A Note on Reachability and Distance Oracles for Transmission Graphs

Mark de Berg
Department of Mathematics and Computer Science, TU Eindhoven, the Netherlands

Abstract

Let P be a set of n points in the plane, where each point $p \in P$ has a transmission radius $r(p) > 0$. The transmission graph defined by P and the given radii, denoted by $\mathcal{G}_{\text{tr}}(P)$, is the directed graph whose nodes are the points in P and that contains the arcs (p, q) such that $|pq| \leqslant r(p)$.

An and Oh [Algorithmica 2022] presented a reachability oracle for transmission graphs. Their oracle uses $O(n^{5/3})$ storage and, given two query points $s, t \in P$, can decide in $O(n^{2/3})$ time if there is a path from s to t in $\mathcal{G}_{\text{tr}}(P)$. We show that the clique-based separators introduced by De Berg *et al.* [SICOMP 2020] can be used to improve the storage of the oracle to $O(n\sqrt{n})$ and the query time to $O(\sqrt{n})$. Our oracle can be extended to approximate distance queries: we can construct, for a given parameter $\varepsilon > 0$, an oracle that uses $O((n/\varepsilon)\sqrt{n}\log n)$ storage and that can report in $O((\sqrt{n}/\varepsilon)\log n)$ time a value $d^*_{\text{hop}}(s,t)$ satisfying $d_{\text{hop}}(s,t) \leqslant d^*_{\text{hop}}(s,t) < (1+\varepsilon)\cdot d_{\text{hop}}(s,t) + 1$, where $d_{\text{hop}}(s,t)$ is the hop-distance from s to t. We also show how to extend the oracle to so-called continuous queries, where the target point t can be any point in the plane.

To obtain an efficient preprocessing algorithm, we show that a clique-based separator of a set F of convex fat objects in $\mathbb{R}^d$ can be constructed in $O(n \log n)$ time.

Keywords and phrases Computational geometry, transmission graphs, reachability oracles, approximate distance oracles, clique-based separators

Digital Object Identifier 10.57717/cgt.v2i1.25

Funding *Mark de Berg*: MdB is supported by the Dutch Research Council (NWO) through Gravitation grant NETWORKS-024.002.003.

1 Introduction

Let P be a set of n points in the plane, where each point $p \in P$ has an associated *transmission radius* $r(p) > 0$. The *transmission graph* defined by P and the given radii, denoted by $\mathcal{G}_{\text{tr}}(P)$, is the directed graph whose nodes are the points in P and that contains an arc (p, q) between two points $p, q \in P$ if and only if $|pq| \leqslant r(p)$, where $|pq|$ denotes the Euclidean distance between p and q. Transmission graphs are a popular way to model wireless communication networks.

The study of transmission graphs leads to various challenging algorithmic problems. In this paper we are interested in so-called *reachability queries*: given two query points $s, t \in P$, is there a path[1] from s to t in $\mathcal{G}_{\text{tr}}(P)$? A closely related query asks for the *hop-distance* from s to t, which is the minimum number of arcs on any path from s to t. Reachability queries can be answered in $O(1)$ time if we precompute for every pair $p, q \in P$ whether or not q is reachable from p and then store that information in a two-dimensional array. This requires quadratic storage. Our goal is to develop a structure using subquadratic storage that allows us to quickly answer reachability queries. Such a data structure is called a *reachability oracle*. If the data structure can also report the (approximate) hop-distance from s to t then it is called an *(approximate) distance oracle*.

[1] Whenever we speak about *paths* in $\mathcal{G}_{\text{tr}}(P)$, we always mean directed paths.

Note that reachability queries for an undirected graph G can be trivially answered in $O(1)$ time after computing the connected components of G. For directed graphs, however, efficient reachability oracles are much harder to design. In fact, for arbitrary directed graphs one cannot hope for an oracle that uses subquadratic storage. To see this, consider the class of directed bipartite graphs $G = (V \cup W, E)$ with $|V| = |W| = n/2$. Then any reachability oracle must use $\Omega(n^2)$ bits of storage in the worst case, otherwise two different graphs will end up with the same encoding and the oracle cannot answer all queries correctly on both graphs. Surprisingly, even for sparse directed graphs no reachability oracles are known that use subquadratic storage and have sublinear query time. Directed planar graphs, on the other hand, admit very efficient reachability oracles: almost three decades of research [4, 5, 7, 8] eventually resulted in the optimal solution by Holm, Rotenberg and Thorup [13], who presented an oracle that has $O(n)$ storage and $O(1)$ query time. Moreover, there are various (approximate) distance oracles for directed planar graphs; see for example [11, 16, 21] and the references therein.

Planar graphs are sparse, but transmission graphs can be dense. Nevertheless, the geometric structure of transmission graphs makes it possible to beat the quadratic lower bound on the amount of storage. Kaplan *et al.* [15] presented three reachability oracles for transmission graphs. Their oracles use subquadratic storage when Ψ, the ratio between the largest and smallest transmission radius, is sufficiently small. The first oracle has excellent performance—it uses $O(n)$ storage and can answer queries in $O(1)$ time—but it only works when $\Psi < \sqrt{3}$. The second oracle works for any Ψ, but it uses $O(\Psi^3 n\sqrt{n})$ storage and has $O(\Psi^3\sqrt{n})$ query time. The third oracle has a reduced dependency on Ψ—it has $O((\log^{1/3}\Psi) \cdot n^{5/3}\log^{2/3} n)$ storage and $O((\log^{1/3}\Psi) \cdot n^{2/3}\log^{2/3} n)$ query time—but an increased dependency on n. This third structure is randomized and answers queries correctly with high probability. Recently, An and Oh [3] presented the first reachability oracle that has subquadratic storage independent of Ψ. It uses $O(n^{5/3})$ storage and has $O(n^{2/3})$ query time. They also presented an oracle that uses $O(n\sqrt{n\log\Psi})$ storage and has $O(\sqrt{n\log\Psi})$ query time.

Our contribution and technique. We present a reachability oracle for transmission graphs that uses $O(n\sqrt{n})$ storage and has $O(\sqrt{n})$ query time. This significantly improves both the storage and the query time of the oracle presented by An and Oh [3]. Our oracle uses a divide-and-conquer approach based on separators. This is a standard approach for reachability oracles—it goes back to (at least) the work of Arikati *et al.* [4] and is also used by Kaplan *et al.* and by An and Oh. It works as follows.

Consider a directed graph $G = (V, E)$ and let $S \subset V$ be a separator for G. Thus $V \setminus S$ can be partitioned into two parts A, B of roughly equal size such that there are no arcs between A and B. The oracle now stores for each pair of nodes $(u, v) \in V \times S$ whether u can reach v in G, and whether v can reach u in G. In addition, there are recursively constructed oracles for the subgraphs $G[A]$ and $G[B]$ induced by A and B, respectively. Answering a reachability query with vertices s, t can then be done as follows: we first check if s can reach t via a node in S, that is, if there is a node $v \in S$ such that s can reach v and v can reach t. Using the precomputed information this takes $O(|S|)$ time. If s can reach t via a node in S, we are done. Otherwise, s can only reach t if s and t lie in the same part of the partition, say A, and s can reach t in $G[A]$. This can be checked recursively.

For graph classes that admit a separator of size $s(n)$, where $n := |V|$, this approach leads to an oracle with $O(n \cdot s(n))$ storage and $O(s(n))$ query time, assuming $s(n) = \Omega(n^\alpha)$ for some $\alpha > 0$. The problem when using this approach for transmission graphs is that they

may not have separators of sublinear size. An and Oh [3] therefore apply the following preprocessing step. Let $\mathcal{D}_P$ be the set of disks defined by the points in P and their ranges. Then An and Oh iteratively remove collections of $\Omega(n^{1/3})$ disks that contain a common point. Note that the disks in such a collection form a clique in the intersection graph defined by $\mathcal{D}_P$. For each collection C, a structure is built to check for two query points $s, t \in P$ if s can reach t via a point corresponding to a disk in C. The set of disks remaining after this preprocessing step has *ply* $O(n^{1/3})$—any point in the plane is contained in $O(n^{1/3})$ of them. This implies that the corresponding transmission graph admits a separator of size $O(n^{2/3})$ [18, 19], leading to a reachability oracle with $O(n^{5/3})$ storage and $O(n^{2/3})$ query time.[2] Our main insight is that we can use the so-called *clique-based separators* recently developed by De Berg *et al.* [6]. (A clique-based separator for a graph $\mathcal{G}$ is a separator that consists of a small number of cliques, rather than a small number of nodes.) This allows us to avoid An and Oh's preprocessing step and integrate the handling of cliques into the global separator approach, giving an oracle with $O(n\sqrt{n})$ storage and $O(\sqrt{n})$ query time.

We also show how to adapt the oracle such that it can answer approximate hop-distance queries. In particular, we show how to construct, for a given parameter $\varepsilon > 0$, an approximate distance oracle that uses $O((n/\varepsilon)\sqrt{n}\log n)$ storage. The oracle can report, for two query points $s, t \in P$, a value $d^*_{\text{hop}}(s,t)$ satisfying $d_{\text{hop}}(s,t) \leqslant d^*_{\text{hop}}(s,t) < (1+\varepsilon)\cdot d_{\text{hop}}(s,t)+1$, where $d_{\text{hop}}(s,t)$ denotes the hop-distance from s to t in $\mathcal{G}_{\text{tr}}(P)$. The query time is $O((\sqrt{n}/\varepsilon)\log n)$.

Finally, we study so-called *continuous reachability queries*, where the target point t can be any point in $\mathbb{R}^2$. Such a query asks, for a query pair $s, t \in P \times \mathbb{R}^2$, if there is a point $q \in P$ with $|qt| \leqslant r(q)$ such that s can reach q. Kaplan *et al.* [15] presented a data structure using $O(n \log \Psi)$ storage such that any reachability oracle can be extended to continuous reachability queries[3] with an additive overhead of $O(\log n \log \Psi)$ to the query time. An and Oh [3] also extended their reachability oracle to continuous reachability queries. The storage of their oracle remained the same, namely $O(n^{5/3})$, but the query increased by a multiplicative polylogarithmic factor to $O(n^{2/3}\log^2 n)$. We present a new data structure to extend any reachability oracle to continuous queries. It uses $O(n \log n)$ storage and has an additive overhead of $O(\log^2 n)$ to the query time. Thus, unlike the method of Kaplan *et al.*, its performance is independent of the ratio Ψ, and unlike the method of An and Oh, we do not incur a penalty on the query time when combined with our reachability oracle: the total query time remains $O(\sqrt{n})$ and the total storage remains $O(n\sqrt{n})$. The extension to continuous queries also applies to approximate distance queries, with an additional additive error of a single hop.

To construct our oracles, we need an algorithm to construct a clique-based separator for the intersection graph $\mathcal{G}^{\times}(\mathcal{D})$ of a set $\mathcal{D}$ of n disks in the plane. De Berg *et al.* [6] present a brute-force construction algorithm for the case where $\mathcal{D}$ is a set of convex fat objects in $\mathbb{R}^d$, running in $O(n^{d+2})$ time. This is sufficient for their purposes, since they use the separator to develop sub-exponential algorithms. For us the separator construction would become the bottleneck in our preprocessing algorithm. Before presenting our oracles, we therefore first show how to construct the clique-based separator in $O(n \log n)$ time. Since we expect that clique-based separators will find more applications where the construction time is relevant, we believe this result is of independent interest. We remark that our algorithm for constructing

[2] Actually, this preprocessing needs to be done at each step in the recursive construction of the oracle. Otherwise the ply would be $O(n_0^{2/3})$ instead of $O(n^{1/3})$, where n_0 is the initial number of points and n is the number of points in the current recursive call, resulting in an extra logarithmic factor in storage.

[3] Kaplan *et al.* used the term *geometric reachability queries*.

the separator depends on an explicit geometric representation of $\mathcal{G}^\times(\mathcal{D})$, that is, we need to know the objects defining $\mathcal{G}^\times(\mathcal{D})$. This seems unavoidable if we want to obtain an $O(n \log n)$ algorithm, since $\mathcal{G}^\times(\mathcal{D})$ may have $\Omega(n^2)$ edges.

2 A fast algorithm to construct clique-based separators

Our distance oracle for transmission graphs will be based on the so-called clique-based separators of De Berg *et al.* [6]. In this section we present an efficient construction algorithm for these separators. Although we only need them for disks in the plane, we will describe the construction algorithm for convex fat objects in $\mathbb{R}^d$, since we expect that an efficient construction of clique-based separators may find other uses.

Let F be a set of n convex α-fat objects in $\mathbb{R}^d$, where an object $o \subset \mathbb{R}^d$ is *α-fat* if there are concentric balls $B_{\text{in}}(o)$ and $B_{\text{out}}(o)$ with $B_{\text{in}}(o) \subseteq o \subseteq B_{\text{out}}(o)$ and $\text{radius}(B_{\text{in}}(o)) \geqslant \alpha \cdot \text{radius}(B_{\text{out}}(o))$. We are interested in sets of objects that are α-fat for some fixed, absolute constant $\alpha > 0$. We will therefore drop the parameter α and simply speak of *fat objects* from now on.

Let $\mathcal{G}^\times(F)$ be the intersection graph induced by F, that is, $\mathcal{G}^\times(F)$ is the undirected graph whose nodes correspond to the objects in F and that has an edge between two objects $o_1, o_2 \in F$ if an only if o_1 and o_2 intersect. A *δ-balanced clique-based separator* of $\mathcal{G}^\times(F)$ is a collection $\mathcal{S} = \{C_1, \ldots, C_{|\mathcal{S}|}\}$ of cliques in $\mathcal{G}^\times(F)$ whose removal partitions $\mathcal{G}^\times(F)$ into two parts of size at most δn, with no edges between the parts. Let γ be a function that assigns a wight to a clique, depending on its size. Then the *weight* of a separator $\mathcal{S}$ is defined as $\sum_{C \in \mathcal{S}} \gamma(|C|)$. In other words, the weight of a separator is the sum of the weights of its constituent cliques.

▶ **Theorem 1** (De Berg *et al.* [6]). *Let $d \geqslant 2$ and $\varepsilon > 0$ be constants and let γ be a weight function such that $\gamma(t) = O(t^{1-1/d-\varepsilon})$. Let F be a set of n convex[4] fat objects in $\mathbb{R}^d$. Then the intersection graph $\mathcal{G}^\times(F)$ has a δ-balanced clique-based separator $\mathcal{S}$ of weight $O(n^{1-1/d})$ for some fixed constant $\delta < 1$.*

De Berg *et al.* [6] show that when the objects have constant complexity, then the clique-based separator of Theorem 1 can be constructed in $O(n^{d+2})$ time, with a balance factor $\delta = 6^d/(6^d+1)$. We show that their algorithm can be implemented to run in $O(n \log n)$ time. Our implementation results in a somewhat larger balance factor, namely $12^d/(12^d+1)$. This is still a fixed constant smaller than 1, which is all that matters in applications of the separator theorem.

Step 1: Finding candidate separators. Step 1 starts by finding a *smallest k-enclosing hypercube* for F—this is a smallest hypercube that contains at least k objects from F—for $k := n/(6^d+1)$. De Berg *et al.* do this in a brute-force manner, taking $O(n^{d+2})$ time. Instead, we compute a 2-approximation[5] of a smallest k^*-enclosing hypercube for $k^* := n/(12^d+1)$, as follows.

Consider a smallest k^*-enclosing hypercube H^* for F. Let q be an arbitrary point in H^*, and let H_q denote the smallest k^*-enclosing hypercube centered at q. Note that H_q is a 2-approximation of H^*, that is, the edge length of H_q is at most twice the edge length

[4] The result also holds for non-convex fat objects if they are similarly sized, but we restrict our attention to convex fat objects.

[5] There is an efficient randomized algorithm to compute a smallest k-enclosing ball for a set of points in $\mathbb{R}^d$ [12], which we can use, but we want to obtain a deterministic algorithm.

of H^*. Moreover, H_q can trivially be computed in $O(n)$ time, for a given q. We can find a point $q \in H^*$ as follows. Pick a representative point $p_o \in o$ for every object $o \in F$ and compute a *weak ε-net* with respect to convex ranges for the set $P_F := \{p_o : o \in F\}$, where $\varepsilon := 1/(12^d + 1)$. (A weak ε-net for convex ranges is a point set $Q \subset \mathbb{R}^d$ such that any convex range—and, hence, any hypercube—containing at least εn points from P_F also contains at least one point from Q.) Since ε and d are fixed constants, there is such an ε-net of size $O(1)$ and it can be constructed[6] in $O(n)$ time [17]. Note that Q is guaranteed to contain a point $q \in H^*$. Thus, for each $q \in Q$, we compute the smallest smallest k^*-enclosing hypercube H_q, and we report the smallest of these hypercubes. This hypercube, which we denote by H_0, is a 2-approximation a smallest k^*-enclosing hypercube.

The hypercube H_0 is used to define a set of potential separators, as explained next. Assume without loss of generality that the edge length of H_0 is 2, so that the edge length of a smallest k^*-enclosing hypercube is at least 1. Define $H(t)$ to be the copy of H_0 that is scaled by a factor t with respect to the center of H_0. Following De Berg *et al.*, we define $F_{\text{large}} \subseteq F$ to be the set of objects that intersect the interior or boundary of $H(3)$ and whose diameter is at least $1/5$. These so-called *large objects* can be stabbed by $O(1)$ points, and so they define $O(1)$ cliques. These cliques will be put into the separator (plus some more cliques, as detailed below), so we can focus on $F \setminus F_{\text{large}}$, the set of remaining objects.

Note that any hypercube $H(t)$ induces a separator $\mathcal{S}(t)$ in a natural way: put the objects from $F \setminus F_{\text{large}}$ that intersect the boundary $\partial H(t)$ into $\mathcal{S}(t)$, suitably grouped into cliques, in addition to the cliques comprising F_{large} that we already put into $\mathcal{S}(t)$. De Berg *et al.* [6] prove that one of the separators $\mathcal{S}(t)$ with $1 \leqslant t \leqslant 3$ is a separator with the desired properties. To this end they first prove that any separator $\mathcal{S}(t)$ with $1 \leqslant t \leqslant 3$ is $(6^d/(6^d + 1))$-balanced. In their proof they work with a smallest $n/(6^d + 1)$-enclosing hypercube H_0. The key argument is that $H(3)$ can be covered by 6^d hypercubes of edge length $9/10$ in such a way that any object in $F \setminus F_{\text{large}}$ is contained in one of them.[7] Since each of the covering hypercubes contains fewer than $n/(6^d + 1)$ objects, this results in a balance factor of $6^d/(6^d + 1)$. We can use exactly the same argument, except that our H_0 is only a 2-approximation of a smallest k^*-enclosing hypercube. Thus we need 12^d hypercubes in our cover; see Fig. 1(i). Since $k^* = n/(12^d + 1)$, this implies that our separators $\mathcal{S}(t)$ are $(12^d/(12^d + 1))$-balanced.

Step 2: Constructing the cliques and finding a low-weight separator. Step 2 starts by partitioning the set of objects that may intersect the separators $\mathcal{S}(t)$ into cliques. To this end the objects are partitioned into size classes, and for each[8] size class F_s a point set Q_s of size $O(n/2^{2sd})$ is generated that stabs all objects in F_s. Each point $q \in Q_s$ defines a clique C_q, which consists of objects from F_s containing q; here an object stabbed by multiple points can be assigned to an arbitrary one of them. The set Q_s consists of grid points of a suitably defined grid, and they are not only guaranteed to stab the objects $o \in F_s$ but they even stab the inner balls $B_{\text{in}}(o)$. Thus, we can assign each object $o \in F_s$ to the point $q \in Q_s$ closest to

[6] Using an ε-net for convex ranges instead of for hypercubes only, is overkill. However, we did not find a reference explicitly stating that a linear-time deterministic algorithm exists that constructs an ε-net for hypercubes.

[7] De Berg *et al.* actually cover $H(3)$ with unit hypercubes instead of hypercubes of edge length $9/10$, because they define an object to be large when its diameter is at least $1/4$. In degenerate cases, however, the smallest hypercube containing at least $n/(6^d + 1)$ objects can, in fact, contain many more than $n/(6^d + 1)$ objects, and so the unit hypercubes used in the covering may contain more objects as well. We solve this minor issue by redefining an object to be large when its diameter is $1/5$ (instead of $1/4$) so that we can use hypercubes of edge length strictly smaller than 1 in the covering.

[8] The smallest size class is handled differently, namely by creating a singleton clique for each object in this class. For simplicity we do not explicitly mention these singleton cliques from now on.

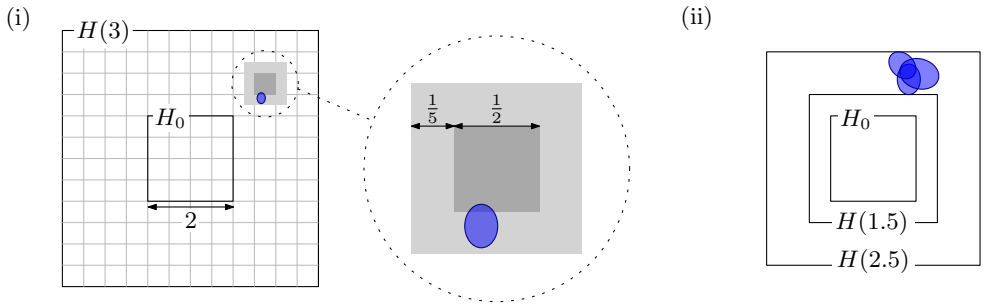

Figure 1 (i) Hypercube $H(3)$ is partitioned into 12^d cells. For each cell, a hypercube of edge length 9/10 with the same center is created; see the dark grey cell and the light grey hypercube. Any object of diameter at most 1/5, such as the blue object, intersecting some cell is completely contained in the corresponding hypercube. (ii) For the clique C shown in blue, we have $I_C = [1.5, 2.5]$.

the center of $B_{\text{in}}(o)$. Since Q_s forms a grid, this can be done in $O(|F_s| \log n)$ time, or even in $O(|F_s|)$ time if we would allow the floor function. The total time to construct the set $\mathcal{C}$ of all cliques over all size classes F_s is therefore $\sum_s O(n/2^{2sd} + |F_s| \log n)$, which is $O(n \log n)$ since $\sum_s |F_s| \leqslant n$.

De Berg *et al.* prove that one of the separators $\mathcal{S}(t)$, for $1 \leqslant t \leqslant 3$ has the desired weight, that is, that the cliques from $\mathcal{C}$ intersected by $\mathcal{S}(t)$ have total weight $O(n^{1-1/d})$. We can find this separator in $O(n \log n)$ time, as follows. For each clique $C \in \mathcal{C}$, define

$$I_C := \{t : 1 \leqslant t \leqslant 3 \text{ and } \partial H(t) \cap \mathcal{U}(C) \neq \emptyset\},$$

where $\mathcal{U}(C)$ denote the union of the objects comprising the clique C. In other words, $I_C \subseteq [1, 3]$ contains the values of t such that $\partial H(t)$ intersect at least one object from the clique C; see Fig. 1(ii). The interval I_C can trivially be computed in $O(|C|)$ time, so computing all intervals takes $O(n)$ time in total. We then find a value $t^* \in [1, 3]$ that minimizes $\sum_{C:t\in I_C} \gamma(|C|)$, which can easily be done in $O(n \log n)$ time, and report $\mathcal{S}(t^*)$ as the desired separator.

In conclusion, both steps of the construction algorithm can be implemented to run in $O(n \log n)$ time, leading to the following theorem.

▶ **Theorem 2.** *Let F be a set of n constant-complexity fat objects in $\mathbb{R}^d$, where d is a fixed constant. Then we can construct a clique-based separator for $\mathcal{G}^\times(F)$ with the properties given in Theorem 1 in $O(n \log n)$ time.*

3 The oracles

Before we describe our oracles in detail, we need to introduce some notation. Recall that P is the set of input points, and that $r(p)$ denotes the transmission radius of a point $p \in P$. We denote the disk of radius r centered at some point z by $D(z, r)$, and for $p \in P$ we define $D_p := D(p, r(p))$. We call D_p the *transmission disk* of p. Note that the arc (p, q) is present in the transmission graph $\mathcal{G}_{\text{tr}}(P)$ if and only if $q \in D_p$. We write $p \rightsquigarrow q$ to indicate that p can reach q in $\mathcal{G}_{\text{tr}}(P)$. In the following we do not distinguish between the points in P and the corresponding nodes in $\mathcal{G}_{\text{tr}}(P)$.

The basic reachability oracle. Let $L = q_1, q_2, \ldots, q_{|L|}$ be a path in $\mathcal{G}_{\text{tr}}(P)$. The first ingredient that we need is an oracle for what we call a *via-path query* with respect to the

given path L: given two query points $s, t \in P$, is it possible for s to reach t via a node in L? In other words, a via-path query asks if there exist a node $q_i \in L$ such that $s \rightsquigarrow q_i \rightsquigarrow t$. We call such an oracle a *via-path oracle*. As observed by An and Oh [3], there is a simple via-path oracle with $O(1)$ query time that uses $O(n)$ storage, irrespective of the length of the path L. The oracle stores for each point $p \in P$, including the points in L, two values:

$$\text{MinIn}_L[p] := \min\{i : p \rightsquigarrow q_i \text{ and } q_i \in L\} \text{ and } \text{MaxOut}_L[p] := \max\{i : q_i \rightsquigarrow p \text{ and } q_i \in L\}.$$

In other words, $\text{MinIn}_L[p]$ is the minimum[9] index of any point $q_i \in L$ that can be reached from p, where the points in L are numbered in order along the path. Note that the path from p to q_{i^*}, where if $i^* := \text{MinIn}_L[p]$, may pass through other points from L (which then must be successors of q_{i^*} along L). A similar statement holds for the path from q_{j^*} to p where $j^* := \text{MaxOut}_L[p]$. Clearly the oracle needs $O(n)$ storage. It is easy to see that a via-path query with points s, t can be answered by checking if $\text{MinIn}_L[s] \leqslant \text{MaxOut}_L[t]$; see the paper by An and Oh [3, Lemma 13].

An and Oh show that a via-path oracle can be constructed in $O(n)$ time, provided a linear-size spanner of $\mathcal{G}_{\text{tr}}(P)$ is available; see the proof of Lemma 14 in their paper. (A spanner for $\mathcal{G}_{\text{tr}}(P)$ is a subgraph such that for any two points $p, q \in P$ we have: p can reach q in $\mathcal{G}_{\text{tr}}(P)$ if and only if p can reach q in the subgraph.) They also show that a spanner can be computed in $O(n \log^3 n)$ time [3, Theorem 5]. Note that the spanner needs to be computed only once, so its construction does not influence the preprocessing time of our final oracle as stated later in Theorem 5.

Now let $\mathcal{D}_P := \{D_p : p \in P\}$ be the set of disks defined by the points in P and their transmission radii. Let $\mathcal{G}^\times(\mathcal{D}_P)$ denote the intersection graph of $\mathcal{D}_P$, that is, the undirected graph with node set $\mathcal{D}_P$ that contains an edge (D_p, D_q) if and only if $D_p \cap D_q \neq \emptyset$. Let C be a clique in $\mathcal{G}^\times(\mathcal{D}_P)$ such that all disks in C can be stabbed by a single point x_C. We call such a clique C, a *stabbed clique*. A crucial observation by An and Oh [3] is that the set $P(C) \subseteq P$ of points corresponding to a stabbed clique C can be covered by six paths in $\mathcal{G}_{\text{tr}}(P)$. More precisely, $P(C)$ can be partitioned into (at most) six subsets $P_1(C), \ldots, P_6(C)$ with the following property: if we order the points in $P_i(C)$ by decreasing transmission radius then there is an arc from each point to its direct successor, so the points form a path in $\mathcal{G}_{\text{tr}}(P)$; see Figure 2. In fact, and this will be relevant later, each point in $P_i(C)$ has an arc to all its successors in the ordering. We call such a path a *transitive path*. (An and Oh use the term *chain*.)

It is folklore and easy to see that any clique in $\mathcal{G}^\times(\mathcal{D}_P)$ can be partitioned into $O(1)$ stabbed cliques. Indeed, if D_p is the smallest disk in the clique and σ is a square of side length $4 \cdot \text{radius}(D_p)$, then a sufficiently fine grid in σ will stab all disks in the clique. Thus we obtain the following lemma.

▶ **Lemma 3.** *Let C be a clique in $\mathcal{G}^\times(\mathcal{D}_P)$ and let $P(C) \subseteq P$ be the points corresponding to the disks in C. Then we can cover the points in $P(C)$ by $O(1)$ transitive paths in $\mathcal{G}_{\text{tr}}(P)$.*

A *via-clique oracle*, for a given clique C in $\mathcal{G}^\times(\mathcal{D}_P)$, is a data structure that can answer *via-clique queries*: given two query points s, t, decide if s can reach t in $\mathcal{G}_{\text{tr}}(P)$ via a point in $P(C)$. It is important to note that the query asks if s can reach t in $\mathcal{G}_{\text{tr}}(P)$, not in $\mathcal{G}^\times(\mathcal{D}_P)$. Lemma 3 implies that we can answer a via-clique query using $O(1)$ via-path queries.

[9] We define the minimum over the empty set to be ∞, which means that $\text{MinIn}_L[p] = \infty$ if no q_i can be reached from p. Similarly, the maximum over the empty set is defined to be $-\infty$, so $\text{MaxOut}_L[p] = -\infty$ if no q_i can reach p.

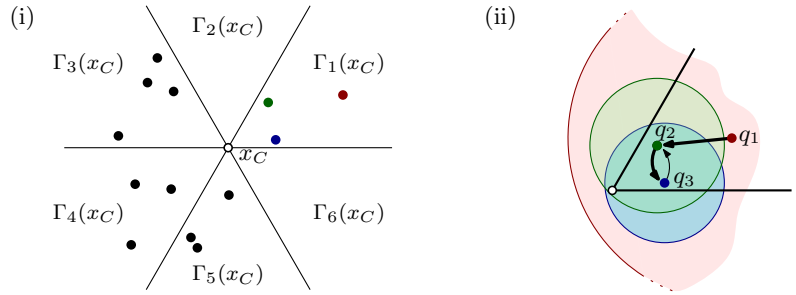

Figure 2 (i) To partition $P(C)$ into $P_1(C), \ldots, P_6(C)$ we partition the plane into six 60-degree cones $\Gamma_1(x_C), \ldots, \Gamma_6(x_C)$ with apex at x_C—we call these the *canonical cones* of x_C—and assign each point from $P(C)$ to the cone containing it, with ties broken arbitrarily. The set $P_i(C)$ then contains the points assigned to $\Gamma_i(x_C)$. In this example the sets $P_2(C)$ and $P_6(C)$ are empty. (ii) If we sort the points assigned to a cone by increasing transmission radius, then each point q_i must have an arc to any successor q_j in the ordering. The reason is that the angle $\angle q_i x_C q_j$ is at most 60 degrees, so $|q_i q_j| \leqslant \max(|q_i x_C|, |q_j x_C|) \leqslant \max(r(q_i), r(q_j)) = r(q_i)$.

The following lemma summarizes (and slightly extends, since they only considered stabbed cliques) the intermediate result from An and Oh on which we build.

▶ **Lemma 4.** *Let C be a clique in $\mathcal{G}^\times(\mathcal{D}_P)$. Then there is a via-clique oracle for C that uses $O(|C|)$ storage and that can answer via-clique queries in $O(1)$ time. The oracle can be constructed in $O(n)$ time, provided we have a spanner of the transmission graph $\mathcal{G}_{\text{tr}}(P)$ available.*

Proof. Since we can partition C into $O(1)$ stabbed cliques in $O(|C|)$ time, the result follows from the result by An and Oh [3]. Lemma 14 in their paper gives a bound of $O(n^{5/3})$ on the total preprocessing time for all cliques in their construction, but the bound for a single clique is $O(n)$; see the last two sentences in their proof. ◀

To obtain a reachability oracle for transmission graphs we combine Lemma 4 with the machinery of clique-based separators. Recall from the previous section that a δ-balanced clique-based separator of the intersection graph $\mathcal{G}^\times(\mathcal{D})$ of a set $\mathcal{D}$ of n disks, is a set $\mathcal{S}$ of cliques in $\mathcal{G}^\times(\mathcal{D})$ whose removal partitions $\mathcal{G}^\times(\mathcal{D})$ into two parts of size at most δn, for some fixed constant $\delta < 1$. Note that if we set the weight function γ to be $\gamma(|C|) = 1$, then the weight of a clique-based separator is simply its number of cliques. Theorem 2 thus implies that we can compute a clique-based separator consisting of $O(\sqrt{n})$ cliques in $O(n \log n)$ time. Clique-based separators provide a simple mechanism to recursively construct an efficient reachability oracle for transmission graphs, as follows.

Construct a clique-based separator $\mathcal{S}$ consisting of $(\sqrt{n})$ cliques for the intersection graph $\mathcal{G}^\times(\mathcal{D}_P)$, where $\mathcal{D}_P$ is the set of disks defined by the points in P and their transmission radii.
- For each clique $C \in \mathcal{S}$, construct a via-clique oracle using Lemma 4.
- Let $P_A, P_B \subset P$ be the two subsets of points corresponding to the partition of $\mathcal{G}^\times(\mathcal{D}_P)$ induced by $\mathcal{S}$. Recursively construct reachability oracles for P_A and P_B.

The recursion ends when $|P| \leqslant 1$; we then simply store the point in P (if any).

Answering a reachability query with query points s and t is done as follows. We first perform a via-clique query with s and t for each clique $C \in \mathcal{S}$. If any of the queries returns

YES then s can reach t. If all queries return NO, and s and t belong to different parts of the partition, then s cannot reach t. If neither of these two cases apply, then we recursively check if s can reach t in $\mathcal{G}_{\text{tr}}(P_A)$ or $\mathcal{G}_{\text{tr}}(P_B)$, depending on whether $s, t \in P_A$ or $s, t \in P_B$. This leads to the following theorem.

▶ **Theorem 5.** *Let P be a set of n points in $\mathbb{R}^2$. Then we can answer reachability queries on P in $O(\sqrt{n})$ time with an oracle using $O(n\sqrt{n})$ storage, which can be constructed in $O(n\sqrt{n})$ time.*

Proof. Observe that a separator in $\mathcal{G}^{\times}(\mathcal{D}_P)$ is also a separator in $\mathcal{G}_{\text{tr}}(P)$. In other words, if P_A, P_B are the two subsets of P corresponding to the parts in the partition of $\mathcal{G}^{\times}(\mathcal{D}_P)$ induced by $\mathcal{S}$, then there are no arcs between the points in P_A and those in P_B. This implies that queries are answered correctly.

The total storage for the via-clique oracles of the cliques $C \in \mathcal{S}$ is $O(n\sqrt{n})$. Hence, the storage $M(n)$ of our oracle satisfies the recurrence $M(n) = O(n\sqrt{n}) + M(n_1) + M(n_2)$, where $n_1, n_2 \leqslant \delta n$ for some constant $\delta < 1$ and $n_1 + n_2 \leqslant n$. It follows that $M(n) = O(n\sqrt{n})$. The query time $Q(n)$ satisfies $Q(n) = O(\sqrt{n}) + Q(n_1)$ where $n_1 \leqslant \delta n$, and so $Q(n) = O(\sqrt{n})$.

It remains to discuss the preprocessing time. By Lemma 4 the time to construct a via-clique oracles is $O(n)$, after $O(n \log^3 n)$ preprocessing to construct a spanner of $\mathcal{G}_{\text{tr}}(P)$, which only needs to be done once. Hence, the total time to construct a via-clique oracle for each of the $O(\sqrt{n})$ cliques is $O(n\sqrt{n})$. Moreover, by Theorem 2 the clique-based separator can be constructed in $O(n \log n)$ time. Thus the preprocessing time satisfies the same recurrence as the amount of storage, which implies that the total construction time is $O(n\sqrt{n})$. ◀

An approximate distance oracle. We now extend our reachability oracle to an approximate distance oracle. Thus the oracle must be able to approximate the hop distance[10] $d_{\text{hop}}(s, t)$ for two query points $s, t \in P$.

To answer approximate distance queries we need to extend the via-path oracle so that, for a given transitive path L and query points $s, t \in P$, it can approximate $d_L(s, t)$, the length of the shortest path from s to t via a point in L. The via-path oracle we presented above is not suitable for that: it only records the first point on L that can be reached from s and the last point from which we reach reach t, and going via these points may take many more hops than a shortest path via L would. To overcome this problem, we will store the first point on L that can be reached from s (and the last point that can reach t) *with a path of a given length*. To avoid increasing the storage too much, we will not do this for all possible lengths, but only for logarithmically many lengths. More precisely, our via-path oracle stores, for all $p \in P$ and $j \in \{-1, \ldots, \lceil \log_{1+\varepsilon} n \rceil\}$, the values

$$\text{MinIn}_L[p, j] := \min\{i : d_{\text{hop}}(p, q_i) \leqslant (1+\varepsilon)^j \text{ and } q_i \in L\}$$

and

$$\text{MaxOut}_L[p, j] := \max\{i : d_{\text{hop}}(q_i, j) \leqslant (1+\varepsilon)^j \text{ and } q_i \in L\}.$$

Here we include $j = -1$, since for $p \in L$ there is a path of zero hops to (resp. from) L, namely to (resp. from) p itself. The next lemma shows that we can get a $(1+\varepsilon)$-approximation of $d_L(s, t)$ using the arrays MinIn_L and MaxOut_L, up to an additional additive error of a single hop.

[10] We define $d_{\text{hop}}(s, t) = \infty$ if s cannot reach t; in this case the oracle should return ∞.

▶ **Lemma 6.** *Let* $d^*_L(s,t) := \min\{(1+\varepsilon)^j + (1+\varepsilon)^k : \mathrm{MinIn}_L[s,j] \leqslant \mathrm{MaxOut}_L[t,k]\}$. *If* $d_L(s,t) = \infty$ *then* $d^*_L(s,t) = \infty$, *and otherwise* $d_L(s,t) \leqslant d^*_L(s,t) < (1+\varepsilon)\cdot d_L(s,t) + 1$.

Proof. First note that for any pair j,k with $\mathrm{MinIn}_L[s,j] \leqslant \mathrm{MaxOut}_L[t,k]$, there is an arc from q_a to q_b for $a := \mathrm{MinIn}_L[s,j]$ and $b := \mathrm{MaxOut}_L[t,k]$. Hence,

$$\text{if} \quad \mathrm{MinIn}_L[s,j] \leqslant \mathrm{MaxOut}_L[t,k] \quad \text{then} \quad d_L(s,t) \leqslant (1+\varepsilon)^j + (1+\varepsilon)^k + 1. \tag{1}$$

To prove the first part of the lemma, suppose $d_L(s,t) = \infty$. Then we must also have $d^*_L(s,t) = \infty$, as claimed. Indeed, if $d^*_L(s,t) < \infty$ there are j,k with $\mathrm{MinIn}_L[s,j] \leqslant \mathrm{MaxOut}_L[t,k]$ which, together with (1), would contradict $d_L(s,t) = \infty$.

To prove the second part of the lemma, suppose $d_L(s,t) \neq \infty$. Then (1) and the definition of $d^*_L(s,t)$ immediately imply that $d_L(s,t) \leqslant d^*_L(s,t)$. To prove $d^*_L(s,t) \leqslant (1+\varepsilon)\cdot d_L(s,t)+1$, let $q_{i^*} \in L$ be a point such that $d_L(s,t) = d_{\mathrm{hop}}(s,q_{i^*}) + d_{\mathrm{hop}}(q_{i^*},t)$. Define

$$j^* := \min\left\{j : -1 \leqslant j \leqslant \lceil \log_{1+\varepsilon} n\rceil \text{ and } d_{\mathrm{hop}}(s,q_{i^*}) \leqslant (1+\varepsilon)^j\right\}$$

and

$$k^* := \min\left\{k : -1 \leqslant k \leqslant \lceil \log_{1+\varepsilon} n\rceil \text{ and } d_{\mathrm{hop}}(s,q_{i^*}) \leqslant (1+\varepsilon)^k\right\}.$$

Then

$$\mathrm{MinIn}_L[s,j^*] \leqslant i^* \leqslant \mathrm{MaxOut}_L[t,k^*]$$

and so $d^*_L(s,t) \leqslant (1+\varepsilon)^{j^*} + (1+\varepsilon)^{k^*} + 1$ by (1). Moreover,

$$d_L(s,t) = d_{\mathrm{hop}}(s,q_{i^*}) + d_{\mathrm{hop}}(q_{i^*},t) > (1+\varepsilon)^{j^*-1} + (1+\varepsilon)^{k^*-1}.$$

Hence,

$$d^*_L(s,t) \leqslant (1+\varepsilon)^{j^*} + (1+\varepsilon)^{k^*} + 1 < (1+\varepsilon)\cdot d_L(s,t) + 1,$$

thus finishing the proof. ◀

The following lemma summarizes the performance of our via-path approximate distance oracle.

▶ **Lemma 7.** *Let* $\mathcal{G}_{\mathrm{tr}}(P)$ *be a transmission graph on* n *points, and let* L *be transitive path in* $\mathcal{G}_{\mathrm{tr}}(P)$. *Then there is an oracle that uses* $O((1/\varepsilon)\cdot n\log n)$ *storage such that, for two query points* $s,t \in P$ *we can compute the value* $d^*_L(s,t)$ *from Lemma 6 in* $O((1/\varepsilon)\log n)$ *time.*

Proof. The storage needed for the arrays MinIn and MaxOut is $O(n\log_{1+\varepsilon} n)$. Since $\log_{1+\varepsilon} n = (\log n)/\log(1+\varepsilon) = O((1/\varepsilon)\log n)$ this proves the bound on the amount of storage of our oracles.

To compute $d^*_L(s,t)$ when answering a query, we note that the values $\mathrm{MinIn}[s,j]$ are non-increasing as j increases, and the values $\mathrm{MaxOut}[t,k]$ are non-decreasing as k increases. Hence, we can find $d^*_L(s,t) = \min\{(1+\varepsilon)^j + (1+\varepsilon)^k : \mathrm{MinIn}_L[s,j] \leqslant \mathrm{MaxOut}_L[t,k]\}$ by scanning the rows $\mathrm{MinIn}[s,\cdot]$ and $\mathrm{MaxOut}[t,\cdot]$, as follows. First, for $j = \lceil \log_{1+\varepsilon} n\rceil$ we find the smallest k such that $\mathrm{MinIn}_L[s,j] \leqslant \mathrm{MaxOut}_L[t,k]$. If no such k exists then $d^*_L(s,t) = \infty$ and we are done. Otherwise we decrease j by 1 and increase k until we again have $\mathrm{MinIn}_L[s,j] \leqslant \mathrm{MaxOut}_L[t,k]$. We continue scanning $\mathrm{MinIn}[s,\cdot]$ and $\mathrm{MaxOut}[t,\cdot]$ in opposite directions, until we have found for each j the smallest k such that $\mathrm{MinIn}_L[s,j] \leqslant \mathrm{MaxOut}_L[t,k]$. One of the pairs j,k thus found must be the pair defining $d^*_L(s,t)$. Thus the query time is linear in the length of the rows $\mathrm{MinIn}[s,\cdot]$ and $\mathrm{MaxOut}[t,\cdot]$, which is $O(\log_{1+\varepsilon} n) = O((1/\varepsilon)\log n)$. ◀

The via-path oracle for approximate distance queries can be extended to a via-clique oracle, in exactly the same way as was done for reachability queries: cover each clique by $O(1)$ stabbed cliques, cover each stabbed clique by at most six (transitive) paths, and construct a via-path oracle for each of them. These via-clique oracles can then be plugged into the separator approach, again exactly as before.

To answer a query with query points s, t on our complete oracle, we first compute an approximate hop-distance from s to t via one of the cliques in the clique-based separator. If s and t are in the same part of the partition, we then also recursively approximate the hop-distance within that part. Finally, we return the minimum of the distances found.

The analysis of the amount of storage and query time is as before; the only difference is that the via-clique oracle has an extra factor $O((1/\varepsilon)\log n)$ in the storage and query time, which carries over to the final bounds. Unfortunately, the preprocessing time does not carry over. The reason is that we cannot afford to work with a spanner. It is possible to do a BFS on $\mathcal{G}_{\mathrm{tr}}(P)$ efficiently, as demonstrated by Kaplan *et al.* [14] and by An and Oh [3], but it is not quite clear how to integrate this into the preprocessing algorithm. Moreover, we also need to do a BFS on the "reversed" transmission graph, which makes things even more difficult. We therefore revert to simply solving the All-Pairs Shortest Path problem (APSP) in a preprocessing step, thus giving all pairwise distances. Once these distances are available, the construction of the oracle can be done within the same time bounds as before. Solving APSP in directed unweighted graphs can be done in $O(n^{2.5302})$ time [10]. We can improve the preprocessing time by observing that, since we are approximating the hop-distance anyway, we may as well solve APSP approximately. Indeed, if we work with $(1+\varepsilon)$-approximations of the distances, then the total (multiplicative) approximation factor of our oracle is bounded by $(1+\varepsilon)^2 < 1+3\varepsilon$. Hence, by working with parameter $\varepsilon' := \varepsilon/3$, we can get a $(1+\varepsilon')$-approximation, for any $\varepsilon' > 0$. Galil and Margalit [9] (see also the paper by Zwick [22]) have shown that one can compute a $(1+\varepsilon)$-approximation of all pairwise distances in $\tilde{O}(n^\omega/\varepsilon)$ time,[11] where $\omega < 2.373$ is the matrix-multiplication exponent [2].

Putting everything together, we obtain the following theorem.

▶ **Theorem 8.** *Let P be a set of n points in $\mathbb{R}^2$. Then there is an oracle for approximate distance queries that uses $O((n/\varepsilon)\sqrt{n}\log n)$ storage and that, given two query points $s, t \in P$ can determine if s can reach t. If so, it can report a value $d^*_L(s,t)$ with $d_L(s,t) \leqslant d^*_L(s,t) \leqslant (1+\varepsilon)\cdot d_L(s,t)+1$ in $O((\sqrt{n}/\varepsilon)\log n)$ time. The oracle can be constructed in $\tilde{O}(n^\omega/\varepsilon)$ time, where $\omega < 2.373$ is the matrix-multiplication exponent.*

Extension to continuous reachability queries. In a continuous reachability query, we are given a query pair $s, t \in P \times \mathbb{R}^2$ and we wish to decide if s can reach t, that is, we wish to decide if there is point $q \in P$ such that $s \rightsquigarrow q$ and $|qt| \leqslant r(q)$. We define the hop-distance from s to t, denoted by $d_{\mathrm{hop}}(s,t)$, as

$$d_{\mathrm{hop}}(s,t) = \min\{d_{\mathrm{hop}}(s,q)+1 : q \in P \text{ and } |qt| \leqslant r(q)\}.$$

Next we present a data structure that, given a query target point $t \in \mathbb{R}^2$, determines a set $Q(t)$ of at most six points that can serve as the last point on a path from any source point s to t. To this end, let $\Gamma_1(t), \ldots, \Gamma_6(t)$ be the six canonical cones of t. (See Fig. 2(i) for an illustration of the concept of canonical cones.) For each cone $\Gamma_i(t)$, let $Q_i(t) \subseteq P$ be the set

[11] The notation $\tilde{O}$ hides $O(\mathrm{polylog}\, n)$ factors.

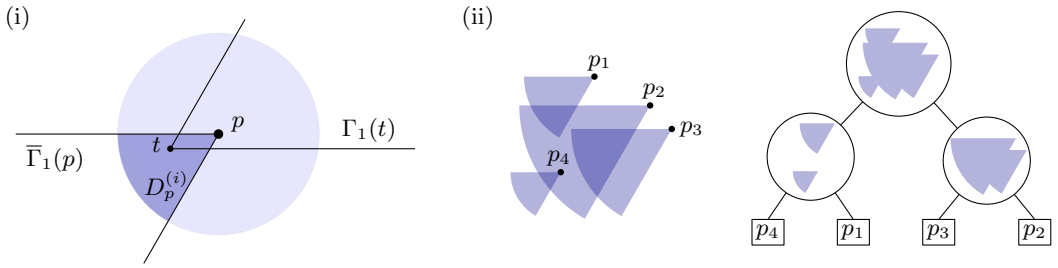

Figure 3 (i) Illustration of the fact that $p \in Q_i(t)$ if and only if $t \in D_p^{(i)}$. (ii) The set $\mathcal{D}^*$ and the corresponding search structure.

of points in $\Gamma_i(t)$ whose transmission disk contains t. (If a point p with $t \in D_p$ lies on the boundary between two cones, we assign it to one of them arbitrarily.) For $Q_i(t) \neq \emptyset$, let $q_i(t)$ be a point in $Q_i(t)$ of minimum radius. Define $Q(t)$ to be set of at most six points $q_i(t)$ selected in this manner.

▶ **Observation 9.** *Let $s \in P$ be a point that can reach t and let $d^*(s,t) := \min_{q \in Q(t)} d_{\text{hop}}(s,q) + 1$. Then $d_{\text{hop}}(s,t) \leqslant d^*(s,t) \leqslant d_{\text{hop}}(s,t) + 1$.*

Proof. Suppose s can reach t, and let $q^* \in P$ be the point immediately preceding t on a shortest path from s to t. Let i be such that $q^* \in Q_i(t)$, and let $q_i(t) \in Q_i(t)$ be the minimum-radius point that was put into $Q(t)$. Then q^* and $q_i(t)$ lie in the same 60-degree cone $\Gamma_i(t)$ and $r(q_i(t)) \leqslant r(q^*)$. Moreover, $|q^*t| \leqslant r(q^*)$ and $|q_i(t)t| \leqslant r(q_i(t))$. Hence, $(q^*, q_i(t))$ is an arc in $\mathcal{G}_{\text{tr}}(P)$—this was also the key property underlying the via-path oracle—and so $d_{\text{hop}}(s, q_i(t)) \leqslant d_{\text{hop}}(s, q^*) + 1$. This implies

$$d^*(s,t) \leqslant d_{\text{hop}}(s, q_i(t)) + 1 \leqslant d_{\text{hop}}(s, q^*) + 2 = d_{\text{hop}}(s,t) + 1.$$

On the other hand, any point $q \in Q(t)$ can reach t with one hop, so

$$d_{\text{hop}}(s,t) \leqslant \min_{q \in Q(t)} d_{\text{hop}}(s,q) + 1 = d^*(s,t).$$

◀

We now describe a data structure that, given a query point t, computes the point $q_i(t)$ mentioned in Observation 9 (if it exists), for a fixed index i. We will construct such a data structure for each $1 \leqslant i \leqslant 6$, and by querying each of these six structures we can compute the set $Q(t)$.

Consider a fixed index i, that is, consider the canonical cone of a fixed orientation. The structure for this fixed index i is defined as follows. For a point $p \in P$, let $\overline{\Gamma}_i(p)$ be the canonical cone opposite $\Gamma_i(p)$—for instance, $\overline{\Gamma}_1(p) = \Gamma_4(p)$—and define $D_p^{(i)} := D_p \cap \overline{\Gamma}_i(p)$. Then $p \in Q_i(t)$ if and only if $t \in D_p^{(i)}$; see Fig. 3(i). Thus we need a data structure on the set $\mathcal{D}^* := \{D_p^{(i)} : p \in P\}$ that, given a query point $t \in \mathbb{R}^2$, can quickly find a point $p \in P$ with minimum transmission radius such that $t \in D_p^{(i)}$. To this end we construct a balanced binary tree $\mathcal{T}_i$, as follows; see Fig. 3(ii).

- The leaves of $\mathcal{T}_i$ store the points $p \in P$ in left-to-right order according to their transmission radii. For example, the leftmost leaf stores a point $p \in P$ of minimum transmission radius and the rightmost leaf stores a point $p \in P$ of maximum transmission radius.

- Let $P(v)$ denote the set of points stored in the subtree rooted at v. Then v stores the union $\mathcal{U}(v) := \bigcup_{p \in P(v)} D_p^{(i)}$, preprocessed such that for a query point $t \in \mathbb{R}^2$ we can decide if $t \in \mathcal{U}(v)$ in $O(\log |\mathcal{U}(v)|)$ time. Using a standard point-location data structure [20] this associated structure needs $(|\mathcal{U}(v)|)$ storage.

A query with point $t \in \mathbb{R}^2$ to find $q_i(t)$ is answered as follows. If $t \notin \mathcal{U}(root(\mathcal{T}_i))$ then $Q_i(t) = \emptyset$ and we are done. Otherwise we descend in $\mathcal{T}_i$, proceeding to the left child v_{left} of the current node v if $t \in \mathcal{U}(v_{\text{left}})$, and proceeding to the right child of v otherwise. We then report the point stored in the leaf where the search ends. It is easy to see that this correctly answers the query. This leads to the following theorem.

▶ **Theorem 10.** *Let P be a set of n points in $\mathbb{R}^2$. There is a data structure of size $O(n \log n)$ that, for any query point $t \in \mathbb{R}^2$, can find in $O(\log^2 n)$ time a set $Q(t) \subset P$ of at most six points with the following property: for any point $s \in P$ we have $d_{\text{hop}}(s,t) \leqslant d^*(s,t) \leqslant d_{\text{hop}}(s,t) + 1$, where $d^*(s,t) := \min_{q \in Q(t)} d_{\text{hop}}(s,q) + 1$. The data structure can be built in $O(n \log^2 n)$ time.*

Proof. Observation 9 states that the set $Q(t)$ defined earlier has the desired properties. It follows from the discussion above that we can find $Q(t)$ by querying the six structures $\mathcal{T}_i$ as described above. It remains to argue that each $\mathcal{T}_i$ uses $O(n \log n)$ storage, has $O(\log^2 n)$ query time, and can be built in $O(n \log^2 n)$ time.

The total amount of storage is $\sum_{v \in \mathcal{T}_i} O(u(v))$, where $u(v)$ is the complexity of the union $\mathcal{U}(v)$. Recall that each $D_p^{(i)}$ is a 60-degree sector of a disk, and that all these sectors have exactly the same orientation. Hence, $\mathcal{U}(v)$ is the union of a set of *homothets*—they are scaled and translated copies of each other—and so their union complexity is linear [1]. Thus $u(v) = O(|P(v)|)$, from which it follows by standard arguments that the total amount of storage is $O(n \log n)$. The query time is $O(\log^2 n)$, since we spend $O(\log n)$ per node as we descend $\mathcal{T}_i$ and the depth of $\mathcal{T}_i$ is $O(\log n)$.

The tree $\mathcal{T}_i$ can be built in a bottom-up manner. Thus at each node v of $\mathcal{T}_i$ we construct $\mathcal{U}(v)$ by taking the union of $\mathcal{U}(v_{\text{left}})$ and $\mathcal{U}(v_{\text{right}})$, where v_{left} and v_{right} are the left and right child of v. This can be done in $O\left((u(v_{\text{left}}) + u(v_{\text{right}})) \log(u(v_{\text{left}}) + u(v_{\text{right}}))\right)$ time with a simple plane-sweep algorithm. After constructing $\mathcal{U}(v)$, we process it for point location in $O(u(v) \log u(v))$ time. Thus constructing the tree $\mathcal{T}_i$ takes $\sum_{v \in \mathcal{T}_i} O(u(v) \log u(v)) = O(n \log^2 n)$ time. Since we have six such trees, the total preprocessing tims is $O(n \log^2 n)$. ◀

▶ **Corollary 11.** *Let P be a set of n points in $\mathbb{R}^2$, each with an associated transmission radius, and let $\mathcal{G}_{\text{tr}}(P)$ be the corresponding the transmission graph.*

- **(i)** *There is an oracle for continuous reachability queries in $\mathcal{G}_{\text{tr}}(P)$ that has $O(\sqrt{n})$ query time, uses $O(n\sqrt{n})$ storage, and can be built in $O(n\sqrt{n})$ time.*
- **(ii)** *There is an oracle for continuous approximate-distance queries in $\mathcal{G}_{\text{tr}}(P)$ that uses $O((n/\varepsilon)\sqrt{n} \log n)$ storage and that, given query points $s \in P$ and $t \in \mathbb{R}^2$ can report a value $d_L^*(s,t)$ with $d_L(s,t) \leqslant d_L^*(s,t) \leqslant (1+\varepsilon) \cdot d_L(s,t) + 2$ in $O((\sqrt{n}/\varepsilon) \log n)$ time. The oracle can be constructed in $\tilde{O}(n^\omega/\varepsilon)$ time, where $\omega < 2.373$ is the matrix-multiplication exponent.*

4 Concluding remarks

We presented oracles for reachability and approximate distance queries in transmission graphs, whose performance does not depend on Ψ, the ratio between the largest and smallest transmission radius. The bounds we obtain are a significantly improvement over the previously best known bounds by An and Oh [3]. However, our bounds for reachability queries are still quite far from what can be achieved when $\Psi < \sqrt{3}$: we obtain $O(\sqrt{n})$ query time with

$O(n\sqrt{n})$ storage, while for $\Psi < \sqrt{3}$ Kaplan *et al.* [14] obtain $O(1)$ query time using $O(n)$ storage. They do this by reducing the problem to one on planar graphs. For $\Psi > \sqrt{3}$ such a reduction seems quite hard, if not impossible, and to make progress a much deeper understanding of the structure of transmission graphs may be needed. A first goal could be to improve on our bounds for small values of Ψ, say $\Psi = 2$. Lower bounds for unbounded Ψ would also be quite interesting.

References

1 Pankaj K. Agarwal, Janos Pach, and Micha Sharir. State of the union (of geometric objects): A review. In *Computational Geometry: Twenty Years Later*, pages 9–48. American Mathematical Society, 2008.

2 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proc. 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539, 2021. doi:10.1137/1.9781611976465.32.

3 Shinwoo An and Eunjin Oh. Reachability problems for transmission graphs. *Algorithmica*, 84:2820–2841, 2022. URL: https://doi.org/10.1007/s00453-022-00985-1.

4 Srinivasa Rao Arikati, Danny Z. Chen, L. Paul Chew, Gautam Das, Michiel H. M. Smid, and Christos D. Zaroliagis. Planar spanners and approximate shortest path queries among obstacles in the plane. In *Proc. 4th Annual European Symposium on Algorithms (ESA)*, volume 1136 of *Lecture Notes in Computer Science*, pages 514–528, 1996. doi:10.1007/3-540-61680-2_79.

5 Danny Z. Chen and Jinhui Xu. Shortest path queries in planar graphs. In *Proc. 32nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 469–478, 2000. doi:10.1145/335305.335359.

6 Mark de Berg, Hans L. Bodlaender, Sándor Kisfaludi-Bak, Dániel Marx, and Tom C. van der Zanden. A framework for Exponential-Time-Hypothesis-tight algorithms and lower bounds in geometric intersection graphs. *SIAM J. Comput.*, 49:1291–1331, 2020. doi:10.1137/20M1320870.

7 Hristo N. Djidjev. Efficient algorithms for shortest path queries in planar digraphs. In *Proc. 22nd International Workshop on Graph-Theoretic Concepts in Computer Science*, volume 1197 of *Lecture Notes in Computer Science*, 1997. doi:10.1007/3-540-62559-3.

8 Greg N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. Comput.*, 16(6):1004–1022, 1987. doi:10.1137/0216064.

9 Zvi Galil and Oded Margalit. All pairs shortest distances for graphs with small integer length edges. *Inf. Comput.*, 134(2):103–139, 1997. doi:10.1006/inco.1997.2620.

10 François Le Gall. Faster algorithms for rectangular matrix multiplication. In *Proc. 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 514–523, 2012. doi:10.1109/FOCS.2012.80.

11 Pawel Gawrychowski, Shay Mozes, Oren Weimann, and Christian Wulff-Nilsen. Better tradeoffs for exact distance oracles in planar graphs. In *Proc. 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 515–529, 2018. doi:10.1137/1.9781611975031.34.

12 Sariel Har-Peled and Benjamin Raichel. Net and prune: A linear time algorithm for euclidean distance problems. *J. ACM*, 62(6):44:1–44:35, 2015. doi:10.1145/2831230.

13 Jacob Holm, Eva Rotenberg, and Mikkel Thorup. Planar reachability in linear space and constant time. In *Proc. 56th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 370–389, 2015. doi:10.1109/FOCS.2015.30.

14 Haim Kaplan, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. Spanners for directed transmission graphs. *SIAM J. Comput.*, 47(4):1585–1609, 2018. doi:10.1137/16M1059692.

15 Haim Kaplan, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. Reachability oracles for directed transmission graphs. *Algorithmica*, 82(5):1259–1276, 2020. doi:10.1007/s00453-019-00641-1.

16 Hung Le and Christian Wulff-Nilsen. Optimal approximate distance oracle for planar graphs. In *Proc. 62nd IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 363–374, 2021. doi:10.1109/FOCS52979.2021.00044.

17 Jirí Matousek and Uli Wagner. New constructions of weak epsilon-nets. *Discret. Comput. Geom.*, 32(2):195–206, 2004. doi:10.1007/s00454-004-1116-4.

18 Gary L. Miller, Shang-Hua Teng, William P. Thurston, and Stephen A. Vavasis. Separators for sphere-packings and nearest neighbor graphs. *J. ACM*, 44(1):1–29, 1997. doi:10.1145/256292.256294.

19 Warren D. Smith and Nicholas C. Wormald. Geometric separator theorems & applications. In *Proc. 39th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 232–243, 1998. doi:10.1109/SFCS.1998.743449.

20 Jack Snoeyink. Point location. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry, Second Edition*, pages 767–785. Chapman and Hall/CRC, 2004. doi:10.1201/9781420035315.pt4.

21 Christian Wulff-Nilsen. Approximate distance oracles for planar graphs with improved query time-space tradeoff. In *Proc. 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 351–362, 2016. doi:10.1137/1.9781611974331.ch26.

22 Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, 2002. doi:10.1145/567112.567114.