15] and the references therein). However, for general graphs, any efficient routing scheme needs to store $\Omega(n^\alpha)$ bits per node, for some $\alpha > 0$ [PU89]. Thus, it is natural to ask whether improved results are possible for specialized graph classes. For example, for trees it is known how to obtain a routing scheme that follows a shortest path and requires $O(\log n)$ bits of information at each node [FG01, SK85, TZ01]. In planar graphs, for any $\varepsilon > 0$ it is possible to store a polylogarithmic number of bits at each node in order to route a packet along a path of length at most $1 + \varepsilon$ times the length of the shortest path [Tho04].

A graph class that is of particular interest for routing problems comes from the study of mobile and wireless networks. Such networks are traditionally modeled as unit disk graphs, cf. Chapter 1. Even though unit disk graphs may be dense, they share many properties with planar graphs, in particular with respect to algorithmic problems. There exists a vast literature on routing in unit disk graphs (cf. [GS04]), but most known

schemes cannot ensure a short routing path in the worst case. Yan, Xiang, and Dragan [YXD12] present a scheme with provable worst case guarantees. They extend a scheme by Gupta et al. [GKR04] for planar graphs to unit disk graphs by using a delicate planarization argument, to obtain small-sized balanced separators. Their planarzation technique is comparable to the one we use in Section 3.3. Even though the scheme by Yan et al. is conceptually simple, it requires a detailed analysis with an extensive case distinction.

We propose an alternative approach to routing in unit disk graphs. Our scheme is based on the well-separated pair decomposition for unit disk graphs [GZ05]. It stores a polylogarithmic number of bits with each node of the graph, and it constructs a routing path that can be made arbitrarily close to a shortest path (see Section 4.1 for a precise statement of our results). This compares favorably with the scheme by Yan et al. [YXD12] which achieves only a constant factor approximation. Moreover, our scheme is arguably simpler to analyze. However, unlike the algorithm by Yan et al., our scheme requires that the packet contain a modifiable *header* with a polylogarithmic number of bits. It is an interesting question whether this header can be removed.

4.1 The Routing Model and Our Results

Let $P \subset \mathbb{R}^2$ be a set of n sites in the plane. We say that P has *density* ϑ if every unit disk contains at most ϑ points from P. The density ϑ of P is *bounded* if $\vartheta = O(1)$.

We consider the unit disk graph $\mathrm{UD}(P)$. Recall that we weight each edge st of $\mathrm{UD}(P)$ by its Euclidean length $|st|$. For the ease of notation we denote by $d(\cdot,\cdot)$ the shortest path distance in $\mathrm{UD}(P)$, i.e., $d(\cdot,\cdot)$ is shorthand for $d_{\mathrm{UD}(P)}(\cdot,\cdot)$.

We would like to obtain a *routing scheme* for $\mathrm{UD}(P)$ with small *stretch* and compact *routing tables*. Formally, this is defined as follows: we can preprocess $\mathrm{UD}(P)$ to obtain for each site $s \in P$ a *label* $\ell(s) \in \{0,1\}^*$ and a *routing table* $\rho(s) \in \{0,1\}^*$. Furthermore, we need to define a *routing function*

$$f : P \times \{0,1\}^* \times \{0,1\}^* \to P \times \{0,1\}^* \times \{0,1\}^*.$$

The function f takes as input a *current site* s, the label $\ell(t)$ of a *target site* t, and a *header* $h \in \{0,1\}^*$. The routing function may use its input and the routing table $\rho(s)$ of s to compute a new site s', a modified header h', and the label of an *intermediate target* $\ell(t')$. The new site s' may be either s or a neighbor of s in $\mathrm{UD}(P)$. Even though the eventual goal of the packet is the target t, we introduce the intermediate target t' into the notation, since it allows for a more succinct presentation of the routing algorithm.

The routing scheme is *correct* if the following holds: let h_0 be the empty header. For any two sites $s, t \in P$, consider the sequence of triples given by $(s_0, \ell_0, h_0) = (s, \ell(t), h_0)$ and $(s_i, \ell_i, h_i) = f(s_{i-1}, \ell_{i-1}, h_{i-1})$ for $i \geqslant 1$. Then there exists a $k = k(s,t) \geqslant 0$ such that $s_k = t$ and $s_i \neq t$ for $i < k$, i.e., the routing scheme reaches t after k steps. We call $s_0, s_1, \ldots, s_k$ the *routing path* between s and t, and we define the *routing distance* $d_\rho(s,t)$ between s and t as $d_\rho(s,t) = \sum_{i=1}^{k} |s_{i-1}s_i|$.

The quality of the routing scheme is measured by four parameters:

- the *label size* $L(n) = \max_{P \subset \mathbb{R}^2, |P|=n} \max_{s \in P} |\ell(s)|$,
- the *table size* $T(n) = \max_{P \subset \mathbb{R}^2, |P|=n} \max_{s \in P} |\rho(s)|$,
- the *header size* $H(n) = \max_{P \subset \mathbb{R}^2, |P|=n} \max_{s \neq t \in P} \max_{i=1,\dots,k(s,t)} |h_i|$,
- and the *stretch* $\varphi(n) = \max_{P \subset \mathbb{R}^2, |P|=n} \max_{s \neq t \in P} d_\rho(s,t)/d(s,t)$.

We show that for any $P \subset \mathbb{R}^2$, $|P| = n$, and any $\varepsilon > 0$ we can construct a routing scheme with stretch $\varphi(n) = 1 + \varepsilon$, label size $L(n) = O(\log n)$, table size $T(n) = O(\varepsilon^{-5} \log^2 n \log^2 D)$, and header size $H(n) = O(\log n \log D)$, where D is the weighted diameter of UD(P), i.e., the maximum length of a shortest path between two sites in UD(P).

4.2 The Well-Separated Pair Decomposition for UD(P)

Our routing scheme uses the well-separated pair decomposition (WSPD) for the unit disk graph metric given by Gao and Zhang [GZ05]. See Section 2.2 for an overview of WSPDs and their applications in computational geometry.

Since our routing scheme relies crucially on the specific structure of the WSPD described by Gao and Zhang, we remind the reader of the main steps of their algorithm and analysis.

First, Gao and Zhang assume that P has bounded density and that UD(P) is connected. They construct the Euclidean minimum spanning tree T for P. It is easy to see that T is a spanning tree for UD(P) with maximum degree 6. Furthermore, T can be constructed in $O(n \log n)$ time [BCvKO08].

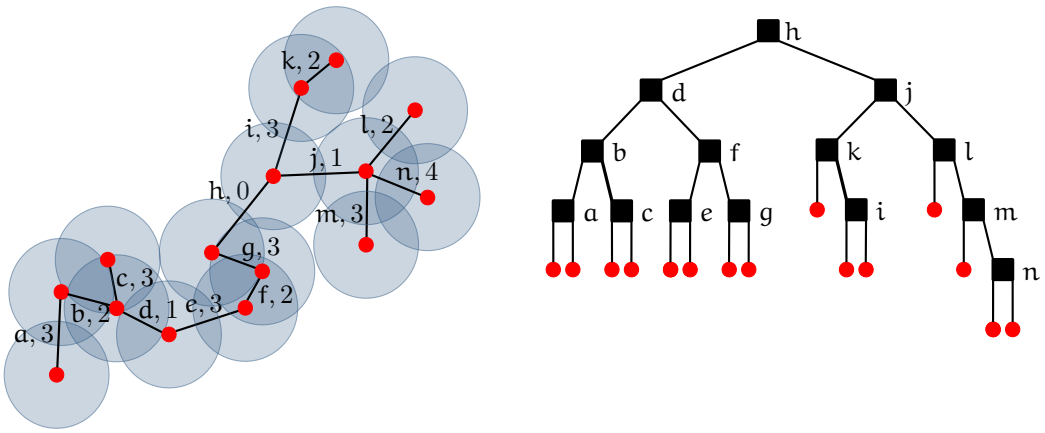


Figure 4.1: An EMST T of UD(P) (left) where the edges are annotated with their level in the hierarchical decomposition H (right).

Since T has maximum degree 6, there exists an edge e in T such that $T \setminus e$ consists of two trees with at least $\lceil (n-1)/6 \rceil$ vertices each. By applying this observation recursively, we obtain a *hierarchical decomposition* H of T. The decomposition H is a binary tree. Each node ν of H represents a subtree T_ν of T with vertex set $P_\nu \subseteq P$ such that (i) the root of H corresponds to T; (ii) the leaves of H are in one-to-one correspondence with the sites in P; and (iii) let ν be an inner node of H with children u and w. Then ν has an *associated edge* $e_\nu \in T_\nu$ such that removing e_ν from T_ν yields the two subtrees T_u and T_w represented by u and w (see Figure 4.1). Furthermore, we have $|P_u|, |P_w| \geqslant \lceil (|P_\nu|-1)/6 \rceil$.

It follows that H has height $O(\log n)$. The *level* $\delta(\nu)$ of a node $\nu \in H$ is defined as the number of edges on the path from ν to the root of H. The level of the associated edge e_ν of ν is the level of ν in H. This uniquely defines a level for each edge in T. Now, for each node $\nu \in H$, the subtree T_ν is a connected component in the forest that is induced in T by the edges of level at least $\delta(\nu)$.

After computing the hierarchical decomposition, the algorithm of Gao and Zhang essentially uses the greedy algorithm of Callahan and Kosaraju (see Algorithm 5.3 in Section 5.2.3) to construct a WSPD, with H in place of the quadtree (or the fair split tree). Let $c \geqslant 1$ be a separation parameter. The algorithm traverses H and produces a sequence $\Xi = (u_1, \nu_1), (u_2, \nu_2), \ldots, (u_m, \nu_m)$ of pairs of nodes of H, with the following properties:

1. The sets $P_{u_1} \times P_{\nu_1}, P_{u_2} \times P_{\nu_2}, \ldots, P_{u_m} \times P_{\nu_m}$ constitute a partition of $P \times P$. This means that for each ordered pair of sites $(s, t) \in P \times P$, there is exactly one pair $(u, \nu) \in \Xi$ with $(s, t) \in P_u \times P_\nu$. We say that (u, ν) *represents* (s, t).

2. Each pair $(u, \nu) \in \Xi$ is c-well-separated, i.e.,we have

$$(c + 2) \max\{|P_u| - 1, |P_\nu| - 1\} \leqslant |\sigma(u)\sigma(\nu)|, \tag{4.1}$$

   where $\sigma(u), \sigma(\nu)$ are arbitrary sites in P_u and P_ν chosen by the algorithm.

Since in the unit disk graph metric the diameter $\operatorname{diam}(P_u)$ is at most $|P_u| - 1$ and since $|\sigma(u)\sigma(\nu)| \leqslant d(\sigma(u), \sigma(\nu))$, (4.1) implies that

$$(c + 2) \max\{\operatorname{diam}(P_u), \operatorname{diam}(P_\nu)\} \leqslant d(\sigma(u), \sigma(\nu)), \tag{4.2}$$

which is (almost) the traditional well-separation condition as described in Section 2.2. However, (4.1) is easier to check algorithmically and has additional advantages that we will exploit in our routing scheme below.

Gao and Zhang show that their algorithm yields a c-WSPD with $m = O(\vartheta c^2 n \log n)$ pairs, where ϑ is the density of P. More precisely, they prove the following lemma:

Lemma 4.1 (Lemma 4.3 and Corollary 4.6 in [GZ05]). *For each node $u \in H$, the WSPD Ξ has $O(\vartheta c^2 |P_u|)$ pairs that contain u.* □

4.3 Further Properties of the WSPD for UD(P)

We begin with two technical lemmas on WSPDs that will be useful later on. The first lemma shows that the choice of the sites $\sigma(u)$ for the nodes $u \in H$ is essentially arbitrary.

Lemma 4.2. *Let Ξ be a c-WSPD for P and let s, t be two sites such that the pair $(u, v) \in \Xi$ represents (s, t). Then $c \operatorname{diam}(P_u) \leqslant c(|P_u| - 1) \leqslant d(s, t)$.*

Proof. By triangle inequality and (4.1) we have

$$\begin{aligned} |st| &\geqslant |\sigma(u)\sigma(v)| - 2\max\{\operatorname{diam}(P_u), \operatorname{diam}(P_v)\} \\ &\geqslant (c+2)\max\{|P_u| - 1, |P_v| - 1\} - 2\max\{\operatorname{diam}(P_u), \operatorname{diam}(P_v)\}. \end{aligned}$$

Since $|P_u| - 1$ and $|P_v| - 1$ are upper bounds for $\operatorname{diam}(P_u)$ and $\operatorname{diam}(P_v)$, respectively, and since $d(s, t) \geqslant |st|$, we get

$$d(s, t) \geqslant c\max\{|P_u| - 1, |P_v| - 1\} \geqslant c\max\{\operatorname{diam}(P_u), \operatorname{diam}(P_v)\},$$

and the claim follows. □

The next lemma shows that short distances are represented by singletons in the WSPD.

Lemma 4.3. *Let Ξ be a c-WSPD for P and let $s, t \in P$ be two sites with $d(s, t) < c$. If $(u, v) \in \Xi$ represents (s, t), then $P_u = \{s\}$ and $P_v = \{t\}$.*

Proof. If $d(s, t) < c$, by Lemma 4.2 we have $c(|P_u| - 1) \leqslant d(s, t) < c$, and thus $|P_u| < 2$. By symmetry the same holds for $|P_v|$. □

4.4 The Routing Scheme

Let ϑ be the density of P. First we describe a routing scheme whose parameters depend on ϑ. Then we show how to remove this dependency and extend the scheme to work with arbitrary density. Our routing scheme uses the WSPD described in Section 4.2, and it is based on the following idea: let Ξ be the c-WSPD for UD(P) and let T be the EMST for P used to compute it. We distribute the information about the pairs in Ξ among the sites in P (in a way to be described later) such that each site stores $O(\vartheta c^2 \log n)$ pairs in its routing table. To route from s to t, we explore T, starting from s, until we find the site r with the pair (u, v) representing (s, t) stored in $\rho(r)$. Our scheme will guarantee that s and r are sites in P_u, and therefore it suffices to walk along T_u to find r (see Figure 4.2). This is called the *local routing*. With (u, v), we store in $\rho(r)$ the *middle site* m on a shortest path from r to $\sigma(v)$, i.e., a vertex "halfway" between r and $\sigma(v)$. We recursively route from r to m and when reaching m from m to t. To keep track of intermediate targets during the recursion, we store a stack in the header. This second step, the recursive routing through the middle site, we call the *global routing*.

We now describe our routing scheme in detail. Let $1 + \varepsilon$, $\varepsilon > 0$, be the desired stretch factor.

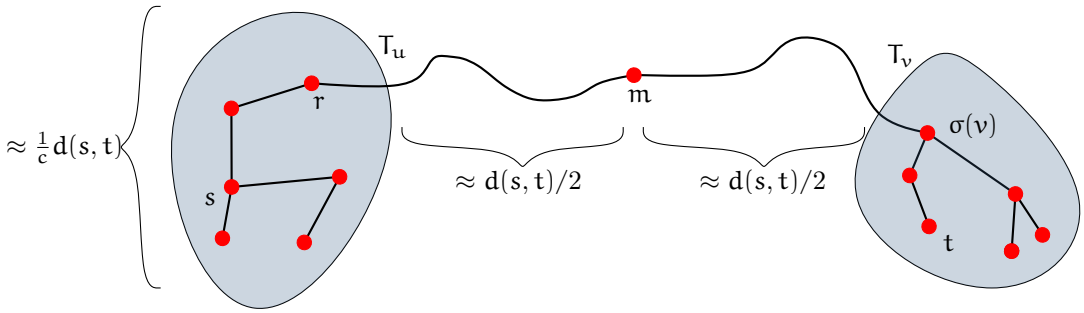


Figure 4.2: To route a packet from s to t, we first walk along T_u until we find r. Then we recursively route from r to m and from m to t.

4.4.1 Preprocessing

The preprocessing phase works as follows. We set $c = (\alpha/\varepsilon)\log D$, where D is the Euclidean diameter of $UD(P)$ and α is a sufficiently large constant we will fix later. Then we compute a c-WSPD for $UD(P)$. As explained in Section 4.2, the WSPD consists of a bounded degree spanning tree T of $UD(P)$, a hierarchical balanced decomposition H of T whose nodes $u \in H$ correspond to subtrees T_u of T, and a sequence $\Xi = (u_1, v_1), (u_2, v_2), \ldots, (u_m, v_m)$ of $m = O(\vartheta c^2 n \log n) = O(\vartheta \varepsilon^{-2} n \log n \log^2 D)$ well-separated pairs that represent a partition of $P \times P$.

First, we determine the *labeling* ℓ for the sites in P. For this, we perform a postorder traversal of H. Let l be a counter which is initialized to 1. Whenever we encounter a leaf of H, we set the label $\ell(s)$ of the corresponding site $s \in P$ to l, and we increment l by 1. Whenever we visit an internal node u of H for the last time, we annotate it with the interval I_u of the labels in T_u. Thus, a site $s \in P$ lies in a subtree T_u if and only if $\ell(s) \in I_u$. Each label has $O(\log n)$ bits.

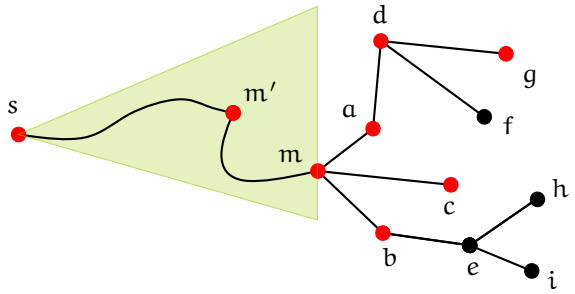
Next, we describe the routing tables. Each routing table consists of two parts, the *local* routing table and the *global* routing table. The local routing table $\rho_L(s)$ of a site s stores the neighbors of s in T, in counterclockwise order, together with the levels in H of the corresponding edges (cf. Section 4.2). Since T has degree at most 6, each local routing table consists of $O(\log n)$ bits. The global routing table $\rho_G(s)$ of a site s is obtained as follows: we go through all $O(\log n)$ nodes u of H that contain s in their subtree T_u. By Lemma 4.1, Ξ contains at most $O(\vartheta c^2 |P_u|)$ well-separated pairs in which u represents one of the sets. We assign $O(\vartheta c^2) = O(\vartheta \varepsilon^{-2} \log^2 D)$ of these pairs to s, such that each pair is assigned to exactly one site in P_u. For each pair (u, v) assigned to s, we store the interval I_v corresponding to P_v. Furthermore, if $\sigma(v)$ is not a neighbor of s in $UD(P)$, we store the label $\ell(m)$ of the *middle site* m of a shortest path π from s to $\sigma(v)$. Here, m is a site on π that minimizes the maximum distance, $\max\{d(s,m), d(m,\sigma(v))\}$, to the endpoints of π. A site s lies in $O(\log n)$ different sets P_u, at most one for each level of H. For each such set, we store $O(\vartheta \varepsilon^{-2} \log^2 D)$ pairs in $\rho_G(s)$, each of which requires $O(\log n)$ bits. Thus, ρ_G has $O(\vartheta \varepsilon^{-2} \log^2 n \log^2 D)$ bits.

Finally, we argue that the routing scheme can be computed efficiently.

Lemma 4.4. *The preprocessing time for the routing scheme that is described above is* $O(n^2 \log n + \vartheta n^2 + \vartheta \varepsilon^{-2} n \log n \log^2 D)$.

Proof. The c-WSPD can be computed in $O(\vartheta c^2 n \log n) = O(\vartheta \varepsilon^{-2} n \log n \log^2 D)$ time using the algorithm by Gao and Zhang [GZ05]. Within the same time bound, we can distribute the WSPD-pairs to the sites in P and compute the labels for P.

It remains to compute the middle sites; we do this for all pairs $(s, t) \in P \times P$ as follows: we first compute $UD(P)$ explicitly. Since P has density ϑ, we have $O(\vartheta n)$ edges in $UD(P)$, and we can compute it naively in time $O(n^2)$. For each $s \in P$, we compute the shortest path tree $\mathcal{T}$ with root s. This takes total time $O(n^2 \log n + \vartheta n^2)$, using n invocations of Dijkstra's algorithm.



$H_a = (g : 12, d : 11, a : 7)$
$H_b = (b : 9)$
$H_c = (c : 8)$
$H_m = (g : 12, d : 11, b : 9, c : 8, a : 7, m : 6)$

$d(s, m') = 4$

Figure 4.3: m is the middle site for g and d. From b onwards m' is a better middle site.

For each $s \in P$, we perform a post-order traversal of the shortest path tree $\mathcal{T}$ to find the middle sites for all s-t-paths. First, for each leaf t of $\mathcal{T}$, we create a max-heap that contains t with $d(s, t)$ as the key. We now describe how to process a site m during the traversal. First, we merge the heaps of all children of m into a new heap H_m and we insert m into H_m with $d(s, m)$ as key, see Figure 4.3. During the traversal, we maintain the invariant that H_m contains all sites that are descendants of m in $\mathcal{T}$ for which we have not yet found a middle site. Furthermore, since $d(s, t)$ increases monotonically along every root-leaf path in $\mathcal{T}$, the sites for which m might be the middle site are a prefix of the decreasingly sorted distances $d(s, t)$ with $t \in H_m$. Thus, to find the sites in H_m for which m is the middle site, we repeatedly perform an extract-max operation on H_m to obtain the next candidate t. Then, we compare the value of $\max\{d(s, m), d(m, t)\}$ with $\max\{d(s, m'), d(m', t)\}$, where m' is the parent of m in $\mathcal{T}$. That is, we check if m' is a "better" middle site than m. If not, m must be the middle site for s-t. Otherwise, m cannot be the middle site for any other site in H_m, and we proceed with our traversal. Using, e.g., Fibonacci Heaps, we can merge two heaps in $O(1)$ time and perform an extract-max operation in $O(\log n)$ amortized time [CLRS09]. Since each element of $\mathcal{T}$ is inserted and extracted at most once, we need $O(n \log n)$ time to find the middle sites for s. Thus, we can find all middle sites in time $O(n^2 \log n)$ and the total preprocessing time is $O(n^2 \log n + \vartheta n^2 + \vartheta \varepsilon^{-2} n \log n \log^2 D)$. □

4.4.2 Routing a Packet

Suppose we are given two sites s and t, and we would like to route a packet from s to t. Recall our overall strategy: we first perform a local exploration of $UD(P)$ in order to discover a site r that stores a pair $(u, v) \in \Xi$ representing (s, t) in its global routing table $\rho_G(r)$. To find r, we consider the subtrees of T that contain s by increasing size, and we perform an Euler tour in each subtree until we find r. In $\rho_G(r)$ we have stored the middle site m of a shortest path from r to $\sigma(v)$. We put the label $\ell(t)$ into the header, and we recursively route the packet from r to m. Once we reach m, we retrieve the original target t from the header and recursively route from m to t, see Algorithm 4.1 for pseudo-code.

Local Routing: The Euler-Tour. We start at s, and we would like to find the site r that stores the pair (u, v) representing (s, t). By construction, both s and r are contained in P_u, and it suffices to perform an Euler tour on T_u to discover r. Since we do not know u in advance, we begin with the leaf in H that contains s, and we explore all nodes on the path to the root until we find u.

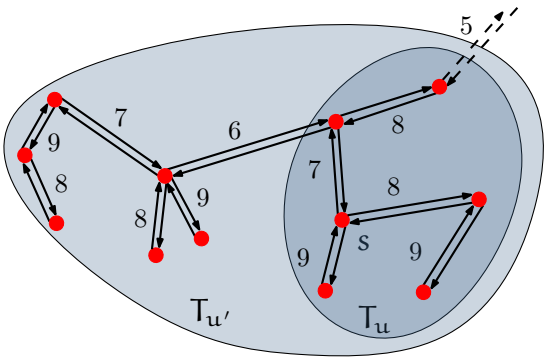


Figure 4.4: To find r we do an Euler Tour on T_u, the subtree that contains s whose edges have level at least 7. Since we do not find r, we search the next larger subtree $T_{u'}$, where u' is the parent of u in H by decreasing the search level to 6.

We store s as the *start site* in the header h. Let $w \in H$ be the node to be explored, and let $l = \delta(w)$ be the level of w in H. We store l in h. Recall that T_w is a connected component of the forest induced by all edges of level at least l. We perform an Euler tour on T_w using the local routing tables as follows: starting at s, we follow the first edge in $\rho_L(s)$ that has level at least l. Every time we visit a site r, we check for all WSPD-pairs (u, v) in $\rho_G(r)$ whether $\ell(t) \in I_v$, i.e., whether $t \in P_v$. If so, we clear the local routing information from h, and we proceed with the global routing. If not, we scan $\rho_L(r)$ for the next edge in $\rho_L(r)$ that has level at least l, going back to the beginning of $\rho_L(r)$ if necessary, and we follow this edge. For this, we must remember in h the edge through which we last entered r (note that we must store only the last edge of the tour). Once we reach s for the last time (i.e., through the last edge in $\rho_L(s)$ with level at least l), we

decrease l by one and restart the process. Decreasing l corresponds to proceeding with the parent of w in H.

Global Routing: The WSPD. Suppose we are at a site s such that $\rho_G(s)$ contains the pair (u, v) with the target t being in P_v. If t is not a neighbor of s, then $\rho_G(s)$ also contains the label of a middle site m for (u, v). We push (the label of) t onto the header stack, and we use $\ell(m)$ as the new target. Then we perform a local routing, starting at s, in order to find a pair (u', v') with $m \in P_{v'}$. If t is a neighbor of s, we go directly to t. Since t may be an intermediate target, we pop the next element from the header stack and set it as the new target label. If the header stack is empty, t is our final destination.

```
Input: currentSite s, targetLabel ℓ(t), header h
Output: nextSite, nextTargetLabel, header
if ℓ(s) = ℓ(t) then                          /* intermediate target reached?  */
    if h.stack = ∅ then                                    /* final target?  */
        return (s, ⊥, ⊥)
    else
        return (s, h.stack.pop(), h)
else if ρ(s) stores a WSPD-pair (u, v) with t ∈ P_v then  /* global routing */
    h.startSite ← ∅
    if s and t are neighbors in UD(P) then
        return (t, ℓ(t), h)
    else
        nextTargetLabel ← label of middle site for (u, v)
        h.stack.push(ℓ(t))
        return (s, nextTargetLabel, h)
else                                                       /* local routing */
    if h.startSite = ∅ then
        h.startSite ← s
        h.level ← δ(s)
    r ← next clockwise neighbor of s with level of edge sr ⩾ h.level
    if r = ⊥ then                                  /* Euler tour is finished */
        h.level ← h.level − 1
        return (s, ℓ(t), h)
    else
        return (r, ℓ(t), h)
```

Algorithm 4.1: The routing algorithm.

4.4.3 Analysis of the Routing Scheme

We now show that our routing scheme is correct and has low stretch, i.e., that for any two sites $s, t \in P$, it yields a routing path $s = s_0, \ldots, s_k = t$ of length at most $(1 + \varepsilon)d(s, t)$.

Correctness. First, we consider only small distances and show that in this case our routing scheme produces an actual shortest path.

Lemma 4.5. *Let* s, t *be two sites in* P *with* $d(s,t) < c$*. Then, the routing scheme produces a routing path* $s_0, s_1, \ldots, s_k$ *with the following properties*

(i) $s_0 = s$ *and* $s_k = t$*,*

(ii) $d_\rho(s,t) = d(s,t)$*, and*

(iii) the header stack is in the same state at the beginning and at the end of the routing path.

Proof. We prove that our routing scheme has properties (i)-(iii) by induction on the rank of $d(s,t)$ in the sorted list of the pairwise distances in $\mathrm{UD}(P)$.

For the base case, consider the edges st in $\mathrm{UD}(P)$, i.e., $d(s,t) = |st| \leqslant 1$. By Lemma 4.3, there exists a pair (u,v) with $P_u = \{s\}$ and $P_v = \{t\}$. Thus, Algorithm 4.1 correctly routes to t in one step and does not manipulate the header stack. All properties are fulfilled.

Now, consider an arbitrary pair s, t with $1 < d(s,t) < c$. By Lemma 4.3, there is a pair (u,v) with $P_u = \{s\}$ and $P_v = \{t\}$. By construction, (u,v) is stored in $\rho_G(s)$ and the routing algorithm directly proceeds to the global routing phase. Since $d(s,t) > 1$, the routing table contains a middle site m and since P_u and P_v are singletons, m is a middle site on a shortest path from s to t. Algorithm 4.1 pushes $\ell(t)$ onto the stack and sets m as the new target. By induction, the routing scheme now routes the packet along a shortest path from s to m (i, ii), and when the packet arrives at m, the target label $\ell(t)$ is at the top of the stack (iii). Thus, Algorithm 4.1 executes line 5, and routes the packet from m to t. Again by induction, the packet now follows a shortest path from m to t (i, ii), and when the packet arrives at t, the stack is in the same state a before pushing $\ell(t)$ (iii). The claim follows. □

Building on Lemma 4.5, we can now prove that our scheme is correct.

Lemma 4.6. *Let* s, t *be two sites in* P*. Then, the routing scheme produces a routing path* $s_0, s_1, \ldots, s_k$ *with the following properties*

(i) $s_0 = s$ *and* $s_k = t$*, and*

(ii) the header stack is in the same state at the beginning and at the end of the routing path.

Proof. Again, we use induction on the rank of $d(s,t)$ in the sorted list of pairwise distances in $\mathrm{UD}(P)$. If $d(s,t) < c$, the claim is immediate by Lemma 4.5.

Now, consider an arbitrary pair $s, t \in P$. By construction, our routing scheme will eventually find a site $r \in P$ whose global routing table stores a WSPD-pair (u,v) that represents (s,t), together with a middle site m (m exists for $d(s,t) \geqslant c$ large enough). So far, the stack remains unchanged (see Figure 4.5). Algorithm 4.1 pushes $\ell(t)$ onto

the stack and sets m as the new target. By induction, the routing scheme routes the packet correctly from s to m (i), and when the packet arrives at m, the target label $\ell(t)$ is at the top of the stack (ii). Thus, Algorithm 4.1 executes line 5, and routes the packet from m to t. Again by induction, the packet arrives at t, with the stack in the same state as before pushing $\ell(t)$ (i, ii). □

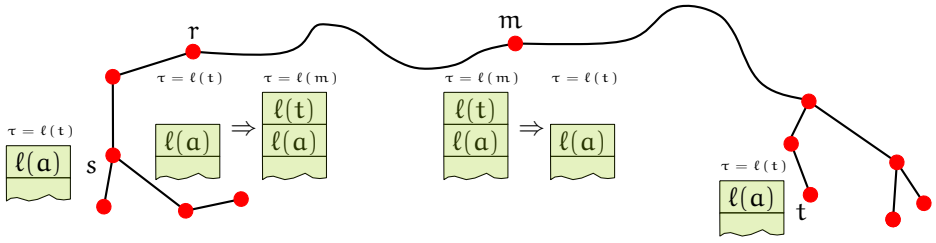


Figure 4.5: How the stack (green) and the target label τ changes due to the global routing. The label $\ell(a)$ is on top of the stack before and after the global routing.

Stretch Factor. The analysis of the stretch factor requires some more technical work. We begin with a lemma that justifies the term "middle site".

Lemma 4.7. *Let s, t be two sites in P with $d(s,t) \geqslant c \geqslant 14$ and let $(u,v) \in \Xi$ be the WSPD-pair that represents (s,t). If m is a middle site of a shortest path from s to $\sigma(v)$ in $UD(P)$, then*

(i) $d(s,m) + d(m,t) \leqslant (1 + 2/c)d(s,t)$, *and*

(ii) $d(s,m), d(m,t) \leqslant (5/8)d(s,t)$.

Proof. For (i) we use that m is the middle site on a shortest s-$\sigma(v)$-path and we apply the triangle inequality twice to get

$$\begin{aligned} d(s,m) + d(m,t) &\leqslant d(s,m) + d(m,\sigma(v)) + d(\sigma(v),t) && \text{(triangle inequality)} \\ &= d(s,\sigma(v)) + d(\sigma(v),t) && (m \text{ is middle site}) \\ &\leqslant d(s,t) + 2d(\sigma(v),t) && \text{(triangle inequality)} \\ &\leqslant (1 + 2/c)d(s,t), && \text{(Lemma 4.2)} \end{aligned}$$

where the last inequality follows from Lemma 4.2 and since $d(\sigma(v),t) \leqslant \operatorname{diam}(P_v)$.

For (ii) let π be a shortest path from s to $\sigma(v)$ that contains m, and let m' be the point on π with distance $d(s,\sigma(v))/2$ from s and from $\sigma(v)$ (m' may lie on an edge of π). Since the edges of π have length at most 1, there is a site m'' on π with $d(m',m'') = |m'm''| \leqslant 1/2$. Furthermore, the triangle inequality and Lemma 4.2 yield $d(s,\sigma(v)) \geqslant (1 - 1/c)d(s,t) \geqslant c - 1$. Hence,

$$\begin{aligned}\max\{d(s,m''),d(m'',\sigma(v)\} &\leqslant d(s,\sigma(v))/2+1/2\\ &\leqslant \big(1/2+1/(2c-2)\big)d(s,\sigma(v)) \qquad (*)\\ &= (c/(2c-2))d(s,\sigma(v)),\end{aligned}$$

where (*) is due to the fact that the triangle inequality, Lemma 4.2, and $d(s,t)\geqslant c$ yield

$$d(s,\sigma(v))\geqslant (1-1/c)d(s,t)\geqslant c-1.$$

For $c\geqslant 14$, the site m'' is distinct from s and $\sigma(v)$, and the distances to and from the middle site m are at most

$$\max\{d(s,m),d(m,\sigma(v))\}\leqslant \max\{d(s,m''),d(m'',\sigma(v))\}\leqslant (c/(2c-2))d(s,\sigma(v)). \quad (4.3)$$

Using the triangle inequality and Lemma 4.2 again, we get

$$(1-1/c)d(m,t)\leqslant d(m,\sigma(v)) \quad (4.4)$$

and

$$d(s,\sigma(v))\leqslant (1+1/c)d(s,t). \quad (4.5)$$

Thus, we can derive

$$\begin{aligned}\max\{d(s,m),d(m,t)\} &\leqslant (1+1/(c-1))\max\{d(s,m),d(m,\sigma(v))\} && \text{(by (4.4))}\\ &\leqslant (c/(2c-2))(1+1/c)(1+1/(c-1))d(s,t) && \text{(by (4.3, 4.5))}\\ &= (c/(2c-2))(1+2/(c-1))d(s,t)\\ &= (c^2+c)/(2(c-1)^2)d(s,t),\end{aligned}$$

and (ii) follows from $c\geqslant 14$. □

In the next lemma, we bound the distance traveled during the local routing.

Lemma 4.8. *Let s,t be two sites in P with $d(s,t)\geqslant c$. Then, the total distance traveled by the packet during the local routing phase before the WSPD-pair representing (s,t) is discovered is at most $(48/c)d(s,t)$.*

Proof. Let (u,v) be the WSPD-pair representing (s,t), and let $u_0,u_1,\ldots,u_k=u$ be the path in H from the leaf u_0 for s to u. Let $T_0,T_1,\ldots,T_k$ and $P_0,P_1,\ldots,P_k$ be the corresponding subtrees of T and sites of P. The local routing algorithm iteratively performs an Euler tour of $T_0,T_1,\ldots,T_k$ (the tour of T_k may stop early). An Euler tour in T_i takes $2|P_i|-2$ steps, and each edge has length at most 1. As described in Section 4.2, for $i=0.\ldots,k-1$, the WSPD ensures that

$$|P_i|\leqslant |P_{i+1}|-\lceil(|P_{i+1}|-1)/6\rceil\leqslant (5/6)|P_{i+1}|+1/6\leqslant (11/12)|P_{i+1}|,$$

since $|P_{i+1}| \geqslant 2$. It follows that the total distance for the local routing is at most

$$\sum_{i=0}^{k} (2|P_i| - 2) \leqslant 2|P_k| \sum_{i=0}^{k} (11/12)^i \leqslant 24|P_k|.$$

By Lemma 4.2, we have $d(s,t) \geqslant c(|P_u| - 1)$ and since $P_k = P_u$ the total distance is bounded by $24|P_u| \leqslant 24(1 + d(s,t)/c) \leqslant (48/c)d(s,t)$, where the last inequality is true for $d(s,t) \geqslant c$. □

Finally, we can bound the stretch factor:

Lemma 4.9. *For any two sites s and t, we have $d_\rho(s,t) \leqslant \big(1+\varepsilon\big)d(s,t)$.*

Proof. We show by induction on $d_\rho(s,t)$ that there is an $\alpha > 0$ with

$$d_\rho(s,t) \leqslant \big(1 + (\alpha/c)\log d(s,t)\big)d(s,t).$$

Since $d(s,t) \leqslant \operatorname{diam}(P) = D$, the lemma then follows from our choice of $c = (\alpha/\varepsilon)\log D$.

If $d(s,t) < c$, the claim follows by Lemma 4.5. If $d(s,t) \geqslant c$, Algorithm 4.1 performs a local routing to find the site r that has the WSPD-pair (u,v) representing (s,t) stored in $\rho_G(r)$. Then the packet is routed recursively from r to the middle site m and from m to t. By Lemma 4.8 the length of the routing path is $d_\rho(s,t) \leqslant (48/c)d(s,t) + d_\rho(r,m) + d_\rho(m,t)$, and by induction we get

$$\begin{aligned} d_\rho(s,t) \leqslant (48/c)d(s,t) &+ \big(1 + (\alpha/c)\log d(r,m)\big)d(r,m) \\ &+ \big(1 + (\alpha/c)\log d(m,t)\big)d(m,t). \end{aligned}$$

Since m is a middle site on a shortest r-$\sigma(v)$-path in $\mathrm{UD}(P)$, Lemma 4.7(i),(ii) and the fact that $\log(5/8) \leqslant -1/2$ imply

$$\begin{aligned} d_\rho(s,t) &\leqslant (48/c)d(s,t) + \Big(1 + (\alpha/c)\log\big((5/8)d(r,t)\big)\Big)\big(d(r,m) + d(m,t)\big) \\ &\leqslant (48/c)d(s,t) + \Big(1 + (\alpha/c)\log(d(r,t)) - \alpha/2c\Big)(1 + 2/c)d(r,t). \end{aligned}$$

By triangle inequality and by Lemma 4.2 we have $d(r,t) \leqslant (1+1/c)d(s,t)$, which gives $(1+2/c)d(r,t) \leqslant (1+2/c)(1+1/c)d(s,t) \leqslant (1+4/c)d(s,t)$, for c large enough. Hence,

$$d_\rho(s,t) \leqslant (48/c)d(s,t) + \Big(1 + (\alpha/c)\log((1+1/c)d(s,t)) - \alpha/2c\Big)(1 + 4/c)d(s,t),$$

and for $\alpha > 192$, we can eliminate the first term to get

$$d_\rho(s,t) \leqslant \Big(1 + (\alpha/c)\log((1+1/c)d(s,t)) - \alpha/4c\Big)(1 + 4/c)d(s,t),$$

and since now $c \geqslant 192$ and hence $\log(1 + 1/c) \leqslant 1/8$, this is

$$\leqslant \Big(1 + (\alpha/c)\log(d(s,t)) - \alpha/8c\Big)(1 + 4/c)d(s,t) = (1 + (\alpha/c)\log d(s,t))d(s,t) + \Delta,$$

with $\Delta = -(\alpha/8c)(1+4/c)d(s,t)+(4/c)d(s,t)\big(1+(\alpha/c)\log d(s,t)\big)$. It remains to show that $\Delta \leqslant 0$, i.e., that

$$(4/c)d(s,t)\big(1+(\alpha/c)\log d(s,t)\big) \leqslant (\alpha/8c)(1+4/c)d(s,t).$$

Now, since we picked $c = (\alpha/\varepsilon)\log D$ and $\alpha \geqslant 192$, we have

$$1+(\alpha/c)\log(d(s,t)) \leqslant 2 \leqslant (\alpha/32)(1+4/c),$$

as desired. This finishes the proof. □

Combining Lemma 4.4 and 4.9 we obtain the following theorem.

Theorem 4.10. *Let P be a set of n sites in the plane with density ϑ. For any $\varepsilon > 0$, we can preprocess P into a routing scheme for $UD(P)$ with labels of size $O(\log n)$ bits and routing tables of size $O(\vartheta\varepsilon^{-2}\log^2 n \log^2 D)$, where D is the diameter of $UD(P)$. For any two sites s,t, the scheme produces a routing path with $d_\rho(s,t) \leqslant (1+\varepsilon)d(s,t)$ and during the routing the maximum header size is $O(\log n \log D)$. The preprocessing time is $O(n^2\log n + \vartheta n^2 + \vartheta\varepsilon^{-2}n\log n\log^2 D)$.*

4.5 Extension to Arbitrary Density

Let $1+\varepsilon$, $\varepsilon > 0$, be the desired stretch factor. To extend the routing scheme to point sets of unbounded density, we follow a strategy similar to Gao and Zhang [GZ05, Section 4.2]: we first pick an appropriate $\varepsilon_1 > 0$, and we compute an ε_1-net $R \subseteq P$, i.e., a subset of sites such that each site in P has distance at most ε_1 to the closest site in R and such that any two sites in R have distance at least ε_1, see Figure 4.6.

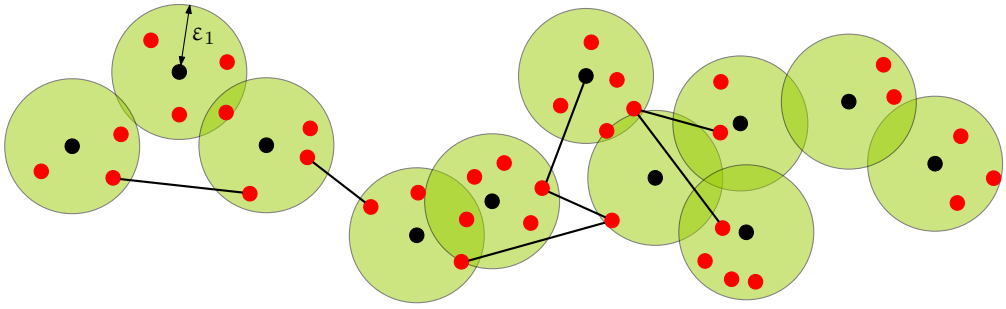


Figure 4.6: The set R (black) and the bridges (endpoints of black edges) form the set Z.

It is easy to see that R has density $O(\varepsilon_1^{-2})$ (see Lemma 4.11), and we would like to represent each site in P by its closest site in R. However, the connectivity in $UD(R)$ might differ from $UD(P)$. To rectify this, we add additional sites to R. This is done as follows: two sites $s,t \in R$ are called *neighbors* if $|st| > 1$, but there are $p,q \in P$ such that s,p,q,t is a path in $UD(P)$ and such that $|sp| \leqslant \varepsilon_1$ and $|qt| \leqslant \varepsilon_1$ (possibly, $s = p$ or $q = t$). In this case, p and q are called a *bridge* for s,t. Let R' be a point set that

contains an arbitrary bridge for each pair of neighbors in R. Set $Z = R \cup R'$. A simple volume argument bounds the density of Z.

Lemma 4.11. *The set R has density $O(\varepsilon_1^{-2})$ and the set Z has density $O(\varepsilon_1^{-3})$.*

Proof. Let D be a unit disk and let D' be the disk with radius $1 + \varepsilon_1/2$ concentric to D. For each $s \in R \cap D$, the disk $D(s, \varepsilon_1/2)$ with center s and radius $\varepsilon_1/2$ is contained in D'. Let $s, t \in R$ be two sites. By construction $|st| \geqslant \varepsilon_1$ and thus the disks $D(s, \varepsilon_1/2)$ and $D(t, \varepsilon_1/2)$ are disjoint. A disk of radius $\varepsilon_1/2$ has area $\pi\varepsilon_1^2/4$ and the area of D' is $\pi(1+\varepsilon_1/2)^2$. Thus we can place $O(\varepsilon_1^{-2})$ such disks into D' disjointly. Hence, the density of R is $O(\varepsilon_1^{-2})$.

Now, let D'' be the disk with radius $1 + \varepsilon_1$ concentric to D. To bound the density of Z, let $p \in (Z \setminus R) \cap D$ be a site that belongs to a bridge. Suppose that $s \in R$ is responsible for p being a bridge site. Then we have $|sp| \leqslant \varepsilon_1$ and by construction $s \in D''$. We charge p to s. The same volume argument as above shows that $|R \cap D''| = O(1/\varepsilon_1^{-2})$. Below we show that s has $O(\varepsilon_1^{-1})$ neighbors and thus can get $O(\varepsilon_1^{-1})$ charges from bridge sites. Hence, the number of bridge sites in $Z \cap D$, and also the density of Z, is $O(\varepsilon_1^{-3})$.

Consider the annulus A around s with inner radius 1 and outer radius $1 + 2\varepsilon_1$. All neighbors of s must lie in A. Let A' the annulus concentric to A with inner radius $1 - \varepsilon_1/2$ and outer radius $1 + (5/2)\varepsilon_1$. The area of A' is

$$\pi(1 + (5/2)\varepsilon_1)^2 - \pi(1 - \varepsilon_1/2)^2 = 6\varepsilon_1 + 6\varepsilon_1^2,$$

and thus, since R is an ε_1-net, we can place $O(\varepsilon_1^{-1})$ sites of R in A. Hence, s has $O(\varepsilon_1^{-1})$ neighbors, as claimed. □

Furthermore, Gao and Zhang show the following:

Lemma 4.12 (Lemma 4.8 and Lemma 4.9 in [GZ05])**.** *Computing Z needs $O((n/\varepsilon_1^2) \log n)$ time, and if $d^Z(\cdot,\cdot)$ denotes the shortest path distance in $UD(Z)$, then, for any $s, t \in R$, we have $d^Z(s, t) \leqslant (1 + 12\varepsilon_1)d(s, t) + 12\varepsilon_1$.*

Now, our extended routing scheme proceeds as follows: first, we compute R and Z as described above, and we perform the preprocessing algorithm for Z with ε_1 as stretch parameter. We assign arbitrary new labels to the sites in $P \setminus Z$. Then, we extend the label $\ell(s)$ of each site $s \in P$, such that it also contains the label of a site in R closest to s. The label size remains $O(\log n)$.

To route between two sites $s, t \in P$, we first check whether we can go from s to t in one step (we assume that this can be checked locally in the routing function). If so, we route the packet directly. Otherwise, we have $d(s, t) > 1$. Let $s', t' \in R$ be the closest sites in R to s and to t. By construction, we can obtain s' and t' from $\ell(s)$ and $\ell(t)$. Now, we first go from s to s'. Then, we use the low-density algorithm to route from s' to t' in $UD(Z)$, and finally we go from t' to t in one step. Using the discussion above, the total routing distance is bounded by

$$d_\rho(s, t) \leqslant |ss'| + d^Z_\rho(s', t') + |t't|,$$

where $d_\rho^Z(\cdot,\cdot)$ is the routing distance in $UD(Z)$. By Lemma 4.9 and 4.12, this is

$$\begin{aligned}&\leqslant \varepsilon_1 + (1+\varepsilon_1)d^Z(s',t') + \varepsilon_1\\ &\leqslant 2\varepsilon_1 + (1+\varepsilon_1)\big((1+12\varepsilon_1)d(s',t') + 12\varepsilon_1\big),\end{aligned}$$

and by using the triangle inequality twice this is

$$\leqslant 2\varepsilon_1 + (1+\varepsilon_1)\big((1+12\varepsilon_1)(d(s,t)+2\varepsilon_1) + 12\varepsilon_1\big).$$

Rearranging and using $d(s,t) > 1$ yields

$$\leqslant (1 + 29\varepsilon_1 + 50\varepsilon_1^2 + 24\varepsilon_1^3)d(s,t) \leqslant (1+\varepsilon)d(s,t),$$

where the last inequality holds for $\varepsilon_1 \leqslant \varepsilon/103$. This establishes our main theorem for this chapter:

Theorem 4.13. *Let P be a set of n sites in the plane. For any $\varepsilon > 0$, we can preprocess P into a routing scheme for $UD(P)$ with labels of $O(\log n)$ bits and routing tables of size $O(\varepsilon^{-5}\log^2 n \log^2 D)$, where D is the diameter of $UD(P)$. For any two sites s,t, the scheme produces a routing path with $d_\rho(s,t) \leqslant (1+\varepsilon)d(s,t)$ and during the routing the maximum header size is $O(\log n \log D)$. The preprocessing time is $O(n^2 \log n + \varepsilon^{-3}n^2 + \varepsilon^{-5} n \log n \log^2 D)$.*

Proof. The theorem follows from the above discussion and from the fact that the set Z has density $O(\varepsilon^{-3})$, by our choice of ε_1. □

4.6 Conclusion

We have presented an efficient routing scheme for unit disk graphs that produces a routing path whose length can be made arbitrarily close to optimal. For this, we used the fact that the unit disk graph metric admits a small WSPD. Our techniques almost solely rely on properties of well-separated pairs and thus we expect our approach to generalize to other graph metrics for which WSPDs can be found. One such example is the *hop-distance* $d_h(\cdot,\cdot)$ in unit disk graphs, in which all edges have length 1. Let P be a set of sites and let $\mathrm{diam}_h(P)$ denote the diameter of P in terms of $d_h(\cdot,\cdot)$. Since $\mathrm{diam}_h(P) \leqslant |P| - 1$ and $|st| \leqslant d_h(s,t)$ for every two sites $s,t \in P$, the well-separation condition (4.1) implies a separation with respect to the hop-distance. Thus, we can also find a routing scheme that approximates the number of hops used in the routing path instead of its Euclidean length.

Various open questions remain. First of all, it would be interesting to improve the size of the routing tables. One way to achieve this might be to decrease the dependency on ε. The ε^{-5}-factor seems to be rather high. It is mostly owed to the ε^{-3}-factor we introduced in Section 4.5 when going from bounded to unbounded density. Further improvements might be on the side of the WSPD: traditional WSPDs have only $O(c^2 n)$

pairs, while the WSPD of Gao and Zhang has an additional logarithmic factor. Whether this factor can be avoided is still an open question and any improvement in the number of pairs would immediately decrease the size of our routing tables by the same amount.

Furthermore, our routing scheme makes extensive use of a modifiable header. While this is consistent with the usual model for routing schemes, the scheme of Yan et al. managed to avoid the need of a header completely. In order to be completely comparable to their result, we would need to have a routing scheme that only requires a small routing table to produce a routing path with stretch $1 + \varepsilon$.

II

Transmission Graphs

Chapter 5

Spanners for Transmission and Disk Graphs

From now on, we consider transmission graphs, a generalization of unit disk graphs. Like almost all geometric intersection graphs, transmission graphs may be very dense and may contain $\Theta(n^2)$ edges. Thus, standard graph algorithms, like breadth first search (BFS), run slowly when applied to a dense transmission graph [CLRS09], since an *explicit* representation of all edges of the graph is required. This is particularly unsatisfying since transmission graphs admit a linear size *implicit* representation: we store the coordinates of each site together with its associated radius.

For some applications a sparse approximation of a transmission graph G that preserves distances suffices, i.e., we would like to have a spanner for G (cf. Section 2.1). Furthermore, we want to construct the spanner efficiently without generating an explicit representation of G. More precisely, our input is a set of sites $P \subset \mathbb{R}^2$, each having an associated radius $r_p > 0$, and a parameter $t > 1$. Our goal is to construct a sparse subgraph $H \subseteq G$ such that for any two sites $p, q \in P$ we have $d_H(p,q) \leqslant t d_G(p,q)$, where d_H and d_G are the shortest path distances in H and G, respectively, when edges are weighted by their Euclidean length. The graph H is called a t-spanner for G and we want to construct it in time almost linear in the size of P.

One might wonder if we can always find a t-spanner for G at all. For unit- and general disk graphs Fürer and Kasivisawnathan show that this is possible [FK12]. Their construction is based on the Yao graph [Yao82] and leads to an efficient algorithm. Peleg and Roditty [PR10] give a spanner construction for transmission graphs in any metric space with bounded doubling dimension. However, their algorithm has two drawbacks, making it not applicable for our purposes: it needs an explicit representation of the graph and the size of the spanner depends logarithmically on the radius ratio Ψ. For the latter, they even show that there are metric spaces with bounded doubling dimension where this dependency cannot be avoided.

We continue these studies by giving an algorithm that constructs a t-spanner for a transmission graph of a planar set of point sites ($P \subset \mathbb{R}^2$) with respect to the Euclidean metric. The algorithm has almost linear running time and constructs a spanner with a linear number of edges (for any constant t), independent of the radius ratio Ψ.

Our construction is also based on the *Yao graph* [Yao82]. The basic Yao graph is a t-spanner for the complete graph defined by n sites in the plane (with Euclidean distances as the weights of the edges). To determine the sites adjacent to a particular site q, we divide the plane by equally spaced rays emanating from q and connect q to its closest site in each wedge (the number of wedges increases as t gets smaller). See Figure 5.1. Adapting this construction to transmission graphs poses a severe computational difficulty, as we want to consider in each wedge only the sites p with $q \in D(p)$ (i.e., that form an incoming edge for q) and to pick the closest site to q only among those. Since finding the exact closest site turns out to be difficult, we need to relax this requirement in a subtle way, without hurting the approximation too much. This makes it possible to construct the spanner efficiently.

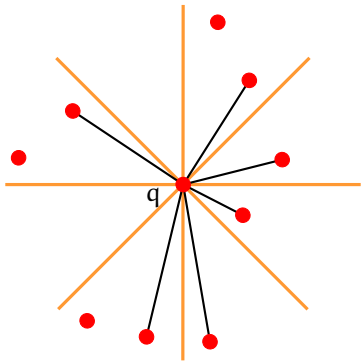


Figure 5.1: The Yao edges for q.

Even with a good t-spanner at hand, we sometimes wish to obtain exact solutions for certain problems on disk graphs. Working in this direction, Cabello and Jejčič gave an $O(n \log n)$ time algorithm for computing a BFS tree in a unit-disk graph, rooted at any given site [CJ15]. For this, they exploited the special structure of the Delaunay triangulation of the disk centers. We show that our spanner admits similar properties for transmission graphs. As a first application of our spanner, we get an efficient algorithm to compute a BFS tree in a transmission graph rooted at any given site.

For other applications, we consider the reachability oracles we construct in Chapter 6. Recall that a reachability oracle is a data structure that can answer *reachability queries*: given two vertices s and t, determine whether there is a directed path from s to t (cf. Chapter 1). The quality of a reachability oracle is measured by its query time, its space requirement, and its preprocessing time. For transmission graphs, we can ask for a more general *geometric* reachability query: given a vertex s and *any* point $q \in \mathbb{R}^2$, determine whether there is a vertex t such that there is a directed path from s to t in G, and q lies in the disk of t. We show how to extend any given reachability oracle to answer geometric queries with a small *additive* increase in space and query time. Furthermore, we will make extensive use of our spanner in Chapter 6 to decrease the preprocessing time of our reachability oracles.

5.1 Our Results

In Section 5.2, we show how to compute, for every fixed $t > 1$, a t-spanner H of G. Our construction is quite generic and can be adapted to several situations. In the simplest

case, if the spread Φ of P is bounded, we can obtain a t-spanner in time $O(n(\log n + \log \Phi))$ (Section 5.2.1). With slightly more work, we can weaken the assumption to a bounded *radius ratio* Ψ (recall that Ψ is the ratio between the largest and smallest radius in P), giving a running time of $O(n(\log n + \log \Psi))$ (Section 5.2.2). Recall that a bound on Φ implies a bound on Ψ, by Observation 2.1. Using even more advanced data structures, we can compute a t-spanner in time $O(n \log^5 n)$, without any dependence on Φ or Ψ (Section 5.2.3). Finally, in Section 5.3.1 we modify the last construction and make it work also for disk graphs. It will produce a spanner for $D(P)$ in expected time $O(n2^{\alpha(n)} \log^{10} n)$ and thus it gives the first non-trivial spanner construction for disk graphs that is independent of the radius ratio, improving the currently best known result of Fürer and Kasivisawnathan [FK12].

In Section 5.4.1 we show how to adapt a result by Cabello and Jejčič [CJ15] to compute a BFS-tree in a transmission graph, from any given vertex $p \in P$, in $O(n \log n)$ time, once we have the spanner ready.

In Section 5.4.2 we show how to use a spanner to extend a reachability oracle to answer geometric reachability queries. Specifically, we show that any reachability oracle for a transmission graph with radius ratio Ψ that requires $S(n)$ space and answers a query in $Q(n)$ time, can be extended in $O(n \log n \log \Psi)$ time to an oracle that can answer geometric reachability queries. The new oracle requires $S(n) + O(n \log \Psi)$ space and answers a query in $Q(n) + O(\log n \log \Psi)$ time.

5.2 Spanner Constructions for Transmission Graphs

First, we give a spanner construction for the transmission graph whose running time depends on the spread. Later, in Section 5.2.2, we will tune this construction so that the running time depends on the radius ratio. The main result which we prove in this section is as follows.

Theorem 5.1. *Let G be the transmission graph for a set P of n points in the plane with spread Φ. For any fixed $t > 1$, we can compute a t-spanner for G in $O(n \log \Phi)$ time. The construction needs $O(n \log \Phi)$ space.*

Let ρ be a ray originating from the origin and let $0 < \alpha < 2\pi$. A *cone* with *opening angle* α and *middle axis* ρ is the closed region containing ρ and bounded by the two rays obtained by rotating ρ clockwise and counterclockwise by $\alpha/2$.

Given a cone C and a point $q \in \mathbb{R}^2$, we write C_q for the copy of C obtained by translating the origin to q. We call q the *apex* of C_q. Ideally, our spanner should look as follows. Let $\mathcal{C}$ be a set of k cones with opening angles $2\pi/k$ that partition the plane. For each site $q \in P$ and each cone $C \in \mathcal{C}$, we pick the site $p \in P \cap C_q$ with $q \in D(p)$ that is closest to q (see Figure 5.2). We add the edge pq to H. The resulting graph has $O(kn)$ edges. Using standard techniques, one can show that H is a t-spanner, if k is

large enough as a function of t. This construction has been reported before and seems to be folklore [Car16, PR10].

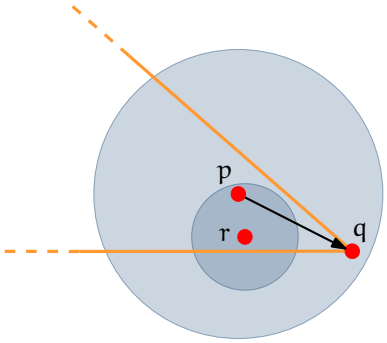


Figure 5.2: A cone C_q (orange) at a site q. Since $q \notin D(r)$, we pick the edge pq.

Unfortunately, the standard algorithms for computing the Yao graph do not seem to adapt easily to our setting without a penalty in their running times [CHT90]. The problem is that for each site q and each cone C_q, we need to search for a nearest neighbor of q only among those sites $p \in C_q$ such that $q \in D(p)$. This seems to be hard to do with the standard approaches. Thus, we modify the construction to search only for an *approximate* nearest neighbor of q and argue that picking an approximately shortest edge in each cone suffices to obtain a spanner.

We partition each cone C_q into "intervals" obtained by intersecting C_q with annuli around q whose inner and outer radii grow exponentially; see Figure 5.3. There can be only $O(\log \Phi)$ non-empty intervals. We cover each such interval by $O(1)$ grid cells whose diameter is "small" compared to the width of the interval. This gives two useful properties. (i) We only need to consider edges from the interval closest to q that contains sites with outgoing edges to q; all other edges to q will be longer. (ii) If there are multiple edges from the same grid cell, their endpoints are close together, and it suffices to consider only one of them.

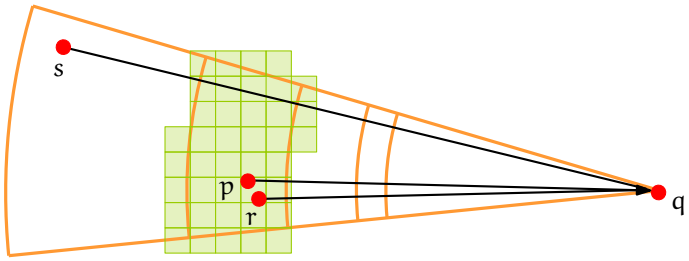


Figure 5.3: A cone C_q covered by discretized intervals. We only need one of the edges pq, rq for H.

To make this approach more concrete, we define a decomposition of P into pairs of subsets of P contained in certain grid cells. These pairs will be well-separated and represent a discretized version of the intervals (see Definition 5.2 below). This is motivated by the spanner construction based on the well-separated pair decomposition (WSPD), as described in Section 2.2.1.

Let $c > 1$ be a parameter and let $\Xi = \{(A_1, B_1), \ldots, (A_m, B_m)\}$ be a c-WSPD for P. To obtain the WSPD spanner for P, we put in the spanner for each pair (A_i, B_i) in the WSPD an edge ab, where a is an arbitrary site in A_i and b is an arbitrary site in B_i

(cf. Section 2.2.1). It turns out that a similar approach works for transmission graphs. However, since they are directed, we need to find for *each* site in B_i an incoming edge from a site in A_i, if such an edge exists, and vice versa. This causes two difficulties: we cannot afford to check all possible edges in $A_i \times B_i$, since this would lead to a quadratic running time, and we cannot control the indegree of a site p since it may belong to many sets A_i and B_i. We address the second problem by taking only $O(1)$ edges into a particular site q, within each of the k cones of the Yao construction described above. For the first problem, we identify in each A_i a special subset that "covers" all edges from a site in A_i to a site in B_i, such that each the total size of these subsets is linear in the size of P.

The concrete implementation of this idea is captured by Definition 5.2. A pair (A_i, B_i) corresponds to sets $P \cap \sigma$ and $P \cap \tau$ for two grid cells σ, τ that have the same diameter and that are well separated (Property (i)). For a grid cell τ, we define a particular subset $R_\tau \subseteq P \cap \tau$ to be the set of sites *assigned* to τ. Property (ii) in Definition 5.2 guarantees that each edge pq of G with $q \in \sigma$ and $p \in \tau$ is "covered" by R_τ in the sense that there is *some* site in R_τ that has an outgoing edge to q (see Figure 5.4).

Definition 5.2. *Let $c > 2$ and let G be the transmission graph of a planar point set P. A* c-separated annulus decomposition *for G consists of a finite set $\mathcal{G} \subset \bigcup_{i=0}^{\infty} \mathcal{G}_i$ of* grid cells*, a symmetric* neighborhood relation $N \subseteq \mathcal{G} \times \mathcal{G}$ *between these cells, and a subset of* assigned sites $R_\tau \subseteq P \cap \tau$ *for each grid cell $\tau \in \mathcal{G}$. A* c-separated annulus decomposition *for G has the following properties:*

(i) For every $(\sigma, \tau) \in N$ we have $\operatorname{diam}(\sigma) = \operatorname{diam}(\tau)$ and $d(\sigma, \tau) = \gamma \operatorname{diam}(\sigma)$, for some $\gamma \in [c-2, 2c)$.

(ii) for every edge pq of G, there is a pair $(\sigma, \tau) \in N$ with $q \in \sigma$, $p \in \tau$, and with a site $r \in R_\tau$ such that $q \in D(r)$.

The following fact is a direct consequence of Definition 5.2. For each cell $\sigma \in \mathcal{G}$, we define its *neighborhood* as $N(\sigma) = \{\tau \mid (\sigma, \tau) \in N\}$.

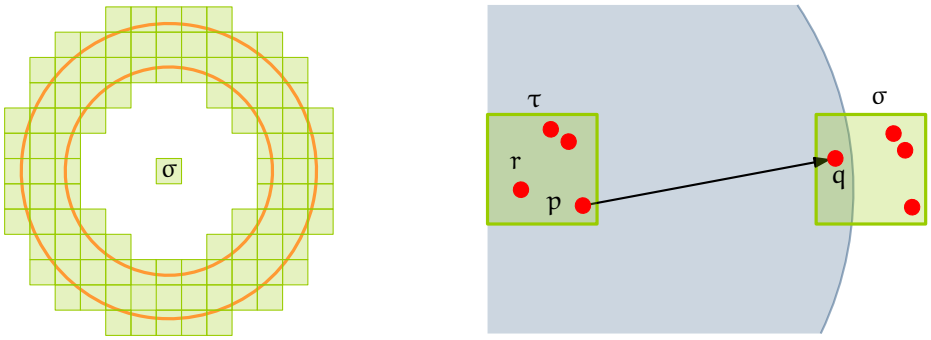
Lemma 5.3. *For each cell $\sigma \in \mathcal{G}$, we have $|N(\sigma)| = O(c^2)$, and for each cell $\tau \in \mathcal{G}$ the number of cells $\sigma \in \mathcal{G}$ such that $\tau \in N(\sigma)$ is $O(c^2)$.*

Proof. This follows from Definition 5.2(i) via a standard volume argument as given, e.g., in the proof of Lemma 4.11. □

Given this decomposition, we first present a simple (and rather inefficient) rule for picking incoming edges such that the resulting graph is a t-spanner. Then we explain how to compute the decomposition using a *quadtree*. Finally, we exploit the quadtree to make the spanner construction efficient.

5.2.1 Efficient Spanner Construction for Sites with Bounded Spread

Let $t > 1$ be the desired stretch factor. We pick a suitable separation parameter c and a number of cones k that depend on t, as specified later. Let $(\mathcal{G}, N, R_\tau)$ be a c-separated



(i) The neighborhood $N(\sigma)$ of a cell σ covers an annulus.

(ii) Since $q \in D(r)$, the edge pq is covered by r if we put r into R_τ.

Figure 5.4: Illustration of Property (i) and (ii) in Definition 5.2.

annulus decomposition for G. For a cone $C \in \mathcal{C}$ and an integer $\ell \in \mathbb{N}$, we define C^ℓ as the cone with the same middle axis as C but with an opening angle ℓ times as large as the opening angle of C. For a grid cell $\sigma \in \mathcal{G}$, let C_σ be the copy of C with the center of σ as the apex.

To obtain a t-spanner $H \subseteq G$, we pick the incoming edges for each site $q \in P$ and each cone $C \in \mathcal{C}$ as follows (see Algorithm 5.1). We consider the cells $\mathcal{G}^q \subseteq \mathcal{G}$ containing q in increasing order of diameter. Let σ be one such cell containing q that we process. We traverse all neighboring cells τ of σ, that are contained in C^2_σ. For each such neighboring cell τ, we check if there exists a site $r \in R_\tau$ that has an outgoing edge to q. If such a site exists, we add to H an edge to q from a single, arbitrary, such site r. After considering *all* neighbors τ of σ we terminate the processing of q and C if we added at least one edge incoming to q. If we have not added any edge into q while processing all neighbors τ of σ, we continue to the next largest cell in $\mathcal{G}^q$ containing q. We use here the extended and translated cones C^2_σ (instead of the cone C_q) to gain certain flexibility that will be useful for later extensions of Algorithm 5.1.

```
1 G^q ← cells of G that contain q
2 Sort the cells in G^q in increasing order by diameter
3 Make q active
4 while q is active do
5 |   σ ← next largest cell in G^q
6 |   foreach cell τ ∈ N(σ) that is contained in C^2_σ do
7 |   |   if there is a site r ∈ R_τ with q ∈ D(r) then
8 |   |   |   Take an arbitrary such r, add the edge rq to H, and set q to inactive
```

Algorithm 5.1: Selecting the incoming edges for q and the cone C.

For each cone $C \in \mathcal{C}$ and each site $q \in P$ there is only one cell $\sigma \in \mathcal{G}^q$ that produces incoming edges for q. We have k cones and $|N(\sigma)| = O(c^2)$ by Lemma 5.3, so q has

$O(c^2k)$ incoming edges. It follows that the size of H is $O(n)$ since c and k are constants.

Next we show that H is a t-spanner. For this, we show that every edge pq of G is represented in H by a path of length approximately $|pq|$. We prove this by induction on the ranks of the edge lengths. This is done in a similar manner as for the standard Yao graphs, but with a few twists that require three additional technical lemmas. Lemma 5.4 deals with the imprecision introduced by taking the cone C^2_σ instead of C_q. It follows from this lemma that if pq is contained in the cone C_q then Algorithm 5.1 picks at least one edge rq with $r \in C^4_q$. Lemma 5.5 and Lemma 5.6 encapsulate geometric facts that are used to bound the distance between the endpoints r and p depending on whether $|rq|$ is larger or smaller than $|pq|$. Lemma 5.6 is due to Bose et al. [BDD+12] and for completeness we include their proof.

Lemma 5.4. *Let $c > 3 + \frac{2}{\sin(\pi/k)}$ and let $\ell \in \{1, \dots, \lfloor k/2 \rfloor\}$. Consider a cell $\sigma \in \mathcal{G}_i$ and a cone $C \in \mathcal{C}$. Fix two points $q, s \in \sigma$. Every cell $\tau \in \mathcal{G}_i$ with $d(\sigma, \tau) \geqslant (c-2)2^i$ that intersects the cone C^ℓ_q is contained in the cone $C^{2\ell}_s$. In particular, any point $p \in C^\ell_q$ with $|pq| \geqslant (c-2)2^i$ lies in a cell that is fully contained in $C^{2\ell}_s$.*

Proof. Let x be a point in $\tau \cap C^\ell_q$. By assumption, $|xq| \geqslant (c-2)2^i$. Let $D = D(x, 2^i)$ be the disk with center x and radius 2^i. Then, $\tau \subseteq D$. We show that $C^{2\ell}_s$ contains D and thus τ. Since σ has diameter 2^i, and C^ℓ_q contains x, the translated copy C^ℓ_s must intersect D. If $D \subset C^\ell_s$, we are done. Otherwise, there is a boundary ray ρ of C^ℓ_s that intersects the boundary of D. Let y be the first intersection of ρ with the boundary of D. See Figure 5.5.

Since $s \in \sigma$ and $x \in \tau$, the triangle inequality gives that $|ys| \geqslant |xs| - |xy| \geqslant (c-3)2^i$. Let ρ' be the boundary ray of $C^{2\ell}_s$ corresponding to ρ and let y' be the orthogonal projection of y onto ρ'. Since $|ys| \geqslant (c-3)2^i$ and since the angle between ρ and ρ' is $\pi\ell/k$, we get that $|yy'| \geqslant (c-3)2^i \sin(\pi\ell/k)$. It follows that $|yy'| \geqslant 2 \cdot 2^i$ for $c > 3 + \frac{2}{\sin(\pi\ell/k)}$. This holds for any $\ell \in \{1, \dots, \lfloor k/2 \rfloor\}$ if $c \geqslant 3 + \frac{2}{\sin(\pi/k)}$. Thus, $\tau \subset D \subset C^{2\ell}_s$. □

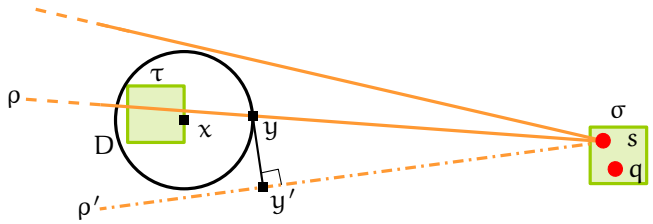
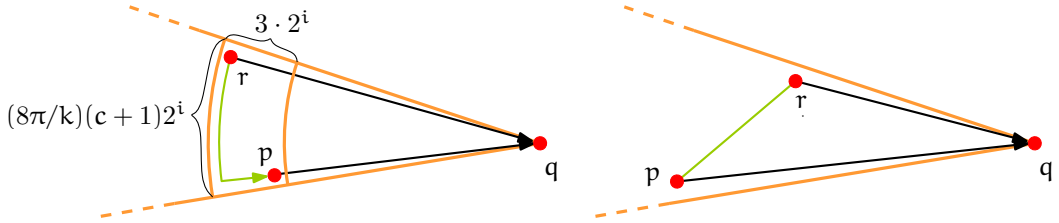


Figure 5.5: The boundary ray ρ of C^ℓ_s intersects the boundary of D in y.

Let p be a site in C_q such that pq is an edge of G, and $p \in \tau \in N(\sigma)$ where σ is a cell containing q. Then by Lemma 5.4, τ is contained in C^2_σ. It follows that Algorithm 5.1 either finds an edge rq before processing σ, or finds an edge rq with $r \in \tau$ while processing σ. By applying Lemma 5.4 again we get that $r \in C^4_q$. This fact is described in greater detail and is being used in the proof of Lemma 5.7 below.

Lemma 5.5. *Let* $C \in \mathcal{C}$*, and let* $q \in \mathbb{R}^2$*. Suppose there are two points* $p, r \in C_q^4$ *with* $(c-2)2^i \leqslant |pq| \leqslant |rq| \leqslant (c+1)2^i$*. Then* $|pr| \leqslant ((8\pi/k)(c+1)+3)2^i$*.*

Proof. The points p and r lie in an annulus around q with inner radius $(c-2)2^i$ and outer radius $(c+1)2^i$. Since $p, r \in C_q^4$, when going from r to p, we must travel at most $(8\pi/k)(c+1)2^i$ units along the circle around q with r on the boundary, then at most $3 \cdot 2^i$ units radially towards p. Thus, $|pr| \leqslant (8\pi/k)(c+1)2^i + 3 \cdot 2^i$. □



(i) Lemma 5.5. Two sites in an annulus are close to each other.

(ii) Lemma 5.6. If α is small and $|rq| \leqslant |pq|$, then $|pr| < |pq|$.

Lemma 5.6 (Lemma 10 in [BDD+12])**.** *Let* $k \geqslant 25$ *be large enough such that*

$$\frac{1+\sqrt{2-2\cos(8\pi/k)}}{2\cos(8\pi/k)-1} = 1 + \Theta(1/k) \leqslant t$$

for our desired stretch factor t*. For any three distinct sites* $p, q, r \in \mathbb{R}^2$ *such that* $|rq| \leqslant |pq|$ *and* $\alpha = \angle pqr$ *is between* 0 *and* $8\pi/k$*, we have* $|pr| \leqslant |pq| - |rq|/t$*.*

Proof. By the law of cosines and since $0 \leqslant \alpha \leqslant 8\pi/k$ we have that

$$|pr|^2 = |pq|^2 + |rq|^2 - 2|pq| \cdot |rq| \cos\alpha \leqslant |pq|^2 + |rq|^2 - 2|pq| \cdot |rq| \cos(8\pi/k)$$

Introducing t by adding and subtracting the same terms, this is

$$\begin{aligned} &= |pq|^2 - \frac{2}{t}|pq| \cdot |rq| + \frac{1}{t^2}|rq|^2 + \frac{t^2-1}{t^2}|rq|^2 - \frac{2(t\cos(8\pi/k)-1)}{t}|pq| \cdot |rq| \\ &= \Big(|pq| - \frac{|rq|}{t}\Big)^2 + \frac{t^2-1}{t^2}|rq|^2 - \frac{2(t\cos(8\pi/k)-1)}{t}|pq| \cdot |rq|. \end{aligned}$$

We complete the proof by showing that under the assumptions of the lemma $\frac{t^2-1}{t^2}|rq|^2 - \frac{2(t\cos(8\pi/k)-1)}{t}|pq| \cdot |rq| \leqslant 0$. We have that

$$\begin{aligned} \frac{t^2-1}{t^2}|rq|^2 - \frac{2(t\cos(8\pi/k)-1)}{t}|pq| \cdot |rq| &= \frac{|rq|^2}{t^2}\Big(t^2 - 1 - 2(t^2\cos(8\pi/k) - t)\frac{|pq|}{|rq|}\Big) \\ &\leqslant \frac{|rq|^2}{t^2}\Big(t^2 - 1 - 2(t^2\cos(8\pi/k) - t)\Big), \end{aligned}$$

where the last inequality follows since $|pq| \geqslant |rq|$ and

$$t \geqslant \frac{1+\sqrt{2-2\cos(8\pi/k)}}{2\cos(8\pi/k)-1} \geqslant \frac{1}{2\cos(8\pi/k)-1} \geqslant \frac{1}{\cos(8\pi/k)} ,$$

so $t\cos(8\pi/k) \geqslant 1$. Now we have that

$$t^2 - 1 - 2(t^2\cos(8\pi/k) - t) = (1-2\cos(8\pi/k))t^2 + 2t - 1 \leqslant 0$$

if $\cos(8\pi/k) > 1/2$ and

$$\frac{1+\sqrt{2-2\cos(8\pi/k)}}{2\cos(8\pi/k)-1} \leqslant t .$$

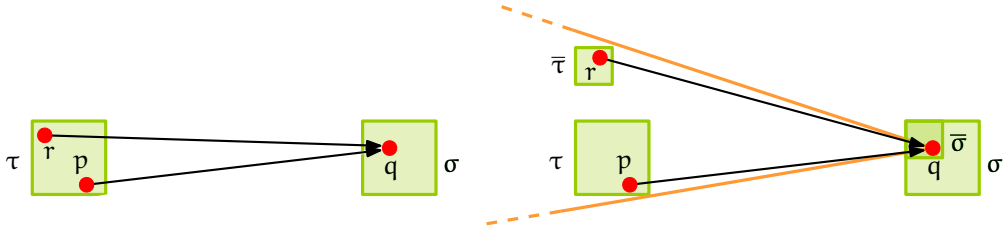
The latter inequality holds by assumption and $\cos(8\pi/k) > 1/2$ for $k \geqslant 25$. □

We are now ready to bound the stretch of the spanner H. This is done in two steps. In the first step (Lemma 5.7) we prove that for any edge pq of G which is not in H, there exists a shorter edge rq in H, such that r is "close" to p. This fact allows us to prove in the second step, via a fairly standard inductive argument, that H is indeed a t-spanner for G.

Lemma 5.7. *Let c and k be such that $c > 3 + \frac{2}{\sin(\pi/k)}$ as required by Lemma 5.4, k satisfies the conditions of Lemma 5.6 and, in addition, $c \geqslant 2 + \frac{2t}{t-1}$ and $k \geqslant \frac{16\pi t}{t-1}$. Let pq be an edge of G. Then either pq is in H or there is an edge rq in H such that $|pr| \leqslant |pq| - |rq|/t$.*

Proof. Let N be the neighborhood relation of the c-separated annulus decomposition used by Algorithm 5.1. Let $(\sigma,\tau) \in N$ be a pair of neighboring cells satisfying requirement (ii) of Definition 5.2 with respect to pq. In particular we have that $q \in \sigma$ and $p \in \tau$. If there is more than one such pair $(\sigma,\tau) \in N$, we consider the pair with minimum diameter. Let $\operatorname{diam}(\sigma) = 2^i$, that is, $\sigma,\tau \in \mathcal{G}_i$.

Let $C \in \mathcal{C}$ be the cone such that $p \in C_q$. Since $p \in C_q \cap \tau$ and since $d(\sigma,\tau) \geqslant (c-2)2^i$, Lemma 5.4 implies that $\tau \subset C^2_\sigma$. Hence, τ is considered for incoming edges for q (line 6 in Algorithm 5.1). We split the rest of the proof into two cases.



(i) Case 1: p and r are in the same cell σ.

(ii) Case 2: p and r are in different cells with different levels but in the same cone C^4_q.

Case 1: q remains active until (σ,τ) is considered. Requirement (ii) of Definition 5.2 guarantees that Algorithm 5.1 finds an incoming edge rq for q with $r \in \tau$. If $r = p$, we

are done, so suppose that $r \neq p$. Since $\operatorname{diam}(\sigma) = 2^i$ and $|rq| \geqslant d(\sigma,\tau) \geqslant (c-2)2^i$ we have

$$\begin{aligned}|pr| &\leqslant 2^i = |pq| - (|pq| - 2^i) \leqslant |pq| - (|rq| - 2\cdot 2^i)\\ &\leqslant |pq| - (|rq| - 2|rq|/(c-2)) \leqslant |pq| - |rq|(1-2/(c-2)) \leqslant |pq| - |rq|/t,\end{aligned}$$

for $c \geqslant 2 + \frac{2t}{t-1}$.

Case 2: q becomes inactive before (σ,τ) is considered. Then Algorithm 5.1 has selected an edge rq while considering a pair $(\bar\sigma,\bar\tau) \in N$ with $q \in \bar\sigma$, $r \in \bar\tau$ and $\operatorname{diam}(\bar\sigma) \leqslant 2^{i-1}$. We now distinguish two subcases.

Subcase 2a: $|rq| \geqslant |pq|$. From Property (i) of Definition 5.2, it follows that $d(\sigma,\tau) \geqslant (c-2)2^i$ and therefore $|pq| \geqslant (c-2)2^i$. It also follows from the same property that $d(\bar\sigma,\bar\tau) \leqslant 2c2^{i-1}$, so $|rq| \leqslant 2c2^{i-1} + 2\cdot 2^{i-1} = (c+1)2^i$. Combining these inequalities we obtain that $(c-2)2^i \leqslant |pq| \leqslant |rq| \leqslant (c+1)2^i$ and therefore $|pq| \geqslant |rq| - 3\cdot 2^i$. Lemma 5.5 implies that $|pr| \leqslant ((8\pi/k)(c+1)+3)2^i$, and thus we have

$$\begin{aligned}|pr| &\leqslant ((8\pi/k)(c+1)+3)2^i\\ &\leqslant |pq| - |pq| + ((8\pi/k)(c+1)+3)2^i\\ &\leqslant |pq| - \big(|rq| - 3\cdot 2^i - ((8\pi/k)(c+1)+3)2^i\big)\\ &\leqslant |pq| - \Big(|rq| - \frac{(8\pi(c+1)-6)2^i}{k}\Big)\\ &\leqslant |pq| - |rq|\Big(1 - \frac{(8\pi(c+1)-6)}{k(c-2)}\Big)\\ &\leqslant |pq| - |rq|\Big(1 - \frac{16\pi}{k}\Big)\,.\end{aligned}$$

The third inequality follows since $|pq| \geqslant |rq| - 3\cdot 2^i$ as we argued above, and the fifth inequality follows since $2^i \leqslant |rq|/(c-2)$. The last inequality holds for $c \geqslant 5$ (which follows from our assumptions). Now for $k \geqslant \frac{16\pi t}{t-1}$ we clearly have that

$$|pq| - |rq|\Big(1 - \frac{16\pi}{k}\Big) \leqslant |pq| - |rq|/t.$$

Subcase 2b: $|rq| < |pq|$. By assumption, we have $p \in C_q \subset C_q^4$. Furthermore, by applying Lemma 5.4 with the midpoint of $\bar\sigma$ as q, r as p, and q as s, in the statement of the lemma, we get that $r \in C_q^4$. Since $p, r \in C_q^4$ and since the opening angle of C_q^4 is $8\pi/k$, it follows from Lemma 5.6 that $|pr| \leqslant |pq| - |rq|/t$. □

Lemma 5.8. *For any $t > 1$, there are constants c and k such that H is a t-spanner for the transmission graph G.*

Proof. We pick the constants c and k so that Lemma 5.7 holds. We prove by induction on the indices of edges when ordered by their lengths, that for each edge pq of G, there is a path from p to q in H of length at most $t|pq|$. For the base case, consider the shortest edge pq in G. By Lemma 5.7, if pq is not in H then there is an edge rq in H such that

$|pr| \leqslant |pq| - |rq|/t$. Since pq is an edge of G, it follows that $r_p \geqslant |pq|$ and therefore pr must also be an edge of G, and it is shorter than pq. This gives a contradiction and therefore pq must be in H.

For the induction step, consider an edge pq of G. If pq is in H we are done. Otherwise by Lemma 5.7 there is an edge rq in H such that $|pr| \leqslant |pq| - |rq|/t$. As argued above, pr is an edge of G shorter than $|pq|$ so by the induction hypothesis, there is a path from p to r in H of length no larger than $t|pr|$. It follows that

$$d_H(p,q) \leqslant d_H(p,r) + |rq| \leqslant t|pr| + |rq| \leqslant t(|pq| - |rq|/t) + |rq| \leqslant t|pq|,$$

as required. □

Finding the Decomposition. We use a quadtree to define the cells of the decomposition [BCvKO08]. We recall that a *quadtree* is a rooted tree T in which each internal node has degree four. Each node ν of T is associated with a cell σ_ν of some grid $\mathcal{G}_i$, $i \geqslant 0$, and if ν is an internal node, the cells associated with its children partition σ_ν into four congruent squares, each with diameter $\operatorname{diam}(\sigma_\nu)/2$. If σ_ν is from $\mathcal{G}_i$ then we say that ν is of *level* i. Note that all nodes of T at the same distance from the root are of the same level.

Let c be the required parameter for the annulus decomposition. We scale P such that the closest pair in P has distance c. (We use P to denote also the scaled point set). Let L be the smallest integer such that we can translate P so that it fits in a single cell σ of $\mathcal{G}_L$. Since c is constant and P has spread Φ, the diameter of P (after scaling) is $c\Phi$ and therefore $L = O(\log \Phi)$. We translate P so that it fits in σ and we associate the root r of our quadtree T with this cell σ, i.e. $\sigma_r = \sigma$. By the definition of a level, r is of level L.

We continue constructing T top down as follows. We construct level $i-1$ of T, given level i, by splitting the cell σ_ν of each node ν, whose cell σ_ν is not empty, into four congruent squares, and associate each of these squares with a child of ν. We stop the construction of T after generating the cells of level 0. The scaling which we did to P ensures that each cell of a leaf node at level 0 contains at most one site.

We now set $\mathcal{G} = \{\sigma_\nu \mid \nu \in T\}$. We define N as the set of all pairs $(\sigma_\nu, \sigma_w) \in \mathcal{G} \times \mathcal{G}$ such that ν and w are at the same level in T and $d(\sigma_\nu, \sigma_w) \in [c-2, 2c)\operatorname{diam}(\sigma_\nu)$.[1] It remains to define the assigned sets R_τ for each $\tau \in \mathcal{G}$. Let pq be an edge in G and let $(\tau, \sigma) \in N$ be two grid cells with $p \in \tau$ and $q \in \sigma$. For a site $r \in \tau \cap P$ to "cover" pq according to Property (ii) in Definition 5.2, we must have $q \in D(r)$ and thus the radius of r needs to be at least $r_r \geqslant d(\sigma,\tau) \geqslant (c-2)\operatorname{diam}(\tau)$. If we include all sites with radius larger than $(c-2)\operatorname{diam}(\tau)$ into R_τ, we cannot control the size of R_τ. However, any disk of a site $\tau \cap P$ whose radius is larger than $2(c+1)\operatorname{diam}(\tau)$ will contain σ completely. These sites with large radius can be handled by including only the site with the largest radius into R_τ. More precisely, for $\tau \in \mathcal{G}$, we define R'_τ to be the set of all sites $p \in \tau \cap P$ with $r_p \in [c-2, 2(c+1))\operatorname{diam}(\sigma_\nu)$ and we let m_τ be the site in $\tau \cap P$ with the largest radius. We set $R_\tau = R'_\tau \cup \{m_\tau\}$.

[1] We denote the interval $[a\operatorname{diam}(\sigma_\nu), b\operatorname{diam}(\sigma_\nu))$ by $[a,b)\operatorname{diam}(\sigma_\nu)$.

Lemma 5.9. $(\mathcal{G}, \mathsf{N}, \mathsf{R}_\tau)$ *is a* c*-separated annulus decomposition for* G.

Proof. Property (i) of Definition 5.2 follows by construction. To prove that Property (ii) holds consider an edge $\mathtt{pq}$ of G. Let $\mathtt{i}$ be the integer such that $|\mathtt{pq}| \in [\mathsf{c}, 2\mathsf{c})2^{\mathtt{i}}$. Let σ, τ be the cells of $\mathcal{G}_{\mathtt{i}}$ with $\mathtt{p} \in \tau$ and $\mathtt{q} \in \sigma$. By construction, σ and τ are assigned to nodes of the quadtree and thus contained in $\mathcal{G}$. Since $\operatorname{diam}(\sigma) = \operatorname{diam}(\tau) = 2^{\mathtt{i}}$, we have

$$(\mathsf{c}-2)2^{\mathtt{i}} \leqslant |\mathtt{pq}| - 2\operatorname{diam}(\sigma) \leqslant \mathsf{d}(\sigma,\tau) \leqslant |\mathtt{pq}| < \mathsf{c}2^{\mathtt{i}+1},$$

and therefore $(\sigma, \tau) \in \mathsf{N}$ by our definition of N. Since $\mathtt{pq}$ is an edge of G, it follows that $\mathtt{r}_{\mathtt{p}} \geqslant |\mathtt{pq}| \geqslant \mathsf{c}2^{\mathtt{i}}$. If $\mathtt{r}_{\mathtt{p}} < (\mathsf{c}+1)2^{\mathtt{i}+1}$, then $\mathtt{p} \in \mathsf{R}'_\tau$ and we are done. Otherwise, $\mathtt{r}_{\mathtt{m}_\tau} \geqslant \mathtt{r}_{\mathtt{p}} \geqslant (\mathsf{c}+1)2^{\mathtt{i}+1}$, and $\mathtt{q} \in \sigma \subset \mathsf{D}(\mathtt{m}_\tau)$. □

Computing the Edges of H**.** We find edges for each cone $\mathsf{C} \in \mathcal{C}$ separately as follows. For each pair of neighboring cells σ and $\tau \in \mathsf{N}(\sigma)$ such that τ is contained in C^2_σ we find all incoming edges to sites in σ from sites in τ simultaneously. To do this efficiently, we need to sort the sites in σ along the $\mathtt{x}$ and $\mathtt{y}$ directions. Therefore, we process the cells bottom-up along T in order of increasing levels. In this way we can obtain a sorted list of the sites in each cell σ by merging the sorted lists of its children. See Algorithm 5.2.

```
1 Make all sites in P active
2 for i = 0, ..., L do
3     foreach ν ∈ T of level i do
4         Q ← active sites in σ_ν ∩ P
          // preproccesing
5         Sort Q in x and y-direction by merging the sorted lists of the ν's children
6         foreach τ ∈ N(σ_ν) contained in C²_{σ_ν} do
              // edge selection
7             For each site q ∈ Q, find an r ∈ R_τ with q ∈ D(r), if it exists; add the
              edge rq to H
8         Set all q ∈ Q for which at least one incoming edge was found to inactive
```

Algorithm 5.2: Selecting the edges for H for a fixed cone C.

Note that the edges selected by Algorithm 5.2 have the same properties as the edges selected by Algorithm 5.1. Thus, by Lemma 5.8, the resulting graph is a t-spanner. Let Q be the set of active sites in σ_ν when processing ν. Let $\tau \in \mathsf{N}(\sigma_\nu)$ such that τ is contained in $\mathsf{C}^2_{\sigma_\nu}$ and let $\mathsf{R} = \mathsf{R}_\tau$. Assume $|\mathsf{Q}| = \mathtt{n}$ and $|\mathsf{R}| = \mathtt{m}$. To find the edges from sites in R to sites in Q efficiently, we use the fact that these sets of sites are separated by a line ℓ parallel to either the $\mathtt{x}$- or the $\mathtt{y}$-axis.

Assume without loss of generality that ℓ is the $\mathtt{x}$-axis, the sites of R are above ℓ and the sites of Q are below ℓ, and assume that Q is sorted along ℓ. For each site $\mathtt{p} \in \mathsf{R}$ we take the part of $\mathsf{D}(\mathtt{p})$ which lies below ℓ and compute the union of these "caps". This union is bounded from above by ℓ and from below by the lower envelope of the arcs of

the boundaries of the caps. The complexity of the boundary of this union is $O(m)$ and it can be computed in $O(m \log m)$ time [SA96]. See Figure 5.8.

Once we have computed this union we check for each $q \in Q$ whether q lies inside it. This can be done by checking whether the intersection, z, of a vertical line through q with the union is above or below q. If q is above z then we add the edge rq to H where r is the site such that $z \in \partial D(r)$. We perform this computation for all sites in Q together by a simple sweep in x-direction while simultaneously traversing the lower envelope of the caps and the sites of Q. This clearly takes $O(m \log m + n)$ time, since we still need to find the lower envelope of the disk corresponding to R. We thus proved the following lemma.

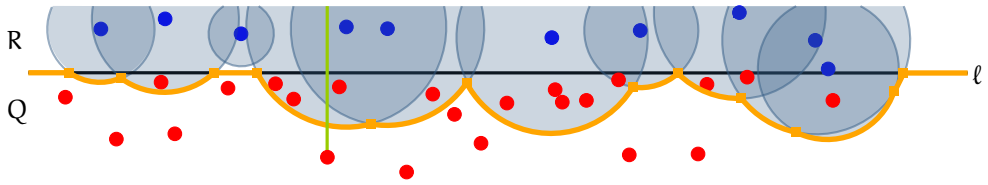


Figure 5.8: The lower envelope (orange), the sites Q (red) and R (blue), and the sweepline (green).

Lemma 5.10. *Let Q, R, and ℓ be as above with $|Q| = n$ and $|R| = m$. Suppose that Q is sorted along ℓ and that ℓ separates Q and R. We can compute in $O(m \log m + n)$ time for each $q \in Q$ one disk from R that contains it, provided that such a disk exists.*

Analysis. We prove that Algorithm 5.2 runs in $O(n \log \Phi)$ time and uses $O(n \log \Phi)$ space. The running time is dominated by the edge selection step that is described in Lemma 5.10. We argue that each site participates in $O(1)$ edge selection steps as a disk center (in R) and in $O(\log \Phi)$ edge selection steps as a site looking for incoming edges. From these observations (and the fact that $\Phi = \Omega(n^{1/2})$) the stated time bound essentially follows.

Lemma 5.11. *We construct the spanner H of the transmission graph G in $O(n \log \Phi)$ time and space.*

Proof. The quadtree T can be computed in $O(n \log \Phi)$ time and space [BCvKO08], and within this time bound we can also compute $N(\sigma_v)$, R_{σ_v}, and m_{σ_v} for each node $v \in T$.

Merging the sorted lists of the sites in σ_w for each child w of v to obtain the sorted list of the sites in σ_v (line 5 in Algorithm 5.2) takes time linear in the number of sites in σ_v. Summing up over all nodes v in a single level of T we get that the total merging time per level is $O(n)$, and $O(n \log \Phi)$ for all levels.

To analyze the time taken by the edge selection steps (line 7 in Algorithm 5.2), consider a particular pair $(\sigma, \tau) \in N$ for which the algorithm runs the edge selection step. By Lemma 5.10, if we charge m_τ by $O(1)$, each disk center in R'_τ by $O(\log n)$ and each active site in $\sigma \cap P$ by $O(1)$ then the total charges cover the cost of the edge selection

step for (σ, τ). There are $O(n \log \Phi)$ nodes in T and therefore $O(n \log \Phi)$ cells τ in $\mathcal{G}$. By Lemma 5.3 each such cell τ participates in an edge selection step of $O(c^2) = O(1)$ pairs. So the total charges to the site m_τ over all cells τ, is $O(n \log \Phi)$.

By construction, each $p \in P$ is assigned to $O(1)$ sets R'_τ and by Lemma 5.3 each τ participates in an edge selection step of $O(c^2) = O(1)$ pairs. It follows that the total charges to a site p from edge selection steps of pairs (σ, τ) such that $p \in R'_\tau$ is $O(\log n)$.

Finally, each site is active for $O(c^2) = O(1)$ pairs in N at each of $O(\log \Phi)$ levels. So the total charges to a site p from edge selection steps of pairs (σ, τ) such that p is active in $\sigma \cap P$ is $O(\log \Phi)$. We conclude that the total running time of all edge selection steps is $O(n \log n + n \log \Phi) = O(n \log \Phi)$, since $\log \Phi = \Omega(\log n)$. □

Theorem 5.1 follows by combining Lemmas 5.8 and 5.11.

5.2.2 From Bounded Spread to Bounded Radius Ratio

Let $P \subset \mathbb{R}^2$ be a set of sites with radius ratio Ψ. We extend our spanner construction from Section 5.2.1 such that the running time depends on Ψ, the ratio between the largest to smallest radius, rather than on the spread Φ. This is a more general result as we may assume that $\Psi \leqslant 2\Phi$ by Observation 2.1. We prove the following theorem.

Theorem 5.12. *Let G be the transmission graph for a set P of n sites in the plane with radius ratio Ψ. For any fixed $t > 1$, we can compute a t-spanner for G in $O(n(\log n + \log \Psi))$ time and $O(n \log \Psi)$ space.*

The main observation which we use is that sites that are close together form a clique in G and can be handled using classic spanner constructions, while sites that are far away from each other belong to distinct components of G and can be dealt with independently.

Given t, we pick sufficiently large constants $k = k(t)$ and $c = c(t)$ as specified in Section 5.2.1. We scale the input such that the *smallest radius* is c. Let $M = c\Psi$ be the largest radius after we did the scaling. First, we partition P into sets that are far apart and can be handled separately.

Lemma 5.13. *We can partition P into sets $P_1, \ldots, P_\ell$, such that each set P_i has diameter $O(n\Psi)$ and for any $i \neq j$, no site of P_i can reach a site of P_j in G. Computing the partition takes $O(n \log n)$ time and $O(n)$ space.*

Proof. We assign to each site $p \in P$ an axis-parallel square S_p that is centered at p and has side-length $2M$. We define the intersection graph G_S that has a vertex for each site in P, and an edge between two vertices p and q if and only if $S_p \cap S_q \neq \emptyset$. ($G_S$ is undirected.)

If follows that if there is no (undirected) path from p to q in G_S, then there is no (directed) path from p to q in G. We can compute the connected components of G_S in $O(n \log n)$ time by sweeping the plane using a binary search tree [PS85]. Let $P_1, \ldots, P_\ell$ be the vertex sets of these connected components. By construction, each set of sites P_i has diameter $O(nM)$ and for any $i \neq j$, no site in P_i can reach a site in P_j in G. □

By Lemma 5.13, we may assume that the diameter of our input set P is $O(n\Psi)$. We compute a hierarchical decomposition T for P as in Section 5.2.1, with a little twist as follows. We translate P so that it fits in a single grid cell σ of diameter $O(n\Psi)$. Starting from σ, we recursively subdivide each non-empty cell into four congruent cells of half the diameter. We do not subdivide cells of level 0 whose diameter is 1. We partition all cells of a particular level in $O(n)$ time and $O(n)$ space.

We construct a quadforest T such that the roots of its trees correspond to the non-empty cells of level $L = \lceil \log \Psi \rceil$ in our decomposition. Each internal node of T corresponds to a non-empty cell obtained when subdividing the cell of its parent. It suffices to store only the lowest L levels, since larger cells cannot contribute any edges to the spanner (as we will argue below). The forest T requires $O(n \log \Psi)$ space and we compute it in $O(n(\log n + \log \Psi))$ time.

We cannot derive from the quadforest T a c-separated annulus decomposition for G, as we did in Section 5.2.1. In particular a cell corresponding to a leaf of T may now contain many sites that are adjacent in G. For edges induced by such pairs of sites we cannot satisfy Property (ii) of Definition 5.2.

We can (and do) derive from T a *partial* c-separated annulus decomposition $(\mathcal{G}, N, R_\tau)$ exactly as described in Section 5.2.1 before Lemma 5.9: we set $\mathcal{G} = \{\sigma_\nu \mid \nu \in T\}$, define N as all pairs $(\sigma_\nu, \sigma_w) \in \mathcal{G} \times \mathcal{G}$ with $\operatorname{diam}(\sigma_\nu) = \operatorname{diam}(\sigma_w)$ and $d(\sigma_\nu, \sigma_w) \in [c-2, 2c) \operatorname{diam}(\sigma_\nu)$, and set $R_\tau = R'_\tau \cup \{m_\tau\}$ where $R'_\tau = \{p \in \tau \cap P \mid r_p \in [c-2, 2(c+1)) \operatorname{diam}(\tau)\}$. This decomposition satisfies Property (ii) of Definition 5.2 for all edges pq with $d(\sigma, \tau) \geqslant (c-2)$, where σ and τ are the cells of level 0 in T that contain q and p, respectively. The proof that Property (ii) of Definition 5.2 holds for these edges is the same as the proof of Lemma 5.9. In particular, in the proof of Lemma 5.9, the argument shows that pairs of cells at level i guarantee Property (ii) of Definition 5.2 for all edges with length in $[c-2, 2c)2^i$. Since the edges of G are of length at most $M = c\Psi$, the cells up to level $L = \lceil \log \Psi \rceil$ suffice to guarantee Property (ii) of Definition 5.2 for all edges pq with $d(\sigma, \tau) \geqslant (c-2)$.

We mark all sites of P as active, and we run Algorithm 5.2 of Section 5.2.1 using T and the partial c-separated annulus decomposition that we derived from it. The resulting graph H is not yet a t-spanner since the decomposition was only partial.

To make H a spanner we add more edges that "take care" of the edges not "covered" by the c-separated annulus decomposition. We consider each pair of level 0 cells σ and τ with $d(\sigma, \tau) < c-2$. The set of sites $Q = (P \cap \sigma) \cup (P \cap \tau)$ form a clique, since the distance between each pair of sites in Q is not larger than c. We compute a Euclidean t-spanner for Q of size $O(|Q|)$ in $O(|Q| \log |Q|)$ time [NS07] (e.g., the WSPD spanner described in Section 2.2.1) and for each (undirected) edge pq of this spanner we add pq and qp to H. As each site $p \in P$ participates in $O(c^2)$ such spanners, we generate $O(n)$ edges in total in $O(n \log n)$ time.

We now prove that H is indeed a t-spanner. The proof is analogous to the proof of Lemma 5.8.

Lemma 5.14. *For any $t > 1$, there are constants $c = c(t)$ and $k = k(t)$ such that H is a t-spanner for the transmission graph G.*

Proof. By construction, H is a subgraph of G. Let pq be an edge of G, and let σ and τ be the cells of level 0 with $q \in \sigma$ and $p \in \tau$. If $d(\sigma, \tau) < c - 2$, then the Euclidean t-spanner for σ and τ contains a path from p to q of length at most $t|pq|$.

For the remaining edges, the lemma is proved by induction on the rank of the edges when we sort them by length, as in Lemma 5.8. The proof is almost verbatim as before; we only comment on the base case. Let pq be the shortest edge in G. If the endpoints p and q lie in level 0 cells whose distance is less than $c - 2$, we have already argued that H contains an approximate path from p to q. Otherwise, the same argument as in Lemma 5.8 applies, and the algorithm includes pq in H. □

Using Lemma 5.14, Theorem 5.12 follows just as Theorem 5.1 in Section 5.2.1. The analysis of the space and time required by our construction is exactly as in Lemma 5.11, but now T has $O(\log \Psi)$ levels.

5.2.3 Spanners for Unbounded Spread and Radius Ratio

We eliminate the dependency of our bounds on the radius ratio at the expense of a more involved data structure and an additional polylogarithmic factor in the running time. Given $P \subset \mathbb{R}^2$ and the desired stretch factor $t > 1$, we choose appropriate parameters $c = c(t)$ and $k(t)$ as in Section 5.2.1 and rescale P such that the distance between the closest pair of points in P is $c + 2$.

To get the spanner for G, we compute a compressed quadtree T for P [HP11]. A *compressed quadtree* is a rooted tree in which each internal node has degree 1 or 4. Each node v is associated with a cell σ_v of a grid $\mathcal{G}_i$. If v has degree 4, then the cells associated with its children partition σ_v into 4 congruent squares of half the diameter, and at least two of them must be non-empty. If v has degree 1, then the cell associated with the only child w of v has diameter at most $\operatorname{diam}(v)/4$ and $(\sigma_v \setminus \sigma_w) \cap P = \emptyset$. Each internal node of T contains at least two sites in its cell and each leaf at most one site. For technical reasons we assume that the cell associated with a leaf v has diameter 1. Since v contains a single point p we can artificially guarantee this by shrinking the cell associated with v to the cell of diameter 1 containing p.

Note that, in contrast with (uncompressed) quadtrees, the diameter of σ_v may be smaller than 2^{L-i}, where i is the distance of v to the root and 2^L is the diameter of the root. A compressed quadtree for P with $O(n)$ nodes can be computed in $O(n \log n)$ time [HP11].

To simplify the notation in the rest of this section, we write $\operatorname{diam}(v)$ instead of $\operatorname{diam}(\sigma_v)$, and for two nodes v, w, we write $d(v, w)$ for $d(\sigma_v, \sigma_w)$.

Our approach is to use the algorithm from Section 5.2.1 on the compressed quadtree T. One problem with this approach is that the depth of T may be linear, so considering all sites for incoming edges at each level, as in Algorithm 5.2, would be too expensive. We tackle this difficulty by using the version of Chan's dynamic nearest neighbor data structure as given in Corollary 2.5 in Section 2.3.2 to speed up this stage.

Another problem arises when we try to use the approach from Section 5.2.1 on the compressed quadtree T. We need to define an appropriate neighborhood relation. The

neighborhood relation from Section 5.2.1 relied on the fact that in a quadtree each point appears for every i in the appropriate range in exactly one cell whose diameter is 2^i. This is no longer the case in a compressed quadtree.

As in Section 5.2.1, the neighborhood relation N which we define here would consist of pairs (σ_v, σ_w) such that $\operatorname{diam}(v) = \operatorname{diam}(w)$ and $d(v, w) \in [c - 2, 2c) \operatorname{diam}(v)$. The set R'_{σ_v} consists of all sites in $\sigma_v \cap P$ whose radius is in $[c - 2, 2(c + 1)) \operatorname{diam}(v)$, and we set $R_\tau = R'_\tau \cup \{m_\tau\}$ where m_τ is the site in τ with the largest radius. To make sure that N and R_τ fulfill Property (ii) of Definition 5.2, we insert $O(n)$ additional nodes into T so that $\mathcal{G}$ contains the appropriate cells. To find these nodes, we adapt the WSPD algorithm of Callahan and Kosaraju [CK95a] (see Algorithm 5.3).

```
   call wspd1(r) on the root of T
1  wspd1(v) :
2  if v is a leaf then
3  |  return ∅
4  else
5  |  Return the union of wspd1(w) and wspd2(w1, w2) for all children w and
   |  pairs of distinct children w1, w2 of v
1  wspd2(v, w) :
2  if d(v, w) ⩾ c max{diam(v), diam(w)} then
3  |  return {v, w}
4  else if  diam(v) ⩽ diam(w) then
5  |  return the union of wspd2(v, u) for all children u of w.
6  else
7  |  return the union of wspd2(u, w) for all children u of v
```

Algorithm 5.3: Computing a well-separated pair decomposition from a compressed quadtree T. We scale the input such that the distance between the closest pair of points is $c + 2$. This guarantees that when v and w are both leaves, wspd2(v, w) returns $\{v, w\}$.

Lemma 5.15. *Given a constant $c > 5$, we can in $O(n \log n)$ time insert $O(n)$ nodes into T so that $\mathcal{G} = \{\sigma_v \mid v \in T\}$ with N and R_σ defined as above is a c-separated annulus decomposition for G. In the same time, we can compute N and all sets R_σ.*

Proof. First, we run the usual algorithm for finding a c-well-separated pair decomposition on T [CK95a]; see Algorithm 5.3 for pseudocode. It is well known [LM12] that the algorithm runs in $O(n)$ time and returns a set Ξ of $O(n)$ pairs $\{v, w\}$[2] of nodes in T with the following properties:

(a) for each two distinct sites p, q, there is exactly one $\{v, w\} \in \Xi$ with $q \in \sigma_v$, $p \in \sigma_w$;

[2]Note that here the pairs of the WSPD are sets instead in tuples as defined in Section 2.2. We use this slightly different version since it is consistent with the Calahan-Kosaraju algorithm.

(b) for each $\{v, w\} \in \Xi$, we have $c \cdot \max\{\mathrm{diam}(v), \mathrm{diam}(w)\} \leqslant d(v, w)$;

(c) for every call wspd2(v, w), we have $\max\{\mathrm{diam}(v), \mathrm{diam}(w)\} \leqslant \min\{\mathrm{diam}(\overline{v}), \mathrm{diam}(\overline{w})\}$, where $\overline{v}$, $\overline{w}$ are the parents of v and w in T;

Note that (a) and (b) are the requirements for Ξ being a c-WSPD and that since we scaled P such that the closest pair has distance $c + 2$, (b) is satisfied by any pair of (non-empty) cells of $\mathcal{G}_0$.

For each pair $\{v, w\} \in \Xi$, we insert two nodes v' and w' into T such that $\mathrm{diam}(v') = \mathrm{diam}(w')$ and such that $d(v', w')$ is approximately $c \cdot \mathrm{diam}(v')$. Suppose that $\{v, w\}$ was generated through a call wspd2$(v, \overline{w})$ in Algorithm 5.3 (the case that $\{v, w\}$ was generated through the call wspd2$(\overline{v}, w)$ is similar). Let $r' = \min\{d(v, w)/c, \mathrm{diam}(\overline{w})\}$ and let r be equal to r' rounded down to the next power of 2.
Observe that

$$r \leqslant \mathrm{diam}(\overline{w}) \leqslant \mathrm{diam}(\overline{v}), \tag{5.1}$$

because $r \leqslant \mathrm{diam}(\overline{w})$ by definition, and $\mathrm{diam}(\overline{w}) \leqslant \mathrm{diam}(\overline{v})$ by (c) and our assumption that wspd2$(v, \overline{w})$ was called.
Furthermore, we have

$$\max\{\mathrm{diam}(v), \mathrm{diam}(w)\} \leqslant r. \tag{5.2}$$

This follows from (c) if $r' = \mathrm{diam}(\overline{w})$ and from (b) if $r' = d(v, w)/c$ (recall that $\mathrm{diam}(v)$ and $\mathrm{diam}(w)$ are powers of two).

It follows from (5.1) and (5.2) that we can insert nodes v' and w' into T between v and $\overline{v}$ and between w and $\overline{w}$, respectively, such that $\mathrm{diam}(v') = \mathrm{diam}(w') = r$ and such that $\sigma_v \subseteq \sigma_{v'} \subseteq \sigma_{\overline{v}}$ and $\sigma_w \subseteq \sigma_{w'} \subseteq \sigma_{\overline{w}}$.

We insert all these new nodes into T efficiently by partitioning them according to the parent-child pair in T that they should be inserted between. We sort all the new nodes x that should be inserted between each particular parent-child pair $\overline{v}, v$ by decreasing diameter and remove "duplicate nodes", i.e., in each group of nodes of the same diameter we leave only one. Finally, we insert into T a path consisting of the remaining nodes in order, making the first node on the path a child of $\overline{v}$ and the last node on the path a parent of v. It takes $O(n \log n)$ time to insert all the $O(n)$ new nodes.

To find the sets $R_\tau = R'_\tau \cup \{m_\tau\}$, we consider each site $p \in P$ and we identify the nodes v in T such that $p \in R_{\sigma_v}$ in $O(\log n)$ time as follows. Since $c > 5$ there are at most two integers i such that $r_p \in [c - 2, 2(c + 1))2^i$. For each such i, we identify (in $O(1)$ time) the cell $\sigma \in \mathcal{G}_i$ containing p and then determine whether σ is associated with a node v in T. The latter step requires $O(\log n)$ time with an appropriate data structure. If indeed there is such a node v, we insert p into R'_{σ_v}. Thus, the total time we spend to find all sets R'_τ is $O(n \log n)$. Within the same time we can find the sites m_τ and thus we can compute the sets $R_\tau = R'_\tau \cup \{m_\tau\}$ in total time $O(n \log n)$. The pairs in N can be computed similarly also in $O(n \log n)$ time.

We now argue that this construction yields a c-separated annulus decomposition for P. Property (i) of Definition 5.2 holds by construction. To prove that Property (ii) of Definition 5.2 holds consider some edge pq in G.

Since Ξ is a c-WSPD, by (a) there is a pair $\{v, w\} \in \Xi$ with $q \in \sigma_v$ and $p \in \sigma_w$. Suppose that $\{v, w\}$ was generated through the call wspd2$(v, \overline{w})$. Thus, we must have inserted nodes v' and w' into T with $\sigma_v \subseteq \sigma_{v'} \subseteq \sigma_{\overline{v}}$, $\sigma_w \subseteq \sigma_{w'} \subseteq \sigma_{\overline{w}}$, and with $\operatorname{diam}(v') = \operatorname{diam}(w') = r$. Hence, $q \in \sigma_{v'}$ and $p \in \sigma_{w'}$.

We claim that $(\sigma_{v'}, \sigma_{w'}) \in N$. To prove this claim observe that since $r \leqslant d(v, w)/c$ it follows that

$$d(v', w') \geqslant d(v, w) - 2r \geqslant cr - 2r = (c - 2)\operatorname{diam}(v'), \tag{5.3}$$

Furthermore, if $r' = d(v, w)/c$, then $d(v, w)/2c < r \leqslant d(v, w)/c$ and therefore

$$d(v', w') \leqslant d(v, w) \leqslant 2cr. \tag{5.4}$$

Since the pair $\{v, w\}$ was generated through a call wspd2$(v, \overline{w})$ we know that $d(v, \overline{w}) \leqslant c \operatorname{diam}(\overline{w})$. So if $r' = \operatorname{diam}(\overline{w})$ (so $r = r' = \operatorname{diam}(w') = \operatorname{diam}(v')$) then we have

$$d(v', w') \leqslant d(v, \overline{w}) + \operatorname{diam}(v') \leqslant (c + 1)r \leqslant 2cr. \tag{5.5}$$

By (5.3),(5.4) and (5.5), we get $(\sigma_{v'}, \sigma_{w'}) \in N$. Finally, since pq is an edge of G, we have $r_p \geqslant d(v', w') \geqslant (c - 2)\operatorname{diam}(w')$, by (5.3). If $r_p < (c + 1)\operatorname{diam}(w')$, then $p \in R'_{\sigma_{w'}}$. Otherwise let m be the site in $\sigma_{w'} \cap P$ with the largest radius. Then, $r_m \geqslant r_p \geqslant (c+1)\operatorname{diam}(w')$, so $D(m)$ contains $\sigma_{v'}$ and thus q. This establishes Property (ii) of Definition 5.2. □

Computing the Edges of H. As already mentioned, to construct the spanner $H \subseteq G$ for a stretch factor $t > 1$, we choose appropriate constants $k = k(t)$ and $c = c(t)$, scale P such that the closest pair has distance $c + 2$, and compute a compressed quadtree T for P. To obtain a c-separated annulus decomposition $(\mathcal{G}, N, R_\sigma)$ for G, we augment T with $O(n)$ nodes as described in the proof of Lemma 5.15.

We select the spanner edges for each cone $C \in \mathcal{C}$ separately, as follows. For each leaf v of T, we create a dynamic nearest neighbor (NN) data structure S_v as in Theorem 2.5 that contains initially the single point $p \in \sigma_v \cap P$. We call a site p *active* if $p \in S_v$ for some node v in T. So initially, all sites of P are active. Then we process the nodes of T in order of increasing diameter similarly to Algorithm 5.2 of Section 5.2.1.

Let w be the child of v such that $|S_w|$, the number of sites in S_w, is largest. We generate S_v from S_w by inserting into S_w all the active sites of the children of v other than w (we call this the *preproccesing* step at v). Then we use S_v to do the edge selection for all $\tau \in N(\sigma_v)$ contained in $C^2_{\sigma_v}$; see Algorithm 5.4. We take a site $r \in R_\tau$ and repeatedly query S_v for the site closest to r. Let q be the result. If rq is an edge in G, we add rq to H, delete q from S_v, and do another query with r. Otherwise, we continue with the next site of R, until all of R is processed. (This step is called the *edge selection* step at v.)

The edges selected by Algorithm 5.4 have the same properties as the edges selected by Algorithm 5.1. Thus, by Lemma 5.8 we obtain a t-spanner H. Next, we analyze the running time.

```
   // preproccesing
 1 Let w be the child of v whose S_w contains the most sites
 2 Insert all active sites of each child w' ≠ w of v into S_w
 3 Set S_v ← S_w
 4 foreach τ ∈ N(σ_v) contained in C^2_{σ_v} do
 5 |   foreach r ∈ R_τ do
   |   |   // edge selection
 6 |   |   q ← NN(v, r) // query S_v with r
 7 |   |   while q ∈ D(r) and q ≠ ∅ do
 8 |   |   |   add the edge rq to H; delete q from S_v; q ← NN(v, r)
 9 |   reinsert all deleted sites into S_v
10 delete all q from S_v for which at least one edge rq was found
```

Algorithm 5.4: Selecting incoming edges for the sites of a node v and a cone C.

Lemma 5.16. *Algorithm 5.4 has a total running time of* $O(n \log^5 n)$ *and it requires* $O(n)$ *space.*

Proof. It takes $O(n \log n)$ time to compute the compressed quadtree and to find the neighboring pairs as in Lemma 5.15. Initializing the nearest neighbor structures S_v at the leaves v takes $O(n)$ time.

Consider now the preprocessing phases at internal nodes v. That is, the construction of S_v from S_w where w is a child of v, by inserting into it the active sites from structures $S_{w'}$ from the children $w' \neq w$ of v. Since S_w is the largest structure among the structures of the children of v, each time a site is inserted, the size of the nearest neighbor structure that contains it increases by a factor of at least two. Thus, each site is inserted $O(\log n)$ times. By Theorem 2.5 each such insertion takes $O(\log^3 n)$ time. So the total time it takes to perform all these insertions is $O(n \log^4 n)$.

For the edge selection, consider two nodes v and w in T whose cells are neighbors. For each site r in $R_{\sigma_w} = R'_{\sigma_w} \cup m_{\sigma_w}$, we perform one nearest neighbor query at line 6 of Algorithm 5.4 (the initial query with r). We now evaluate what is the total time spent performing these *initial queries*.

By Lemma 5.3 each cell has $O(c^2)$ neighbors, so each site m_{σ_w} generates $O(c^2)$ queries. The total number of sites m_{σ_w} is equal to the number of nodes in T, which is $O(n)$. Therefore the total number of initial nearest neighbor queries generated by sites m_{σ_w} is $O(n)$. Each site is assigned to R'_{σ_w} for at most two nodes w and may generate $O(c^2)$ nearest neighbor queries when we process the neighboring cells of each such cell σ_w. Therefore, the total number of initial nearest neighbor queries generated by sites in sets R'_{σ_w} is also $O(n)$. By Theorem 2.5 the time it takes to perform a query is $O(\log^2 n)$ so the total time spent by initial queries is $O(n \log^2 n)$.

For each edge that we create in the while-loop of line 7, we perform at most two deletions, one insertion and one additional nearest neighbor query. Since H has $O(n)$ edges, the total time required to perform these operations is $O(n \log^5 n)$ by Theorem 2.5.

The total size of the compressed quadtree and of the associated data structures is $O(n)$. Furthermore, a dynamic nearest neighbor structure with m elements requires $O(m)$ space by Corollary 2.5. Thus, since at any time each site lies in at most one dynamic nearest neighbor structure, the total space requirement is $O(n)$. □

We conclude this section with the following theorem that follows from Lemma 5.16 and the discussion preceding it.

Theorem 5.17. *Let G be the transmission graph for an n-point set $P \subset \mathbb{R}^2$. For any $t > 1$, we can compute a t-spanner for G in $O(n \log^5 n)$ time and $O(n)$ space.*

5.3 Spanners for Disk Graphs

In this section we slightly tweak Algorithm 5.4 from Section 5.2.3 to make it work with disk graphs. Let $P \subset \mathbb{R}^2$ be a set of n sites with radius ratio Ψ. Given $t > 1$, the algorithm by Fürer and Kasivisawnathan can compute a t-spanner for the disk graph $D(P)$ in time $O(n^{4/3+\varepsilon} \log^{2/3} \Psi)$, where $\varepsilon > 0$ can be made arbitrarily small [FK12]. Their approach is also based on the Yao graph and quite similar to the one we used in Section 5.2. We think of $D(P)$ as a directed graph by directing each edge from the larger to the smaller disk. We pick an appropriate parameter $k = k(t)$ and a set $\mathcal{C}$ of k cones that partition the plane, as described in Section 5.2.1. For each site $q \in P$ and each cone $C \in \mathcal{C}$ we pick the closest site $p \in C \cap P$ that has an "outgoing" edge to q. More precisely, we pick the site $p \in C \cap P$ such that

(i) pq is an edge in $D(P)$,

(ii) $r_p \geqslant r_q$, and

(iii) among all sites in $C \cap P$ with properties (i) and (ii), p minimizes the distance to q.

Using standard arguments from, e.g., the Yao Graph algorithm, one can prove that this construction yields a t-spanner for $D(P)$, but as with transmission graphs, we do not know how to implement it efficiently. However, as we will see below, applying Algorithm 5.4 on the directed version of $D(P)$ will quickly give an approximately shortest edge inside each cone. The same analysis as in Section 5.2.1 will reveal that this is still sufficient to obtain a t-spanner for $D(P)$. This gives the first spanner construction for disk graphs in near-linear time that is independent of the radius ratio Ψ. In particular we prove the following theorem.

Theorem 5.18. *Let $D(P)$ be the disk graph for a set P of n points in the plane. For any $t > 1$, we can compute a t-spanner for $D(P)$ in $O(n 2^{\alpha(n)} \log^{10} n)$ expected time, where $\alpha(n)$ is the inverse Ackermann function. The expected storage is $O(n \log^3 n)$.*

5.3.1 Constructing the Spanner

We pick suitable constants $k = k(t)$ and $c = c(t)$, depending on our desired stretch factor $t > 1$. Let G be the directed graph on P obtained from $D(P)$ by directing each edge from the larger to the smaller disk. If two disks have the same radius, we include directed edges in both directions. Suppose we have a c-separated annulus decomposition $(\mathcal{G}, N, R_\tau)$ for G, as given in Definition 5.2. Note that Definition 5.2 is not restricted to transmission graphs, but it can be used with any directed geometric graph.[3]

We run Algorithm 5.1 with G and $(\mathcal{G}, N, R_\tau)$. The result is a directed graph $\overrightarrow{H} \subseteq G$. Let H be the graph obtained by substituting each directed edge in $\overrightarrow{H}$ by an undirected edge. Consider an edge pq in G. By Lemma 5.7 there is an edge rq in $\overrightarrow{H}$ such that $|pr| \leqslant |pq| - |rq|/t$, and the same holds for the corresponding undirected edges in $D(P)$ and in H. This allows us to show that H is a t-spanner for $D(P)$.

Lemma 5.19. *For any $t > 1$, there are constants $c = c(t)$ and $k = k(t)$ such that H is a t-spanner for the disk graph $D(P)$.*

Proof. The proof goes exactly as the proof of Lemma 5.8, i.e., by induction on the rank of the edge lengths in $D(P)$. The only difference is in the inductive step. Let pq be an edge in $D(P)$. Suppose $r_p \geqslant r_q$, such that pq is a directed edge in G. As argued above, by Lemma 5.7, $\overrightarrow{H}$ contains an edge rq with $|pr| \leqslant |pq| - |rq|/t$, and rq is an undirected edge in H. We argue that pr is an edge of $D(P)$, the rest then follows by induction. Since rq is an edge in $\overrightarrow{H}$, we have $r_r \geqslant r_q$. Hence, $|pr| \leqslant |pq| \leqslant r_p + r_q \leqslant r_p + r_r$ and pr is an edge in $D(P)$. Now the inductive argument is exactly the same as in the proof of Lemma 5.8. □

Remark: It is not necessarily true that $\overrightarrow{H}$ is a t-spanner for G. In the proof of Lemma 5.19, we argue that pr is an edge in $D(P)$. However, in G we might only have a directed edge from r to p (and not from p to r). This would not give a short p-q-path in H.

5.3.2 Efficient Construction

We now give an efficient implementation of the spanner construction from the previous section. We follow closely the approach from Section 5.2.3: we compute a compressed quadtree T, augment it with WSPD pairs as in Lemma 5.15, and traverse T in level-order using a dynamic nearest neighbor data structure to efficiently detect spanner edges between neighboring grid cells. However, since the edges of G are defined by the intersection of disks and not by sites contained in disks, this approach faces three problems that need to be addressed:

P1) To "cover" each edge pq of G according to Property (ii) in Definition 5.2, we need to adjust the interval that defines the sites R'_τ.

[3] And with slight changes, Definition 5.2 also works with undirected geometric graphs.

P2) When selecting spanner edges from a site r (edge selection in Algorithm 5.4), the *Euclidean* nearest neighbor is not sufficient to decide if r has outgoing edges (and to find them). We use the dynamic additively weighted nearest neighbor structure from Corollary 2.7 instead.

P3) Third, when selecting edges for r, we only want to consider sites with radius at most r_r, i.e., we only want to consider edges in G, not all edges in $D(P)$.

We now describe the detailed implementation of the algorithm, focusing on how to solve the three problems. As in Section 5.2.3, we scale our input such that the closest pair in P has distance $c-2$. We compute a compressed quadtree T, and we augment it with $O(n)$ cells from the c-WSPD, exactly as in Lemma 5.15. This gives a c-separated annulus decomposition as follows. The cells $\mathcal{G}$ are all cells assigned to nodes of T, i.e., $\mathcal{G} = \{\sigma_v \mid v \in T\}$. The neighborhood relation N consists of all pairs $(\sigma_v, \sigma_w) \in \mathcal{G} \times \mathcal{G}$ with $\operatorname{diam}(v) = \operatorname{diam}(w)$ and with $d(v,w) \in [c-2, 2c)\operatorname{diam}(v)$. It remains to define the sets R_{σ_v} for each node $v \in T$. For this, let pq be a *directed* edge in G and let $\sigma, \tau \in \mathcal{G}_i$ be the two cells with $p \in \tau$ and $q \in \sigma$. Since any edge rq in G has $r_r \geqslant r_q$, the minimum radius for sites in $\tau \cap P$ that could "cover" the edge pq as required by Property (ii) in Definition 5.2 is $d(\sigma,\tau)/2$. Thus, for $v \in T$ we use for R'_{σ_v} all sites in $\sigma_v \cap P$ whose radius is in $[c/2-1, 2(c+1))\operatorname{diam}(v)$, i.e., the lower bound is exactly half the minimum distance between neighboring grid cells. We set $R_{\sigma_v} = R'_{\sigma_v} \cup \{m_{\sigma_v}\}$, where m_{σ_v} is the site in $\sigma_v \cap P$ with the largest radius. We can now prove that this gives a c-separated annulus decomposition and thus problem P1 is solved.

Lemma 5.20. $(\mathcal{G}, N, R_\tau)$ *as defined above is a* c*-separated annulus decomposition for* G*. It can be computed in time* $O(n \log n)$ *using* $O(n)$ *space.*

Proof. First, we show that $(\mathcal{G}, N, R_\tau)$ is a c-separated annulus decomposition. Property (i) of Definition 5.2 holds by construction. For Property (ii), consider an edge pq of G. The proof of Lemma 5.15 shows that there is a pair $(\sigma,\tau) \in N$ with $p \in \tau$ and $q \in \sigma$. Since pq is an edge of G and by the definition of N, we have

$$r_p \geqslant |pq|/2 \geqslant d(\sigma,\tau)/2 \geqslant (c-2)\operatorname{diam}(\tau)/2.$$

If $r_p \leqslant 2(c+1)\operatorname{diam}(\tau)$ we have $p \in R_\tau$ by definition. Otherwise, the largest site m_τ in $\tau \cap P$ has radius greater than $2(c+1)\operatorname{diam}(\tau)$. Thus, the disk $D(m_\tau)$ contains the whole cell σ and hence $m_\tau q$ is an edge in G. This establishes Property (ii) of Definition 5.2.

The time and space bounds follow by the analysis done in the proof of Lemma 5.15 and the fact that for each site p there are now at most three integers i such that $r_p \in [c/2-1, 2(c+1))2^i$. □

To construct $\overrightarrow{H}$ efficiently, we run Algorithm 5.5, a modified version of Algorithm 5.4, for each cone $C \in \mathcal{C}$. We perform a level-order traversal of T, starting from the lowest level, and for each node $v \in T$, we perform the edge selection step, see Algorithm 5.5. Let $\tau \in N(\sigma_v)$ and let $r \in R_\tau$ be a site for which we search outgoing edges to *active* sites

(sites for which we have not found incoming edges yet) in $\sigma_v \cap P$. To find these edges, it is not sufficient anymore to maintain a *Euclidean* dynamic nearest neighbor structure S_v for the active sites in $\sigma_v \cap P$, as done in Algorithm 5.4: the nearest neighbor of r might have a small radius and might not form an edge with r, while another active site could do so, c.f. Figure 5.9. Thus, instead of the Euclidean metric we use a dynamic *additively weighted Euclidean* nearest neighbor structure as given by Corollary 2.7. We weight each site in P by its radius, and the distance from a point $x \in \mathbb{R}^2$ to a site $q \in P$ is defined as $\delta_q(x) = |qx| - r_q$. Then rq is an edge in $D(P)$ if and only if $\delta_q(r) \leqslant r_r$. In particular, by Lemma 2.8, if $r \in R_\tau$ has a neighbor in $D(P)$ that lies in $\sigma_v \cap P$, then also the additively weighted nearest neighbor of r in $\sigma_v \cap P$ is a neighbor of r in $D(P)$.

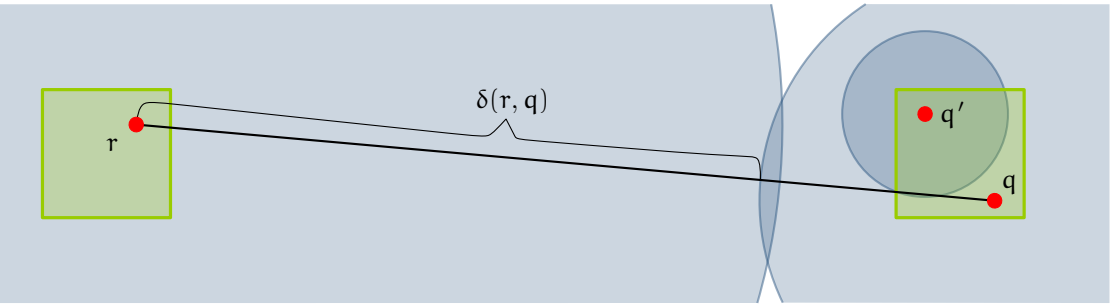


Figure 5.9: The Euclidean nearest neighbor q' of r does not form an edge, but the nearest neighbor according to δ does.

The last problem P3 occurs when querying a nearest neighbor structure S_v with a site $r \in R_\tau$ (while-loop at line 8). Since we work with the directed graph G, we want to consider only active sites q with radius smaller than the radius of r. To address this issue, we maintain the active sites of a node v, sorted by increasing radius, in a list L_v. We use L_v to insert into our nearest neighbor structure only the active sites q with $r_q \leqslant r_r$ when processing r. In the beginning, we build for each leaf $w \in T$ a list L_w containing the single point in σ_w. Furthermore, we create an *empty* nearest neighbor structure S_w according to Corollary 2.7. When processing an inner node $v \in T$, we obtain L_v and S_v by merging the structures of the children of v, see line 1 and line 2 in Algorithm 5.5.

We maintain the following invariant on S_v: when querying S_v with a site $r \in R_\tau$, all sites $q \in S_v$ have radius $r_q \leqslant r_r$. In the leaf nodes, the invariant holds by construction. To maintain it during the algorithm, we process the sites in R_τ by increasing radius. Before we perform the edge selection with a site $r \in R_\tau$, we insert into S_v all active sites with radius at most r_r using L_v (line 6 in Algorithm 5.5). Furthermore, after we finished processing the neighbor τ of σ_v, we remove from S_v all sites with radius larger than $(c/2 - 1)\,\mathrm{diam}(v)$ (line 10 in Algorithm 5.5). This ensures that S_v never contains sites whose radius is too large.

The edges in the graph $\overrightarrow{H}$ as computed by Algorithm 5.5 have the same properties as described in Section 5.3.1. Thus, if we substitute each edge pq in $\overrightarrow{H}$ by an undirected

```
   // preproccesing
1  let w be the child of v with the most active sites in L_w; for each child w' ≠ w,
   insert all actives sites of L_w' into L_w
2  let w be the child of v with the most sites in S_w; for each child w' ≠ w, insert all
   sites of S_w' into S_w
3  set L_v = L_w and S_v = S_w
4  foreach τ ∈ N(σ_v) contained in C^2_σv do
5      foreach r ∈ R_τ by increasing radius do
6          insert into S_v all sites q ∈ L_v with r_q ⩽ r_r not in S_v yet
           // edge selection
7          q ← NN(v, r) // query S_v with r
8          while |rq| ⩽ r_q + r_r and q ≠ ∅ do
9              add the edge rq to H⃗; delete q from S_v; q ← NN(v, r)
10     delete from S_v all sites q with r_q ⩾ (c/2 − 1) diam(v)
11 delete from S_v and L_v all sites q for which at least one edge rq was found
```

Algorithm 5.5: Selecting incoming edges for the sites of a node v and a cone C.

edge between p and q, the resulting graph H is a t-spanner for $D(P)$ by Lemma 5.19.

We now analyze the running time.

Lemma 5.21. *Algorithm 5.5 has a total expected running time of* $O(n2^{\alpha(n)}\log^{10} n)$ *and it requires* $O(n\log^3 n)$ *expected space.*

Proof. It takes $O(n\log n)$ time to compute the compressed quadtree and to find the neighboring pairs as in Lemma 5.15. Initializing the lists L_w and the nearest neighbor structures S_w at the leaves w of the quadtree T takes $O(n)$ time.

Let v be an internal node of T. At the preproccesing phase of v, we construct S_v from S_w where w is a child of v having the most sites in its nearest neighbor structure S_w. In the same way, we construct the list L_v. As argued in the proof of Lemma 5.16, each site is inserted $O(\log n)$ times into a nearest neighbor structure or into a sorted list. By Corollary 2.7, an insertion into a nearest neighbor structure takes $O(2^{\alpha(n)}\log^6 n)$ expected amortized time. Using, e.g., binary search trees, an insertion into a sorted list takes $O(\log n)$ worst-case time. Thus, the total expected time for all preprocessing steps is $O(n2^{\alpha(n)}\log^7 n)$.

Now, consider the edge selection step (lines 7–9). Recall that for a node v of T we denote by R'_{σ_v} the set of sites in $\sigma_v \cap P$ with radius in $[c/2-1, 2(c+1))\,\mathrm{diam}(v)$ and that we have $R_{\sigma_v} = R'_{\sigma_v} \cup m_{\sigma_v}$. As in the proof of Lemma 5.16, we charge the initial nearest neighbor queries (line 7) to the nodes of T and to sites in the sets R'_{σ_v}. The queries and deletions done in line 9 are charged to the edges of $\overrightarrow{H}$ we create. Since T has $O(n)$ nodes, and since each site is assigned to R'_{σ_v} for at most three nodes v of T, and since $\overrightarrow{H}$ has $O(n)$ edges, we perform $O(n)$ queries and deletions. By Corollary 2.7, a query needs $O(\log^2 n)$ worst-case time and a deletion needs $O(2^{\alpha(n)}\log^{10} n)$ expected amortized time. Thus,

the total expected time needed for the edge selection is $O(n2^{\alpha(n)} \log^{10} n)$.

It remains to count the insertions and deletions that are required to maintain the invariant on S_v (line 6 and line 10). Since at the end of the algorithm every site is contained in at most one nearest neighbor structure, the number of these insertions and deletions differ by at most n. Thus, it suffices to count only the deletions done in line 10. Let q be a site that is deleted from S_v while processing σ_v for some node v of T. We claim that $q \in R'_{\sigma_v}$ and hence we can charge the deletions to the sites in R'_{σ_v}. Since q was deleted, $r_q \geqslant (c/2 - 1)\operatorname{diam}(v)$. Furthermore, q was inserted by a site $r \in R_\tau = R'_\tau \cup \{m_\tau\}$. If $r \in R'_\tau$, then

$$r_q \leqslant r_r < 2(c+1)\operatorname{diam}(\tau) \leqslant 2(c+1)\operatorname{diam}(v),$$

and thus $q \in R'_{\sigma_v}$. Otherwise, $r = m_\tau$ and $m_\tau \notin R'_\tau$, i.e., $r_{m_\tau} \geqslant 2(c+1)\operatorname{diam}(v)$. Then, since $d(\tau, \sigma_v) \leqslant 2c \operatorname{diam}(v)$, the disk $D(m_\tau)$ contains σ_v. Hence, $m_\tau q$ is an edge in G and Algorithm 5.5 finds an incoming edge for q in the edge selection step. In particular, q will be deleted from S_v in line 9, contradicting the assumption that q is deleted at line 10. Hence, the number of insertions and deletions needed to maintain the invariant on S_v is proportional to R'_{σ_v}. As argued above, the total size of the sets R'_{σ_v} is $O(n)$. Thus, by Corollary 2.7 we need $O(n2^{\alpha(n)} \log^{10} n)$ expected time for this step in total.

The total size of the compressed quadtree and of the associated sorted lists is $O(n)$. Furthermore, a dynamic nearest neighbor structure with m elements needs $O(m \log^3 m)$ expected space by Corollary 2.7. Thus, since at any time each site lies in at most one dynamic nearest neighbor structure, the expected space requirement is $O(n \log^3 n)$. □

5.4 Applications

We present two applications of our spanner construction. We show how to use it to compute a breadth first search (BFS) tree from a particular vertex, and we show how to use it to extend a given reachability data structure for additional queries specific to transmission graphs. Both applications rely on the *power diagram*, which is a weighted version of the Voronoi Diagram that represents the union of a set of disks (in our case these are the disks $D(p)$ for $p \in P$) as a planar subdivision. More specifically, the *power distance* between a point q, and a disk with center p and radius r, is $(d(p, q))^2 - r^2$. The power diagram partitions the plane into n regions, such that all points in a specific region have the same closest disk in power distance. The power diagram of a set of n disks is of size $O(n)$ and can be constructed in $O(n \log n)$ time. If augmented with a point location structure then we can locate the disk D that minimizes the power distance from a query point q in $O(\log n)$ time. In particular we can determine in $O(\log n)$ time if q is in the union of the disks by checking if $q \in D$ [IIM85, Kir83].

5.4.1 From Spanners to BFS Trees

We show how to compute the BFS tree in a transmission graph G from a given root $s \in P$ using the spanner constructions from the previous section. We adapt a technique

that Cabello and Jejčič developed for unit-disk graphs [CJ15]. Denote by $d_h(s,p)$ the BFS distance (also known as hop distance) from s to p in G. Let $W_i \subseteq P$ be the sites $p \in P$ with $d_h(s,p) = i$. Cabello and Jejčič used the Delaunay triangulation (DT) to efficiently identify W_{i+1}, given $W_0, \ldots, W_i$. We use our t-spanner in a similar manner for transmission graphs.

Lemma 5.22. *Let t be small enough, and let H be the t-spanner for a transmission graph G as in Theorem 5.1, 5.12 or 5.17. Let $v \in W_{i+1}$, for some $i \geqslant 1$. Then, there is a site $u \in W_i$ and a path $u = q_\ell, \ldots, q_1 = v$ in H with $d_h(s, q_j) = i + 1$ for $j = 1, \ldots, \ell$.*

Proof. We focus on the spanner from Theorem 5.12, since it has the most complicated structure. The proof for the other constructions is similar and simpler.

Since $v \in W_{i+1}$, there is a site $w \in W_i$ with $v \in D(w)$. If H contains the edge wv, the claim follows by setting $u = q_2 = w$ and $q_1 = v$. Otherwise, we construct the path backwards from v (see Figure 5.10). Suppose we have already constructed a sequence $v = q_1, q_2, \ldots, q_k$ of sites in P such that (i) for $j = 1, \ldots, k-1$, $q_{j+1}q_j$ is an edge of H; (ii) for $j = 1, \ldots k$, we have $q_j \in D(w)$ and $d_h(s, q_j) = i+1$; and (iii) for $j = 1, \ldots, k-1$, $|wq_{j+1}| < |wq_j|$. We begin with the sequence $q_1 = v$ satisfying the invariant.

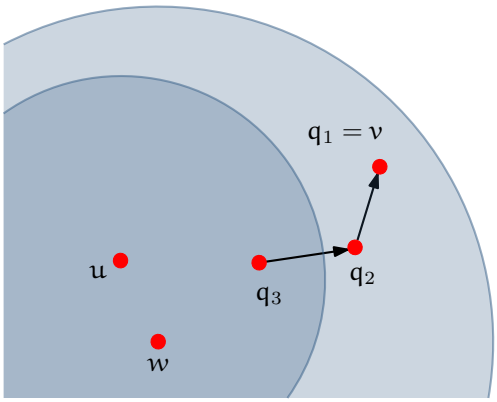


Figure 5.10: The partial path constructed backwards from v. Setting $q_4 = u$ will complete it.

Let c be the constant from the spanner construction of Section 5.2.2. We scale everything such that the smallest radius in P is c. Suppose that we have $q_1, \ldots, q_k$ and that wq_k is not an edge of H (otherwise we could finish by setting $u = w$). Let $\sigma, \tau \in \mathcal{G}_0$ be the cells such that $w \in \tau$ and $q_k \in \sigma$. We distinguish two cases, depending on $d(\sigma, \tau)$, and we either show how to find u to complete the path from u to v or how to choose q_{k+1}.

Case 1: $d(\sigma, \tau) < c - 2$. Let $Q = (P \cap \sigma) \cup (P \cap \tau)$. We have that $w, q_k \in Q$. The algorithm of Section 5.2.2 constructs a Euclidean spanner for Q and adds its edges to H. In particular, there is a directed path π from w to q_k that uses only sites of Q. By construction, the pairwise distances between the sites of Q are all at most c. Thus, for

each $p \in Q$ we have $p \in D(w)$ and $q_k \in D(p)$, and therefore $i \leqslant d_h(s,p) \leqslant i+1$. We set u to be the last site of π with $d_h(s,u) = i$. To obtain the desired path from u to v we take the subpath of π starting at u and concatenate it to the the partial path $q_k, \ldots, q_1 = v$.

Case 2: $d(\sigma,\tau) \geqslant c-2$. Since wq_k is not an edge of H, by Lemma 5.7 there exists an edge rq_k in H with $|wr| < |wq_k|$. We set $q_{k+1} = r$. Since $q_k \in D(w)$, we have $q_{k+1} \in D(w)$ and $i \leqslant d_h(s,q_{k+1}) \leqslant i+1$. If $d_h(s,q_k) = i$, we set $u = q_{k+1}$ and are done. Otherwise, q_{k+1} satisfies properties (i)–(iii) and we continue to extend the path.

Since the distance to w decreases in each step and since P is finite, this process eventually stops and the lemma follows. □

```
1  W_0 ← {s}; dist[s] = 0; π[s] = s; i = 0
2  for p ∈ P \ {s}, dist[p] = ∞ and π[p] = NIL
3  while W_i ≠ ∅ do
4  |  compute power diagram with point location structure PD_i of W_i
5  |  queue Q ← W_i ; W_{i+1} ← ∅
6  |  while Q ≠ ∅ do
7  |  |  p ← dequeue(Q)
8  |  |  foreach edge pq of H do
9  |  |  |  u ← PD_i(q) // query PD_i with q, D(u) minimizes the power
   |  |  |     distance from q
10 |  |  |  if q ∈ D(u) and dist[q] = ∞ then
11 |  |  |  |  enqueue(Q, q); dist[q] = i + 1; π[q] = u; add q to W_{i+1}
12 |  i ← i + 1
```

Algorithm 5.6: Computing the BFS tree for G with root s using the spanner H.

The BFS tree for s is computed iteratively; see Algorithm 5.6 for pseudocode. Initially, we set $W_0 = \{s\}$. Now assume we have computed $W_0, \ldots, W_i$. By Lemma 5.22, all sites in W_{i+1} can be reached from W_i in the subgraph of H induced by $W_i \cup W_{i+1}$. Thus, we can compute W_{i+1} by running a BFS search in H from the points of W_i using a queue Q. Every time we encounter a new vertex q, we check if it lies in a disk around a site of W_i, and is not yet in the BFS tree for s. If so, we add q to W_{i+1} and to Q. Otherwise, we discard q. To test whether q lies in a disk of W_i, we compute a power diagram for W_i in time $O(|W_i| \log |W_i|)$ and query it with q.

A site p at level i is traversed by at most two BFS searches in H. In the first search we discover that p is in W_i, and in the second search p is a starting point — this is the search to discover W_{i+1}. It follows that an edge pq of H is considered twice by Algorithm 5.6. Each time we consider the edge pq we spend $O(\log n)$ time for querying a power diagram with q. Since H is sparse, the total time required is $O(n \log n)$. This establishes the following theorem.

Theorem 5.23. *Let* G *be the transmission graph of a set* $P \subset \mathbb{R}^2$ *of* n *points. Given a spanner* H *for* G *as in Theorem 5.1, Theorem 5.12, or Theorem 5.17, we can compute in* $O(n \log n)$ *additional time a BFS tree rooted at any given site* $s \in P$*.*

Unfortunately, we cannot derive a similar result for disk graphs in the same way. In the proof of Lemma 5.22 in Case 2, we used the fact that if wq_k is not an edge in H, then there is an edge rq_k with $|wr| < |wq_k|$. For disk graphs, our spanner construction can guarantee such an r only when $r_w \geqslant r_{q_k}$, but not in the other case. The reason for this is quite similar to the reason why the graph $\overrightarrow{H}$ from the previous Section Section is not a spanner for the directed version of a disk graph $D(P)$ (cf. Section 5.3.1).

However, using the dynamic additively weighted Euclidean nearest neighbor structure from Corollary 2.7, there is a more direct way to compute a BFS trees for a disk graph $D(P)$. This strategy was already an observation of Roditty and Segal in the context of unit disk graphs [RS11]. Let a root site $s \in P$ be given. We initialize a dynamic additively weighted nearest neighbor structure, where the weights correspond to the radii of the sites. We insert all sites from $P \setminus \{s\}$. At each point of the BFS-algorithm, the dynamic nearest neighbor data structure contains all sites that are not yet part of the BFS-tree. To find all new neighbors of a site p of the partial BFS-tree T, we repeatedly find and delete a nearest neighbor of p in $P \setminus T$, until the next nearest neighbor is not adjacent to p in $D(P)$. By Lemma 2.8, this will give us all new neighbors of p in T. A successful query and deletion yields a new edge of the BFS-tree. Thus, these queries and deletions can be charged to the edges of T. The last unsuccessful query can be charged to p itself. Hence, the total number of insertions, deletions, and queries is $O(n)$. By Corollary 2.7, we get the following theorem.

Theorem 5.24. *Let* $D(P)$ *be the disk graph of a set* $P \subset \mathbb{R}^2$ *of* n *points. We can compute in* $O(n2^{\alpha(n)} \log^{10} n)$ *expected time a BFS tree rooted at any given site* $s \in P$*, where* $\alpha(n)$ *is the inverse Ackermann function.*

5.4.2 Geometric Reachability Oracles

Let G be a directed graph. If there is a directed path from a vertex s to a vertex t in G, we say s *can reach* t (in G). A *reachability oracle* for a graph G is a data structure that can answer efficiently for any given pair s, t of vertices of G whether s can reach t. Reachability oracles have been studied extensively over the last decades (see, e.g., [HRT14, Tho04] and the references therein) and we will do so in Chapter 6.

When G is a transmission graph we are interested in a more general type of reachability query where the target t is not necessarily a vertex of G, but an arbitrary point in the plane. We say that a site s can reach a point $t \in \mathbb{R}^2$ if there is a site q in G such that $t \in D(q)$ and such that s can reach q in G. A data structure that supports this type of queries is called a *geometric reachability oracle*. We can use our spanner construction from Theorem 5.12 to extend any reachability oracle for a transmission graph to a geometric reachability oracle with a small overhead in space and query time. More precisely, we prove the following theorem.

Theorem 5.25. *Let G be the transmission graph for a set P of n points in the plane with radius ratio Ψ. Given a reachability oracle for G that requires $S(n)$ space and has query time $Q(n)$, we can obtain in $O(n \log n \log \Psi)$ time a geometric reachability oracle that requires $S(n) + O(n \log \Psi)$ space and can answer a query in $O(Q(n) + \log n \log \Psi)$ time.*

Given a query s, t with a target $t \in \mathbb{R}^2$, our strategy is to find a small subset $Q \subseteq P$ such that for each $q \in Q$, $t \in D(q)$, and such that Q "covers the space around t" in the following sense. For any disk $D(p)$ such that $t \in D(p)$ there is a site $q \in Q$ with $q \in D(p)$. In particular the edge pq is in G.

Such a set Q satisfies that s can reach t if and only if s can reach some site $q \in Q$. Once we have computed Q we decide whether s can reach t by querying the given reachability oracle with s, q for all $q \in Q$. The answer is positive if and only if it is positive for at least one site $q \in Q$.

In what follows, we construct a data structure of size $O(n \log \Psi)$ that allows to find such a set Q of size $O(1)$ in $O(\log n \log \Psi)$ time. Theorem 5.25 is then immediate.

The Data Structure. We compute a 2-spanner H for G as in Theorem 5.12. Let k (the number of cones) and c (the separation parameter) be the two constants used by the construction of H, and recall that we scaled P such that the smallest radius of a site in P is c. Let T be the quadforest used by the construction of H. The trees in T have depth $O(\log \Psi)$ and each node $\nu \in T$ corresponds to a grid cell σ_ν from some grid $\mathcal{G}_i$, $i \geqslant 0$. Our data structure is obtained by augmenting each node $\nu \in T$ by a power diagram PD_{σ_ν} for the sites in $\sigma_\nu \cap P$, together with a point location data structure. This requires $O(|\sigma_\nu \cap P|)$ space and $O(|\sigma_\nu \cap P| \log |\sigma_\nu \cap P|)$ time for each node ν [IIM85, Kir83]. Since any site of P is in $O(\log \Psi)$ cells of T, we need $O(n \log \Psi)$ space and $O(n \log n \log \Psi)$ time in total.

```
1 L ← depth of T
2 for i = 0, ..., L do
3 |   σ ← cell of G_i with t ∈ σ
4 |   foreach τ ∈ N(σ) contained in C²_σ do
5 |   |   q ← PD_τ(t) // query PD_τ with t
6 |   |   if t ∈ D(q), add q to Q
7 |   Stop if at least one q was added to Q
```

Algorithm 5.7: Query Algorithm for a cone C and a point t.

Performing a Query. Let a query point $t \in \mathbb{R}^2$ be given. Let σ be the cell in $\mathcal{G}_0$ that contains t. To find Q, we first traverse all non-empty cells $\tau \in \mathcal{G}_0$ with $d(\sigma, \tau) \leqslant c - 2$. From each such cell τ, if there exists a site $q \in \tau \cap P$ such that $t \in D(q)$ then we add one, arbitrary, such site to Q. To determine if such a site exists, and to find one if it exists, we query PD_τ with t. Second, we go through all cones $C \in \mathcal{C}$, and we run Algorithm 5.7 with C and t to find the remaining sites for Q. Algorithm 5.7 is similar

to Algorithms 5.1 and 5.2, and computes the incoming edges of t if it would have been inserted into the spanner. We go through the grids at all levels of T. For each level we consider the cell σ that contains t and for each cell $\tau \in N(\sigma)$ that is contained in C^2_σ we select a site with an edge to t if there is one. Lemma 5.7 holds for the incoming edges of t and using this fact, we can prove that our data structure has the desired properties.

Lemma 5.26. *Let P be a set of n points in the plane with radius ratio Ψ. We can construct in $O(n \log n \log \Psi)$ time a data structure that finds for any given query point $t \in \mathbb{R}^2$ a set $Q \subseteq P$ such that $|Q| = O(1)$ and for any site $p \in P$, if $t \in D(p)$ we have that $D(p) \cap Q \neq \emptyset$. The query time is $O(\log n \log \Psi)$ and the space requirement is $O(n \log \Psi)$.*

Proof. The construction time and the space requirement are immediate. For the query time recall that T has depth $O(\log \Psi)$ and by Lemma 5.3, at each level we make $O(c^2)$ queries to power diagrams. It follows that it takes $O(\log n \log \Psi)$ time to compute Q.

By construction, Q has size $O(1)$. Indeed, at the first step, we add at most one site for every cell of distance at most $c-2$ from σ, and there are $O(c^2)$ such cells. In the second step, for each cone, we only add sites from $O(c^2)$ cells at exactly one level of T.

Now let $p \in P$ be a site with $t \in D(p)$. It remains to show that $D(p) \cap Q \neq \emptyset$. If $p \in Q$, we are done. If not, we let $\sigma, \tau \in \mathcal{G}_0$ be the cells with $t \in \sigma$ and $p \in \tau$. If $d(\sigma, \tau) \leqslant c-2$ then there must be a site $q \in \tau \cap Q$. Since $\operatorname{diam}(\tau) = 1$ and $r_p \geqslant c$, we have $q \in D(p)$. If $d(\sigma, \tau) > c-2$ then pt is an edge in the transmission graph of $P \cup \{t\}$ that is not selected by Algorithm 5.7. Applying Lemma 5.7 to pt guarantees that there is an edge qt with $q \in Q$ and $|pq| < |pt|$. Since $t \in D(p)$, we also have $q \in D(p)$. This finishes the proof. □

5.5 Conclusion

We have described the first construction of spanners for transmission graphs that runs in near-linear time. Our techniques are quite general, and they allowed us to extend our results to disk graphs. This significantly improves the bounds of Fürer and Kasiviswanathan [FK12].

To demonstrate the usefulness of the spanner construction, we described two applications. One of them will be of great benefit in Chapter 6 when we consider reachability oracles for transmission graphs. Then, we describe several constructions for reachability oracles, that will provide many opportunities to apply Theorem 5.25. Also, in this context our spanner construction plays a crucial role in obtaining fast preprocessing algorithms.

Chapter 6

Reachability Oracles for Transmission Graphs

Representing the connectivity of a graph in a space efficient, succinct manner, while supporting fast queries, is one of the most fundamental data structuring questions on graphs. For an undirected graph, it suffices to compute the connected components (cf. Chapter 3) and to store with each vertex a label for its respective component. This leads to a linear-space data structure that can decide in constant time if any two given vertices are connected. For undirected graphs, however, connectivity is not a symmetric relation any more, and the problem turns out to be much more challenging. Thus, if G is a directed graph, we say that a vertex s can *reach* a vertex t if there is a directed path in G from s to t. Our goal is to construct a *reachability oracle*, a space efficient data structure that answers *reachability queries*, i.e., that determines for any pair of query vertices s and t whether s can reach t. The quality of a reachability oracle for a graph with n vertices is measured by three parameters: the *space* $S(n)$, the *query time* $Q(n)$ and the *preprocessing time*. The simplest solution stores for each pair of vertices whether they can reach each other, leading to a reachability oracle with $\Theta(n^2)$ space and constant query time. For sparse graphs with $O(n)$ edges, storing just the graph and performing a breadth first search for a query yields an $O(n)$ space oracle with $O(n)$ query time. Interestingly, other than that, we are not aware of any better solutions for general directed graphs, even sparse ones. Thus, any result that simultaneously achieves subquadratic space and sublinear query time would be of great interest. A lower bound by Pătraşcu [Păt11] shows that we cannot hope for $o(\log n)$ query time with linear space, but it does not rule out constant time queries with slightly superlinear space. In the absence of progress towards non-trivial reachability oracles or better lower bounds, solutions for special cases become important. For directed planar graphs, after a long line of research [ACC+96, CX00, Dji96, Fed87, Tho04], Holm, Rotenberg and Thorup presented a reachability with optimal parameters [HRT14]. This result, as well as most

other previous results, is actually not merely a reachability oracle but it can also return the approximate shortest path distance between the query vertices.

Naturally, one would like to discover further graph classes, besides planer graphs, for which efficient reachability oracles exist. A good candidate for such a class are *transmission graphs*. Even though they are in general highly non-planar, they share many geometric properties with planar graphs. These properties allow us to translate several approaches known from planar graphs to the case of transmission graphs. Recall that in the geometric context of transmission graphs, it is natural to consider a more general type of query where the target point is an arbitrary point in the plane rather than a vertex of the graph. In this case, a vertex $s \in P$ can reach a *point* $q \in \mathbb{R}^2$ if there is a *vertex* $t \in P$ such that s reaches t and such that $|tq| \leqslant r_t$. We call such queries *geometric reachability queries* and oracles for them *geometric reachability oracles*. To avoid ambiguities, we sometimes use the term *standard* reachability query/oracle when referring to the case where the query consists of two vertices.

6.1 Our Results

It turns out that the radius ratio Ψ, the ratio of the largest and the smallest transmission radius in P, is an important parameter for our constructions. We present three different reachability oracles, each of them will perform best for a certain range of Ψ. If Ψ is less than $\sqrt{3}$, we can turn the transmission graph into a planar graph in $O(n \log n)$ time, while preserving the reachability structure and keeping the number of vertices linear in n. As mentioned above, for planar graphs there is a linear time construction of a reachability oracle with linear space and constant query time [HRT14]. This construction yields a standard reachability oracle. However, as argued in Section 5.4.2, any standard reachability oracle can be transformed into a geometric one by paying an *additive* overhead of $O(\log n \log \Psi)$ in the query time and of $O(n \log \Psi)$ in the space (see Theorem 5.25). Our final construction needs $O(n)$ space and has query time $O(\log n)$ for geometric queries and $O(1)$ for standard queries. It can be found in Section 6.2.

When $\Psi \geqslant \sqrt{3}$, we do not know how to find a planar graph representing the reachability of G. Fortunately, we can use a theorem by Alber and Fiala that allows us to find a small and balanced separator with respect to the area of the union of the disks [AF04]. This leads to a standard reachability oracle with query time $O(\Psi^3\sqrt{n})$ and space and preprocessing time $O(\Psi^3 n^{3/2})$, see Section 6.3. When Ψ is even larger, we can use random sampling combined with a quadtree of logarithmic depth to obtain a standard reachability oracle with query time $O(n^{2/3} \log^{1/3} \Psi \log^{2/3} n)$, space $O(n^{5/3} \log^{1/3} \Psi \log^{2/3} n)$, and preprocessing time $O(n^{5/3}(\log \Psi + \log n) \log^{1/3} \Psi \log^{2/3} n)$. Refer to Section 6.4. Again, we can transform both oracles into geometric reachability oracles using Theorem 5.25 from Section 5.4.2. Since the overhead is additive, this time the transformation does not affect the performance bounds.

6.2 Ψ is less than $\sqrt{3}$

Let $P \subset \mathbb{R}^2$ be a set of sites such that each site in P has a radius in the interval $[1, \sqrt{3})$, i.e., the radius ratio of P is $\Psi < 3$. We show that in this restricted case we can make the transmission graph of P planar by first removing unnecessary edges and then resolving edge crossings by adding $O(n)$ additional vertices. This will not change the reachability between the original vertices. The existence of efficient reachability oracles then follows from known results for directed planar graphs. The main goal is to prove the following lemma.

Lemma 6.1. *Let G be the transmission graph for a planar n-point set P with $\Psi < \sqrt{3}$. In $O(n \log n)$ time, we can find a plane graph $\widetilde{H} = (V, E)$ such that*

(i) $|V| = O(n)$ and $|E| = O(n)$;

(ii) $P \subseteq V$; and

(iii) for any $p, q \in P$, p can reach q in G if and only if p can reach q in $\widetilde{H}$.

Given Lemma 6.1, we can obtain our reachability oracle from known results.

Theorem 6.2. *Let G be the transmission graph for a two-dimensional set P of n points, and suppose the radius ratio Ψ is less than $\sqrt{3}$. Then, we can construct in $O(n \log n)$ time a standard reachability oracle for G with $S(n) = O(n)$ and $Q(n) = O(1)$ or a geometric reachability oracle for G with $S(n) = O(n)$ and $Q(n) = O(\log n)$.*

Proof. We apply Lemma 6.1 and construct the distance oracle of Holm, Rotenberg, and Thorup for the resulting graph [HRT14]. This distance oracle can be constructed in linear time, it needs linear space, and it has constant query time. The result for the geometric reachability oracle follows from Theorem 5.25. □

We prove Lemma 6.1 in three steps. First, we show in Section 6.2.1 how to make G sparse without changing its reachability. Then, we show in Section 6.2.2 how to turn G into a planar graph. Finally, we argue in Section 6.2.3 that we can combine these two operations to get the desired graph $\widetilde{H}$ from Lemma 6.1 that is sparse and planar.

6.2.1 Obtaining a Sparse Graph

We construct a subgraph $H \subseteq G$ with the same reachability as G but with $O(n)$ edges and $O(n)$ edge crossings. The bound on the number of crossings will allow us to obtain a planar graph with only a linear number of edges later on. Consider the grid $\mathcal{G} = \mathcal{G}_0$ whose cells have diameter 1, and let $\sigma \in \mathcal{G}$ be a grid cell. We say that an edge of G *lies in* σ if both endpoints are contained in σ. The *neighborhood* $N(\sigma)$ of σ consists of the 7×7 block of cells in $\mathcal{G}_0$ with σ at the center. Two grid cells are *neighboring* if they lie in each other's neighborhood. Since a cell in $\mathcal{G}$ has side length $\sqrt{2}/2$ and since every edge in G has length less than $\sqrt{3}$, the endpoints of every edge in G must lie in neighboring grid cells.

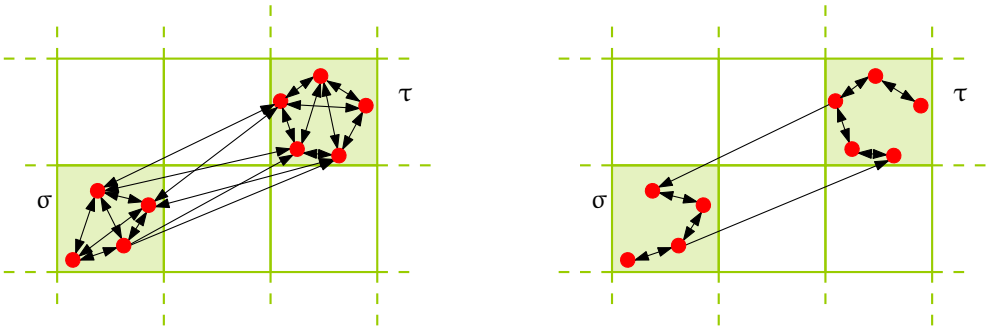


Figure 6.1: The vertices and edges of two neighboring cells of G (left) and of H (right)

We now construct the subgraph $H \subseteq G$. It has vertex set P, and we pick the edges as follows (see also Figure 6.1): for each non-empty cell $\sigma \in \mathcal{G}$, we set $P_\sigma = P \cap \sigma$, and we compute the Euclidean minimum spanning tree (EMST) T_σ of P_σ. For each edge pq of T_σ, we add the directed edges pq and qp to H. Then, for every cell $\tau \in N(\sigma)$, we check if there are any edges from σ to τ in G. If so, we add an arbitrary such edge to H. The following lemma states properties of H.

Lemma 6.3. *The graph* H

(i) has the same reachability as G*;*

(ii) has $O(n)$ *edges;*

(iii) can be constructed in $O(n \log n)$ *time; and*

(iv) the straight line embedding of H *in the plane with vertex set* P *has* $O(n)$ *edge crossings.*

Proof. (i): All edges of H are also edges of G: inside a non-empty cell σ, P_σ induces a clique in G, and the edges of H between different cells are edges in G by construction. It follows that H does not increase the reachability. Now let pq be an edge in G. We show that there is a path from p to q in H: if pq lies in a cell σ of $\mathcal{G}_0$, we take the path along the EMST T_σ. If pq goes from a cell σ to another cell τ, then there is an edge uv from σ to τ in H, and we take the path in T_σ from p to u, then the edge uv, and finally the path in T_τ from v to q.

(ii): For a nonempty cell σ, we create $|P_\sigma| - 1$ edges inside σ. Furthermore, since $|N(\sigma)|$ is constant, there are at most $O(1)$ edges between σ and other cells. Thus, H has $O(n)$ edges.

(iii): Since we assumed that our model of computation supports finding the cell for a vertex $p \in P$ in constant time, we can easily compute the sets P_σ, $\sigma \in \mathcal{G}$ nonempty, in time $O(n \log n)$. Computing the EMST T_σ for a cell σ needs time $O(|P_\sigma| \log |P_\sigma|)$, for a total of $O(n \log n)$. To find the edges between neighboring cells, we build a Voronoi diagram together with a point location structure for each set P_σ. Again, this takes

$O(n \log n)$ total time [BCvKO08]. Let σ and τ be two neighboring cells. For each vertex in P_σ, we locate the nearest neighbor in P_τ using the Voronoi diagram. If there is a vertex $p \in P_\sigma$ whose nearest neighbor $q \in P_\tau$ lies in the disk $D(p)$, we add the edge pq to H, and we proceed to the next pair of neighboring cells. Since $|N(\sigma)|$ is constant, a vertex participates in $O(1)$ point locations, of $O(\log n)$ time each. The total running time is $O(n \log n)$.

(iv): We distinguish two kinds of crossings. First, if at least one edge of a crossing lies inside a grid cell σ, then the other edge must go between different cells of $N(\sigma)$, because T_σ is crossing-free. There are $O(1)$ such edges, so there are $O(n)$ crossings of the first kind, since H has $O(n)$ edges.

In the second kind of crossing, both edges go between different grid cells. As Ψ is constant, each edge of H can participate in at most $O(1)$ such crossings. Since there are $O(n)$ edges in H, the total number of crossings is $O(n)$. □

6.2.2 Making G Planar

We now describe how to turn G into a planar graph. The general strategy is similar to the planarization argument for unit disk graphs from Chapter 3. Suppose an edge pq and an edge uv of G cross at a point x. To eliminate the crossing, we add x as a new vertex to the graph, and we replace pq and uv by the four new edges px, xq, ux and xv. Furthermore, if qp is an edge of G, we replace it by the two edges qx, xp, and if vu is an edge of G, we replace it by the two edges vx, xu. See Figure 6.2. We say that this *resolves* the crossing between p, q, u and v. Let $\widetilde{G}$ be the graph obtained by iteratively resolving all crossings in G.

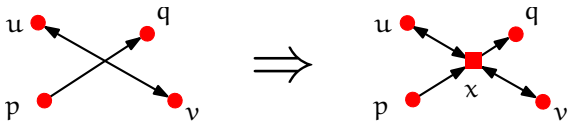


Figure 6.2: Resolving a crossing. Since the edge vu exists, we also add vx and xu as edges.

First, we want to show that resolving crossings keeps the *local* reachability between the four vertices of the crossing edges. Intuitively speaking, the restriction $\Psi < 3$ forces the vertices to be close together. This guarantees the existence of additional edges between p, q, u, v in G, and these edges cover the new paths introduced by resolving the crossing.

To formally prove this, we first need a geometric observation. Recall that for a vertex $p \in P$, we denote by $D(p, r)$ and by $C(p, r)$ the disk and the circle with center p and with radius r.

Lemma 6.4. *Let p, q be two points in $\mathbb{R}^2$ with $|pq| = \sqrt{3}$.*

(i) Let $a \in C(p, 1) \cap C(q, 1)$. Then, for any $r \in [1, \sqrt{3})$, if $b \in C(p, r) \cap C(q, r)$ lies on the other side of the line through p and q from a, then $|ab| \geqslant r$.

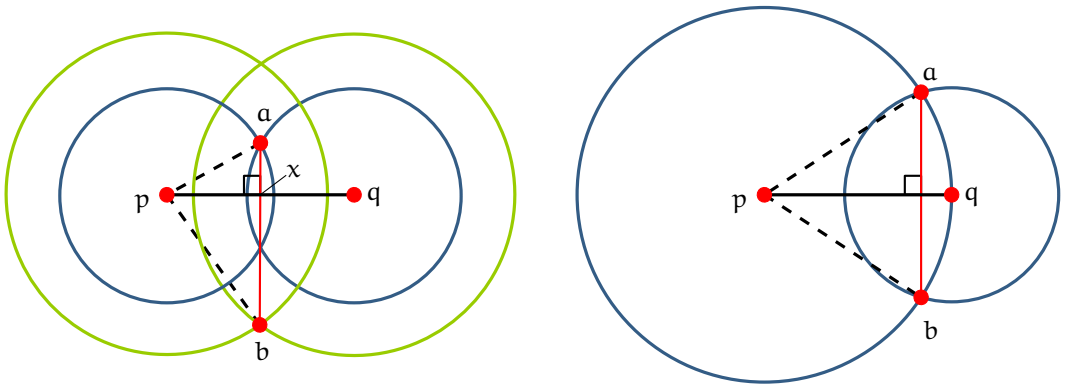
(ii) Let $\{a, b\} = C(p, \sqrt{3}) \cap C(q, 1)$. Then, $|ab| > \sqrt{3}$.

Proof. (i): Let x be the intersection point of the line segments $\overline{pq}$ and $\overline{ab}$. Then $|ab| = |ax| + |xb|$. Using that $|pa| = 1$ and $|px| = \sqrt{3}/2$, the Pythagorean Theorem gives $|xa| = 1/2$. Similarly, we can compute $|xb|$ as a function of r: with $|pb| = r$ we get $|xb| = \sqrt{r^2 - 3/4}$. We want to show that

$$\begin{aligned} & r \leqslant |ab| = 1/2 + \sqrt{r^2 - 3/4} \\ \Leftrightarrow & r^2 \leqslant 1/4 + \sqrt{r^2 - 3/4} + r^2 - 3/4 \\ \Leftrightarrow & 1/2 \leqslant \sqrt{r^2 - 3/4} \\ \Leftrightarrow & 1 \leqslant r^2, \end{aligned}$$

which holds since $r \in [1, \sqrt{3})$.

(ii): Use the Pythagorean Theorem with the right angles marked in Figure 6.3(ii). □



(i) (ii)

Figure 6.3: The cases (i) and (ii) of Lemma 6.4.

Lemma 6.5. *Suppose that* pq *and* uv *are edges in* G *that cross. Let* $G' \subseteq G$ *be the transmission graph induced by* p, q, u *and* v*. If* $\Psi < \sqrt{3}$*, then* p *reaches* v *in* G' *and* u *reaches* q *in* G'*.*

Proof. We may assume that $r_p \geqslant r_u$. Furthermore, we set $r_q = r_v = 1$. This does not add any new edges and thus reachability in the new graph implies reachability in G'. We show that if either u does not reach q (case 1) or p does not reach v (case 2), then $|uv| > r_u$. Hence uv cannot be an edge of G' despite our assumption.

Case 1: u does not reach q. Then we have $p \notin D(u)$, $q \notin D(u)$, $p \notin D(v)$ and $q \notin D(v)$. Equivalently this gives $u \notin D(p, r_u) \cup D(q, r_u)$ and $v \notin D(p, 1) \cup D(q, 1)$. Thus, the positions of u and v that minimize $|uv|$ are the intersections $u \in C(p, r_u) \cap C(q, r_u)$ and $v \in C(p, 1) \cap C(q, 1)$ on different sides of the line through p and q. To further minimize $|uv|$, observe that $|uv|$ depends on the distance of p and q and that $|uv|$ strictly decreases as $|pq|$ grows, i.e., as $|pq|$ approaches $\sqrt{3}$. For the limit case $|pq| = \sqrt{3}$, we are

in the situation of Lemma 6.4(i) with $a = u$ and $b = v$ and thus we would get $|uv| \geqslant r_u$. But since $\Psi < \sqrt{3}$, we must have $|pq| < \sqrt{3}$ and by strict monotonicity, it follows that $|uv| > r_u$, as desired.

Case 2: p does not reach v. Then we have $u \notin D(p)$, $v \notin D(p)$, $u \notin D(q)$ and $v \notin D(q)$. We scale everything, such that $r_p = \sqrt{3}$, and we reduce r_v, r_q once again to 1. Now, the positions of u and v minimizing $|uv|$ are $\{u, v\} = C(p, \sqrt{3}) \cap C(q, 1)$. As above, further minimizing $|uv|$ gives $|pq| = \sqrt{3}$. By Lemma 6.4(ii), we have $|uv| > \sqrt{3}$ and thus uv cannot be an edge of G' (note that even after scaling we have $r_u \leqslant \sqrt{3}$). □

We iteratively resolve crossings in G. Call the resulting graph $\widetilde{G}$. Let $p, q \in P$ be two vertices such that p can reach q in G. Since resolving crossings cannot destroy any path in G, p can also reach q in $\widetilde{G}$. Next, we show the reverse: for any $p, q \in P$, if p can reach q in $\widetilde{G}$, then p can also reach q in G. This seems to be a bit more difficult than one might expect, because when resolving the crossings, we introduce new vertices and edges to which Lemma 6.5 is not directly applicable. To simply the arguments, we make the following general position assumptions: no three vertices in P are collinear and no three edges in G have a common intersection point. These assumptions can be omitted by making a finer case distinction.

Lemma 6.6. *For any two vertices p and q in P, if p can reach q in $\widetilde{G}$ then p can reach q in G.*

Proof. Each edge e of $\widetilde{G}$ lies on an edge of G with the same direction. We call it the *supporting edge* of e. A pair $p, q \in P$ such that p can reach q in $\widetilde{G}$, but not in G is called a *bad pair*. Among all bad pairs, we pick p, q such that there is a path π in $\widetilde{G}$ from p to q with the minimum number of *support switches*, where π changes from one supporting edge to another (see Figure 6.4). Let $p_1q_1, \ldots, p_kq_k$ be the sequence of supporting edges as they are visited along π ($p_1 = p$, $q_k = q$).

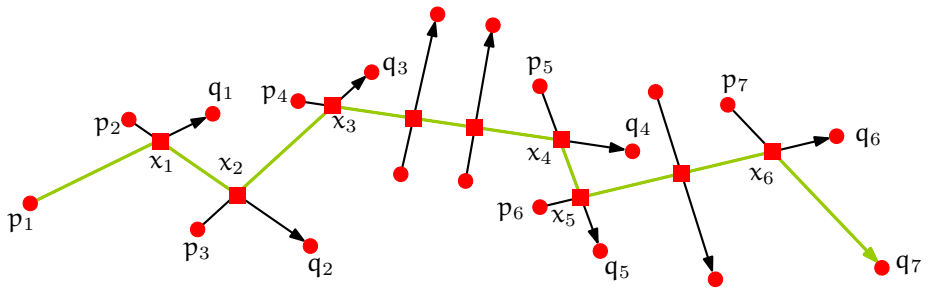


Figure 6.4: A path (green) with $k = 7$ supporting edges that is in $\widetilde{G}$ but not in G.

Claim 6.7. *The following holds in G:*

(P1) p_1 *reaches* $q_2, \ldots, q_{k-1}$

(P2) $p_2, \ldots, p_k$ *reach* q_k

(P3) p_1 *does not reach* $p_2, \ldots, p_k$

(P4) *there is no edge* $q_i p_i$ *for* $i \geqslant 2$.

(P5) *for* $i = 1, \ldots, k-1$ *the line segments* $\overline{p_i q_i}$ *and* $\overline{p_{i+1} q_{i+1}}$ *have a common intersection point* x_i *in their interior*

(P6) *for* $i = 1, \ldots, k-1$ *the intersection point* x_{i+1} *lies in the interior of the line segment* $\overline{x_i q_{i+1}}$.

Proof. ***(P1)*** and ***(P2)*** follow from the minimality of π, and ***(P3)*** follows from ***(P2)***. For ***(P4)***, assume that G contains an edge $q_i p_i$, for $i \geqslant 2$. By ***(P1)***, p_1 reaches q_i in G and thus p_1 reaches p_i, despite ***(P3)***. For ***(P5)***, since $p_i q_i$ and $p_{i+1} q_{i+1}$ are consecutive along π, they must intersect in a point x_i. By our general position assumption, x_i is in their interior, or otherwise we would have three collinear vertices in P. Finally, we can show ***(P6)***: by ***(P4)*** all supporting edges exist only in one direction. Since resolving a crossing preserves the direction of the edges, x_{i+1} must come after x_i along the edge $p_{i+1} q_{i+1}$, i.e., x_{i+1} lies on the line segment $x_i q_{i+1}$. By our general position assumption it must lie in the interior. □

By Lemma 6.5, we have $k \geqslant 3$ supporting edges along π. We now argue that the path π cannot exist and thus there cannot be any bad pair. Since $p_1 q_1$ and $p_2 q_2$ cross, the proof of Lemma 6.5 shows that G contains one of the following edges: $p_1 p_2$, $q_1 p_2$, $p_1 q_2$, or $q_1 q_2$. By ***(P3)***, neither $p_1 p_2$ nor $q_1 p_2$ exist. There are two cases, depending on whether G contains $p_1 q_2$, or $q_1 q_2$ (see Figure 6.5). Each case will lead to a contradiction to the minimality of π.

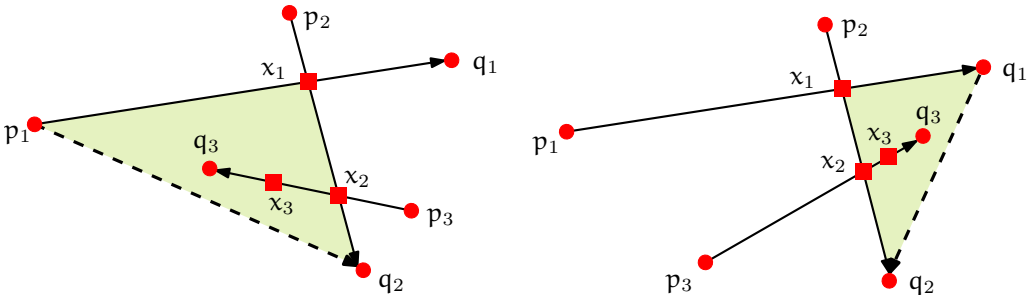


Figure 6.5: Either $p_1 q_2$ or $p_1 q_2$ locks all edges in the corresponding triangle.

Case 1. G contains $p_1 q_2$. Consider the triangle $\triangle = p_1 x_1 q_2$. Since $q_2, x_1 \in D(p_1)$, we have $\triangle \subset D(p_1)$. Thus, by ***(P3)***, none of $p_2, \ldots, p_k$ may lie inside $\triangle$. By ***(P6)***, $p_3 q_3$ intersects the boundary of $\triangle$ in the interior of the line segment $\overline{x_1 q_2}$. First, suppose that $k = 3$. In this case, we have $p_3, q_3 \notin \triangle$ (otherwise p_1 could reach q_3). Thus, $p_3 q_3$ intersects the boundary of $\triangle$ twice, so $p_3 q_3$ either intersects $p_1 q_1$ or $p_1 q_2$. In both cases, Lemma 6.5 shows that p_1 reaches q_3. Thus, we must have $k \geqslant 4$.

We now claim that the intersection x_3 of $p_3 q_3$ and $p_4 q_4$ lies in $\triangle$: if $p_3 q_3$ intersects $\triangle$ once, then $q_3 \in \triangle$, as we already observed that $p_3 \notin \triangle$. ***(P6)*** then gives $x_3 \in \triangle$. Now

suppose p_3q_3 intersects the boundary of $\triangle$ twice, and let y be the second intersection point. We claim that y comes after x_2 along p_3q_3: otherwise, since x_3 comes after x_2 on p_3q_3 by ***(P6)***, we can construct a path with fewer support switches than π: if $y \in \overline{p_1x_1}$, we omit p_2q_2; if $y \in p_1q_2$, we omit p_2q_2 and substitute p_1q_1 with p_1q_2. By the same argument, x_3 cannot come after y on p_3q_3. Thus, x_3 lies on the line segment $\overline{x_2y} \subset \triangle$. This establishes $x_3 \in \triangle$ for both subcases. Now, consider the segment $\overline{p_4x_3}$. Since we observed $p_4 \notin \triangle$, we have that $\overline{p_4x_3}$ intersects $\triangle$, and we can again reroute π to have fewer support switches.

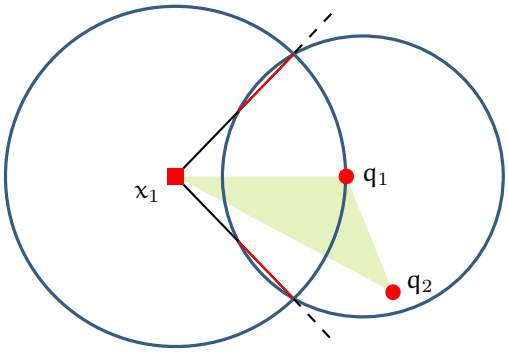


Figure 6.6: In Case 2) the triangle $x_1q_1q_2$ is contained in the union of the disk $D(q_1)$ and $D(x_1)$.

Case 2. G contains q_1q_2. Consider the triangle $\triangle = x_1q_1q_2$. We claim that $\triangle \subset D(p_1) \cup D(q_1)$. Then the remaining argument is analogous to Case 1. Let $D(x_1) \subset D(p_1)$ be the disk with center x_1 and q_1 on its boundary. Let $C(x_1)$ be the boundary of $D(x_1)$. We show that $\triangle \subset D(x_1) \cup D(q_1)$. If $x_1 \in D(q_1)$, then $\triangle \subset D(q_1)$ and we are done. Otherwise, since the two rays from x_1 through $C(x_1) \cap C(q_1)$ intersect $D(q_1)$ inside $D(x_1)$ (see red parts in Figure 6.6), all line segments from x_1 to a point on $C(q_1)$ lie in $D(x_1) \cup D(q_1)$. The claim follows. □

6.2.3 Putting it Together

To prove Lemma 6.1, we first construct the sparse subgraph H of G as in Lemma 6.3 in time $O(n \log n)$. Then we iteratively resolve the crossings in H to obtain a planar graph $\widetilde{H}$. Since H has $O(n)$ crossings that can be found in $O(n)$ time, this takes $O(n)$ time.

Let $p, q \in P$. We must argue that p can reach q in G if and only if p can reach q in $\widetilde{H}$. Let $\widetilde{G}$ be the graph obtained by resolving the crossings in G, as in Lemma 6.6. We know that the reachability between p and q is the same in G, H, and $\widetilde{G}$. Furthermore, if p can reach q in H, then also in $\widetilde{H}$ since resolving the crossing cannot destroy any paths that are in H. Finally, if p can reach q in $\widetilde{H}$, then also in $\widetilde{G}$, because (a subdivision of) every edge of $\widetilde{H}$ is present in $\widetilde{G}$. Thus, $\widetilde{H}$ and G have the same reachability properties. See also Figure 6.7 for a schematic overview of the arguments.

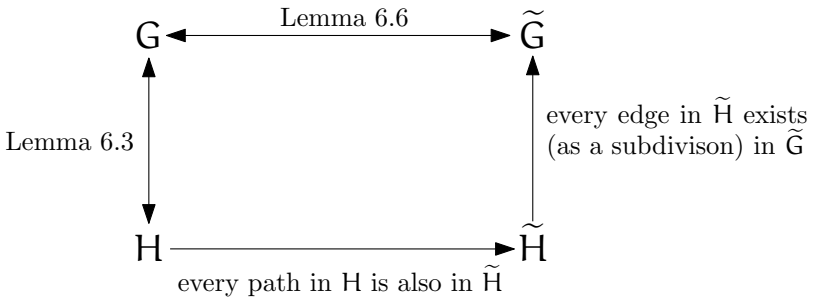


Figure 6.7: The black arrows indicate the reachability between sites in P in the graphs G, $\widetilde{G}$, H, and $\widetilde{H}$.

6.3 Polynomial Dependence on Ψ

We now present a standard reachability oracle whose performance parameters depend polynomially on the radius ratio Ψ. Together with Theorem 5.25 we will obtain the following result:

Theorem 6.8. *Let G be the transmission graph for a set $P \subset \mathbb{R}^2$ of n points. We can construct a geometric reachability oracle for G with $S(n) = O(\Psi^3 n^{3/2})$ and $Q(n) = O(\Psi^3\sqrt{n})$ in time $O(\Psi^3 n^{3/2})$.*

Our approach is based on a geometric separator theorem for planar disks. Let P be a set of n points, each having an assigned radius. We may assume that the smallest radius in P is 1. Let $\mathcal{D}$ be the set of disks induced by P. We write $\bigcup\mathcal{D} := \bigcup_{D\in\mathcal{D}} D$ and we let $\mu(\mathcal{D})$ be the area occupied by $\bigcup\mathcal{D}$. Alber and Fiala show how to find a separator for $\mathcal{D}$ with respect to $\mu(\cdot)$ [AF04].

Theorem 6.9 (Theorem 4.12 in [AF04])**.** *There exist positive constants $\alpha < 1$ and β such that the following holds: let $\mathcal{D}$ be a set of n disks and Ψ the ratio of the largest and the smallest radius in $\mathcal{D}$. Then we can find in time $O(\Psi^2 n)$ a partition $\mathcal{A} \cup \mathcal{B} \cup \mathcal{S}$ of $\mathcal{D}$ satisfying (i) $\bigcup\mathcal{A} \cap \bigcup\mathcal{B} = \emptyset$, (ii) $\mu(\mathcal{S}) \leqslant \beta\Psi^2\sqrt{\mu(\mathcal{D})}$ and (iii) $\mu(\mathcal{A}), \mu(\mathcal{B}) \leqslant \alpha\mu(\mathcal{D})$.*

Let G be the transmission graph of P. Since any directed path in G lies completely in $\bigcup\mathcal{D}$, any path from a vertex of a disk in $\mathcal{A}$ to a vertex of a disk in $\mathcal{B}$ needs to use at least one vertex of a disk in $\mathcal{S}$, see Figure 6.8. Since $\mu(\mathcal{S})$ is small, we can approximate $\bigcup\mathcal{S}$ with few grid cells. We choose the diameter of the cells small enough such that all vertices in one cell form a clique and are equivalent in terms of reachability. We can thus pick one vertex per cell and store the reachability information for it. Applying this idea recursively gives a separator tree of logarithmic depth that lets us answer reachability queries. Details follow.

Preprocessing Algorithm and Space Requirement

For the preprocessing phase, consider the grid $\mathcal{G} = \mathcal{G}_0$ whose cells have diameter 1. All vertices in a single cell form a clique in G, so it suffices to determine the reachability for

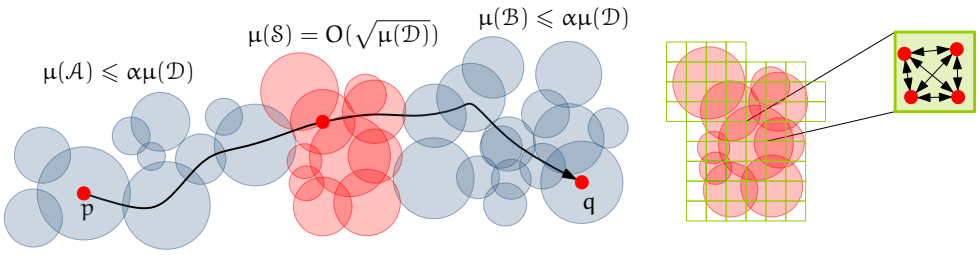


Figure 6.8: Any path from $\mathcal{A}$ to $\mathcal{B}$ needs to use at least one vertex of $\mathcal{S}$. Since $\mu(\mathcal{S})$ is small, we can approximate $\bigcup\mathcal{S}$ with few grid cells.

one such vertex. For each non-empty cell $\sigma \in \mathcal{G}$, we pick an arbitrary vertex $p_\sigma \in P \cap \sigma$ as the *representative* of σ.

Starting with $\mathcal{D}$, we recursively create a separator tree T that contains all the required reachability information: we create a root node ν of the separator tree and we set $\mathcal{D}_\nu = \mathcal{D}$. Let $P_\nu \subseteq P$ be the sites corresponding to disks in $\mathcal{D}_\nu$. We compute $\mathcal{A}, \mathcal{B}$, and $\mathcal{S}_\nu = \mathcal{S}$ for $\mathcal{D}_\nu$ according to Theorem 6.9. Let Q_ν be all cells in $\mathcal{G}$ that intersect $\bigcup\mathcal{S}_\nu$. Let R_ν be the representatives of Q_ν. For each $r \in R_\nu$, we store at ν all sites in P_ν that r can reach and all the sites in P_ν that can reach r in the transmission graph induced by P_ν. To obtain the reachability information, we compute a 2-spanner H_ν for the transmission graph induced by P_ν, as in Theorem 5.12. Since we are only interested in the reachability properties of the spanner, a 2-spanner (or any spanner with constant stretch) suffices. For each representative $r \in R_\nu$, we compute a BFS tree in H_ν with root r. Next, we reverse all edges in H_ν, and we again compute BFS-trees for all $r \in R_\nu$ in the transposed graph. This gives the required reachability information for ν. If $\mu(\mathcal{D}) = O(1)$ we store at ν all sites of P_ν and we stop. Otherwise, we recursively compute separator trees for $\mathcal{A}$ and $\mathcal{B}$, and we connect them to ν.

As T has $O(\log(\Psi n))$ levels by Theorem 6.9 and since $\mu(D) = O(\Psi^2 n)$, the total running time for computing the 2-spanners is $O(n(\log n + \log\Psi)^2)$ by Theorem 5.12. Since the spanners are sparse, computing one BFS tree requires linear time. Furthermore, every time we compute BFS tree for a representative $r \in R_\nu$, we store the reachability information for the complete tree at ν. Thus, the total time for computing the BFS-trees is proportional to the total storage we need. Below, we will show that this is at most $O(\Psi^3 n^{3/2})$. Then, the total preprocessing time is $O(n(\log n + \log\Psi)^2 + \Psi^3 n^{3/2}) = O(\Psi^3 n^{3/2})$.

To bound the space requirement, we show that a set $\mathcal{D}$ of disks has $O(\mu(\mathcal{D}))$ representatives.

Lemma 6.10. *Let $\mathcal{D}$ be a set of n disks with radius at least 1. Then the number of cells in $\mathcal{G}$ that intersect $\bigcup\mathcal{D}$ is $O(\mu(\mathcal{D}))$.*

Proof. Suppose that a cell $\sigma \in \mathcal{G}$ intersects a disk $D \in \mathcal{D}$. Then D contains a disk of radius 1 that intersects the boundary of σ. Thus, the intersection of $\bigcup\mathcal{D}$ and the region consisting of σ and its eight surrounding cells has area at least 1. Since there can be only $O(\mu(D))$ different regions of this kind, the claim follows. □

Theorem 6.9(ii) and Lemma 6.10 imply that $|R_v| = O(\Psi^2\sqrt{\mu(\mathcal{D}_v)})$ and so the size of the reachability table at a node v is $O(\Psi^2\sqrt{\mu(\mathcal{D}_v)}|P_v|)$. Thus, we obtain the following recursion with respect to $\mu(\cdot)$ for the space requirement $S(\mu(\mathcal{D}), n)$ for a set $\mathcal{D}$ of n disks:

$$S(\mu(\mathcal{D}), n) = S(\mu(\mathcal{A}), n_1) + S(\mu(\mathcal{B}), n_2) + O(\Psi^2\sqrt{\mu(\mathcal{D})}n), \tag{6.1}$$

where $n_1 = |\mathcal{A}|$, $n_2 = |B|$ and $S(O(1), n) = O(n)$. Since $n_1 + n_2 \leqslant n$ and since Theorem 6.9 gives $\max\{\mu(\mathcal{A}), \mu(\mathcal{B})\} \leqslant (1-\alpha)\mu(\mathcal{D})$, the recursion in (6.1) solves to $S(\mu(\mathcal{D}), n) = O(\Psi^2\sqrt{\mu(\mathcal{D})}n)$. As $\mu(\mathcal{D}) = O(n\Psi^2)$, the total space is $O(\Psi^3 n^{3/2})$.

Query Algorithm

Let $p, q \in P$ be given. We let v be the node in T with $p \in R_v$, or, if no such node exists, we let v be the leaf node of T where $p \in P_v$ is stored. In the same way we pick the node w for q. Let u be the least common ancestor of v and w. It can be found in $O(\Psi \log n)$ time by walking up the tree T. Let L be the path from u to the root of T. We check for each representative $r \in \bigcup_{x\in L} R_x$ whether p can reach r and whether r can reach q. If so, we return YES. If there is no such r, we return NO. Since $|R_x|$ increases geometrically along L, the running time is dominated by the time for processing the root, which is $O(\Psi^2\sqrt{\mu(\mathcal{D})})$. Bounding $\mu(\mathcal{D})$ by $O(\Psi^2 n)$, the total query time is $O(\Psi^3\sqrt{n})$.

It remains to argue that our query algorithm is correct. By construction, it follows that we return YES only if there is a path from p to q. Now, suppose there is a path π in G from p to q. Let v, w be the (leaf) nodes in T for p and q as defined above. Let u be their least common ancestor, and let L be the path from u to the root. By construction, $\bigcup_{x\in L} S_x$ contains a disk for a vertex r in π. We pick r such that the corresponding node $x \in T$ is closest to the root. Let r' be the representative for the cell σ containing r. Since the vertices in σ constitute a clique, p can reach r' and r' can reach q in the subgraph of G induced by P_x. Thus, when walking along L, the algorithm will discover r' and the path from p to q. Theorem 6.8 now follows.

6.4 Logarithmic Dependence on Ψ

Finally, we improve the dependence on Ψ to be logarithmic, at the cost of a slight increase of the exponent for n. We can show the following theorem by constructing a standard reachability oracle and then using Theorem 5.25.

Theorem 6.11. *Let G be the transmission graph for a set P of n points in the plane. We can construct a geometric reachability oracle for G with $S(n) = O(n^{5/3}\log^{1/3}\Psi\log^{2/3} n)$ and $Q(n) = O(n^{2/3}\log^{1/3}\Psi\log^{2/3} n)$. All queries are answered correctly with high probability. The preprocessing time is $O(n^{5/3}(\log\Psi + \log n)\log^{1/3}\Psi\log^{2/3} n)$.*

We scale everything such that the smallest radius in P is 1. Our approach is as follows: let $p, q \in P$. If there is a p-q-path with "many" vertices, we detect this by taking a large enough random sample $S \subseteq P$ and by storing the reachability information for every

vertex in S. If there is a path from p to q with "few" vertices, then p must be "close" to q, where "closeness" is defined relative to the largest radius along the path. The radii from P can lie in $O(\log \Psi)$ different scales, and for each scale we store local information to find such a short path.

Long Paths

Let $0 < \alpha < 1$ be a parameter to be determined later. First, we show that a random sample can be used to detect paths with many vertices.

Lemma 6.12. *We can sample a set $S \subseteq P$ of size $O(n^\alpha \log n)$ such that the following holds with high probability at least $1 - 1/n$: for any $p, q \in P$, if there is a path π from p to q in G with at least $n^{1-\alpha}$ vertices, then $\pi \cap S \neq \emptyset$.*

Proof. Let $m = 4n^\alpha \ln n$. We construct S by including each $p \in P$ independently with probability m/n. Using Chernoff bounds [Mul15], we get

$$\Pr[|S| \geqslant 8n^\alpha \ln n] \leqslant e^{-n^\alpha \ln n} \leqslant \frac{1}{2n}.$$

Thus, S has size $O(n^\alpha \log n)$ with probability at least $1 - 1/2n$. Now fix p and q and let π be a path from p to q with $k \geqslant n^{1-\alpha}$ vertices. The probability that S contains no vertex from π is at most $(1-m/n)^k \leqslant e^{-mk/n} \leqslant 1/n^4$, by our choice of m. Since there are $n(n-1)$ ordered vertex pairs, the union bound shows that the probability that S fails to detect a pair of vertices connected by a long path is at most $n(n-1)/n^4 \leqslant 1/2n$, for all $n \geqslant 1$. The lemma follows by a union bound. □

We compute a sample S as in Lemma 6.12, and for each $s \in S$, we store two Boolean arrays that indicate for each $p \in P$ whether p can reach s and whether s can reach p. This needs space $O(n^{1+\alpha} \log n)$. It remains to deal with vertices that are connected by a path with less than $n^{1-\alpha}$ vertices.

Short Paths

Let $L = \lceil \log \Psi \rceil$. We consider the L grids $\mathcal{G}_0, \ldots, \mathcal{G}_L$ (recall that the cells in $\mathcal{G}_i$ have diameter 2^i). For each cell $\sigma \in \mathcal{G}_i$, let $R_\sigma \subseteq P$ be the vertices $p \in P \cap \sigma$ with $r_p \in [2^i, 2^{i+1})$. The set R_σ forms a clique in G, and for each $p \in R_\sigma$, the disk $D(p)$ contains the cell σ. The *neighborhood* $N(\sigma)$ of σ is defined as the set of all cells in $\mathcal{G}_i$ that have distance at most $2^{i+1}n^{1-\alpha}$ from σ. We have $|N(\sigma)| = O(n^{2-2\alpha})$. Let $P_\sigma \subseteq P$ be the vertices that lie in cells of $N(\sigma)$. For every $i = 0, \ldots, L$ and for every $\sigma \in \mathcal{G}_i$ with $R_\sigma \neq \emptyset$, we fix an arbitrary *representative point* $r_\sigma \in R_\sigma$. For every vertex $p \in P$, we store for every $i \in \{0, \ldots, L\}$ two sorted lists of cells $\sigma \in \mathcal{G}_i$ with $p \in P_\sigma$: the first list contains all corresponding representatives r_σ that can be reached from p; the second list contains all corresponding representatives r_σ that can reach p. A vertex p appears in at most $O(n^{2-2\alpha} \log \Psi)$ point sets P_σ, so the total space is $O(n^{3-2\alpha} \log \Psi)$.

Query Algorithm

Let $p, q \in P$ be given. To decide whether p can reach q, we first check the Boolean tables for all $O(n^\alpha \log n)$ points in S. If there is an $s \in S$ such that p reaches s and s reaches q, we return YES. If not, for $i \in \{0, \dots, L\}$, we consider the list of representatives that are reachable from p in the neighborhood at level i and the list of representatives that can reach q in the neighborhood at level i. We check whether these lists contain a common element. Since the lists are sorted, this can be done in time linear in their size. If we find a common representative at for some i, we return YES. Otherwise, we return NO.

We now prove the correctness of the query algorithm. First note that we return YES, only if there is a path from p to q. Now suppose that there is a path π from p to q. If π has at least $n^{1-\alpha}$ vertices, then by Lemma 6.12, the sample S hits π with probability at least $1 - 1/n$, and the algorithm returns YES. If π has less than $n^{1-\alpha}$ vertices, let r be the vertex of π with the largest radius, and let i be such that the radius of r lies in $[2^i, 2^{i+1})$. Let σ be the cell of $\mathcal{G}_i$ that contains r. Since π has at most $n^{1-\alpha}$ vertices, and since each edge of π has length at most 2^{i+1}, the path π lies entirely in the cells of $N(\sigma)$. In particular, both p and q are contained in cells of $N(\sigma)$. Since $r \in R_\sigma$ and since R_σ forms a clique in G, the representative point r_σ of σ can be reached from p and can reach q. By the symmetry of the neighborhood relation, r_σ is contained in the list of reachable representatives from p and in the lists of representatives that can reach q. This is detected when checking the corresponding lists for p and q at level i.

Time and Space Requirements

For querying long paths we need $O(n^\alpha \log n)$ time: for every $s \in S$ we test in $O(1)$ time whether p can reach s and whether s can reach q. For querying short paths there are $O(\log \Psi)$ levels, and at each level we step through two lists of size $O(n^{2-2\alpha})$. Thus, the tradeoff-point is achieved for

$$n^\alpha \log n = n^{2-2\alpha} \log \Psi \Leftrightarrow n^\alpha = n^{2/3} (\log \Psi / \log n)^{1/3}.$$

This yields $Q(n) = O(n^{2/3} \log^{1/3} \Psi \log^{2/3} n)$. This choice of α gives a space bound of $O(n^{5/3} \log^{1/3} \Psi \log^{2/3} n)$.

For the preprocessing algorithm, we first compute the reachability arrays for each $s \in S$. To do so, we build a 2-spanner H for G as in Theorem 5.12 in time $O(n(\log n + \log \Psi))$. Then, for each $s \in S$ we perform a BFS search in H and its transposed graph. This gives all vertices that s can reach and that can be reached by s in $O(n^{5/3} \log^{1/3} \Psi \log^{2/3} n)$ total time. For the short paths, the preprocessing algorithm goes as follows: For each $i = 0, \dots, L$ and for each cell $\sigma \in \mathcal{G}_i$ that has a representative r_σ, we compute a 2-spanner H_σ as in Theorem 5.12 for P_σ. For each representative r_σ, we do a BFS search in H_σ and the transposed graph, each starting from r_σ. This gives all $p \in P_\sigma$ that can reach r_σ and that are reachable from r_σ. The running time is dominated by the time for constructing the spanners. Since each point $p \in P$ is contained in $O(n^{2-2\alpha} \log \Psi) = O(n^{2/3} \log^{1/3} \Psi \log^{2/3} n)$ different P_σ, and since constructing

H_σ takes $O(|P_\sigma|(\log \Psi + \log |P_\sigma|))$ time, it follows that the total preprocessing time is $O(n^{5/3}(\log \Psi + \log n) \log^{1/3} \Psi \log^{2/3} n)$.

6.5 Conclusion

Transmission graphs constitute a natural class of directed graphs for which non-trivial reachability oracles can be constructed. As mentioned at the beginning of the chapter, it seems to be very difficult to obtain similar results for general directed graphs. We believe that our results only scratch the surface of the possibilities offered by transmission graphs, and several interesting open problems remain.

All our results depend on the radius ratio Ψ and the major question is whether this dependency can be avoided. Our most efficient reachability oracle is for $\Psi < \sqrt{3}$. In this case the reachability of a transmission graph with n vertices can be represented by a planar graph with $O(n)$ vertices. However, it is not clear to us that the bound of $\sqrt{3}$ is tight. Can we obtain a similar result for, say, $\Psi = 100$? Or is there even a way to represent *any* transmission graph, regardless of Ψ, by a planar graph with $o(n^2)$ vertices? This would immediately imply a non-trivial reachability oracles for all ranges of Ψ.

Conversely, it would be interesting to see if we can represent the reachability of arbitrary directed graphs using transmission graphs. If this is possible, the relevant questions are how many vertices the transmission must have, what the required radius ratio is, and how fast it can be computed. A representation that achieves both few vertices and low radius ratio would lead to efficient reachability oracles for general directed graphs.

Parting Thoughts

In this thesis we considered three types of disk intersection graphs, namely unit disk graphs, disk graphs, and transmission graphs, and we explored their algorithmic capabilities. We gave efficient data structures and algorithms for several problems related to connectivity and (approximately) shortest paths in these graphs.

In the first part of this thesis, we studied unit disk graphs, and we gave a data structure that solves the dynamic connectivity problem for them (see Chapter 3). Our solution yields a significant improvement in the update time, compared to the previous best result by Chan et al. [CPR11]. However, there is no reason to believe that our bounds, especially for the update time, are optimal. Thus, we would like to know if there are solutions with improved update and/or query time. Also, we would like to derive some lower bounds for the problem.

In Chapter 4 we presented the first efficient routing scheme with stretch $1 + \varepsilon$ for unit disk graphs. This improves a previous routing scheme by Yan et al. [YXD12] that has stretch slightly larger than 3. The main geometric tool we used was the well-separated pair decomposition for the unit disk graph metric by Gao and Zhang [GZ05]. Already Yan et al. [YXD12] asked whether this well-separated pair decomposition can be helpful to obtain a routing scheme with $1+\varepsilon$ stretch. Now, we can answer this question affirmatively. The main drawback of our routing scheme is the need of a modifiable header at the packet. In future work, we will try to find a headerless routing scheme for unit disk graphs that also has stretch $1 + \varepsilon$.

The second part of the thesis contains our results for transmission graphs. We saw several spanner constructions in Chapter 5. The most general one can construct a spanner in time $O(n \log^5 n)$, where n is the number of sites. However, the usual spanner constructions for the complete Euclidean graph (e.g., Yao graphs, Θ-graphs, WSPD spanners) require only $O(n \log n)$ time [NS07]. The major open question from Chapter 5 is: can we construct spanners for transmission graphs also in time $O(n \log n)$? Or have transmission graphs any structural barriers that rule out such an algorithm?

In Chapter 6 we presented three different reachability oracles for transmission graphs. All of them had a dependence on the radius ratio Ψ, and performed best for a certain range of Ψ. However, once Ψ gets large enough, all of our oracles fail to beat the trivial solutions. The existence of non-trivial reachability oracles that are independent of Ψ is the most compelling open question that remains from Chapter 6.

We managed to extend our dynamic connectivity structure and the spanner construction to disk graphs. These extensions required a dynamic data structure for additively weighted Euclidean nearest neighbors. For this, we used a more general data structure that is capable of solving the dynamic nearest neighbor problem for a wide range of

distance functions (see Section 2.3.2 and Theorem 2.6). This data structure has a rather high update time and this directly reflects to our solutions. It is likely that for the special case of additively weighted Euclidean distance functions we can find a better solution. This is particularly reasonable since Euclidean and additively weighted Euclidean distance functions induce very similar surfaces in three dimensions (cones versus shifted cones, cf. Section 2.3.2), and for Euclidean distances there already is a more efficient dynamic nearest neighbors structure available (see Corollary 2.5).

In conclusion, disk intersection graphs constitute an interesting class of graphs to investigate and we managed to solve several natural problems for these graphs, but still there is further work to be done. Unit disk graphs have already been studied for the last decades, but even for them we still can find compelling open problems that are worth to study. For transmission graphs, the picture looks quite different: even though they are a very natural generalization of unit disk graphs, they received not much attention by the research community, yet. Except for the range assignment problems mentioned in the introduction (see Section 1.3), we are not aware of any another work where transmission graphs are studied. Certainly, our results just scratched the surface and there are much more algorithmic problems in the context of transmission graphs that want to be discovered and solved. We hope that some of our results will encourage further people to study these fascinating graphs, and that this will be the forerunner of a longer line of research in the near future.

Bibliography

[AC09] Peyman Afshani and Timothy M. Chan. Optimal halfspace range reporting in three dimensions. In *Proc. 20th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 180–186, 2009.

[ACC+96] Srinivasa Arikati, Danny Z. Chen, L. Paul Chew, Gautam Das, Michiel H. M. Smid, and Christos D. Zaroliagis. Planar spanners and approximate shortest path queries among obstacles in the plane. In *Proc. 4th Annu. European Sympos. Algorithms (ESA)*, pages 514–528, 1996.

[AES99] Pankaj K. Agarwal, Alon Efrat, and Micha Sharir. Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications. *SIAM J. Comput.*, 29(3):912–953, 1999.

[AESW91] Pankaj K. Agarwal, Herbert Edelsbrunner, Otfried Schwarzkopf, and Emo Welzl. Euclidean minimum spanning trees and bichromatic closest pairs. *Discrete Comput. Geom.*, 6(5):407–422, 1991.

[AF04] Jochen Alber and Jirí Fiala. Geometric separation and exact solutions for the parameterized independent set problem on disk graphs. *J. Algorithms*, 52(2):134–151, 2004.

[Afs14] Vala Afshar. 50 incredible WiFi tech statistics that businesses must know [slide deck], 2014. http://www.huffingtonpost.com/vala-afshar/50-incredible-wifi-tech-s_b_4775837.html Accessed: 2016-05-16.

[AKL13] Franz Aurenhammer, Rolf Klein, and Der-Tsai Lee. *Voronoi Diagrams and Delaunay Triangulations*. World Scientific Publishing, 2013.

[AM95] Pankaj K. Agarwal and Jirí Matoušek. Dynamic half-space range reporting and its applications. *Algorithmica*, 13(4):325–345, 1995.

[AMS94] Sunil Arya, David M. Mount, and Michiel H. M. Smid. Randomized and deterministic algorithms for geometric spanners of small diameter. In *Proc. 35th Annu. IEEE Sympos. Found. Comput. Sci. (FOCS)*, pages 703–712, 1994.

[AW05] Christoph Ambühl and Uli Wagner. The clique problem in intersection graphs of ellipses and triangles. *Theory Comput. Syst.*, 38(3):279–292, 2005.

[BBE+14] Yves Brise, Kevin Buchin, Dustin Eversmann, Michael Hoffmann, and Wolfgang Mulzer. Interference minimization in asymmetric sensor networks. In *Proc. 10th Int. Symp. on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics (ALGOSENSORS)*, pages 136–151, 2014.

[BDD+12] Prosenjit Bose, Mirela Damian, Karim Douïeb, Joseph O'Rourke, Ben Seamone, Michiel H. M. Smid, and Stefanie Wuhrer. $\pi/2$-angle Yao graphs are spanners. *Internat. J. Comput. Geom. Appl.*, 22(1):61–82, 2012.

[BBK16] Mark de Berg, Hans Bodlaender, and Sándor Kisfaludi-Bak. Connected dominating set in unit-disk graphs is W[1]-hard. In *Proc. 32nd European Workshop on Comput. Geom. (EWCG)*, 2016.

[BCvKO08] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 3rd edition, 2008.

[BJ00] Gerth Stølting Brodal and Riko Jacob. Dynamic planar convex hull with optimal query time and $O(\log n \cdot \log \log n)$ update time. In *Proc. 7th Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 57–70, 2000.

[BJ02] Gerth Stølting Brodal and Riko Jacob. Dynamic planar convex hull. In *Proc. 43rd Annu. IEEE Sympos. Found. Comput. Sci. (FOCS)*, pages 617–626, 2002.

[BK98] Heinz Breu and David G. Kirkpatrick. Unit disk graph recognition is NP-hard. *Comput. Geom.*, 9(1-2):3–24, 1998.

[Bou08] Azzedine Boukerche. *Algorithms and Protocols for Wireless Sensor Networks*. Wiley Series on Parallel and Distributed Computing). John Wiley & Sons, 1st edition, 2008.

[Car16] Paz Carmi. Excercise sheet for Geometric Spanners class, 2016. http://www.cs.bgu.ac.il/~gs161/wiki.files/GeoSpann2106Ass1.pdf Accessed: 2016-03-09.

[CC15] Paz Carmi and Lilach Chaitman-Yerushalmi. On the minimum cost range assignment problem. In *Proc. 26th Annu. Internat. Sympos. Algorithms Comput. (ISAAC)*, pages 95–105, 2015.

[CCJ90] Brent N. Clark, Charles J. Colbourn, and David S. Johnson. Unit disk graphs. *Discrete Math.*, 86(1-3):165–177, 1990.

[CCT15] Paz Carmi, Lilach Chaitman-Yerushalmi, and Ohad Trabelsi. On the bounded-hop range assignment problem. In *Proc. 14th Algorithms and Data Structures Symposium (WADS)*, pages 140–151, 2015.

[Cha01] Timothy M. Chan. Dynamic planar convex hull operations in near-logarithmic amortized time. *J. ACM*, 48(1):1–12, 2001.

[Cha06] Timothy M. Chan. A dynamic data structure for 3-D convex hulls and 2-D nearest neighbor queries. In *Proc. 17th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 1196–1202, 2006.

[Cha10] Timothy M. Chan. A dynamic data structure for 3-D convex hulls and 2-D nearest neighbor queries. *J. ACM*, 57(3):Art. 16, 15, 2010.

[Che13] Shiri Chechik. Compact routing schemes with improved stretch. In *Proc. 32nd ACM Symp. on Principles of Distributed Computing (PODC)*, pages 33–41, 2013.

[CHT90] M. S. Chang, N. F. Huang, and C. Y. Tang. An optimal algorithm for constructing oriented Voronoi diagrams and geographic neighborhood graphs. *Inform. Process. Lett.*, 35(5):255–260, 1990.

[CJ15] Sergio Cabello and Miha Jejčič. Shortest paths in intersection graphs of unit disks. *Comput. Geom.*, 48(4):360–367, 2015.

[CK93] Paul B. Callahan and S. Rao Kosaraju. Faster algorithms for some geometric graph problems in higher dimensions. In *Proc. 4th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 291–300, 1993.

[CK95a] Paul B. Callahan and S. Rao Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *J. ACM*, 42(1):67–90, 1995.

[CK95b] Paul B. Callahan and S. Rao Kosaraju. Algorithms for dynamic closest pair and n-body potential fields. In *Proc. 6th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 263–272, 1995.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 3rd edition, 2009.

[CPR11] Timothy M. Chan, Mihai Pătraşcu, and Liam Roditty. Dynamic connectivity: connecting to networks and geometry. *SIAM Journal on Computing*, 40(2):333–349, 2011.

[CPS04] Andrea E. F. Clementi, Paolo Penna, and Riccardo Silvestri. On the power assignment problem in radio networks. *Mobile Networks and Applications (MONET)*, 9(2):125–140, 2004.

[CT15] Timothy M. Chan and Konstantinos A. Tsakalidis. Optimal deterministic algorithms for 2-d and 3-d shallow cuttings. In *Proc.31st Int. Sympos. Comput. Geom. (SoCG)*, pages 719–732, 2015.

[CX00] Danny Z. Chen and Jinhui Xu. Shortest path queries in planar graphs. In F. Frances Yao and Eugene M. Luks, editors, *Proc. 32nd Annu. ACM Sympos. Theory Comput. (STOC)*, pages 469–478, 2000.

[DGN07] Gautam K. Das, Sasthi C. Ghosh, and Subhas C. Nandy. Improved algorithm for minimum cost range assignment problem for linear radio networks. *Int. J. Found. Comput. Sci.*, 18(3):619–635, 2007.

[Dji96] Hristo N. Djidjev. Efficient algorithms for shortest path queries in planar digraphs. In *Proc. 22nd International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 151–165, 1996.

[EIT+92] David Eppstein, Giuseppe F. Italiano, Roberto Tamassia, Robert E. Tarjan, Jeffery Westbrook, and Moti Yung. Maintenance of a minimum spanning forest in a dynamic plane graph. *J. Algorithms*, 13(1):33–54, 1992.

[Epp95] David Eppstein. Dynamic Euclidean minimum spanning trees and extrema of binary functions. *Discrete Comput. Geom.*, 13:111–122, 1995.

[Fed87] Greg N. Federickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM Journal on Computing*, 16(6):1004–1022, 1987.

[FG01] Pierre Fraigniaud and Cyril Gavoille. Routing in trees. In *Proc. 28th Internat. Colloq. Automata Lang. Program. (ICALP)*, pages 757–772, 2001.

[FK12] Martin Fürer and Shiva Prasad Kasiviswanathan. Spanners for geometric intersection graphs with applications. *J. Comput. Geom.*, 3(1):31–64, 2012.

[FWZ04] Martin Fussen, Roger Wattenhofer, and Aaron Zollinger. On interference reduction in sensor networks. Technical Report Technical Report 453, ETH Zürich, Department of Computer Science, 2004.

[GKR04] Anupam Gupta, Amit Kumar, and Rajeev Rastogi. Traveling with a Pez dispenser (or, routing issues in MPLS). *SIAM Journal on Computing*, 34(2):453–474, 2004.

[GS04] Silvia Giordano and Ivan Stojmenovic. Position based routing algorithms for ad hoc networks: A taxonomy. In *Ad Hoc Wireless Networking*, volume 14 of *Network Theory and Applications*, pages 103–136. Springer-Verlag, 2004.

[GZ05] Jie Gao and Li Zhang. Well-separated pair decomposition for the unit-disk graph metric and its applications. *SIAM J. Comput.*, 35(1):151–169, 2005.

[HdLT01] Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, 2001.

[HM06] Sariel Har-Peled and Manor Mendel. Fast construction of nets in low-dimensional metrics and their applications. *SIAM J. Comput.*, 35(5):1148–1184, 2006.

[HP11] Sariel Har-Peled. *Geometric Approximation Algorithms*. American Mathematical Society, 2011.

[HR16] Tony Hoffmann and Laarn Almendrala Ragaza. The 10 best wireless printers of 2016, 2016. http://www.pcmag.com/article2/0,2817,2379649,00.asp Accessed: 2016-05-16.

[HRT14] Jacob Holm, Eva Rotenberg, and Mikkel Thorup. Planar reachability in linear space and constant time. *CoRR*, arXiv:1411.5867, 2014.

[HS95] M. L. Huson and A. Sen. Broadcast scheduling algorithms for radio networks. In *IEEE Military Communications Conference (MILCOM)*, volume 2, pages 647–651, 1995.

[IIM85] Hiroshi Imai, Masao Iri, and Kazuo Murota. Voronoi diagram in the Laguerre geometry and its applications. *SIAM J. Comput.*, 14(1):93–105, 1985.

[Inc16] Brewie Inc. Brewie product description, 2016. http://www.brewie.org/machine/ Accessed: 2016-05-16.

[Jur15] Nico Jurran. Android wear mit wlan-unterstützung und "handgelenkgesten", 2015. http://www.heise.de/newsticker/meldung/Android-Wear-mit-WLAN-Unterstuetzung-und-Handgelenkgesten-2614327.html Accessed: 2016-05-16.

[Kan15] Axel Kannenberg. Ifa 2015: Vernetzte waschmaschine schlägt waschmittelbestellung vor, 2015. http://www.heise.de/newsticker/meldung/IFA-2015-Vernetzte-Waschmaschine-schlaegt-Waschmittelbestellung-vor-2804947.html Accessed: 2016-05-16.

[Kir83] D. Kirkpatrick. Optimal search in planar subdivisions. *SIAM J. Comput.*, 12(1):28–35, 1983.

[KKKP00] Lefteris M. Kirousis, Evangelos Kranakis, Danny Krizanc, and Andrzej Pelc. Power consumption in packet radio networks. *Theoret. Comput. Sci.*, 243(1-2):289–305, 2000.

[KM12] Ross J. Kang and Tobias Müller. Sphere and dot product representations of graphs. *Discrete Comput. Geom.*, 47(3):548–568, 2012.

[KMR+16] Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. Dynamic planar Voronoi diagrams for general distance functions and their algorithmic applications. Manuscript submitted to FOCS'16, 2016.

[KMRS15] Haim Kaplan, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. Spanners and reachability oracles for directed transmission graphs. In *Proc. 31st Int. Sympos. Comput. Geom. (SoCG)*, pages 156–170, 2015.

[KMRS16a] Haim Kaplan, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. Dynamic connectivity for unit disk graphs. In *Proc. 32nd European Workshop on Comput. Geom. (EWCG)*, 2016.

[KMRS16b] Haim Kaplan, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. Routing in unit disk graphs. In *Proc. 12th Latin American Theoretical Informatics Symposium (LATIN)*, pages 536–548, 2016.

[KTT01] Haim Kaplan, Robert E. Tarjan, and Kostas Tsioutsiouliklis. Faster kinetic heaps and their use in broadcast scheduling (extended abstract). In *Proc. 12th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 836–844, 2001.

[LM12] Maarten Löffler and Wolfgang Mulzer. Triangulating the square and squaring the triangle: quadtrees and Delaunay triangulations are equivalent. *SIAM Journal on Computing*, 41(4):941–974, 2012.

[LT80] Richard J. Lipton and Robert Endre Tarjan. Applications of a planar separator theorem. *SIAM J. Comput.*, 9(3):615–627, 1980.

[Mar06] Dániel Marx. Parameterized complexity of independence and domination on geometric graphs. In *Proc. 2nd Int. Workshop on Parameterized and Exact Computation (IWPEC)*, pages 154–165, 2006.

[Mat92] Jirí Matoušek. Reporting points in halfspaces. *Comput. Geom.*, 2:169–186, 1992.

[Mat14] Jirí Matoušek. Intersection graphs of segments and $\exists\mathbb{R}$. *CoRR*, abs/1406.2636, 2014.

[Meh84] Kurt Mehlhorn. *Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry*, volume 3 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1984.

[Mul15] Wolfgang Mulzer. Lecture notes on Chernoff-bounds, 2015. http://page.mi.fu-berlin.de/mulzer/notes/misc/chernoff.pdf Accessed: 2016-05-10.

[Niv10] Gabriel Nivasch. Improved bounds and new techniques for Davenport–Schinzel sequences and their generalizations. *J. ACM*, 57(3), 2010.

[NS07] Giri Narasimhan and Michiel H. M. Smid. *Geometric spanner networks*. Cambridge University Press, 2007.

[OvL80] Mark H. Overmars and Jan van Leeuwen. Dynamically maintaining configurations in the plane (detailed abstract). In *Proc. 12th Annu. ACM Sympos. Theory Comput. (STOC)*, pages 135–145, 1980.

[OvL81] Mark H. Overmars and Jan van Leeuwen. Maintenance of configurations in the plane. *J. Comput. System Sci.*, 23(2):166–204, 1981.

[Păt11] Mihai Pătraşcu. Unifying the landscape of cell-probe lower bounds. *SIAM Journal on Computing*, 40(3):827–847, 2011.

[Phi16] Phillips. P6000 series Smart Ultra HDTV product description, 2016. http://www.usa.philips.com/c-p/55PFL6900_F7/6000-series-smart-ultra-hdtv/specifications Accessed: 2016-05-16.

[PR10] David Peleg and Liam Roditty. Localized spanner construction for ad hoc networks with variable transmission range. *ACM Transactions on Sensor Networks (TOSN)*, 7(3), 2010.

[PS85] Franco P. Preparata and Michael Ian Shamos. *Computational geometry. An introduction.* Springer-Verlag, 1985.

[PU89] David Peleg and Eli Upfal. A trade-off between space and efficiency for routing tables. *J. ACM*, 36(3):510–530, 1989.

[Ram99] Edgar A. Ramos. On range reporting, ray shooting and k-level construction. In *Proc. 15th Annu. Sympos. Comput. Geom. (SoCG)*, pages 390–399, 1999.

[RS11] Liam Roditty and Michael Segal. On bounded leg shortest paths problems. *Algorithmica*, 59(4):583–600, 2011.

[RT15] Liam Roditty and Roei Tov. New routing techniques and their applications. In *Proc. 34th ACM Symp. on Principles of Distributed Computing (PODC)*, pages 23–32, 2015.

[SA96] Micha Sharir and Pankaj K. Agarwal. *Davenport-Schinzel sequences and their geometric applications.* Cambridge University Press, 1996.

[Sch13] Ben Schwan. Multifunktionswaage mit wlan-anschluss, 2013. http://www.heise.de/tr/artikel/Multifunktionswaage-mit-WLAN-Anschluss-1952262.html Accessed: 2016-05-16.

[Sha85] Micha Sharir. Intersection and closest-pair problems for a set of planar discs. *SIAM J. Comput.*, 14(2):448–468, 1985.

[SK85] Nicola Santoro and Ramez Khatib. Labelling and implicit routing in networks. *Comput. J.*, 28(1):5–8, 1985.

[Son15] Sony. Playstation 4 product description, 2015. http://www.sony.de/electronics/playstation-systeme/playstation-4/specifications Accessed: 2016-05-16.

[Sto05] Ivan Stojmenovic. *Handbook of sensor networks: Algorithms and architectures*, volume 49. John Wiley & Sons, 2005.

[Tho04] Mikkel Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *J. ACM*, 51(6):993–1024, 2004.

[TZ01] Mikkel Thorup and Uri Zwick. Compact routing schemes. In *Proc. 13th ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, pages 1–10, 2001.

[Wil16] Andreas Wilkens. Ces 2016: Samsung macht kÃijhlschrank zum digitalen schwarzen brett, 2016. http://www.heise.de/newsticker/meldung/CES-2016-Samsung-macht-Kuehlschrank-zum-digitalen-Schwarzen-Brett-3063778.html Accessed: 2016-05-16.

[Yao82] Andrew Chi-Chih Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM J. Comput.*, 11(4):721–736, 1982.

[YXD12] Chenyu Yan, Yang Xiang, and Feodor F. Dragan. Compact and low delay routing labeling scheme for unit disk graphs. *Comput. Geom.*, 45(7):305–325, 2012.

Zusammenfassung

Sei $P \subset \mathbb{R}^2$ eine Menge von n Punkten. Der *Unit Disk Graph* von P, $UD(P)$, hat P als Knotenmenge. Zwei Knoten p, q bilden eine Kante in $UD(P)$ genau dann, wenn p und q euklidischen Abstand $|pq| \leqslant 1$ haben. Eine Verallgemeinerung von Unit Disk Graphen sind *Transmissionsgraphen*. Jeder Punkt $p \in P$ hat einen Radius r_p, und wir betrachten einen *gerichteten* Graphen G mit Knotenmenge P. Es existiert eine gerichtete Kante pq in G genau dann, wenn $|pq| \leqslant r_p$, also wenn q im Kreis um p mit Radius r_p liegt. Wir studieren die folgenden Probleme für Unit Disk und Transmissionsgraphen.

Dynamischer Zusammenhang. Wir beschreiben eine Datenstruktur zum Verwalten der Zusammenhangskomponenten eines Unit Disk Graphen $UD(P)$ wenn Punkte in P eingefügt oder gelöscht werden können. Die Datenstruktur kann zu jedem Zeitpunkt Zusammenhangsanfragen beantworten: Gegeben zwei Anfragepunkte $p, q \in P$, existiert ein Pfad von p zu q in $UD(P)$? Alle Operationen benötigen $O(\text{polylog}(n))$ Zeit.
Routing. Wir zeigen, dass es möglich ist, Datenpakete durch einen Unit Disk Graphen zu routen, indem in jedem Schritt der aktuelle Knoten *lokal* bestimmt, zu welchem seiner Nachbarn er das Paket weiter sendet. Das Routingschema besitzt dazu eine Menge von lokalen Informationen an jedem Knoten (Routingtabellen), ein Bezeichner an jedem Knoten, und einen veränderlichen Vorspann im Datenpaket. Die benötigte Größe ist jeweils $O(\text{polylog}(n))$. Wir zeigen, dass ein Datenpaket zwischen beliebigen Knoten p und q in $UD(P)$ geroutet werden kann und es dabei einem approximativ kürzesten Pfad von p nach q folgt.
Spanngraphen. Transmissionsgraphen können eine quadratische Anzahl an Kanten besitzen. Wir zeigen, dass wir zu jedem Transmissionsgraphen G einen aufspannenden Teilgraphen $H \subseteq G$ finden können, der nur eine lineare Anzahl von Kanten besitzt und gleichzeitig die kürzesten Pfade zwischen allen Paaren von Knoten in G approximiert. Der Teilgraph H wird als *Spanngraph* bezeichnet. Unser Algorithmus berechnet einen Spanngraph in Zeit $O(n \,\text{polylog}(n))$, wenn G als Punktmenge mit Radien gegeben ist.
Erreichbarkeitsorakel. Wir beschreiben drei Datenstrukturen, die Erreichbarkeitsanfragen in einem Transmissionsgraphen G beantworten können: gegeben zwei Anfragepunkte $p, q \in P$, existiert ein *gerichteter* Pfad von p zu q in G? Wir bezeichnen solche Datenstrukturen als *Erreichbarkeitsorakel*. Unsere Orakel hängen von dem Verhältnis Ψ des größten und des kleinsten Radius von Punkten in P ab. Wenn $\Psi < \sqrt{3}$ ist, dann können wir ein Orakel mit $O(1)$ Anfragezeit und $O(n)$ Platz konstruieren. Für größere Ψ präsentieren wir ein Orakel mit Anfragezeit $O(\Psi^3\sqrt{n})$ und Platzbedarf $O(\Psi^3 n^{3/2})$. Unser letztes Orakel hat nur noch logarithmische Abhängigkeit von Ψ. Die Anfragezeit ist $O(n^{2/3}\log^{1/3}\Psi\log^{2/3} n)$ und der Platzbedarf $O(n^{5/3}(\log\Psi+\log n)\log^{1/3}\Psi\log^{2/3} n)$.