Reachability Oracles for Disk Transmission Graphs

Haim Kaplan* Wolfgang Mulzer† Liam Roditty‡ Paul Seiferth†

Abstract

Let $P \subseteq \mathbb{R}^d$ be a set of n points, each with an associated radius $r_p > 0$. This induces a directed graph on P with an edge from p to q if and only if q lies in the ball with radius r_p around p.

We show that for $d = 1$ there is a data structure that answers *reachability queries* (given two vertices, is there a directed path between them?) in time $O(1)$ using $O(n)$ space and $O(n \log n)$ preprocessing time. With different techniques we can get a similar result for $d = 2$ as long as the radii are between 1 and $\sqrt{3}$.

1 Introduction

Let $G = (V, E)$ be a directed graph. A *reachability oracle* for G is a data structure for *reachability queries*: given $u, v \in V$, is there a directed path $u \rightsquigarrow v$ from u to v? The quality of the reachability oracle is measured by the preprocessing time $P(n)$, the space requirement $S(n)$, and the query time $Q(n)$. For planar digraphs Thorup showed the existence of efficient oracles [4]:

Theorem 1 *Let G be a planar digraph on n nodes. We can construct in time $O(n \log n)$ a reachability oracle for G with $S(n) = O(n \log n)$ and $Q(n) = O(1)$.*

We consider the problem for *transmission* graphs: let $P \subseteq \mathbb{R}^d$ be a set of n points. Each point $p \in P$ has an associated radius $r_p > 0$. We define a *directed* graph G with vertex set P. There is an edge from p to q if and only if $q \in B(p, r_p)$, where $B(p, r_p)$ is the closed ball around p with radius r_p. For notational convenience we define $B(p) := B(p, r_p)$ and denote by $C(p, r_p)$ its boundary.

For $d = 1$ these graphs admit a rich structure that can be exploited to construct $S(n) = O(n)$ reachability oracles with $Q(n) = 1$ in time $O(n \log n)$. Unfortunately, for $d = 2$ this structure vanishes. However, if the ratio between the radii is small (i.e., less than $\sqrt{3}$) we can planarize the transmission graphs without increasing their size significantly. Thus, using Thorup's Theorem, we get a similar result for the restricted $d = 2$ case (although with a slight increase in $S(n)$).

*School of Computer Science, Tel Aviv University, haimk@post.tau.ac.il

†Institut für Informatik, Freie Universität Berlin, {mulzer,pseiferth}@inf.fu-berlin.de

‡Department of Computer Science, Bar Ilan University, liamr@macs.biu.ac.il

2 The One-Dimensional Case

First, we consider the case $d = 1$. For this, we decompose G into a set $\mathcal{C}$ of strongly connected components (SCCs). A component $C \in \mathcal{C}$ can *reach* a component $D \in \mathcal{C}$ if there is a point in C that can reach a point in D. Then, by strong connectivity, every point in C can reach every point in D. Fix $C \in \mathcal{C}$. We define three points related to C: the leftmost point $\mathrm{l}(C)$ of C; the *left reachpoint* $\mathrm{lr}(C)$, that is, the leftmost point in $\mathbb{R}$ that C can reach; and the *direct* left reachpoint $\mathrm{dl}(C) := \min_{p \in C} p - r_p$, the leftmost point C reaches directly. The right versions $\mathrm{r}(C)$, $\mathrm{rr}(C)$, and $\mathrm{dr}(C)$ are defined analogously. We call $I_C = [\mathrm{l}(C), \mathrm{r}(C)]$ the *interval* of C.

Observation 1 *Let $p, q \in P$ and let C be the SCC of p. Then p reaches q if and only if $q \in [\mathrm{lr}(C), \mathrm{rr}(C)]$.*

Proof. W.l.o.g let q be to the left of p. If p reaches q we have $q \in [\mathrm{lr}(C), \mathrm{rr}(C)]$ by the definition of $\mathrm{lr}(C)$.

Conversely, let $q \in [\mathrm{lr}(C), \mathrm{rr}(C)]$. Let $p' \in P$ such that $\mathrm{lr}(C) = p' - r_{p'}$. A path from p to p' is a sequence of points $p_1, p_2, \ldots, p_k$ with $p_1 = p$, $p_k = p'$ and $d(p_i, p_{i+1}) \le r_{p_i}$, for $i = 1, \ldots, k-1$. Thus, the balls $B(p_i, r_{p_i})$ cover $[\mathrm{lr}(C), p]$, so p reaches q. □

Obs. 1 suggests the following $O(n)$ space oracle: for each $C \in \mathcal{C}$, store the left- and right reachpoint of C. Then, for two given query points p, q, let C be the SCC of p. We say YES if and only if $q \in [\mathrm{lr}(C), \mathrm{rr}(C)]$. Thus, a query can be answered in $O(1)$ time.

2.1 The Structure of the Components

To compute the reachpoints efficiently, we investigate the structure of the SCCs.

Observation 2 *The intervals I_C for $C \in \mathcal{C}$ form a laminar family, i.e., for any two distinct $C, D \in \mathcal{C}$, we have either $I_C \cap I_D = \emptyset$, $I_C \subseteq I_D$, or $I_D \subseteq I_C$.*

Proof. Since C is strongly connected, for every $x \in I_C$, there exists a point $p \in C$ with $d(p, x) \le r_p$. The same holds for D. Suppose $I_C \cap I_D \ne \emptyset$. If neither $I_C \subseteq I_D$ nor $I_D \subseteq I_C$, then one endpoint of I_C must lie in I_D and vice versa. Since the endpoints of I_C and I_D lie in P, strong connectivity implies that C can reach D and that D can reach C. But then, $C = D$, although we assumed them to be distinct. □

By Obs. 2, the components in $\mathcal{C}$ induce a forest. We add a root node to obtain a tree T.

Lemma 2 *For all $C \in \mathcal{C}$ the left reachpoint equals either $\mathrm{dl}(C)$ or $\mathrm{dl}(D)$, with D being a sibling of C in T. The situation for the right reachpoints is analogous.*

Proof. We argue for $\mathrm{lr}(C)$. Let $\overline{C}$ be the parent of C in T. Since $I_C \subseteq I_{\overline{C}}$, the parent $\overline{C}$ can reach C. Thus, C cannot reach $\overline{C}$, as C and $\overline{C}$ are distinct. Furthermore, since the endpoints of $I_{\overline{C}}$ lie in $\overline{C}$, this implies that C cannot reach any component outside $I_{\overline{C}}$, since by Obs. 1, C would then also reach $\overline{C}$.

By the definition of (direct) left reachpoint, there is a $D \in \mathcal{C}$ with $\mathrm{lr}(C) = \mathrm{dl}(D)$. Note that it may be that $D = C$. The argument above gives $I_D \subseteq I_{\overline{C}}$, so D is a descendant of $\overline{C}$. Assume that D neither equals C nor is its sibling. Then, by Obs. 2, there is a sibling D' of C, s.t. $I_D \subseteq I_{D'}$. Since $\mathrm{lr}(C) = \mathrm{dl}(D)$, C can reach D and, by Obs. 1, D' as well. But now Obs. 2 implies $\mathrm{lr}(D') < \mathrm{l}(D') < \mathrm{dl}(D)$. A contradiction. □

2.2 Computing Reachability Between Siblings

By Lem. 2 it suffices to search for $\mathrm{lr}(C)$ and $\mathrm{rr}(C)$ among the siblings of C in T. Let $C_1, \dots, C_k$ be children of a node in T, sorted from left to right according to their intervals. To compute the left reachpoints, we initially set $\mathrm{lr}(C_i) \leftarrow \mathrm{dl}(C_i)$. Furthermore, we initialize a stack S with C_1 and do the following:

```
for i = 2 → k do
    while S ≠ ∅ and lr(C_i) ≤ r(top(S)) do
        D ← pop(S); lr(C_i) = min{lr(C_i), lr(D)}
    end while
    push C_i onto S
end for
```

Computing the right reachpoints is done analogously.

Lemma 3 *We can compute the reachability between all siblings of nodes in T in $O(n \log n)$ time.*

Proof. Sorting the intervals requires $O(n \log n)$ time. Computing $\mathrm{dl}(C_i)$ is linear in the size of C_i, so $O(n)$ time in total. While processing the components, each is pushed/popped at most once onto/from S, taking again $O(n)$ time.

For correctness, consider the sorted siblings $C_1, \dots, C_k$. We maintain the following invariant: all components C_j with $j < i$ have the correct left reachpoint and S contains precisely those components C_j that cannot be reached by any component C_l with $j < l < i$. This is true for C_1: if $\mathrm{dl}(C_1) \neq \mathrm{lr}(C_1)$, then there would be another component C' with $\mathrm{dl}(C') = \mathrm{lr}(C_1)$. The component C' cannot be to the left of C_1, as C_1 is the leftmost sibling, and it cannot be to the right of C_1, since then $I_{C_1} \subseteq [\mathrm{lr}(C'), \mathrm{rr}(C')]$ and both would collapse to one SCC by Obs. 1. Thus $\mathrm{dl}(C_1) = \mathrm{lr}(C_1)$.

For general i, let $p \in P$ be the point with $\mathrm{lr}(C_i) = p - r_p$ and let π be a path from C_i to p. We define the *component path* π' by listing the distinct components π visits. Let F be the first component of π' after C_i, then $\mathrm{lr}(C_i) = \mathrm{lr}(F)$. Note that F must be to the left of C_i. If F is on the stack, we are done. Otherwise, by the invariant, there exists a component C_l on S that can reach F, i.e., $\mathrm{lr}(C_l) = \mathrm{lr}(F)$ and that is between F and C_i. The latter implies $I_{C_j} \subseteq [\mathrm{lr}(C_i), \mathrm{rr}(C_i)]$, and by Obs. 1 C_i can reach C_j. Thus, the algorithm sets $\mathrm{lr}(C_i) = \mathrm{lr}(C_j) = \mathrm{lr}(F)$, as desired. The while-loop ensures that the invariant for S is maintained. □

To summarize, we state our main theorem for $d = 1$.

Theorem 4 *For 1-dimensional transmission graphs we can construct a reachability oracle in time $O(n \log n)$ with $S(n) = O(n)$ and $Q(n) = O(1)$.*

Proof. The only point that is not obvious is how to determine the SCCs without explicitly constructing the transmission graph G. Recall the Kosaraju-Sharir algorithm [1]: first, it performs a DFS of G and records the finishing times of the vertices. Then it performs a second DFS in the transpose graph G'. The second DFS is initiated with the reversed order of the finishing times.

In order to implement this algorithm, we need two operations: given a point p, find an unvisited point q such that pq is an edge of G or an edge of G'. For G, this can easily be done in $O(\log n)$ time: store the points of P in a balanced search tree. When a point p is visited for the first time, remove it from the tree. When looking for an edge, determine the predecessor and the successor of p in the current set, and check the distance. For G', we proceed similarly, but we use an interval tree to store the r_p-balls around the points in P [2]. When a point is visited for the first time, we remove the corresponding r_p-ball from p. When we need to find a neighbor for p, we use the interval tree to find one ball that is pierced by p. Again, this can be done in $O(\log n)$ time.

Thus, $\mathcal{C}$ can be computed in $O(n \log n)$ time. The space requirement follows by construction. □

3 Two Dimensions with Small Radii

For $d = 2$, we restrict ourselves to radii in $[1, \sqrt{3})$. We show that in this restricted case, G can be planarized by first removing superfluous edges and then resolving edge crossings by adding $O(n)$ additional vertices. This will not change the reachability between the original vertices. Using Thorup's Theorem, the existence of efficient reachability oracles follows.

Let (uv) be a directed edge of G. If both, (uv) and (vu), are edges, we say there is an *undirected* edge $\{uv\}$ between u and v. Let $\mathcal{G}_{1/2}$ be the grid with cells of side length $1/2$. A vertex or edge lying (completely) inside a grid cell $\square$ *belongs* to it. If u belongs to $\square$ and v does not, (uv) is said to be *originating* from $\square$. The *neighborhood* $N(\square)$ are all cells in the 9×9 block of cells centered at $\square$. If one grid cell is in the neighborhood of another, they are *neighboring*. Note that for an edge from u to v, the two points belong to either the same or two neighboring grid cells.

3.1 Pruning the Graph

Consider the grid $\mathcal{G}_{1/2}$. We distribute the n points of P among the grid cells. Let us construct a graph $\bar{G}$ on P by doing the following for each non-empty grid cell $\square$: let $U \subseteq P$ be the vertices belonging to $\square$. First, we compute the euclidean minimum spanning tree (EMST) T of U and add all edges of T as undirected edges to $\bar{G}$. Second, for each non-empty cell $\square_N$ in $N(\square)$, we check if there are one or more edges originating from $\square$ and going to $\square_N$. If so, we add an arbitrary one of those edges to $\bar{G}$.

Lemma 5 *$\bar{G}$ has the following properties:* ***a)*** *it has the same reachability as G;* ***b)*** *it has $O(n)$ edges;* ***c)*** *if embedded on P, there are $O(n)$ edge crossings; and* ***d)*** *it can be constructed in $O(n \log n)$ time.*

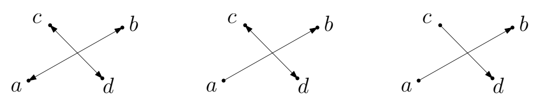
Proof. **a)** Since for any two vertices u, v belonging to the same grid cell $d(u, v) < 1$, all of them form a clique in G. Thus, our construction does not create new edges inside each cell. Also, any edge in $\bar{G}$ from $\square_1$ to $\square_2$ is an edge of G as well. Therefore, $E(\bar{G}) \subseteq E(G)$ and every path $u \rightsquigarrow v$ in $\bar{G}$ is also present in G. On the other hand, for an edge (uv) in G, there is a path in $\bar{G}$: either (uv) belongs to a cell $\square$, then we take the path along the EMST inside $\square$, or (uv) originates from $\square_1$ and goes to $\square_2$. In this case, there is an edge $(u'v')$ from $\square_1$ going to $\square_2$ in $\bar{G}$ and we take the path (using the EMSTs of $\square_1$ and $\square_2$) from u to u', then the edge $(u'v')$ and finally from v' to v.
b) For each cell $\square$ with m vertices we create $m - 1$ edges. Also, since $|N(\square)|$ is constant, at most $O(1)$ edges originate from $\square$. Altogether, we have at most $O(n)$ non-empty grid cells, and thus $\bar{G}$ is sparse.
c) We distinguish whether an edge e belongs to some grid cell $\square$ or not. In the former case it cannot be intersected by any other edge belonging to $\square$, since the EMST is non-crossing. It might be intersected by other edges, but these must originate from either $\square$ itself or a cell in $N(\square)$. This is a constant number of cells, each having $O(1)$ originating edges. It follows that e is intersected $O(1)$ times.

In the latter case, it remains to count edges crossing e that do not belong to some grid cell. Let A be the region where all endpoints of those edges may lie in. It follows by the bounded radii of the disks that A is covered by constant many grid cells, each contributing $O(1)$ to the number of edges crossing e. Therefore, each edge not belonging to some grid cell is also intersected $O(1)$ times and, using b), we have $O(n)$ edge intersections in $\bar{G}$ overall.
d) Using universal hashing and the floor function, we can distribute the points among the grid cells in time $O(n)$ [3]. Computing the EMST for a cell with m vertices needs $O(m \log m)$ time and altogether $O(n \log n)$. To check if an edge between two cells exists, assume we know for every vertex v in $\square_1$ its nearest neighbor v' in $\square_2$. Then, there is an edge from v to $\square_2$ if and only if $d(v, v') < r_v$. Thus, if $\square_1$ has m vertices, we can check in time $O(m)$ if an edge exists.

To obtain the nearest neighbor information we compute for each cell the Voronoi diagram together with a point location structure in overall time $O(n \log n)$ [2]. Afterwards, for each vertex v in a cell $\square$, we query the nearest neighbor of v in each cell of $N(\square)$ with a point location in its Voronoi diagram. Since $|N(\square)|$ is constant, a vertex participates in $O(1)$ point locations, taking $O(\log n)$ time each. Hence, we can compute the nearest neighbor information in time $O(n \log n)$. $\square$

3.2 Removing the Crossings

Consider a crossing of two edges between the vertices a, b and c, d. To eliminate it, we add a new vertex x and replace the two edges by four new ones. If the edge between a and b is directed we add (ax) and (xb), otherwise $\{ax\}$ and $\{xb\}$. For c and d we apply the same rule, and we call this procedure *resolving* a crossing. There are three types of crossings: **(i)** undirected–undirected, **(ii)** undirected–directed, and **(iii)** directed–directed



(i) (ii) (iii)

To argue that resolving crossings preserves reachability, we need the calculations in Obs. 3 as well as Obs. 4 about the local reachability under the presence of additional edges. For space reasons we omit/sketch the proofs of Obs. 3 & 4, but we note that Obs. 3 is the reason for the $\sqrt{3}$–restriction on the size of the radii.

Observation 3 *Let a, b be two points in $\mathbb{R}^2$.*
a) *If $d(a, b) = 1$ and c, d are the two intersection points of $C(a, 1)$ and $C(b, 1)$, then $d(c, d) = \sqrt{3}$.*
b) *If $d(a, b) = \sqrt{3}$ and c, d are the two intersection points of $C(a, \sqrt{3})$ and $C(b, 1)$, then $d(c, d) > \sqrt{3}$.*
c) *If $d(a, b) = \sqrt{3}$ and d is an intersection point of $C(a, 1)$ and $C(b, 1)$, then for any value $r_c \in [1, \sqrt{3}]$ the following holds: let c be the intersection point of $C(a, r_c)$ and $C(b, r_c)$ on the side of the line through a and b opposite to d, then $d(c, d) \geq r_c$.* $\square$

Observation 4 *Resolving a type (i),(ii) or (iii) crossing does not change the reachability if one of the edges*

$\{ac\}$, $\{ad\}$, $\{cb\}$ or $\{bd\}$ exists. For type (iii) it is also sufficient if (cb) and either (ac) or (ad) exists.

Proof. See Fig. 1 for type (i) and $\{ac\}$. Resolving introduces the blue and green connections. But they already existed by going along the dashed paths. The remaining cases can be verified analogously. □

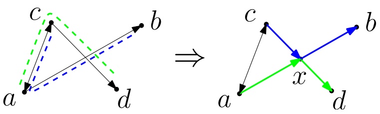


Figure 1: Since $\{ac\}$ exists, reachability is preserved.

We claim that resolving crossings retains reachability: **type (i)**: We may assume that $d(a, b) \leq d(c, d)$. Since $\{ab\}$ is undirected we can also, without losing generality, decrease r_a and r_b to $\max\{1, d(a, b)\}$. This does not add new connections between vertices. If now $d(a, b) > 1$, everything is scaled by a factor $1/d(a, b)$, so $d(a, b) = r_a = r_b = 1$ after scaling. Note that if either c or d lie in $B(a) \cup B(b)$, we are done by Obs. 4. Thus, assume this is not the case. To intersect $\{ab\}$, c and d must lie on opposite sides of the line through a and b. Then, the positions for c and d minimizing their distance are the two intersections of $C(a)$ and $C(b)$. But now, if $d(a, b) = 1$, we are in the situation of Obs. 3a). Otherwise, if $d(a, b) < 1$, the distance between d and c only can increase. Hence, since all radii are at most $\sqrt{3}$, such an edge cannot exist.
type (ii): This case can be reduced to a type (iii) crossing: assume w.l.o.g that $r_c \geq r_d$. Then, we decrease r_d to 1 and treat it as a type (iii) crossing.
type (iii): Assume w.l.o.g that $r_a > r_c$. Then, (cd) being directed implies $r_a > r_c > r_d$. Furthermore, everything can be scaled so that $r_a = \sqrt{3}$. We distinguish three cases: 1) $c \in B(a)$, 2) $d \in B(a)$, 3) or neither of them. See Fig. 2 for the three cases and where the points c and d must lie in each case to minimize their distance. If either $c \in B(a, r_c)$, $c \in B(b, 1)$, $d \in B(a, 1)$, or $d \in B(b, 1)$, then we are done by Obs. 4. Thus, assume this not be the case.
Case 1): The edge (ac) exists. If (cb) is also an edge, we are done by Obs. 4. So assume it is not, i.e. $b \notin B(c)$ or, dually, $c \notin B(b, r_c)$. Thus, the positions minimizing $d(c, d)$ are intersections of $C(a, r_c)$ and $C(b, r_c)$ for c and $C(a, 1)$ and $C(b, 1)$ for d. Minimizing $d(c, d)$ further leads to $d(a, b) = \sqrt{3}$. But then, by Obs. 3c), $d(c, d) > r_c$ and (cd) is not an edge.
Case 2): The edge (ad) exists. Similar to Case 1) there cannot be the edge (cb) at the same time, i.e $c \notin D(b, r_c)$, by Obs. 4. Again, the best position for d minimizing the distance to c is the intersection point of $C(a, 1)$ and $C(b, 1)$. Since $c \notin B(a)$, the best position for c is the intersection of $C(a)$ and $C(b, r_c)$. But $r_a > r_c$ and thus in any case we have that $d(c, d)$ is greater than $d(c_1, d)$ for c_1 being the c from Case 1). Hence, (cd) cannot be an edge by Obs. 3c).
Case 3): This is impossible by Obs. 3b): the positions for c and d minimizing their distance are the intersection points of $C(a)$ and $C(b, 1)$. Further minimizing their distance leads again to $d(a, b) = \sqrt{3}$. This is exactly the situation of Obs. 3b).

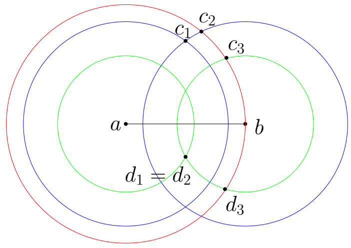


Figure 2: The edge (ab) of a type (iii) crossing and the circles $C(a)$, $C(a, r_c), C(b, r_c)$ and $C(a, 1), C(b, 1)$. Optimal positions for c and d in case j) are c_j and d_j.

The above argumentation shows that resolving crossings preserves the *local* reachability for edges. We can extend this to *global* reachability, i.e., that any two vertices reaching each other in the pruned graph have a path between them in the original graph. The complete argument is be contained in the full version.

Lemma 6 *We can planarize $\bar{G}$ by adding $O(n)$ vertices without changing the reachability.* □

3.3 Putting Things Together

We are now able to prove our main theorem for $d = 2$.

Theorem 7 *For 2-dimensional transmission graphs with radii in $[1, \sqrt{3})$ we can compute in time and space $O(n \log n)$ a reachability oracle with $S(n) = O(n \log n)$ and $Q(n) = O(1)$.*

Proof. We prune G as in Lem. 5, compute all intersections in time $O(n \log n)$ using a sweepline approach and resolve them as shown in Sec 3.2, to get a planar graph [2]. Now, construct the oracle by Thm. 1. □

Acknowledgements: This work is supported by GIF project 1161 & DFG project MU/3501/1. We also thank Günter Rote for valuable comments.

References

[1] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2 edition, 2001.

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

[3] M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F. Meyer auf der Heide, H. Rohnert, and R. E. Tarjan. Dynamic Perfect Hashing: Upper and Lower Bounds. *SIAM J. Comput.*, 23(4):738–761, August 1994.

[4] M. Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *J. ACM*, 51(6):993–1024, November 2004.