Testing Transmission Graphs for Acyclicity*

Haim Kaplan[1], Katharina Klost[2], Wolfgang Mulzer[2], Liam Roditty[3], and Micha Sharir[1]

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

Abstract

Let S be a set of n *point sites* in the plane, such that each site $s \in S$ has an *associated radius* $r_s > 0$. The *transmission graph* on S, denoted $T(S)$, is the directed graph with vertex set S where st is a directed edge if and only if $|st| \leq r_s$, i.e., if t lies in the disk D_s with center s and radius r_s. A basic question is to decide whether $T(S)$ is *acyclic*, i.e., whether $T(S)$ does not contain a directed cycle. We show that if our notion of directed cycle also includes cycles with two edges, then this problem can be solved in $O(n \log n)$ expected time, independent of the number of edges in $T(S)$.

Along the way, we encounter a batched range searching problem that may be interesting in its own right: given $O(n)$ *query triples* of the form (p, r_1, r_2), with $p \in \mathbb{R}^2$ and $0 < r_1 < r_2$, report for every query (p, r_1, r_2) one site $s \in S$ with $p \in D_s$ and $r_s \in [r_1, r_2)$, if it exists. We show how to solve this range searching problem in $O(n \log n)$ expected time.

1 Introduction

Transmission graphs are a popular model for directed sensor networks with different transmission radii (see, e.g., [8] and the references therein). We are given a set S of n *point sites* in the plane, representing the locations of the sensors. Each site $s \in S$ has an *associated radius* $r_s > 0$ that models the transmission strength of the corresponding sensor. The *disk for the site* s, denoted D_s, is the disk with center s and radius r_s. The *directed transmission graph* $T(S)$ has vertex set S and a directed edge from a site s to a site t if and only if $t \in D_s$, i.e., if the sensor s can reach the sensor t. Throughout, we will assume that S is in *general position*, which means that no site lies on the disk boundary of any other site and that all associated radii are pairwise distinct. Even though transmission graphs may contain $\Omega(n^2)$ edges, it turns out that many problems on them can be solved without explicitly generating all those edges [7, 8].

Here, we consider the basic problem of testing whether $T(S)$ contains a *directed cycle*. This is a sequence $s_1, \ldots, s_k$ of $k \geq 2$ sites such that $s_1 \in D_{s_k}$ and such that $s_{i+1} \in D_{s_i}$, for $i = 1, \ldots, k-1$. There are two reasonable variants of this question, depending on whether we allow the case $k = 2$ or insist on $k \geq 3$. We show that if $k = 2$ is permitted, this problem can be solved in $O(n \log n)$ expected time, independent of the number of edges.

Our algorithm is based on the observation that if $T(S)$ contains a directed cycle, then it must contain a directed cycle with two edges. Furthermore, we identify a batched range query problem that may be interesting by itself. Given n disks in the plane, we want to efficiently answer a batch of n queries, each consisting of a point and a radius range. For

* Partially supported by ERC STG 757609 and GIF grant 1367/2015.

35th European Workshop on Computational Geometry, Utrecht, The Netherlands, March 18–20, 2019.
This is an extended abstract of a presentation given at EuroCG'19. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

each such query, we want to find a disk whose radius lies in the given range and that contains the given query point. Kaplan et al. [7] considered a specialized version of this range query, using a nearest neighbor data structure distributed on a balanced binary search tree. Their solution achieves worst-case running time $O(n \log^2 n)$. Another related query structure is due to Imai et al. [6]. Their structure stores a set of n disks. The preprocessing time is $O(n \log n)$, and it can find for any point $p \in \mathbb{R}^2$ in $O(\log n)$ time a disk that contains p, if it exists. Our structure uses a similar high-level approach as the structure of Kaplan et al. [7]. To achieve a better running time, we lift the problem to $\mathbb{R}^3$ and draw on previous results on convex hulls and the intersection of convex polyhedra.

2 Range Queries

We begin by describing our batched range searching problem. The setting is as follows: let S be a set of n sites in $\mathbb{R}^2$, each with an associated radius $r_s > 0$. Let D_s be the disk with radius r_s and center s. We will consider the following query problem:

(R) We are given $O(n)$ query triples (p, r_1, r_2), where $p \in \mathbb{R}^2$ and $r_2 \geq r_1 > 0$. For every query triple, we would like to find a site $s \in S$ with $r_s \in [r_1, r_2)$ and $p \in D_s$, if it exists.

2.1 Canonical Intervals, Paths, and Nodes

The queries in (R) concern sites whose associated radii lie in given intervals. Just as in range trees [1, Chapter 5], we subdivide each such *query interval* $[r_1, r_2)$ into $O(\log n)$ pieces from a fixed set of *canonical intervals*. For this, we build a balanced binary tree B on S. For an example of the tree, illustrating the concepts introduced in this section see Figure 1 The leaves of B are the sites of S, sorted from left to right by increasing associated radius. The tree B has n leaves, $O(n)$ vertices, and height $O(\log n)$. For each vertex $v \in B$, let $\mathcal{I}_v$ be the sorted list of sites in the leaves of the subtree rooted at v. We call the sets $\mathcal{I}_v$, for $v \in B$, the *canonical intervals* of S. There are $O(n)$ of them.

Next, we define *canonical paths* and *canonical nodes* for a query $q = (p, r_1, r_2)$. For a radius $r > 0$, the *predecessor* of r in S is the site $s \in S$ with the largest radius $r_s \leq r$. The *proper predecessor* of r is the site $s \in S$ with the largest radius $r_s < r$. The *successor* and *proper successor* of r are defined analogously. Let $[r_1, r_2)$ be the query interval of q. We consider the path π_1 in B from the root to the leaf with the proper predecessor t_1 of r_1 and the path π_2 in B from the root to the leaf with the successor t_2 of r_2. If t_1 does not exist, we take π_1 as the left spine of B, and if t_2 does not exist, we take π_2 as the right spine of B. Then, π_1 and π_2 are called the *canonical paths* for q. The set of *canonical nodes* for q is defined as follows: for each vertex v in $\pi_1 \setminus \pi_2$, we include the right child of v if it is not in π_1, and for each v in $\pi_2 \setminus \pi_1$, we include the left child of v if it is not in π_2. Furthermore, we include the last node of π_1 if t_1 does not exist, and the last node of π_2 if t_2 does not exist.

▶ **Lemma 2.1.** *The total size of the canonical intervals is $O(n \log n)$, and the tree B together with the sorted canonical intervals can be built in $O(n \log n)$ time. For any query q, there are $O(\log n)$ canonical nodes, and they can be found in $O(\log n)$ time. The canonical intervals for the canonical nodes of q constitute a partition of the query interval for q.*

Proof. Since a site $s \in S$ appears in $O(\log n)$ canonical intervals, the total size of the canonical intervals is $O(n \log n)$. To construct B, we sort S according to the associated radii r_s, and we build B on top of the sorted list. To find the sorted canonical intervals, we perform a bottom-up traversal of B, obtaining the canonical interval for each internal node by copying and joining the canonical intervals of its children.

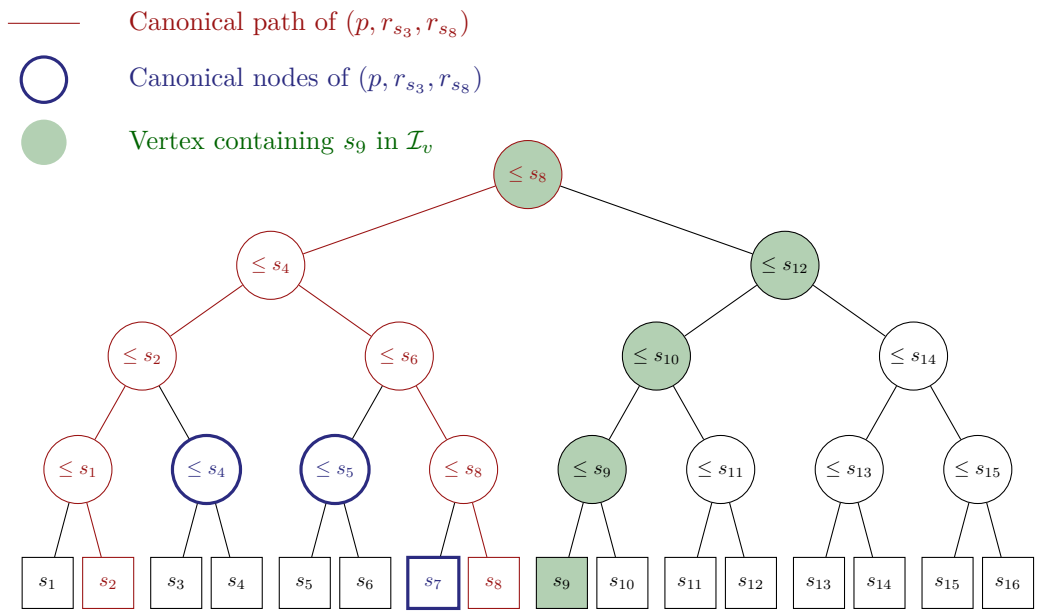

Figure 1 An example of canonical intervals, paths, and nodes.

The bound on the number of canonical nodes for q follows, since B has height $O(\log n)$. To find them, we trace the canonical paths for q in B. The partition property holds directly by construction. ◀

2.2 The Query Procedure

In order to solve a batch of queries of type (R), we build a dedicated search structure for each canonical interval, and we solve all queries that encompass one canonical interval in a single go. This means that we have a set of disks in the plane and a set of query points, and we need to determine for each query point whether it is contained in the union of the given disks. To solve this batched query efficiently, we exploit the well-known *lifting map*. Let $U = \{(x, y, z) \mid x^2 + y^2 = z\}$ be the three-dimensional unit paraboloid. For a point $p \in \mathbb{R}^2$, the lifted version $\hat{p}$ of p is its vertical projection onto U. Each disk D_s, for $s \in S$, is mapped to an upper halfspace $\widehat{D}_s$, so that the projection of $\widehat{D}_s \cap U$ onto the xy-plane is the set $\mathbb{R}^2 \setminus D_s$;[1] see Figure 2. Now, the union of a set of disks in $\mathbb{R}^2$ is represented as the intersection of the lifted upper halfspaces in $\mathbb{R}^3$.

▶ **Lemma 2.2.** *The range searching problem (R) can be solved in $O(n \log n)$ expected time.*

Proof. For each $v \in B$, we construct a three-dimensional representation of the union of the disks in the canonical interval $\mathcal{I}_v$. As explained, this is the intersection $\mathcal{E}_v$ of the lifted three-dimensional halfspaces $\widehat{D}_s$, for $s \in \mathcal{I}_v$. The intersection of two three-dimensional convex polyhedra with a total of m vertices can be computed in $O(m)$ time [2, 3]. Therefore, we can construct all the polyhedra $\mathcal{E}_v$, for $v \in B$, in overall $O(n \log n)$ time, by a bottom-up traversal of B (by Lemma 2.1, the total number of vertices of these polyhedra is $O(n \log n)$).

For the batched query processing, we computed a polytope $\widehat{Q}_v$ for each $v \in B$. The polytope $\widehat{Q}_v$ is obtained by determining all the points p that appear in a query (p, r_1, r_2)

[1] This halfspace is bounded by the plane $z = 2x_s x - x_s^2 + 2y_s y - y_s^2 + r_s^2$, where $s = (x_s, y_s)$.

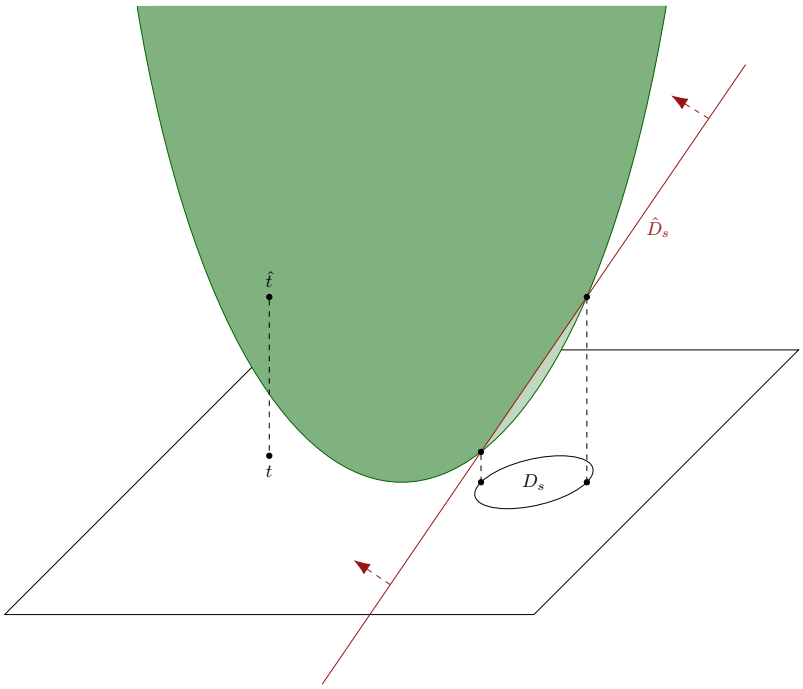

Figure 2 Lifting disks and points. For $\hat{D}$ only the bounding plane is shown.

that has v as a canonical node, lifting those point points p to their three-dimensional representations $\hat{p}$, and taking the convex hull of the resulting three-dimensional point set. The lifted query points all lie on the unit paraboloid U, so every lifted query point appears as a vertex on $\widehat{Q}_v$. To find all polytopes $\widehat{Q}_v$, for $v \in B$, efficiently, we proceed as follows: let A be the three-dimensional point set obtained by taking all points that appear in a query and by lifting them onto the unit paraboloid. We compute the convex hull of A in $O(n \log n)$ time. Then, for each $v \in B$, we find the convex hull of all lifted queries that have v in their canonical path. This can be done in $O(n \log n)$ total expected time by a top-down traversal of B. We already have the polytope for the root of B. To compute the polytope for a child node, given that the polytope for the parent node is available, we use the fact that for any polytope $\mathcal{E}$ in $\mathbb{R}^3$ with m vertices, we can compute the convex hull of any subset of the vertices of $\mathcal{E}$ in $O(m)$ expected time [4]. Once we have for each $v \in B$ the convex hull of the lifted query points that have v on their canonical *path*, we can compute for each $v \in B$ the polytope $\widehat{Q}_v$ that is the convex hull of the lifted query points that have v as a canonical *node*. For this, we consider the canonical path polytope stored at the parent node of v, and we again use the algorithm from [4] to extract the convex hull for the lifted query points that have v as a canonical node.

Now that the polyhedra $\widehat{Q}_v$ and the polytopes $\mathcal{E}_v$ are available, for all $v \in B$, we can answer the queries as follows: for each node $v \in B$, we must find the vertices of $\widehat{Q}_v$ that do not lie inside of $\mathcal{E}_v$. These are exactly the vertices of $\widehat{Q}_v$ that are not vertices of $\widehat{Q}_v \cap \mathcal{E}_v$. As mentioned, the intersections $\widehat{Q}_v \cap \mathcal{E}_v$ can be found in linear time for each node $v \in B$, for a total time $O(n \log n)$, and once the intersection is available, we can easily find all vertices $\hat{p}$ of $\widehat{Q}_v$ that are not vertices of $\widehat{Q}_v \cap \mathcal{E}_v$ (e.g., using radix sort). For any such vertex, we need to find an answer for the corresponding query, if is has not yet been found. For this, we create a data structure that supports ray shooting queries on $\mathcal{E}_v$ in $O(\log n)$ time. This

can be done in $O(|\mathcal{E}_v|)$ time [5], so the overall construction time is $O(n \log n)$. We answer a vertical ray shooting query for the query point p in $\mathcal{E}_v$ in $O(\log n)$ to find a disk in the given radius range that contains p. Since this is done at most once for each query triple, the overall time remains $O(n \log n)$, as desired. ◀

3 Testing for Acyclicity

Now we consider the problem of testing if a given transmission graph is acyclic, where we allow cycles with two edges. Let s, t be two sites such that the edges st and ts both exist in $T(S)$. We call s, t a *double edge*. Double edges in a transmission graph can be characterized by certain configurations between sites:

▶ **Lemma 3.1.** *Let $s, t \in S$ with $t \in D_s$ and $r_s \leq r_t$. Then s, t is a double edge in $T(S)$.*

Proof. The edge ts exists, since $t \in D_s$, or, equivalently, $|st| \leq r_s$. By the second assumption, we get $|st| \leq r_s \leq r_t$, and thus both edges exist. ◀

Now we use Lemma 3.1 to reduce the problem of deciding if $T(S)$ is acyclic to finding double edges in $T(S)$.

▶ **Lemma 3.2.** *A transmission graph $T(S)$ contains a cycle if and only if it contains a double edge.*

Proof. If $T(S)$ contains a double edge, we already have a cycle of length two. Now let $C = s_1, \ldots, s_k$ be a cycle in $T(S)$, such that s_1 is the site of minimum radius in C. We have $s_2 \in D_{s_1}$ by the definition of C and $r_{s_1} \leq r_{s_2}$ by assumption. By Lemma 3.1, we have that s_1, s_2 is the desired double edge. ◀

We use the range query data structure from Section 2.2 to check if a given transmission graph is acyclic.

▶ **Lemma 3.3.** *Let $T(S)$ be a transmission graph on a set S with n sites. In $O(n \log n)$ expected time, we can check if $T(S)$ is acyclic.*

Proof. By Lemma 3.2, it suffices to check if $T(S)$ contains a double edge. We can do this by using the range query data structure as follows. Let $r_{\min}$ be the minimum radius of any input disk. For each site t, we create a query triple $(t, r_{\min}, r_t)$ and perform a batched range query (R) with these triples on all sites. If there is a site t for which the query returns a value, we have, by the choice of the queries, that t is contained in the disk for some smaller site. This directly implies the existence of a double edge by Lemma 3.1. In the other case, we know that there are no double edges and we can conclude that the graph is acyclic. ◀

4 Conclusion

We showed how to check a transmission graph for acyclicity in $O(n \log n)$ time, when we include cycles of length two. Right now, we are working on the same setting, while disallowing these cycles.

Regarding the range query data structure, it would be interesting to derandomize it within the same time bound. Furthermore, at the moment we are only able to find one disk containing each given point. In particular with regard to applications in transmission graphs, it would be useful to be able to find all disks containing each query point, in overall $O((n + k) \log n)$ time, where k is the overall number of disks reported.

References

1 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark H. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, third edition, 2008.

2 Timothy M. Chan. A simpler linear-time algorithm for intersecting two convex polyhedra in three dimensions. *Discrete Comput. Geom.*, 56(4):860–865, December 2016.

3 Bernard Chazelle. An optimal algorithm for intersecting three-dimensional convex polyhedra. *SIAM J. Comput.*, 21(4):671–696, 1992.

4 Bernard Chazelle and Wolfgang Mulzer. Computing hereditary convex structures. *Discrete Comput. Geom.*, 45(4):796–823, 2011.

5 David P. Dobkin and David G. Kirkpatrick. Fast detection of polyhedral intersections. In *Proc. 9th Internat. Colloq. Automata Lang. Program. (ICALP)*, pages 154–165, 1982.

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

7 Haim Kaplan, Katharina Klost, Wolfgang Mulzer, and Liam Roditty. Finding the girth in disk graphs and a directed triangle in transmission graphs. In *Proc. 34th European Workshop Comput. Geom. (EWCG)*, pages 68:1–6, 2018.

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