Efficient Construction of Spanners and BFS Trees for Disk Transmission Graphs

Haim Kaplan[1,a)] Wolfgang Mulzer[2,b)] Liam Roditty[3,c)] Paul Seiferth[2,d)]

Abstract: Let $P \subset \mathbb{R}^2$ be a set of n points in the plane, each having an associated radius $r_p > 0$. This induces a directed graph T on P with an edge from p to q if and only if q lies in the disk with radius r_p around p.
A *t-spanner* for T is sparse subgraph G of T that approximates each path: for any two vertices p, q connected in T by a path of length l, there path in G of length at most tl. For any constant $t > 1$, we show how to construct a t-spanner in time $O(n(\log n + \log \Phi))$, where Φ is the spread of P (ratio of maximum to minimum pairwise distances in P) and where the constant hidden in the O-notation depends on t. Then, for any $s \in P$, we show how to use G to construct a BFS tree for T with root s within the same time bound.

1. Introduction

The usual way to tackle algorithmic problems on wireless sensor networks is to model them as unit-disk graphs, i.e., intersection graphs of disks with radius 1. Even though this provides a rigorous and robust approach from a mathematical and algorithmic point of view, it does not cover all physical properties of a sensor network. One major drawback is that unit-disk graphs imply that each sensor must have the same transmission radius. Using intersection graphs of disks with arbitrary radii is one possible model that circumvents this issue. Furthermore, intersection graphs are undirected while in practical settings directed links may occur. We suggest *transmission graphs* as another model to address these problems: for a given set of points $P \subset \mathbb{R}^2$, each $p \in P$ having an associated radius r_p, the transmission graph T is a *directed* graph on P with an edge from p to q if and only if q lies in the disk $D(p)$ around p with radius r_p. Transmission graphs where introduced, among others, by Peleg and Roditty [9].

All three models suffer from one deficiency. Despite their rather efficient geometric description, the resulting graphs may have $\Theta(n^2)$ edges. This slows down even basic graph algorithms, like breadth first search (BFS), when executed on the graphs. Thus, one often tries to approximate these graphs without explicitly constructing them. A natural expectation is that the geometric properties of disk graphs allow us to do so efficiently.

One such an approximation is a *t-spanner*. Let T be a weighted Euclidean graph. For any $t > 1$, a sparse, spanning subgraph G of T is a t-spanner if for any path from p to q in T of length $d_T(p, q)$, there is a $p - q$-path in G of length at most $td_T(p, q)$. See [8] for an introduction and overview of spanners for geometric graphs. Fürer and Kasivisawnathan show how to efficiently compute a t-spanner for unit- and general disk intersection graphs using a variant of the Yao graph [4, 11]. In the case of transmission graphs, Peleg and Roditty give a construction for t-spanners when the points reside in a metric space with bounded doubling dimension [9]. However, except for the unit-disk case all the algorithms have a running time dependent on the number of edges of the intersection graph, which we try to avoid. We improve upon this by giving a subquadratic algorithm for transmission graphs in the special case of the Euclidean metric. Our result uses, similar to Fürer and Kasivisawnathan, a variant of the Yao graph.

Even having a good approximation in terms of a t-spanner at hand, we sometimes wish to obtain exact solutions for problems on disk graphs. Cabello and Jejčič worked in this direction by giving an $O(n \log n)$ algorithm for BFS trees in unit-disk graphs [2]. Given a vertex s, their algorithm computes the BFS with root s by exploiting a special structure of the Delaunay triangulation of the disk centers. We show that the spanner we constructed admits the same properties for transmission graphs as the Delaunay Triangulation does for unit-disk graphs. Thus, using the same techniques, we can also compute directed BFS trees for transmission graphs efficiently.

2. Organization and Contribution

Let P be a point set in $\mathbb{R}^2$ with $|P| = n$. Throughout this paper we assume that each $p \in P$ has an associated radius $r_p > 0$. Thus, P can also be seen as a set of disks in the plane. We say that P is a point set with *with radii* and denote by $D(p)$ the disk around $p \in P$ with radius r_p. The transmission graph T on P is obtained by adding a directed edge $\overrightarrow{pq}$ to T for any pair of distinct points p, q with $q \in D(p)$.

The spread Φ of P is the ratio between the pairwise maximum and minimum distances between points in P, i.e., $\Phi =$

[1] School of Computer Science, Tel Aviv University
[2] Institut für Informatik, Freie Universität Berlin
[3] Department of Computer Science, Bar Ilan University
[a)] haimk@post.tau.ac.il
[b)] mulzer@inf.fu-berlin.de
[c)] liamr@macs.biu.ac.il
[d)] pseiferth@inf.fu-berlin.de

$\max_{p,q\in P} |pq| / \min_{p\neq q\in P} |pq|$, where $|\cdot|$ is the Euclidean distance.

In Section 3 we present a modification of the Yao graph that yields a t-spanner for T. We first show that its construction can be carried out efficiently and then prove the approximation ratio. The running time depends logarithmically on the spread of P. In particular, we prove the following theorem.

Theorem 2.1 *Let T be the transmission graph of a two-dimensional n-point set P with radii and let Φ be the spread of P. Let $k \geq 14$ be a constant and set*

$$t = (1 + \sqrt{2 - 2\cos(4\pi/k)})/(2\cos(4\pi/k) - 1).$$

We can compute a t-spanner for T in time $O_k(n(\log n + \log \Phi))$.

Here the O_k notation suppresses constant factors depending on k. Note that t can be made arbitrarily close to 1, at the expense of a higher running time, by increasing k. In Section 4 we give an algorithm that uses this spanner to compute for a given vertex $s \in P$ the *exact* BFS tree for T with root s.

Theorem 2.2 *Let T be the transmission graph for a two-dimensional n-point set P with spread Φ. For every $s \in P$ we can compute a BFS tree of T with root s in time $O(n(\log \Phi + \log n))$.*

Our algorithm to establish this result is very similar to the one used by Cabello and Jejčič. Nevertheless, for the sake of completness we analyze its correctness and running time in Section 4.

3. Efficient Spanner Construction

Let T be the transmission graph of a point set $P \subset \mathbb{R}^2$ with radii. Let Φ be the spread of P. Our spanner construction creates a subgraph G of T and is similar to the Yao graph construction [11], but taking into account radii of the points. Let $k \geq 7$ be an integer and C be a set of k equally sized cones with the origin as apex that partition the plane. We attach the cones in C to each vertex $q \in P$. In each cone we pick the closest point to p such that $q \in D(p)$ and add the edge $\overrightarrow{pq}$ to G. This gives a graph with $O(kn)$ edges.

With the next Lemma one shows that the Yao graph is a t-spanner, where increasing k decreases t [1].

Lemma 3.1 *Let $k \geq 7$ and let*

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

For any three distinct points $p, q, r \in \mathbb{R}^2$ such that $|qr| \leq |qp|$ and $\alpha = \angle pqr$ is between 0 and $2\pi/k$, we have $|pr| \leq |qp| - |qr|/t$.

Proving the spanning property of Yao graphs goes by induction on the rank of the edge length. One argues that for each edge $\overrightarrow{pq}$ in T, there exists a path of length at most $t|pq|$ in G. Then, fixing a path in G and summing up the edge lengths, one sees that each path in T is t-approximated. The same arguments hold for our transmission graph version of the Yao graph, as we will see.

However, we do not know how to find the closest point p in each cone quickly when the additional constraint $q \in D(p)$ comes into play. To circumvent this issue, we use a more sophisticated construction that gives a similar graph with the same properties. During the construction, we need to solve efficiently the following subproblem: given set of points Q and a set of disks R, find for each $q \in Q$ one disk of R that contains q, if such a disk exists. There is an efficient algorithm with running time linear in $|Q|$ when the centers of the disks in R and the points in Q are separated by a directed line ℓ and Q is already sorted in the direction of ℓ.

Lemma 3.2 *Let Q, R and ℓ as above with $|Q| = n$ and $|R| = m$. If Q is sorted according to the direction of ℓ and ℓ separates Q and the centers of disks in R, then we can compute in time $O(m \log m + n)$ for each $q \in Q$ one disk of R that contains q if such a disk exists.*

Proof: Consider a coordinate system whose x-axis is ℓ. We will use the lower envelope of the boundaries of the disks and ℓ w.r.t. to the x-axis (see Fig.1). This lower envelope consists of at most $2m - 1$ arcs [10]. To compute it in $O(m \log m)$ time, we use a standard divide and conquer approach. Observe that the envelope is monotone in ℓ direction: each arc and ℓ can be interpreted as a function of x and the lower envelope is the minimum over all these functions.

Now, let S be the points at which the arcs change. We merge Q and S in time $O(m + n)$ and sweep over $Q \cup S$ in ℓ direction to compute the point-disk incidences for Q and R. We initialize D to be the disk belonging to the first arc and q to be the first point of Q. Whenever we reach a point in $p \in S \cup Q$, we update D or q, depending on whether $p \in S$ or $p \in Q$. In the former case, we set D to be the disk of the new arc. In the latter one, we first set $q = p$ and then check if $q \in D$. If so, we associate D to q. This sweep can be done in $O(m + n)$ time. By the monotonicity of the lower envelope, it is sufficient to check for each $q \in Q$ only the disk of the arc intersected by the line through q orthogonal to ℓ. Exactly this is done during the sweep. ■

3.1 The Construction

Let an integer $k \geq 14$ be given and let $c = c(k)$ be a constant to be determined later. Since we assume the spread of P is Φ, we can construct a *quadtree* for P with depth $O(\log \Phi)$: scale everything, so that the closest pair in P has distance c. We choose an integer L and an axis-parallel square $\square$ with diameter 2^L that contains all points of P. [*1] Since c is constant and P has spread Φ, $L \in O(\log \Phi)$. Furthermore, we set all radii $r_p > 2^L$ to be exactly 2^L without changing the graph T. Our quadtree $\mathcal{Q}$ is a sequence $(\mathcal{Q}_0, \ldots, \mathcal{Q}_L)$, where $\mathcal{Q}_i$ is a set of disjoint squares with diameter 2^i that we call *cells* of $\mathcal{Q}_i$. We construct $\mathcal{Q}$ recursively by partitioning each non-empty cell of $\mathcal{Q}_i$ into four smaller cells of $\mathcal{Q}_{i-1}$ with diameter 2^{i-1}. We start with $\mathcal{Q}_L = \{\square\}$. The cells in $\mathcal{Q}$ are axis-parallel and the scaling ensures that each cell of $\mathcal{Q}_0$ contains at most one point. A cell has *level i* if it is contained in $\mathcal{Q}_i$ and the distance $d_Q(\square, \square')$ between the two cells $\square$ and $\square'$ is defined to be the minimum $|pq|$ with with $p \in \square$ and $q \in \square'$. We compute various additional information for each cell $\square \in \mathcal{Q}_i$: a list $N(\square)$ of all cells $\square'$ of the same level with $d_Q(\square, \square') \in [(c-2)2^i, c2^{i+1})$, i.e., all cells near to $\square$; the points $R_\square \subseteq P$ in $\square$ with radius in $[(c-2)2^i, (c+1)2^{i+1})$, i.e., all these disks may intersect $N(\square)$; and the point $m_\square \in P$ with the maximum radius.

Recall that we have k congruent cones partitioning the plane. The following is done for each cone $C \in C$ Fix one and let C_q be

[*1] Don't worry! The $\square$ is neither an error of your PDF reader nor related to missing fonts. It is a commonly used notation for a cell (or a square) of a quadtree or a grid. See, e.g., [5]

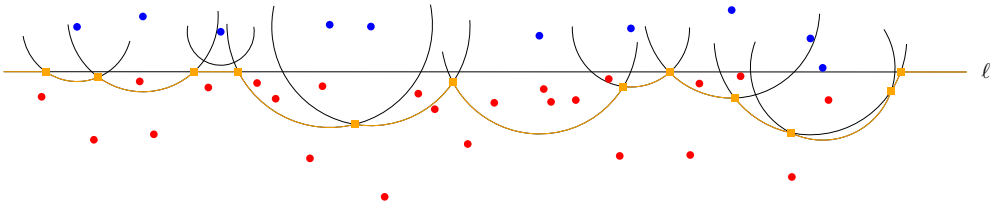

Fig. 1 The lower envelope and S (orange), with the points Q (red), and the centers of R (blue).

the cone with apex q obtained by translating C.

We process the quadtree level by level in increasing order, starting with Q_0. The goal is to select for each $q \in P$ an ingoing edge $\overrightarrow{pq}$ with p lying in C_q, $q \in D(p)$, and $|pq|$ being minimal. Since we do not know how to find the minimal p efficiently, we also select some additional ingoing edges for q. In doing so, each point can be in one of three states: *white*, *gray* or *black*. In the beginning, all points are white and we select only ingoing edges for white and gray points. Once the first ingoing edge for q is selected, say at level i, q becomes gray. After we finished processing level $i+1$, q becomes (and remains) black. It will never be considered again for an ingoing edge.

We perform three steps for each i, $1 \le i \le L$: **1)** preproccesing the points; **2)** handling for each non-empty cell $\square \in Q_i$ the disks with large radius, for which we can be sure that they completely contain $\square$; **3)** handling the disks with small radius which may intersect $\square$.

Step 1) (Preprocessing) Let $\square \in Q_i$ be a non-empty cell. We sort the points in $\square$ in x and y direction by merging the appropriate lists from the four children of $\square$. Then, for each gray point that became gray in level $i-2$, we color it black.

Step 2) (Handling large disks) Let $\square \in Q_i$ be a cell with white or gray points. For each $\square' \in N(\square)$ let Q be the white and gray points of $\square$ whose cone intersects $\square'$, i.e., $C_q \cap \square' \neq \emptyset$ for all $q \in Q$ (see Fig. 2). If the largest disk $D(m_{\square'})$ of $\square'$ contains $\square$, then we add all edges $\overrightarrow{mq}$ for $q \in Q$ and color q gray (if necessary). Otherwise, we proceed with **Step 3)**.

Step 3) (Handling small disks) Consider the set $R_{\square'}$. First, we check if there is an $r \in R_{\square'}$ whose disk contains $\square$. If so, similar to **Step 2)**, we add all edges $\overrightarrow{rq}$ for $q \in Q$ and color q gray if necessary. Otherwise, we use Lemma 3.2 to compute for all $q \in Q$ one $r \in R_{\square'}$ whose disk contains q if such an r exists. As separating line ℓ we take a line supporting one of the four sides of $\square$. Since $\square$ is axis-parallel, ℓ is either parallel to the x- or the y-axis and we sorted the points in Q as needed for the lemma in **Step 1)**. As usual, we add all edges $\overrightarrow{rq}$ we computed and set each q to gray if necessary.

3.2 The Resulting Graph is a t-Spanner

First, we argue that the construction gives $O(kc^2)$ ingoing edges for each point q and thus $O(kc^2n)$ edges in total. Let $\square$ be the level i cell of q and note that $|N(\square)| = O(c^2)$. To see this, consider the annulus A centered at the center of $\square$ with inner radius $(c-2)2^i$ and outer radius $(c+1)2^{i+1}$. A contains all cells of $N_\square$ and the area of A is $O(2^ic^2)$. Thus, it can be covered by $O(c^2)$ cells with diameter 2^i. During the construction we add for each of the k cones ingoing edges for q in at most two consecutive levels, since after the second one q is black. In each level we add $|N(\square)|$ edges, which is $O(kc^2)$ in total. The claim follows.

Next we show that every path in T is approximated well in G if c is a sufficiently large constant.

Lemma 3.3 *Let* $t = (1+\sqrt{2-2\cos(4\pi/k)})/(2\cos(4\pi/k)-1)$. *If* $c > \max\{2+(t+1)/(t-1), 5\}$, *then G is a t-spanner for T.*

Proof: First, by induction on the rank of the length of the edges, we show that for each edge $\overrightarrow{pq}$ in T there is a p-q-path of length at most $t|pq|$ in G. Then we show that this holds for paths, too

For the base case consider the shortest edge $\overrightarrow{pq}$ and let C_q be the cone containing p. Furthermore, let i be the level at which the cells $\square, \square'$ of q and p, respectively, have a distance in $[(c-2)2^i, c2^{i+1})$. Note that by our choice of this interval, such an i exists for any edge. Also note that C_q intersects $\square'$ and therefore, if q is not black, we select an ingoing edge for q from $\square'$. Indeed, q cannot be black at level i. Otherwise, there would be an ingoing edge for q at level at most $i-2$. By construction, this edge has length at most $(c+1)2^{i-1}$. But this cannot be, since $\overrightarrow{pq}$ is the shortest edge and has length at least $(c-2)2^i > (c+1)2^{i-1}$ for $c > 5$. Thus, one ingoing edge for q is selected from a point in $\square'$. This point must be p since its is the only point in $\square'$: every other point r would form an edge $\overrightarrow{pr}$ shorter than $\overrightarrow{pq}$. Hence, $\overrightarrow{pq}$ is an edge in G.

For the induction step, consider an arbitrary edge $\overrightarrow{pq}$ together with the cone C_q containing p and again let i be the level where the distance between $\square$ and $\square'$ is in $[(c-2)2^i, c2^{i+1})$. We distinguish two cases, depending on whether q is black at level i or is not (see Fig. 3).

Case 1: q is not black. Therefore, since $q \in D(p)$ and C_q intersects $\square'$, we select one ingoing edge for q from a point in $\square'$. If this point is p, we are done. Thus, assume we select the edge $\overrightarrow{rq}$ with $r \neq p$. Since $|pr| < 2^i$, we inductively assume a path from p to r in G of length at most $t2^i$. Using this, we estimate the length $d(p,q)$ from p to q in G by

$$d(p,q) \le t2^i + |rq| \le t2^i + |pq| + 2^i = |pq| + (1+t)2^i,$$

where the second inequality is due to triangle inequality. The lower bound $|pq| > (c-2)2^i$ gives

$$|pq| + (1+t)2^i \le (1+(1+t)/(c-2))|pq|, \qquad (1)$$

which less than $t|pq|$ for $c > 2+(1+t)/(t-1)$.

Case 2: q is black. Therefore, there is an edge $\overrightarrow{rq}$ that was selected for q in a level $j \le i-2$. Let $\square''$ be the cell of r at level j.

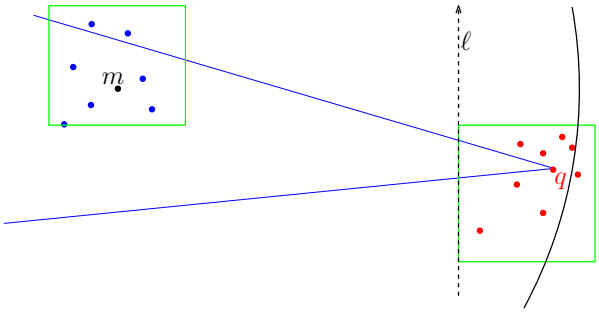

Fig. 2 The cell □ right and the cell □′ left. The set Q are the red points in □. Since the largest disk $D(m)$ (black) does not contain □, we need to do step 3) with ℓ as separating line.

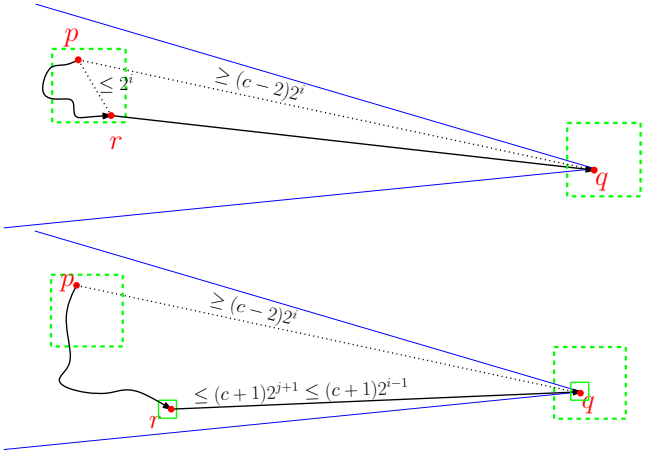

Fig. 3 The two cases of Lemma 3.3. The cells of level i are dashed green and the cells of level j are green.

By construction, the cone C_q either contains or intersects □″. If it only intersects □″, C_q must not necessarily contain r. To overcome this technicality, we double the angle of C_q from $2\pi/k$ to $4\pi/k$. We show in Obs. A.0.1 that C_q now contains □″ and thus r. Now, since $|rq| \le (c+1)2^{i-1} < (c-2)2^i \le |pq|$, we can use Lemma 3.1 to deduce that

$$d(p,q) \le t|pr| + |rq| \le t(|qp| - |rq|/t) + |rq| = t|qp|.$$

To finish the proof, let $\pi = (p_1, \ldots, p_l)$ be a directed path in T with length $|\pi|$. Then there is a path from p_1 to p_l in G with length $d(p_1, p_l) = \sum_{i=1}^{l-1} d(p_i, p_{i+1}) \le t \sum_{i=1}^{l-1} |p_i p_{i+1}| = t|\pi|$. ■

3.3 Running Time of the Construction

We prove the following lemma.

Lemma 3.4 *The construction of the spanner G of T takes $O_k(n(\log\Phi + \log n))$ time.*

Proof: Since c depends only on k, the quadtree can be computed in time $O_k(n \log \Phi)$ [3]. In the same time, we can compute the additional information $N(□)$,$R_□$ and $m_□$ for each cell of Q. The only non-obvious one is $N(□)$: for this, fix some $□ \in Q_i$. There are $O(c^2)$ level i cells $□' \in Q_i$ with $d_Q(□, □') < c2^{i+1}$. To find them, we do a post-order traversal of Q, starting from Q_L. We stop if we either reach level i or the current cell exceeds the maximum distance to □. For $N(□)$ we keep only those cells with distance at least $(c-2)2^i$ to □. Since each cell in $N(□)$ can be connected by a path of length $O(\log\Phi)$ to Q_L in the quadtree Q, we can find all lists in time $O_k(n \log \Phi)$.

For **Step 1)** we need $O(n)$ time at each level: let $P_□$ be the points in a cell $□ \in Q_i$. We can sort $P_□$ in x and y-directions $O(|P_□|)$ time, assuming the points in the four children are already sorted. Thus, we need $O(n)$ time for sorting at each level. Coloring the points takes also $O(n)$ time. Hence, since the depth of Q is $O_k(\log\Phi)$, we need $O_k(n \log \Phi)$ time for **Step 1)** in total.

Consider a cell $□ \in Q_i$ that we process at **Step 2)**. For each $□' \in N(□)$, we compute the set $Q_{□,□'} \subseteq P_□$ of points whose cone intersects □′ in $O(|P_□|)$ time. Since the $P_□$ partition P, we spend $O(n)$ time at each level for this. Furthermore, checking if $D(m_{□'})$ contains □ needs $O(1)$ time. Hence, since there are $O(n)$ cells in Q_i, **Step 2)** takes $O_k(n \log \Phi)$ time.

For **Step 3)** the crucial observation is that each radius participates only in $O(c^2)$ executions of the algorithm from Lemma 3.2. Let $□ \in Q_i$ and let $□' \in N(□)$. Recall that $R_{□'}$ are the points in □′ with radius in $[(c-2)2^i, (c+1)2^{i+1})$. Checking if the disk of an $r \in R_{□'}$ contains □ is dominated by the algorithm from Lemma 3.2. Thus, the total time spent for **Step 3)** is

$$\sum_{i=1}^{L} \sum_{□ \in Q_i} \sum_{□' \in N(□)} O(|R_{□'}| \log |R_{□'}| + |Q_{□,□'}|) =$$
$$\sum_{i=1}^{L} \sum_{□' \in Q_i} \sum_{□ \in N(□)} O(|R_{□'}| \log |R_{□'}| + |P_□|),$$

where equality holds by using $|Q_{□,□'}| \le |P_□|$ and rearranging the sum. Since $L = O(\log\Phi)$, $|N(□)| = O(c^2)$ and there are $O(n)$ cells at each level, splitting the sum and bounding $\log |R_{□'}|$ by $\log n$ gives

$$O_k(n \log \Phi) + O_k(\log n) \sum_{i=1}^{L} \sum_{□' \in Q_i} O(|R_{□'}|).$$

Finally, we can use that at level i we consider only radii in $[(c-2)2^i, (c+1)2^{i+1})$ and therefore each radius is in at most two levels. Thus, the last double sum sums up to $O(n)$ and the total running time for **Step 3)** is $O_k(n(\log\Phi + \log n))$ as well.

We need to repeat all steps for k cones, but the total running time still is $O_k(n(\log\Phi + \log n))$. ■

Theorem 2.1 follows by combining Lemmas 3.3 and 3.4.

4. From Spanners to BFS Trees

In this section we show how to compute BFS trees for a transmission graph T. Assume that the underlying point set has spread Φ. Let the desired root $s \in P$ be given.

To compute the BFS tree with root s we apply a technique used by Cabello and Jejčič in the case of unit-disk graphs [2]. Denote by $d_h(s,p)$ the BFS distance (also known as hop distance) from s to p in T and let W_i be the set of vertices $p \in P$ with $d_h(s,p) = i$. Assume we already computed $W_0, \ldots, W_i$. Cabello and Jejčič showed that the Delaunay triangulation can be used to efficiently identify W_{i+1}. We prove that our t-spanner from Theorem 2.1 provides the same properties for transmission graphs as the Delaunay triangulation does for unit disk graphs.

Lemma 4.1 *Let G be the t-spanner for T from Theorem 2.1 with $k = 14$ and let $v \in W_{i+1}$. There exists a $u \in W_i$ and a path $u = q_1, \ldots, q_l = v$ in G with $d_h(s, q_j) = i + 1$ for $1 < j \leq k$.*

Proof: Since $d_h(s,v) = i+1$, there is a $w \in W_i$ with $v \in D(w)$, i.e., T has an edge $\overrightarrow{wv}$. Assume this edge is not in G, otherwise we are done by setting $u = w$. We construct iteratively, backwards, a path from some $u \in W_i$ to v, s.t. (i) each inner vertex is contained in $D(w)$; (ii) has BFS distance $i+1$ to s; and (iii) we have $|wq_j| < |wq_{j+1}|$. By our assumption, this is true for v. Assume we constructed $q_{j+1}, \ldots, q_l = v$. We choose a vertex q_j as follows: consider the cone $C_{q_{j+1}}$ used by the construction of G (see Section 3) that contains w. If $\overrightarrow{wq_{j+1}}$ is an edge of G, we set $u = q_j = w$ and we are done. Otherwise, let C' be the cone obtained by doubling the angle of $C_{q_{j+1}}$. By Obs. A.0.1 and the way we constructed G, there must be at least one ingoing edge for q_{j+1} contained in C'. We set $q_j = p$ s.t. p minimizes $|wp|$ and the edge $\overrightarrow{pq_{j+1}}$ is contained in C'.

We argue that this choice of q_j guarantees $|wq_j| < |wq_{j+1}|$, and therefore, since $q_{j+1} \in D(w)$, we also have $q_j \in D(w)$. Consider the quadtree Q from the spanner construction algorithm and let i^* be the smallest integer such that the cells $\square$ of w and $\square'$ of p_{j+1} have distance in $[(c-2)2^{i^*}, c2^{i^*+1})$. There are two reasons for $\overrightarrow{wq_{j+1}}$ not being an edge: either q_{j+1} is white/gray at level i^* and another edge is chosen or q_{j+1} is black and not considered for incoming edges anymore. These are exactly the two cases we considered in the proof of Lemma 3.3, where in both it turned out that $|wq_j| < |wq_{j+1}|$.

Furthermore, we now know that $d_h(s,q_j)$ is either i or $i+1$: since $q_j \in D(w)$ we get $d_h(s,q_j) \leq i+1$ and since $d_h(s,p_{j+1}) = i+1$ we have $d_h(s,p_j) \geq i$. If it is i, we set $u = q_j$ and we are done. Otherwise, we proceed choosing the next edge as above. Since the distance to w decreases in each step and since T is finite, this process eventually stops and the lemma follows. ■

To compute the BFS tree with root s, assume we computed everything up to level W_i. By Lemma 4.1, W_{i+1} can be reached by W_i in the subgraph induced by $W_i \cup W_{i+1}$. This suggest the following approach to compute W_{i+1}: start a BFS search in G from each vertex in W_i. Every time we encounter a vertex we have not visited yet, check if it contained in one of the disk of W_i. If so, we add to W_{i+1}, if not, we stop the search. To efficiently check whether a point is contained 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 planar subdivision, can be computed in time $O(|W_i| \log |W_i|)$ and can be augmented by a point location structure to support the following queries in time $O(\log |W_i|)$: given a point p, return a disk in W_i that contains p or nothing if no such disk exists [6, 7].

The complete algorithm starts with $W_0 = \{s\}$ and computes the BFS tree level-wise. The running time of $O(n \log n)$ follows by observing that each edge $\overrightarrow{pq}$ of G is considered at most twice by this procedure, and each time we need do to query a power diagram with q (taking $O(\log n)$ time). Since G is sparse, this takes $O(n \log n)$ in total.

The details can be found in Algorithm 1. It is similar to the one used by Cabello and Jejčič [2].

```
   Data: spanner G and s ∈ P
1  W_0 ← {s}; dist[s] = 0; π[s] = s; i = 0 ;
2  for p ∈ P do
3  |  dist[p] = ∞; π[p] = NIL
4  while W_i ≠ ∅ do
5  |  compute power diagram with point location structure PD_i of W_i ;
6  |  Queue Q ← W_i ;
7  |  W_{i+1} ← ∅ ;
8  |  while Q ≠ ∅ do
9  |  |  p ← pop(Q);
10 |  |  foreach edge pq of G do
11 |  |  |  u ← PD_i(q) ;                    /* query PD_i with q */
12 |  |  |  if q ∈ D(u) and dist[q] = ∞ then
13 |  |  |  |  push(Q, q); dist[q] = i + 1; π[q] = u; add q to W_{i+1}
14 |  i ← i + 1
```

Algorithm 1: Compute the BFS tree for T with root s using G.

4.1 Correctness

We prove correctness of the algorithm by induction on the level of the BFS . Clearly, level 0 consists only of s, as set in the algorithm. Assume we computed correctly $W_0, \ldots, W_i$. We show that the algorithm computes $W_{i+1} = \{q \in P \mid d_h(s,q) = i+1\}$. Since it checks in line 12 that each vertex we add to W_{i+1} is a neighbor of a vertex in W_i in T and has not been added to some previous level yet ($\text{dist}[q] = \infty$), we get $W_{i+1} \subseteq \{q \in P \mid d_h(s,q) = i+1\}$.

For the other direction, let v be a vertex with distance $d_h(s,v) = i+1$. By Lem. 4.1 there exists a vertex $u \in W_i$ and a path $u = q_1, \ldots, q_l = v$ in G with $d_h(s,q_j) = i+1$ for $1 < j \leq l$. We argue that we discover all vertices of this path in iteration i of the inner while loop. Since we know that $u \in W_i$, we add $u = q_1$ to Q in step 6 and because q_2 is a neighbor of q_1 in G, we will discover q_2 in the for loop of line 10. Since $d_h(s,q_2) = i+1$, both checks in line 12 will be true and q_2 is added correctly to W_{i+1}. But at the same time we also add q_2 to Q and we can inductively follow that we discover to whole path up to $q_l = v$ and add it to W_{i+1}. Hence, also $W_{i+1} \supseteq \{q \in P \mid d_h(s,q) = i+1\}$ and the algorithm correctly computes W_{i+1}.

4.2 Running Time

The spanner G for T can be computed in time $O(n(\log \Phi + \log n))$ by Theorem 2.1. Computing the power diagram together with a point location data structure for a fixed W_i in step 5 takes $O(|W_i| \log |W_i|)$ time [6, 7]. Since the W_i partition P, this step needs $O(n \log n)$ time in total.

Now fix $p \in Q$ and let q be a neighbor of p in G. Step 11–13 can be done in $O(\log n)$ for q: we need to do a point location in a power diagram at step 11 and the remaining steps need $O(1)$ time.

Thus the for loop at step 10 needs $\deg_G(p)O(\log n)$ for a fixed p, where $\deg_G(\cdot)$ is the outdegree of p in G. We observe that each point p is added at most twice to Q. When we discover p the first time at level i, we add it to Q and to W_{i+1}. However, we also set $\text{dist}[q] = i + 1$ and thus never discover it again, since the check at step 12 will always fail. Adding p to W_{i+1} causes it to be in Q one more time at the next iteration. For the total running time of the outer while loop we get

$$\sum_{p \in P} 2 \deg_G(p) O(\log n).$$

We use that the spanner G has only $O(n)$ edges to conclude Algorithm 1 runs in time $O(n \log n)$. The above discussion with the analysis of the correctness gives Theorem 2.2.

5. Concluding Remarks

We showed how to construct a t-spanner for disk transmission graphs. The running time depends on Φ, the spread of the underlying point set. An obvious improvement one can ask for is: can we relax (or even get rid of) this dependency. Let us denote by Ψ the ratio of the largest and smallest radius of the points in P. First note that Φ bounds Ψ: all radii larger then the diameter of P can be set to be exactly the diameter. This does not change the graph T. Let x be the distance of the closest pair. Then every radius smaller than x can be set to, say, $x/2$. Then we have $\Psi = O(\Phi)$ without changing the graph.

However, the reverse is not true in general, but for case of our spanner construction we can establish a similar result. To sketch the idea, assume we scaled everything such that the smallest radius is 1 and the largest M. Then $\Psi = M$. Note that the $\log \Phi$ factor in the running time is only due to the depth of the quadtree. We want to replace this by $\log \Psi$. First consider the grid $\mathcal{Q}_0$ and observe that all points in one cell of $\mathcal{Q}_0$ form a clique in T. Thus, we can forget about the radii among these points and treat them with a spanner algorithm for the complete Euclidean graph (see e.g. [8]). Next we want to decompose P such that each part can be enclosed by a bounding box with diameter polynomial in n and M. To do so, consider the grid $\mathcal{G}_{10nM}$ whose cells have diameter $10nM$. For each non-empty cell $\square$, let $P_\square$ be the points in $\square$ together with the points in the eight cells surrounding $\square$. Since the largest radius is M, there cannot be any path from a point in $\square$ to a point outside $P_\square$. We use the algorithm from Section 3 for $P_\square$, for each non-empty $\square$. In doing so, each point takes part in $O(1)$ executions of the algorithm. This adds $O(1)$ additional edges to each point and the total running time does not increase. By the choice of $\mathcal{G}_{10nM}$, the depth of the quadtree becomes $O(\log nM) = O(\log n + \log \Psi)$ and the desired running time follows. We defer the details to the full version.

It is still open whether can tweak our construction to become independent of both, the spread of the points and the ratio of the radii at the same time, or if a different approach is required to solve the problem in its full generality.

Another open problem is how to compute BFS trees for (undirected) disk intersection graphs that are not unit-disk graphs. The spanner constructed by Fürer and Kasivisawnathan is very similar to ours and thus we expect an analogues result to Lemma 4.1 (using slightly more sophisticated arguments) to hold there as well. Then the framework from Cabello and Jejčič used in Algorithm 1 can be applied there, too. The main problem is that the spanner construction of Fürer and Kasivisawnathan uses the adjacency list of the intersection graph, i.e., they construct the graph explicitly which results in a running time of $\Omega(n^2)$. To improve upon this, we suggest a similar construction as used in Section 3. Providing a new version Lemma 3.2 for the case of disk intersection graphs is the main hurdle one has take. We believe this can be done by computing the power diagram of the disks and then "walking" through it in a direction of the separation line ℓ.

Acknowledgments This work is supported by GIF project 1161 & DFG project MU/3501/1. We also like to thank Matias Korman for his kind hospitality before and during our stay in Japan and especially for the invitation to this workshop.

References

[1] P. Bose, M. Damian, K. Doueb, J. O'Rourke, B. Seamone, M. H. M. Smid, and S. Wuhrer. $\pi/2$-Angle Yao Graphs are Spanners. *Internat. J. Comput. Geom. Appl.*, 22(1):61–82, 2012.

[2] S. Cabello and M. Jejčič. Shortest Paths in Intersection Graphs of Unit Disks. `arXiv:1402.4855v1`, February 2014.

[3] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 2nd edition, April 2000.

[4] M. Fürer and S. P. Kasiviswanathan. Spanners for geometric intersection graphs with applications. *J. Comput. Geom.*, 3(1):31–64, 2012.

[5] S. Har-Peled. *Geometric Approximation Algorithms*. American Mathematical Society, 2011.

[6] H. Imai, M. Iri, and K. Murota. Voronoi Diagram in the Laguerre Geometry and its Applications. *SIAM J. Comput.*, 14(1):93–105, 1985.

[7] D. Kirkpatrick. Optimal Search in Planar Subdivisions. *SIAM J. Comput.*, 12(1):28–35, 1983.

[8] G. Narasimhan and M. H. M. Smid. *Geometric spanner networks*. Cambridge University Press, 2007.

[9] D. Peleg and L. Roditty. Localized spanner construction for ad hoc networks with variable transmission range. *TOSN*, 7(3), 2010.

[10] M. Sharir and P. K. Agarwal. *Davenport-Schinzel sequences and their geometric applications*. Cambridge University Press, New York, NY, USA, 1996.

[11] A. C.-C. Yao. On Constructing Minimum Spanning Trees in k-Dimensional Spaces and Related Problems. *SIAM J. Comput.*, 11(4):721–736, 1982.

Appendix

Observation A.0.1 *Let $\mathcal{Q}_0$ be a grid with cell diameter 1 and $k \geq 14$ be a constant. Let C_p be a cone with apex q and angle $2\pi/k$. Let N be the cells of $\mathcal{Q}_0$ intersected by the bounding rays of C_q that have distance at least $c - 1$ from q for a constant $c > 2 + 1/(\sin 2\pi/k)$. Then the cone C' obtained from C by doubling its angle to $4\pi/k$ contains all cells of N.*

Proof: Let $\square \in N$ be such a cell. Let x be the first point where a ray r of C_q intersects $\square$. Consider the disk D with radius 1 centered at x and note that D contains $\square$. We show that the new cone contains D and therefore $\square$. Let r' be the ray corresponding to r and let y be the orthogonal projection of x onto r'. By definition of sine and since $|qx| \geq c - 1$ we get

$$|xy| \geq (c - 1) \sin \pi/k \geq 1,$$

for $c > 1 + 1/\sin \pi/k$. Thus, r' does not intersect D and the observation follows. ■