Spanners and Reachability Oracles for Directed Transmission Graphs*

Haim Kaplan[1], Wolfgang Mulzer[2], Liam Roditty[3], and Paul Seiferth[2]

1 **School of Computer Science, Tel Aviv University, Israel**
haimk@post.tau.ac.il
2 **Institut für Informatik, Freie Universität Berlin, Germany**
{mulzer,pseiferth}@inf.fu-berlin.de
3 **Department of Computer Science, Bar Ilan University, Israel**
liamr@macs.biu.ac.il

Abstract

Let $P \subset \mathbb{R}^d$ be a set of n points, each with an *associated radius* $r_p > 0$. The *transmission graph* G for P has vertex set P and an edge from p to q if and only if q lies in the ball with radius r_p around p. Let $t > 1$. A *t-spanner* H for G is a sparse subgraph of G such that for any two vertices p, q connected by a path of length ℓ in G, there is a p-q-path of length at most $t\ell$ in H. We show how to compute a t-spanner for G if $d = 2$. The running time is $O(n(\log n + \log \Psi))$, where Ψ is the ratio of the largest and smallest radius of two points in P. We extend this construction to be independent of Ψ at the expense of a polylogarithmic overhead in the running time. As a first application, we prove a property of the t-spanner that allows us to find a BFS tree in G for any given start vertex $s \in P$ in the same time.

After that, we deal with *reachability oracles* for G. These are data structures that answer *reachability queries*: given two vertices, 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. For $d = 1$, we show how to compute an oracle with $Q(n) = O(1)$ and $S(n) = O(n)$ in time $O(n \log n)$. For $d = 2$, the radius ratio Ψ again turns out to be an important measure for the complexity of the problem. We present three different data structures whose quality depends on Ψ: (i) if $\Psi < \sqrt{3}$, we achieve $Q(n) = O(1)$ with $S(n) = O(n)$ and preproccesing time $O(n \log n)$; (ii) if $\Psi \geq \sqrt{3}$, we get $Q(n) = O(\Psi^3\sqrt{n})$ and $S(n) = O(\Psi^5 n^{3/2})$; and (iii) if Ψ is polynomially bounded in n, we use probabilistic methods to obtain an oracle with $Q(n) = O(n^{2/3} \log n)$ and $S(n) = O(n^{5/3} \log n)$ that answers queries correctly with high probability. We employ our t-spanner to achieve a fast preproccesing time of $O(\Psi^5 n^{3/2})$ and $O(n^{5/3} \log^2 n)$ in case (ii) and (iii), respectively.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems – Geometrical Problems and Computations

Keywords and phrases Transmission Graphs, Reachability Oracles, Spanner, Intersection Graph

Digital Object Identifier 10.4230/LIPIcs.SOCG.2015.156

1 Introduction

A common model for wireless sensor networks is the *unit-disk graph*: each sensor p is modeled by a unit disk centered at p, and there is an edge between two sensors iff their disks intersect [7]. Intersection graphs of disks with arbitrary radii have also been used to model

* This work is supported by GIF project 1161 & DFG project MU/3501/1.

31st International Symposium on Computational Geometry (SoCG'15).
Editors: Lars Arge and János Pach; pp. 156–170

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

sensors with different transmission radii [2, Chapter 4]. Intersection graphs of disks are undirected, however. For some networks we may want a directed model. In such networks, a sensor p that can transmit information to a sensor q may not be able to receive information from q. This motivated various researchers to consider what we call here *transmission graphs*[16, 15]. A transmission graph G is defined for a set of points $P \subset \mathbb{R}^2$ where each point $p \in P$ has a (transmission) radius r_p associated with it. Each vertex of G corresponds to a point of P, and there is a directed edge from p to q iff q lies in the disk $D(p)$ of radius r_p around p. We can weight each edge pq of G by its Euclidean length $|pq|$ and treat G as a weighted graph. We study (approximate) shortest path and reachability problems for transmission graphs.

Even though transmission graphs have a linear size representation, they may be very dense, even with $\Theta(n^2)$ edges (similar to many other geometric intersection graphs). Thus, if one applies a standard graph algorithm, like breadth first search (BFS), to a dense transmission graph, it runs slowly, since it requires an explicit representation of all the edges in the graph. Thus, given an transmission graph G implicitly as points with radii, it is desirable to construct a sparse approximation of G that preserves connectivity and proximity properties. For any $t > 1$, a subgraph H of G is a *t-spanner* for G if the distance between any pair of vertices p and q in H is at most t times the distance between p and q in G, i.e., $d_H(p, q) \leq t \cdot d_G(p, q)$ for any pair p, q (see [14] for an overview of spanners for geometric graphs). Fürer and Kasivisawnathan show how to compute a t-spanner for unit- and general disk graphs using a variant of the Yao graph [9, 17]. Peleg and Roditty [15] give a construction for t-spanners in transmission graphs in any metric space with bounded doubling dimension. However, except for the unit-disk case, the running times of these algorithms depend on the number of edges in the intersection graph. We avoid this dependency and give an almost linear time algorithm that constructs a t-spanner of a transmission graph for the Euclidean metric in the plane. Our construction is based on the *Yao graph*[17]. The basic Yao graph is a spanner for the complete graph defined by n points in the plane (with Euclidean distances). To determine the points adjacent to a particular point q, we divide the plane by equally spaced rays emanating from q and connect q to the closest point in each wedge (the number of wedges increases as t gets smaller). Transmission graphs, being directed, pose a severe computational difficulty as we want to consider, in each wedge, only the points p with $q \in D(p)$ and pick the closest to q only among those. Our spanner construction generalizes the Yao graph in this manner. We further need to relax this construction in a subtle way, without hurting the approximation too much, in order to construct the spanner efficiently. Even with a good approximation in terms of a 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 vertex [3]. 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.

A classical data structure problem for a directed graph G is to construct a space efficient *reachability oracle* that can answer *reachability queries* quickly. In a reachability query we get two vertices p and q and we would like to determine if there is a directed path from p to q. The quality of a reachability oracle for a graph G with n vertices is measured by three parameters: the query time $Q(n)$, the space requirement $S(n)$, and the preprocessing time. In the planar case, efficient reachability oracles exist and a recent result by Holm, Rotenberg and Thorup achieves optimal parameters [11]. However, for general directed graphs, there are no nontrivial results, and special cases, such as transmission graphs, are of great interest.

We give efficient constructions of reachability oracles for transmission graphs by exploiting their geometry. For points in 1D, we give an $O(n)$ space oracle with query time $O(1)$. In 2D it turns out that the ratio Ψ of the largest and smallest radius of points in P is an important complexity measure for transmission graphs. We give three oracles for different ranges of Ψ.

Our Contribution and Organization of the Paper. In Section 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 (i.e., the ratio between the largest and the smallest distance in P) is bounded, we can obtain a t-spanner in time $O(n(\log n + \log \Phi))$ (Section 2.1). With a little more work, we can weaken the assumption to a bounded *radius ratio* Ψ (the ratio between the largest and smallest radius in P), giving a running time of $O(n(\log n + \log \Psi))$ (Section 2.2). Using even more advanced data structures, we can compute a t-spanner in expected time $O(n \log^6 n)$, without any dependence on Φ or Ψ (Section 2.3). Our spanners have several applications. For one, we adapt a result by Cabello and Jejčič [3] to show that once a spanner is at hand, we can compute the BFS-tree of any given vertex $p \in P$ with additional time $O(n \log n)$ (Section 2.4). Furthermore, we use t-spanners to obtain efficient preprocessing algorithms for reachability oracles.

The remaining paper is dedicated to these reachability oracles. We will see that in 1D transmission graphs admit a rich structure that can be exploited to construct a simple linear space reachability oracle with constant query time and $O(n \log n)$ preprocessing time. This construction is described in Section 3. Unfortunately, in 2D most of their structure vanishes. However, if the radius ratio Ψ is less than $\sqrt{3}$, we show how to make the transmission graph planar in $O(n \log n)$ time, while preserving the reachability structure and keeping the number of vertices linear in n. Now we can construct a reachability oracle for the resulting planar graph. A recent construction of Holm, Rotenberg and Thorup [11] gives a distance oracle for planar graphs in linear time that takes linear space and can answer a query in $O(1)$ time. This construction is in Section 4.1. When $\Psi \geq \sqrt{3}$ we do not know how to planarize G. Fortunately, we can use a separation 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 [1]. This allows us to build a reachability oracle with query time $O(\Psi^3\sqrt{n})$ and space and preprocessing time $O(\Psi^5 n^{3/2})$. See Section 4.2. When Ψ is even larger but still polynomially bounded in n, we use random sampling combined with a quad tree of logarithmic depth to obtain a reachability oracle with query time $O(n^{2/3} \log n)$, space $O(n^{5/3} \log n)$, and preprocessing time $O(n^{5/3} \log^2 n)$. Refer to Section 4.3.

Many of our constructions rely on planar grids. For $i = 0, 1, \ldots$, we define $\mathcal{Q}_i$ 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*). $\mathcal{Q}_i$ is aligned so that the origin is a vertex of the grid. The *distance* between two grid cells is the smallest distance of any two points contained in them. Furthermore, we assume that the input is scaled so that the distance of the closest pair in P is 1. We assume that in our model of computation we can find for any given point the grid cell that contains it in $O(1)$ time. For space reasons, all proofs in this extended abstract are omitted. We refer the interested and ambitious reader to the full version.

2 Spanners and BFS Trees

2.1 Efficient Spanner Construction

Let $P \subset \mathbb{R}^2$ be a point set with radii, and let $\Phi = \max_{p,q \in P} |pq| / \min_{p \neq q \in P} |pq|$ be the *spread* of P. First, we give a spanner construction for the transmission graph G of P that depends on the spread of P. In Section 2.2, we will weaken this to a dependence on the radius ratio.

▶ **Theorem 2.1.** *Let G be the transmission graph for a two-dimensional n-point set P with spread Φ. For any $t > 1$, we can compute a t-spanner for G in time $O(n(\log n + \log \Phi))$.*

Our construction creates a subgraph H of G that is similar to the Yao graph [17], but modified to take the disks into account. Ideally, our spanner should look as follows: we pick a suitable integer k, and we take a set $\mathcal{C}$ of k cones with opening angle $2\pi/k$ that partition the plane and that have the origin as apex. For each vertex $q \in P$, we attach the cones in $\mathcal{C}$ to q, and in each translated cone we pick the closest vertex $p \in P$ with $q \in D(p)$. We add the edge pq to H. The resulting graph has $O(kn)$ edges, and using standard techniques, one can show that it is a t-spanner for large enough k. This construction seems to be folklore [5, 15].

However, the standard algorithms for computing the Yao graph do not seem to adapt easily for our setting without affecting the running time. Thus, we need a more sophisticated construction that gives a graph with similar properties. The idea is to partition each cone C_q attached to q into "intervals" obtained by intersecting C_q with annuli centered at q whose inner and outer radius grows exponentially; see Figure 1. Each of these intervals is discretized by covering it with $O(1)$ grid cells whose diameter is relatively small compared to the distance between the interval and q. This enforces two properties that help us to find an approximately shortest incoming edge for q in C_q: we only need to consider edges from the interval that is closest to q since these edges will be shorter than any edge from any later interval; and if there are multiple edges from the same grid cell to q, it suffices to pick only one of them since their endpoints are close together.

We define a decomposition of P that represents the discretized intervals by a neighborhood relation between grid cells. Given this decomposition, there is a simple (rather inefficient) rule how to pick incoming edges for each $q \in P$ such that the resulting graph H is a spanner. We first give the definition of the decomposition and prove that H is a t-spanner if we pick the parameters appropriately. Then we show how compute the decomposition using a quadtree T. Finally, we use the structure of T to find the edges within the desired time bound. Let $c > 2$ be a large constant. For a grid cell σ, let m_σ be the point in $P \cap \sigma$ with the largest radius.

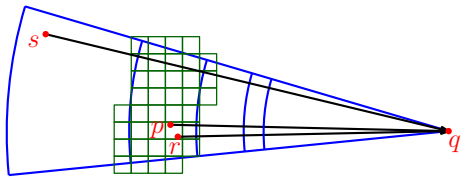

Figure 1 A cone C_q covered by discretized intervals. We only need one of the edges $\overrightarrow{pq}$, $\overrightarrow{rq}$ for H.

▶ **Definition 2.2.** Let G be the transmission graph of a point set $P \subset \mathbb{R}^2$. A *c-separated annulus decomposition* for G consists of a finite set $\mathcal{Q} \subset \bigcup_{i=0}^{\infty} \mathcal{Q}_i$, a symmetric neighborhood relation $N \subseteq \mathcal{Q} \times \mathcal{Q}$, and assigned sets $R_\sigma \subseteq P \cap \sigma$ for each $\sigma \in \mathcal{Q}$ so that (i) for all $(\sigma, \sigma') \in N$, $\operatorname{diam}(\sigma) = \operatorname{diam}(\sigma')$ and $d(\sigma, \sigma') \in [(c-2)\operatorname{diam}(\sigma), 2c\operatorname{diam}(\sigma))$; and (ii) for every edge $\overrightarrow{pq}$ of G, there is a $(\sigma, \sigma') \in N$ with $p \in \sigma$, $q \in \sigma'$, and with either $p \in R_\sigma$ or $q \in D(m_\sigma)$.

For $\sigma \in \mathcal{Q}$, we define $N(\sigma) = \{\sigma' \mid (\sigma, \sigma') \in N\}$. Definition 2.2(i) implies $|N(\sigma)| = O(1)$.

Getting a Spanner. Let $t > 1$ be the desired stretch. Depending on t, we pick suitable constants c (separation parameter) and k (number of cones). Let $\mathcal{Q}$ be a c-separated annulus decomposition for G. To obtain a t-spanner $H \subseteq G$, we pick the incoming edges for each point $q \in P$ and each cone $C \in \mathcal{C}$ as in Alg. 1. For $\sigma \in \mathcal{Q}$ let C_σ be the translated copy of C that has the center of σ as apex and let C_σ^2 be the cone obtained by doubling the opening angle of C_σ. Instead of C_q we use the cones C_σ^2 with $q \in \sigma$ to find incoming edges for q. This gives the generality needed for later extensions of this algorithm.

```
1 Q_q ← cells of Q that contain q
2 Sort Q_q by the diameter of the cells in increasing order; give q the status active
3 while q is active do
4 |   σ ← next largest cell in Q_q
5 |   foreach cell σ' ∈ N(σ) that is contained in C_σ^2 do
6 |   |   if there is a r ∈ R_σ' ∪ {m_σ'} with q ∈ D(r) then
7 |   |   |   take an arbitrary such r, add the edge rq to H, and set q to be inactive.
```

Algorithm 1: Selecting the incoming edges for q and the cone C_q.

For each cone $C \in \mathcal{C}$ and each $q \in P$ there is only one $\sigma \in \mathcal{Q}_q$ that produces incoming edges for q: after σ is processed, q is inactive. Since we have k cones and since $|N(\sigma)| = O(1)$, q has $O(k)$ incoming edges, for a total of $O(n)$ edges in H. To show that H is a t-spanner, we use induction on the rank of the edge lengths. The proof is done in a similar manner as for standard Yao graphs, but with a few additional twists.

▶ **Lemma 2.3.** *For any $t > 1$, there are constants c and k such that H is a t-spanner for G.*

Finding the Decomposition. Let $c > 3$ be the separation parameter. We scale P s.t. the smallest distance in P is c. A *quadtree* for P is a rooted tree T in which each internal node has degree four. Each node v of T has an associated cell σ_v from a grid $\mathcal{Q}_i$, $i \geq 0$, and we say that v has *level* i. If v is an internal node, the cells of its four children partition σ_v into four congruent squares, each with half the diameter of σ_v. We describe how to compute a quadtree T for P s.t. the cells of T form the set $\mathcal{Q}$ for the c-separated annulus decomposition.

We construct T level-wise. Let L be the smallest integer such that there is a cell $\sigma \in \mathcal{Q}_L$ that (possibly after shifting) contains P. Since c is constant and since P has spread Φ, the scaled point set has diameter $c\Phi$, and we can take $L = O(\log \Phi)$. We create the root node v and set $\sigma_v = \sigma$. This will be level L. To construct level $i-1$, given level i, we do the following for each level i node v whose cell σ_v is non-empty: take the four cells of $\mathcal{Q}_{i-1}$ that partition σ_v and create four nodes $w_1, \ldots, w_4$. To each of the four nodes $w_1, \ldots, w_4$ we assign one of the four cells, and we make $w_1, \ldots, w_4$ children of v. This process stops at level 0. The scaling of P ensures that a cell of a level 0 node contains at most one point of P.

We now set $\mathcal{Q} = \{\sigma_v \mid v \in T\}$. We let $(\sigma_v, \sigma_w) \in N$ if v and w have the same level and if $d(\sigma_v, \sigma_w) \in [(c-2)\operatorname{diam}(\sigma_v), 2c\operatorname{diam}(\sigma_v))$. As R_{σ_v} we take all points in $\sigma_v \cap P$ whose radius is between $(c-2)\operatorname{diam}(\sigma_v)$ and $2(c+1)\operatorname{diam}(\sigma_v)$. To see that this satisfies Definition 2.2(ii), consider an edge pq of G with $q \in \sigma_v$ and $p \in \sigma' \in N(\sigma_v)$. Since $D(p)$ must intersect σ_v, we have $r_p \geq (c-2)\operatorname{diam}(\sigma_v)$. Thus, we have either $p \in R_{\sigma'}$ or $r_p > 2(c+1)\operatorname{diam}(\sigma')$. In the second case for any $r \in \sigma'$ with radius $r_r \geq r_p$ the disk $D(r)$ fully contains σ_v. In particular this holds for $r = m_{\sigma'}$. Since Def. 2.2(i) is satisfied by construction, we get the next lemma.

▶ **Lemma 2.4.** *The set $\mathcal{Q}$ with N and the assignment R_σ described above is a c-separated annulus decomposition for G.*

Finding the Edges. To find the edges for the spanner H more quickly, we use the cells of $\mathcal{Q}$ to group the points and find incoming edges for multiple points at once. We process the cells of $\mathcal{Q}$ by increasing diameter, following the structure of the quadtree T.

Fix one cone $C \in \mathcal{C}$ of the k cones we want to process. For $\sigma \in \mathcal{Q}$, let C_σ^2 be the cone with opening angle $4\pi/k$ whose apex is the center of σ obtained by translating C and doubling its opening angle. We give all points in P the status *active*. We process T in level-order,

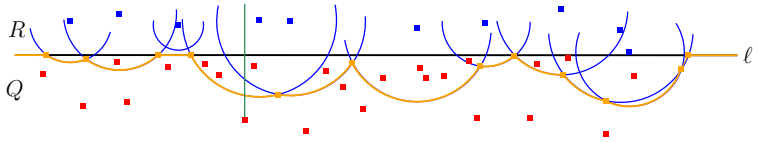

Figure 2 The lower envelope (orange), the points Q (red) and R (blue), and the sweepline (green).

starting with level 0. For each $v \in T$, we select incoming edges for the active points Q in $\sigma_v \cap P$ as in Algorithm 2. First we sort Q by x and y-direction in linear time, using the sorted lists of v's children (preproccesing). Let $\sigma' \in N(\sigma_v)$ be a neighbor of σ_v. The sorting enables us to efficiently find incoming edges for points in Q from points in $R = R_{\sigma'} \cup \{m_{\sigma'}\}$ (edge selection): Q and R are separated by a line ℓ that is parallel to either the x or the y axis, namely one of the supporting lines of the boundary of σ_v. We can compute the lower envelope E of the disks in R and sweep over Q in ℓ direction, see Fig. 2. This takes time linear in $|Q|$ since Q is sorted in ℓ direction. To check whether the current point $q \in Q$ is contained in a disk of R, we only need to test the disk of the arc of E intersected by the sweepline through q orthogonal to ℓ. We summarize the above discussion in Lemma 2.5.

▶ **Lemma 2.5.** *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 time $O(m \log m + n)$ for each $q \in Q$ one disk from R that contains it, provided that such a disk exists.*

The edges selected by Algorithm 2 have the same properties as the edges selected by Algorithm 1. Thus, by Lemma 2.3, the resulting graph is a t-spanner.

```
1 for i = 0, ..., L do
2     foreach v ∈ T of level i do
3         Q ← active points in σ_v ∩ P
          // preproccesing
4         Sort Q in x and y-direction by merging the sorted lists of the children of v
          foreach σ' ∈ N(σ_v) contained in C^2_{σ_v} do
5             R ← R_{σ'} ∪ {m'_σ}
              // edge selection
6             For each q ∈ Q find a r ∈ R with q ∈ D(r), if it exists; add the edge r→q to H
7         Set all q ∈ Q for which at least one incoming edge was found to inactive
```

Algorithm 2: Selecting the edges for H for a fixed cone C.

Running Time. By Lemma 2.5, we can argue that the running time of Algorithm 2 is dominated by the edge selection step. Since T has depth $O(\log \Phi)$, each $p \in P$ takes part in $O(\log \Phi)$ edge selections as a point in Q for incoming edges, taking $O(1)$ time for that point (by Lemma 2.5). Furthermore, each point is in $O(1)$ different sets R_σ and thus takes part in $O(1)$ edges selections as a disk-center in R, taking $O(\log |R|) = O(\log n)$ time for that point. Thus, we have a total running time of $O(n(\log \Phi + \log n))$, as stated in the next lemma.

▶ **Lemma 2.6.** *The construction of the spanner H of G takes $O(n(\log \Phi + \log n))$ time.*

Theorem 2.1 follows by combining Lemmas 2.3 and 2.6.

2.2 From Bounded Spread to Bounded Radius Ratio

Let $P \subset \mathbb{R}^2$ be a point set with radii and let Ψ be the radius ratio of P. We extend the spanner construction from Section 2.1 to be depended on the radius ration Ψ of P.

▶ **Theorem 2.7.** *Let G be the transmission graph for a n-point set $P \subset \mathbb{R}^2$ with radius ratio Ψ. For any $t > 1$, we can compute a t-spanner for G in $O(n(\log n + \log \Psi))$ time.*

The main observation is that the spread is irrelevant in our setting: points that are close together form a clique in G and can be handled through classic spanners, and points that are far away from each other form distinct components and can be dealt with independently.

Given t, we pick large enough constants k and c. Then, we scale the input such that the smallest radius is c. Let $M = O(\Psi)$ be the largest radius. First, we partition P into sets that are far away from each other and can be handled separately.

▶ **Lemma 2.8.** *In $O(n \log n)$ time, we can partition P into sets $P_1, \ldots, P_\ell$ so that each P_i has diameter $O(n\Psi)$ and so that for any $i \neq j$, no point in P_i can reach a point of P_j in G.*

By Lemma 2.8, we may assume that our input point set has diameter $O(n\Psi)$. As in Section 2.1, we can compute a quadtree T for P with L levels and $L = O(\log(n\Psi))$: take a large enough grid cell that contains P and recursively subdivide each non-empty cell into four cells of half the diameter. We stop when the diameter of the cells is 1. Unlike in Section 2.1, the set of the cells of all nodes of T does not yield a c-separated annulus decomposition for G. In particular, Definition 2.2(ii) is not true anymore. Therefore, there can be edges in G that do not go between neighboring cells. These are the *short edges*.

First, we handle *very short edges*: let $v \in T$ be a level 0 node and let $\sigma_v \in \mathcal{Q}_0$ be the cell of v. Let $Q \subseteq P$ be all points that lie in cells of $\mathcal{Q}_0$ with distance at most $c/2 - 3$ from σ. Since any pair of points in Q has distance at most c, the set Q forms a clique in G. We compute a (classic) t-spanner for Q in $O(|Q| \log |Q|)$ time [14]. Since any $p \in P$ participates in $O(c^2)$ such spanners, we generate $O(n)$ edges in total and require $O(n \log n)$ time.

Second, we handle *not quite so short edges*: for each $q \in P$, let v be the level 0 node of T whose cell σ_v contains q. For any cell $\sigma \in \mathcal{Q}_0$ with $d(\sigma_v, \sigma) \in (c/2 - 3, c - 2)$, we take an arbitrary point $r \in \sigma \cap P$ and add the edge $\overrightarrow{rq}$ to our spanner. All these edges have length at most c and are therefore valid edges in G. This takes $O(n)$ time and creates $O(n)$ edges.

Finally, we handle the remaining edges: for this, we mark all points of P as active, and we run Algorithm 2 from Section 2.1 starting from level 0 of T. Call the resulting graph H.

As in Lemma 2.3, induction on the rank of the edges lengths shows that H is a t-spanner.

▶ **Lemma 2.9.** *The graph H is a t-spanner for G if c and k are large enough constants.*

Using Lemma 2.9, Theorem 2.7 follows in the same way as in Section 2.1. The running time analysis goes exactly as in Lemma 2.6, but the quadtree now has $O(\log n + \log \Psi)$ levels.

2.3 Spanners for Unbounded Spread and Radius Ratio

We show how eliminate the bounded radius ratio assumption at the expense of using a more involved data structure and of losing a polylog factor in the running time. Let $P \subset \mathbb{R}^2$ and the desired stretch factor $t > 1$ be given. Assume that the closest pair in P has distance 1.

First we compute a *compressed quadtree* T for P. It is a rooted tree in which each internal node has degree 1 or 4. Each node v has an associated cell σ_v from a grid $\mathcal{Q}_i$. To keep the notation simple, we write $\operatorname{diam}(v)$ for $\operatorname{diam}(\sigma_v) = 2^i$, and for two nodes v, w, we write $d(v, w)$ for $d(\sigma_v, \sigma_w)$. If v has degree 4, then the associated cells of its children partition

σ_v into 4 congruent squares of half the diameter, and at least two of them are non-empty. If v has degree 1, then the associated cell of the only child w of v has diameter at most $\operatorname{diam}(v)/4$. Furthermore, there are no points from P in $\sigma_v \setminus \sigma_w$. Each internal node of T contains at least 2 points from P in its cell and each leaf at most 1 point. A compressed quadtree for P with $O(n)$ nodes can be computed in $O(n \log n)$ time [10].

Our goal is to use the algorithm from Section 2.1 on the compressed quadtree T. There are two problems with this: since the depth of T can be linear, we cannot consider all points for incoming edges in each level, as in Algorithm 2. Instead we use Chan's dynamic nearest neighbor data structure to quickly identify the relevant points. It has the following properties.

▶ **Theorem 2.10** (Chan [6]). *There exists a dynamic data structure that maintains a planar point set S such that* (i) *we can insert a point into S in expected, amortized time $O(\log^3 n)$;* (ii) *we can delete a point from S in expected, amortized time $O(\log^6 n)$; and* (iii) *given a query point q, we can find the nearest neighbor for q in S in worst-case time $O(\log^2 n)$.*

Furthermore, the cells of T do not form a c-separated annulus decomposition anymore. The notion of neighborhood needs to be adapted to accommodate internal nodes of degree 1 and to ensure that Definition 2.2(ii) holds. We fix this by inserting $O(n)$ additional nodes into T that have the desired properties. To find these nodes, we use the well-separated pair decomposition algorithm of Callahan and Kosaraju [4]. Let a large enough constant c be given. As in Section 2.1, we define the neighborhood relation N as the pairs (σ_v, σ_w) whose nodes v and w have the same level in T and that satisfy $d(\sigma_v, \sigma_w) \in [(c-2)\operatorname{diam}(\sigma_v), 2c \operatorname{diam}(\sigma_v))$. The set R_{σ_v} are all points in $\sigma_v \cap P$ whose radius is between $(c-2)\operatorname{diam}(\sigma_v)$ and $2(c+1)\operatorname{diam}(\sigma_v)$.

▶ **Lemma 2.11.** *For any $c > 0$ we can in $O(n \log n)$ time insert $O(n)$ nodes into the compressed quadtree T s.t. $\mathcal{Q} = \{\sigma_v \mid v \in T\}$ with N and the assignment R_σ is a c-separated annulus decomposition for G. In the same time we can compute N and R_{σ_v} for each $\sigma_v \in \mathcal{Q}$.*

Finding the Edges. To find the edges for the spanner $H \subseteq G$, we choose constants k and c depending on t. The algorithm proceeds as follows: we compute a compressed quadtree T for P. To obtain a c-separated annulus decomposition $\mathcal{Q}$, N, R_{σ_v} for G, we augment T with $O(n)$ nodes as in Lemma 2.11. We create the dynamic nearest neighbor (NN) data structure from Theorem 2.10 for each leaf node v of T whose cell σ_v is non-empty. We sort all cells of nodes of T by increasing diameter. A point is called *active* if it is in the NN data structure of some v, thus initially all points of P are active. Fix a cone C. For $\sigma \in \mathcal{Q}$ let C^2_σ be the cone whose apex is the center of σ and such that C^2_σ is obtained from C by translating and doubling the opening angle to $4\pi/k$. To select the spanner edges for C, we consider the nodes of T in increasing order and perform two steps for each node v, similar to Algorithm 2 of Section 2.1: let w be the child of v that has the most active points in its NN structure. To get the NN data structure for v, we insert all active points of the remaining children of v into the NN data structure of w (preproccesing). Since w has the most points, overall each point is inserted $O(\log n)$ times in some NN structure. Then we do the edge selection for all $\sigma' \in N(\sigma_v)$ contained in $C^2_{\sigma_v}$ using the NN structure of v; see Algorithm 3. We take each point $r \in R_{\sigma'} \cup \{m_{\sigma'}\}$ and repeatedly query the NN structure of v. Let q be the result. If rq constitutes an edge in G, we call the query *successful*, add rq to H, delete q, and do another query with r. Otherwise, we proceed with the next point of R. Each such query causes $O(1)$ additional insertion/deletions to a NN structure. If it was successful, we charge these costs to the created edge. Otherwise, we charge the costs to this point r. Since each point $p \in P$ is in $O(1)$ sets R_σ, it can only be responsible for $O(1)$ unsuccessful queries. Thus, since H has n vertices and $O(n)$ edges, we can to prove the next lemma.

▶ **Lemma 2.12.** *The algorithm has total expected running time $O(n \log^6 n)$.*

The edges selected by this procedure have the same properties as the edges selected by Algorithm 1. Thus, by Lemma 2.3 we obtain a t-spanner H, which establishes Theorem 2.10.

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

Algorithm 3: Selecting incoming edges for the points of a node v of T and a cone C.

2.4 From Spanners to BFS Trees

We show how to compute BFS trees for a transmission graph G. Let the desired root $s \in P$ be given. We apply a technique that Cabello and Jejčič used for unit-disk graphs [3]. Denote by $d_h(s, p)$ the BFS distance from s to p in G. For $i \in \mathbb{N}_0$ let $W_i \subseteq P$ be the vertices $p \in P$ with $d_h(s, p) = i$. Cabello and Jejčič used the Delaunay triangulation (DT) to efficiently identify W_{i+1}, given $W_0, \ldots, W_i$. Our t-spanner provides similar properties for transmission graphs as the DT does for unit-disk graphs.

▶ **Lemma 2.13.** *Let H be the t-spanner for G from Theorem 2.7 for t small enough, and let $v \in W_{i+1}$, for some $i \geq 0$. Then there is a vertex $u \in W_i$ and a path $u = q_l, \ldots, q_1 = v$ in H with $d_h(s, q_j) = i + 1$ for $j = l - 1, \ldots, 1$.*

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

Algorithm 4: Compute the BFS tree for G with root s using H.

The BFS tree for s is computed iteratively; see Alg. 4 for pseudocode. Initially, we set $W_0 = \{s\}$. Now assume we computed everything up to W_i. By Lemma 2.13, all vertices 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} as follows: for each $u \in W_i$, start a BFS search in H from u. Every time we encounter a new vertex q, we check if it lies in a disk of W_i. If so, we add q to W_{i+1} and add the new neighbors of q to the queue. Otherwise, we discard q for now. To test whether q lies in a disk of W_i, we use the *power diagram*. This weighted version of the Voronoi Diagram represents the union of the W_i-disks as a planar subdivision. It takes $O(|W_i| \log |W_i|)$ time to compute, and augmented with a point location structure it supports the following queries in time $O(\log |W_i|)$: given a point q, find a disk in W_i that contains it, if it exists [12, 13].

Each edge pq of H is considered at most twice by Alg. 4, and each time we query a power diagram with q (in $O(\log n)$ time). Since H is sparse, the total time is $O(n \log n)$.

3 Reachability Oracles for 1-dimensional Graphs

In this section we prove the following theorem.

▶ **Theorem 3.1.** *Let G be the transmission graph of an n-point set $P \subset \mathbb{R}$. We can construct in $O(n \log n)$ time a reachability oracle for G with $S(n) = O(n)$ and $Q(n) = O(1)$.*

Let $\mathcal{C}$ be the set of strongly connected components (SCCs) of G and let $C \in \mathcal{C}$. We say that C can *reach* a point $p \in P$ if there is a path in G from a point in C to p. We say that C can reach an SCC $D \in \mathcal{C}$ if C can reach a point in D. By strong connectivity, this means that all points in C can reach all points in D. Next, we define several points related to C: the *leftmost point* of C, $\mathrm{l}(C)$, is the point in C with the smallest x-coordinate; the *left reachpoint* of C, $\mathrm{lr}(C)$, is the leftmost point in $\mathbb{R}$ that lies in a ball around a point in P reachable from C; and the *direct left reachpoint* of C, $\mathrm{dl}(C)$, is the leftmost point in $\mathbb{R}$ that lies in a ball around a point in C, i.e., $\mathrm{dl}(C) = \min_{p \in C}(p - r_p)$. The right versions $\mathrm{r}(C)$, $\mathrm{rr}(C)$, and $\mathrm{dr}(C)$ are defined analogously. The *interval* of C, I_C, is defined as $I_C = [\mathrm{l}(C), \mathrm{r}(C)]$.

▶ **Lemma 3.2.** *Let $C \in \mathcal{C}$ be an SCC, and let $p \in C$ a point in C. For any $q \in P$, there is a path in G from p to q if and only if $q \in [\mathrm{lr}(C), \mathrm{rr}(C)]$.*

Lemma 3.2 suggests the following reachability oracle with $O(n)$ space and $O(1)$ query time: for each $C \in \mathcal{C}$, store the reachpoints $\mathrm{lr}(C)$ and $\mathrm{rr}(C)$; and for each point $p \in P$, store the SCC of G that contains it. Given two query points p, q, we look up the SCC C for p, and we return YES iff $q \in [\mathrm{lr}(C), \mathrm{rr}(C)]$. It remains to describe an efficient preprocessing algorithm. To find the reachpoints quickly, we investigate the structure of the SCCs in G.

▶ **Lemma 3.3.** *The intervals $\{I_C \mid C \in \mathcal{C}\}$ for the SCCs form a laminar set family, i.e., for any $C, D \in \mathcal{C}$, we have either $I_C \cap I_D = \emptyset$, $I_C \subseteq I_D$, or $I_D \subseteq I_C$.*

By Lemma 3.3, we can obtain a forest with vertex set $\mathcal{C}$ by considering the set containment relation on the intervals $\{I_C \mid C \in \mathcal{C}\}$. If necessary, we add a common root node to get a tree T. The next lemma characterizes the left and right reachpoints.

▶ **Lemma 3.4.** *Let $C \in \mathcal{C}$. The left reachpoint $\mathrm{lr}(C)$ of C is either $\mathrm{dl}(C)$ or $\mathrm{dl}(D)$, where D is a sibling of C in T. The situation for the right reachpoints is analogous.*

Reachability Between Siblings. By Lemma 3.4, for an SCC $C \in \mathcal{C}$, it suffices to search for $\mathrm{lr}(C)$ and $\mathrm{rr}(C)$ among the siblings of C in T. Let $C_1, \ldots, C_k$ be the children of a node in T, sorted from left to right according to their intervals. To compute the left reachpoints of $C_1, \ldots, C_k$, we set $\mathrm{lr}(C_1) = \mathrm{dl}(C_1)$ and we push C_1 onto an empty stack S. Then we go

through $C_2, \ldots, C_k$, from left to right. For the current child C_i, we initalize the tentative left reachpoint $\mathrm{lr}(C_i) = \mathrm{dl}(C_i)$. While the current tentative reachpoint lies to the left of the right interval endpoint for the top of the stack, we pop the stack and we update the tentative reachpoint of C_i to the left reachpoint of the popped component, if it lies further to the left. Then we push C_i onto the stack and proceed to the next child; see Algorithm 5.

```
1 lr(C_1) ← dr(C_1); push C_1 onto an empty stack S
2 for i ← 2 to k do
3 |   lr(C_i) ← dr(C_i)
4 |   while S ≠ ∅ and lr(C_i) ≤ r(top(S)) do
5 |   |   D ← pop(S); lr(C_i) ← min{lr(C_i), lr(D)}
6 |   push C_i onto S
```

Algorithm 5: Computing left reachpoints.

The right reachpoints are computed analogously. Since each SCC is pushed/popped at most once onto/from S, and sorting the SSCs needs $O(n \log n)$ time, we get the following lemma.

▶ **Lemma 3.5.** *We can compute the reachability for all nodes in T in $O(n \log n)$ time.*

It remains to find the SCCs without explicitly constructing G. To do so, we can use the Kosaraju-Sharir algorithm [8] together with geometric data structures that allow us to efficiently find unvisited edges. See the full version for details. This establishes Theorem 3.1.

4 Reachability Oracles for 2-dimensional Graphs

4.1 Ψ is less than $\sqrt{3}$

Suppose that $\Psi \in [1, \sqrt{3})$. We show that we can *planarize* G 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 planar graphs. We prove the following lemma.

▶ **Lemma 4.1.** *Let G be the transmission graph for a planar n-point set P, such that $\Psi < \sqrt{3}$. In time $O(n \log n)$, we can find a plane graph $H = (V, E)$ s.t.* (i) *$|V| = O(n)$ and $|E| = O(n)$;* (ii) *$P \subseteq V$;* (iii) *for any $p, q \in P$, p can reach q in G iff p can reach q in H.*

Given Lemma 4.1, we can obtain the following result by constructing the distance oracle from Holm, Rotenberg and Thorup for H[11]. It has $O(1)$ query time and needs $O(n)$ space.

▶ **Theorem 4.2.** *Let G be the transmission graph for a two-dimensional set P of n points and let Ψ be the ratio between the largest and smallest radius in P. If $\Psi < \sqrt{3}$, we can construct in $O(n \log n)$ time a reachability oracle for G with $S(n) = O(n)$ and $Q(n) = O(1)$.*

We prove Lemma 4.1 in three steps. First, we show how to reduce the number of edges in G to $O(n)$ without changing the reachability. Then we show how to remove the crossings from G. Finally, we argue that we can combine these two operations to get the desired result.

Pruning the Graph. We construct a sparse subgraph $H \subseteq G$ with the same reachability as G but with $O(n)$ edge crossings. Consider the grid $\mathcal{Q}_0$ whose cells have side length $1/\sqrt{2}$. Let $\sigma \in \mathcal{Q}_0$ 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 $\mathcal{Q}_0$ with σ at the center. Two grid cells are *neighboring* if they lie in each other's neighborhood. For any edge in G, its two endpoints must lie in neighboring grid cells. We assign each point in P to the cell of $\mathcal{Q}_0$ that contains it. The subgraph H has P as vertex set, and we pick the edges as follows: for each non-empty cell σ, let $P_\sigma \subseteq P$ be the points in σ. We compute the Euclidean minimum spanning tree (EMST) T_σ of P_σ, and for each edge pq of T_σ, we add the directed edges pq and qp to H. Then, for cell $\sigma' \in N(\sigma)$, we check if there are any edges from σ to σ' in G. If so, we add an arbitrary such edge to H. The following lemma states properties of H.

▶ **Lemma 4.3.** *The graph H **a)** has the same reachability as G; **b)** has $O(n)$ edges; **c)** can be constructed in $O(n \log n)$ time; and **d)** has $O(n)$ edge crossings if it is drawn in the plane with vertex set P.*

Removing the Crossings. Suppose an edge pq of G and an edge uv of G cross at a point x. To eliminate the crossing, we add x as a new site 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. 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. We can show that the reachability on the vertices of G stays the same in $\widetilde{G}$. Intuitively speaking, the $\Psi < 3$ restriction forces the vertices to be close together. This guarantees the existence of additional edges between p, q, u, v in G and these edges are always sufficient to cover all new paths introduced by resolving the crossing.

▶ **Lemma 4.4.** *For any two sites $p, q \in P$, if p can reach q in $\widetilde{G}$ then p can reach q in G.*

Putting it together. To prove Lemma 4.1, we first construct the sparse subgraph H as in Lemma 4.3 in time $O(n \log n)$. Then we iteratively resolve all crossings in H to obtain $\widetilde{H}$. Since H has $O(n)$ crossings that can be found in the same 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 4.4. 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 $\widetilde{H}$, and 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.

4.2 Ψ is constant

Our goal is to prove the following theorem:

▶ **Theorem 4.5.** *Let G be the transmission graph for an n-point set $P \subset \mathbb{R}^2$ and let Ψ be the ratio between the largest and smallest radius of the points in P. We can construct a reachability oracle for G with $S(n) = O(\Psi^5 n^{3/2})$ and $Q(n) = O(\Psi^3 \sqrt{n})$ in time $O(\Psi^5 n^{3/2})$.*

Let $\mathcal{D}$ be the disks induced by P. Let $\mu(\mathcal{D})$ be the area occupied by $\bigcup \mathcal{D} := \bigcup_{D \in \mathcal{D}} D$. Alber and Fiala show how to compute a separator for disks with respect to $\mu(\cdot)$ [1].

▶ **Theorem 4.6** (Theorem 4.12 in [1])**.** *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) $\mu(\mathcal{S}) \leq \Psi^2 \beta \sqrt{\mu(\mathcal{D})}$ *and* (iii) $\mu(\mathcal{A}), \mu(B) \leq \alpha \mu(\mathcal{D})$.

To obtain the data structure, consider the grid $\mathcal{Q} = \mathcal{Q}_0$ whose cells have diameter 1. All vertices in one cell form a clique in G, so it suffices to determine the reachability for one of them. For each non-empty cell $\sigma \in \mathcal{Q}$ we pick an arbitrary vertex as the *representative* of σ. Let $P_{\mathcal{D}}$ be the set of all representatives for $\mathcal{D}$. We recursively create a separator tree T that contains all needed reachability information: compute $\mathcal{A}, \mathcal{B}$, and $\mathcal{S}$ according to Theorem 4.6. We create a node v of the separator tree. Let Q_v be all cells in $\mathcal{Q}$ that intersect $\bigcup \mathcal{S}$, and let P_v be their representatives and $\mathcal{D}_v$ all disks with centers in Q_v. For each $s \in P_v$, we store all representatives of $P_{\mathcal{D}}$ that s can reach and all the representatives that can be reached by s in the transmission graph induced by $\mathcal{D}$ (this graph is a subgraph of G). We recursively compute separator trees for $\mathcal{A} \setminus \mathcal{D}_v$ and $\mathcal{B} \setminus \mathcal{D}_v$, and we connect them to v.

For the space requirement, we can show that $|P_{\mathcal{D}}| = O(\mu(\mathcal{D}))$ for any set of disks $\mathcal{D}$.

▶ **Lemma 4.7.** *Let $\mathcal{D}$ be a set of n disks with radius at least 1. Then the number of cells in $\mathcal{Q}_0$ that intersect $\bigcup \mathcal{D}$ is $O(\mu(\mathcal{D}))$.*

Then, the space requirement $S(\mu(\mathcal{D}))$ for a set of disks $\mathcal{D}$ with respect to $\mu(\cdot)$ is

$$S(\mu(\mathcal{D})) = S((1-\alpha)\mu(\mathcal{D})) + S(\alpha\mu(\mathcal{D})) + O(\Psi^2 \mu(\mathcal{D})^{3/2}), \tag{1}$$

where the last term accounts for storing reachability between the $O(\Psi^2\sqrt{\mu(\mathcal{D})})$ vertices of $P_{\mathcal{S}}$ and the $O(\mu(\mathcal{D})$ vertices of $P_{\mathcal{A}} \cup P_{\mathcal{B}}$. For $\mu(\mathcal{D}) = O(1)$, we have $S(\mu(\mathcal{D})) = O(1)$, and Eqn. 1 solves to $S(\mu(\mathcal{D})) = O(\Psi^2\mu(\mathcal{D})^{3/2})$. Since $\mu(\mathcal{D}) = O(n\Psi^2)$, the total space is $O(\Psi^5 n^{3/2})$.

Performing a Query. Let $p, q \in P$ be given. We may assume that p and q are representative for their cells. If $p = q$, we say YES. If $p \neq q$, we let v_p and v_q be the nodes in T with $p \in P_{v_p}$ and $q \in P_{v_q}$, respectively. Let u be least common ancestor of v_p and v_q. It can be found in $O(\log n)$ time by walking up the tree. Let L be the path from u to the root of T. We check for each $s \in \bigcup_{v \in L} P_v$ whether p can reach s and s can reach q. If so, we say YES. If there is no such s, we say NO. Since $|P_v|$ decreases geometrically along L, the running time is dominated by the root, and it is $O(\Psi^2\mu(\mathcal{D})^{1/2})$. Bounding $\mu(\mathcal{D})$ by $O(\Psi^2 n)$, the total query time is be $O(\Psi^3\sqrt{n})$. We now argue correctness. First, note that we will say YES only if there is a path from p to q. Now suppose there is a path π in G from p to q, where $p \neq q$ and p, q are representatives. Let v_p, v_q be the nodes in T for p and q, let u be their least common ancestor, and L be the path from u to the root. By construction, $\bigcup_{v \in L} \mathcal{D}_v$ must contain a disk for a point r in π. We pick r such that the corresponding node v is closest to the root. Let s be the representative for the cell containing r. Then there is an edge from r to s and from s to r, so p can reach s and s can reach q in the transmission graph of v. Thus, when walking along L, the algorithm will discover s and the connection between p and q.

Preprocessing Time. We compute for each node v in T a spanner H_v for the corresponding transmission graph, as in Theorem 2.7. Since we are only interested in the reachability H_v, we can choose $t > 1$ to be some small constant. Since T has $O(\log n)$ levels, the total running time for this step is $O(n \log n(\log n + \log \Psi))$. Then we go through all the nodes $v \in T$. For each $s \in P_v$, we compute a BFS tree in H_v with root s. Next, we reverse all edges in H_v and we again compute BFS-trees for all $s \in P_v$ in the transposed graph. This gives the necessary information we want to store for s. Since the amount of work is proportional to the total size of the BFS-trees, we get a total running time of $O(\Psi^5 n^{3/2})$. Theorem 4.5 now follows.

4.3 Ψ is polynomially bounded

Now we assume that Ψ is bounded by some polynomial in n. Then we can show the following.

▶ **Theorem 4.8.** *Let G be the transmission graph for a two-dimensional set P of n points and let Ψ be the ratio between the largest and smallest radii of the points in P. If $\Psi = O(\text{poly}(n))$, we can construct a reachability oracle for G in $O(n^{5/3}\log^2 n)$ time with $S(n) = O(n^{5/3}\log n)$ and $Q(n) = O(n^{2/3}\log n)$. All queries are answered correctly with high probability.*

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 few local information to find such a "short" path.

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

▶ **Lemma 4.9.** *We can sample a set $S \subset P$ of size $O(n^\alpha \log n)$ s.t. the following holds w.h.p.: for any $p, q \in P$, if there is a path π from p to q in G of length at least $n^{1-\alpha}$, then $\pi \cap S \neq \emptyset$.*

We find such a sample S, 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 $O(n^{1+\alpha}\log n)$.

Now we treat short paths. Let $L = \lceil \log \Psi \rceil$. We consider L grids $\mathcal{Q}_0, \ldots, \mathcal{Q}_L$, s.t. the cells in $\mathcal{Q}_i$ have diameter 2^i. For each $\sigma \in \mathcal{Q}_i$, let $Q_\sigma \subseteq P$ be the vertices $p \in P \cap \sigma$ with $r_p \in [2^i, 2^{i+1})$. Q_σ forms a clique in G, and for each $p \in Q_\sigma$, the disk $D(p)$ covers σ. The *neighborhood* $N(\sigma)$ is defined as the set of all cells from $\mathcal{Q}_i$ 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 points that lie in cells of $N(\sigma)$. For every $i = 0, \ldots, L$ and for every $\sigma \in \mathcal{Q}_i$ with $Q_\sigma \neq \emptyset$, we fix an arbitrary *representative point* $q_\sigma \in Q_\sigma$. For every point $p \in P$, we store for every $i \in \{0, \ldots, L\}$ a sorted list of all cells $\sigma \in \mathcal{Q}_i$ with $p \in P_\sigma$ such that q_σ can be reached from p and a list of all cells $\sigma \in \mathcal{Q}_i$ with $p \in P_\sigma$ such that q_σ can reach p. A point in P appears in at most $O(n^{2-2\alpha}\log \Psi)$ point sets P_σ, so the total space requirement is $O(n^{3-2\alpha}\log \Psi)$.

Performing a Query. Let $p, q \in P$ be given. To decide whether p can reach q, we first check all $O(n^\alpha \log n)$ points in S. If there is an $s \in S$ such that p reaches s and such that s reaches q, we return YES. Otherwise, for $i = 0, \ldots, L$, we walk through the lists of cells whose representative point is reachable from p at level i and through the list of cells whose representative point can reach q at level i to check whether they contain a common element. Since the lists are sorted, this can be done in time linear in the list size, as in merge sort. If any of these pairs of lists contains a common cell, we return YES. Otherwise, we return NO.

For correctness, first note that we return YES only if there is a path from p to q. Now assume that there is a path π from p to q. If π has more than $n^{1-\alpha}$ vertices, then by Lemma 4.9, the sample S hits π w.h.p., and the algorithm returns YES. Otherwise, let r be the vertex of π with the largest radius, and let i be such that $r_r \in [2^i, 2^{i+1})$. Let σ be the cell of $\mathcal{Q}_i$ that contains r. Since π has at most $n^{1-\alpha}$ vertices and each edge of π has length at most 2^{i+1}, the path π lies in $N(\sigma)$. In particular, both p and q are contained in cells of $N(\sigma)$. Since $r \in Q_\sigma$ and since Q_σ forms a clique in G, the representative point q_σ of σ can be reached from p and can reach q. By the symmetry of neighborhood definition, σ is contained in the list of reachable cells from p and in the lists of cells that can reach q. This common cell will be detected when checking the corresponding lists for p and q at level i.

Time and Space Requirements. For 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 short paths

there are $O(\log \Psi)$ levels, and at each level we step through two lists of size $O(n^{2-2\alpha})$. Since we assume $\log \Psi = O(\log n)$, the tradeoff for the query time is at $\alpha = 2/3$, yielding $Q(n) = O(n^{2/3} \log n)$. The same α is the tradeoff for the space usage, which is $O(n^{5/3} \log n)$.

For the preprocessing, we first compute the reachability arrays for each $s \in S$. To do so, we build the spanner H for G from Section 2.2 in time $O(n \log n)$. Then, for each $s \in S$ we do 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 $O(n^{3/2} \log n)$ total time. Now, we do the preprocessing for short paths. For each $i = 0, \ldots, L$ and each cell $\sigma \in \mathcal{Q}_i$ that has a representative q_σ we do the following: consider the points P_σ. We compute the spanner H_σ from Section 2.2 for P_σ. For each q_σ, we do a BFS search in H_σ and its transposed graph starting from q_σ. This gives all $p \in P_\sigma$ that reach q_σ and that are reachable from q_σ. The running time is dominated by constructing the spanners. Since each point $p \in P$ is contained in $O(n^{2-2\alpha} \log \Psi) = O(n^{2/3} \log n)$ different P_σ, and since constructing H_σ takes $O(|P_\sigma|(\log \Psi + \log |P_\sigma|))$ time, the preprocessing time for the short paths is $O(n^{5/6} \log^2 n)$.

Acknowledgements. We thank Paz Carmi and Günter Rote for valuable comments.

References

1 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.
2 Azzedine Boukerche. *Algorithms and Protocols for Wireless Sensor Networks*. Wiley Series on Parallel and Distributed Computing). Wiley-IEEE Press, 1st edition, 2008.
3 Sergio Cabello and Miha Jejčič. Shortest paths in intersection graphs of unit disks. *Comput. Geom.*, 48(4):360–367, 2015.
4 Paul Callahan and 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.
5 Paz Carmi, 2014. Personal communication.
6 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.
7 Brent N. Clark, Charles J. Colbourn, and David S. Johnson. Unit disk graphs. *Discrete Math.*, 86(1-3):165–177, 1990.
8 Thomas Cormen, Charles Leiserson, Ronald Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001.
9 Martin Fürer and Shiva Prasad Kasiviswanathan. Spanners for geometric intersection graphs with applications. *J. Comput. Geom.*, 3(1):31–64, 2012.
10 Sariel Har-Peled. *Geometric Approximation Algorithms*. AMS, 2011.
11 Jacob Holm, Eva Rotenberg, and Mikkel Thorup. Planar Reachability in Linear Space and Constant Time. *CoRR*, arXiv:1411.5867, 2014.
12 Hiroshi Imai, Masao Iri, and Kazuo Murota. Voronoi Diagram in the Laguerre Geometry and its Applications. *SICOMP*, 14(1):93–105, 1985.
13 D. Kirkpatrick. Optimal Search in Planar Subdivisions. *SICOMP*, 12(1):28–35, 1983.
14 G. Narasimhan and M. Smid. *Geometric spanner networks*. Cambridge Univ. Press, 2007.
15 David Peleg and Liam Roditty. Localized spanner construction for ad hoc networks with variable transmission range. *TOSN*, 7(3), 2010.
16 P. v. Rickenbach, R. Wattenhofer, and A. Zollinger. Algorithmic Models of Interference in Wireless Ad Hoc and Sensor Networks. *IEEE ACM T NETWORK*, 17(1):172–185, 2009.
17 Andrew Chi-Chih Yao. On Constructing Minimum Spanning Trees in k-Dimensional Spaces and Related Problems. *SICOMP*, 11(4):721–736, 1982.