Triangles and Girth in Disk Graphs and Transmission Graphs

Haim Kaplan
School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel
haimk@tau.ac.il

Katharina Klost
Institut für Informatik, Freie Universität Berlin, 14195 Berlin, Germany
kathklost@inf.fu-berlin.de

Wolfgang Mulzer
Institut für Informatik, Freie Universität Berlin, 14195 Berlin, Germany
mulzer@inf.fu-berlin.de

Liam Roditty
Department of Computer Science, Bar Ilan University, Ramat Gan 5290002, Israel
liamr@macs.biu.ac.il

Paul Seiferth
Institut für Informatik, Freie Universität Berlin, 14195 Berlin, Germany
pseiferth@inf.fu-berlin.de

Micha Sharir
School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel
michas@tau.ac.il

Abstract

Let $S \subset \mathbb{R}^2$ be a set of n *sites*, where each $s \in S$ has an *associated radius* $r_s > 0$. The *disk graph* $D(S)$ is the undirected graph with vertex set S and an undirected edge between two sites $s, t \in S$ if and only if $|st| \leq r_s + r_t$, i.e., if the disks with centers s and t and respective radii r_s and r_t intersect. Disk graphs are used to model sensor networks. Similarly, the *transmission graph* $T(S)$ is the directed graph with vertex set S and a directed edge from a site s to a site t if and only if $|st| \leq r_s$, i.e., if t lies in the disk with center s and radius r_s.

We provide algorithms for detecting (directed) triangles and, more generally, computing the length of a shortest cycle (the *girth*) in $D(S)$ and in $T(S)$. These problems are notoriously hard in general, but better solutions exist for special graph classes such as planar graphs. We obtain similarly efficient results for disk graphs and for transmission graphs. More precisely, we show that a shortest (Euclidean) triangle in $D(S)$ and in $T(S)$ can be found in $O(n \log n)$ expected time, and that the (weighted) girth of $D(S)$ can be found in $O(n \log n)$ expected time. For this, we develop new tools for batched range searching that may be of independent interest.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Graph algorithms analysis

Keywords and phrases disk graph, transmission graph, triangle, girth

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.64

Related Version A full version is available on the arXiv (`https://arxiv.org/abs/1907.01980`).

Funding Supported in part by grant 1367/2016 from the German-Israeli Science Foundation (GIF).
Wolfgang Mulzer: Partially supported by ERC STG 757609.
Paul Seiferth: Partially supported by DFG grant MU/3501/1.
Micha Sharir: Partially supported by ISF Grant 892/13, the Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11), the Blavatnik Research Fund in Computer Science at Tel Aviv University, and the Hermann Minkowski-MINERVA Center for Geometry at Tel Aviv University.

Acknowledgements We like to thank Günther Rote and Valentin Polishchuk for helpful comments.

27th Annual European Symposium on Algorithms (ESA 2019).
Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 64; pp. 64:1–64:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Given a graph G with n vertices and m edges, does G contain a *triangle* (a cycle with three vertices)? This is one of the most basic algorithmic questions in graph theory, and many other problems reduce to it [12, 21]. The best known algorithms use fast matrix multiplication and run in either $O(n^\omega)$ time or in $O\big(m^{2\omega/(\omega+1)}\big)$ time, where $\omega < 2.37287$ is the matrix multiplication exponent [1, 10, 12]. Despite decades of research, the best available "combinatorial" algorithm[1] needs $O\big(n^3 \operatorname{polyloglog}(n)/\log^4 n\big)$ time [22], only slightly better than checking all vertex triples. This lack of progress can be explained by a connection to Boolean matrix multiplication (BMM): if there is a truly subcubic combinatorial algorithm for finding triangles, there is also a truly subcubic combinatorial algorithm for BMM [21]. Itai and Rodeh [12] reduced computing the *girth* (the length of a shortest cycle) of an unweighted undirected graph to triangle detection. For integer edge weights, Roditty and V. Williams [19] gave an equivalence between finding a minimum weight cycle (the weighted girth) and finding a minimum weight triangle.

For the special case of *planar* graphs, significantly better algorithms are known. Itai and Rodeh [12] and, independently, Papadimitriou and Yannakakis [17] showed that a triangle can be found in $O(n)$ time, if it exists. Chang and Lu [7] presented an $O(n)$ time algorithm for computing the girth. The weighted girth can be found in $O(n \log\log n)$ time both in an undirected and in a directed planar graph [15, 16].

In computational geometry, there are two noteworthy graph classes that generalize planar graphs: *disk graphs* and *transmission graphs*. We are given a set S of n planar point *sites*. Each $s \in S$ has an *associated radius* $r_s > 0$ and an *associated disk* D_s with center s and radius r_s. The *disk graph* $D(S)$ is the undirected graph on S where two sites $s, t \in S$ are adjacent if and only if D_s and D_t intersect, i.e., $|st| \leq r_s + r_t$, where $|\cdot|$ is the Euclidean distance. In a *weighted disk graph*, the edges are weighted according to the Euclidean distance between their endpoints. The *transmission graph* $T(S)$ is the directed graph on S where there is an edge from s to t if and only if t lies in D_s, i.e., $|st| \leq r_s$. Again, there is a weighted variant. Both graph classes have received a lot of attention, as they give simple and natural theoretical models for geometric sensor networks (see, e.g., [13, 14]).

Motivated by the vastly better algorithms for planar graphs, we investigate triangle detection and girth computation in disk graphs and transmission graphs. We will see that in a disk graph, a triangle can be found in $O(n \log n)$ time, using a simple geometric observation to relate disk graphs and planar graphs. By a reduction from ε-CLOSENESS [18], this is optimal in the algebraic decision tree model, a contrast to planar graphs, where $O(n)$ time is possible. Our method generalizes to finding a shortest triangle in a weighted disk graph in $O(n \log n)$ expected time. Moreover, we can compute the unweighted and weighted girth in a disk graph in $O(n \log n)$ time, with a deterministic algorithm for the unweighted case and a randomized algorithm for the weighted case. The latter result requires a method to find a shortest cycle that contains a given vertex. Finally, we provide an algorithm to detect a directed triangle in a transmission graph in $O(n \log n)$ expected time. For this, we study the geometric properties of such triangles in more detail, and we develop several new techniques for batched range searching that might be of independent interest, using linearized quadtrees and three-dimensional polytopes to test for containment in the union of planar disks. As before, this algorithm extends to the weighted version. We will assume *general position*, meaning that all edge lengths (and more generally shortest path distances) are pairwise

[1] An algorithm is "combinatorial" if it does not need algebraic manipulations to achieve its goal.

distinct, that no site lies on a disk boundary, and that all radii are pairwise distinct. Due to space reasons, some proofs in this extended abstract are only sketched. The complete proofs can be found in the full version of this paper.

2 Finding a (Shortest) Triangle in a Disk Graph

We would like to decide if a given disk graph contains a triangle. If so, we would also like to find a triangle of minimum Euclidean perimeter.

2.1 The Unweighted Case

The following property of disk graphs, due to Evans et al. [9], is the key to our algorithm. For a proof refer to the full version.

▶ **Lemma 2.1.** *Let $D(S)$ be a disk graph that is not plane, i.e., the embedding that represents each edge by a line segment between its endpoints has two segments that cross in their relative interiors. Then, there are three sites whose associated disks intersect in a common point.*

If $D(S)$ is not plane, it contains a triangle by Lemma 2.1. If $D(S)$ is plane, we can construct it explicitly and then search for a triangle in $O(n)$ time [12, 17]. To check whether $D(S)$ is plane, we begin an explicit construction of $D(S)$ and abort if we discover too many edges.

▶ **Theorem 2.2.** *Let $D(S)$ be a disk graph on n sites. We can find a triangle in $D(S)$ in $O(n \log n)$ worst-case time, if it exists.*

Proof (Sketch). We compute the edges of $D(S)$, using a sweepline approach. If at some point we find more than $3n - 6$ edges, we stop and proceed with the partial graph (which is not plane). If there are at most $3n - 6$ edges, we check for edge crossings, again by a plane sweep. If there is a crossing, we report the resulting triangle in $O(1)$ time. If not, $D(S)$ is plane and we determine if it contains a triangle in $O(n)$ time [12, 17]. ◀

2.2 The Weighted Case

Suppose the edges in $D(S)$ are weighted by their Euclidean lengths. We would like to find a triangle of minimum perimeter, i.e., of minimum total edge length. For this, we solve the decision problem: given $W > 0$, does $D(S)$ contain a triangle with perimeter at most W? Once a decision algorithm is available, the optimization problem can be solved with Chan's randomized geometric optimization framework [5].

To decide if $D(S)$ contains a triangle with perimeter at most W, we use a grid with diameter $W/3$. We look for triangles whose vertices lie in a single grid cell, using the algorithm from Section 2.1. If no cell contains such a triangle, then $D(S)$ will be sparse and we will need to check only $O(n)$ further triples. Details follow.

Set $\ell = W/(3\sqrt{2})$. Let G_1 be the grid whose cells are pairwise disjoint, axis-parallel squares with side length ℓ, aligned so that the origin $(0, 0)$ is a vertex of G_1. The cells of G_1 have diameter $\sqrt{2} \cdot \ell = W/3$, so any triangle whose vertices lie in a single cell has perimeter at most W. We make three additional copies G_2, G_3, G_4 of G_1, and we shift them by $\ell/2$ in the x-direction, in the y-direction, and in both the x- and y-directions, respectively. In other words, G_2 has $(\ell/2, 0)$ as a vertex, G_3 has $(0, \ell/2)$ as a vertex, and G_4 has $(\ell/2, \ell/2)$ as a vertex, see Figure 1. This ensures that if all edges in a triangle are "short", the triangle lies in a single grid cell.

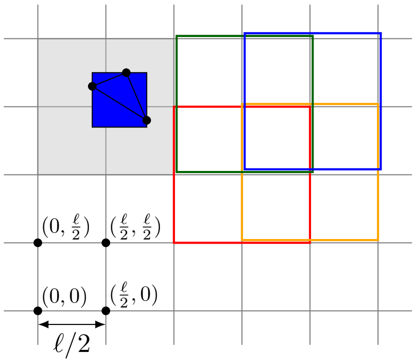


Figure 1 The four shifted grids, with a cell from each grid shown in red, orange, green, and blue, respectively. Every square with side length at most $\ell/2$ is wholly contained in a single grid cell.

▶ **Lemma 2.3.** *Let Δ be a triangle formed by three vertices $a, b, c \in \mathbb{R}^2$ such that each edge of Δ has length at most $\ell/2$. There is a cell $\sigma \in \bigcup_{i=1}^4 G_i$ with $a, b, c \in \sigma$.*

Proof. We can enclose Δ with a square of side length $\ell/2$. This square must be completely contained in a cell of one of the four grids, see Figure 1. ◀

We go through all nonempty grid cells $\sigma \in \bigcup_{i=1}^4 G_i$, and we search for a triangle in the disk graph $D(S \cap \sigma)$ induced by the sites in σ, with Theorem 2.2. Since each site lies in $O(1)$ grid cells, and since we can compute the grid cells for a given site in $O(1)$ time (using the floor function), the total running time is $O(n \log n)$. If a triangle is found, we return YES, since the cells have diameter $W/3$ and thus such a triangle has perimeter at most W. If no triangle is found, Lemma 2.3 implies that any triangle in $D(S)$ has one side of length more than $\ell/2$ and hence at least one vertex with associated radius at least $\ell/4$. We call a site $s \in S$ *large* if $r_s > \ell/4$. A simple volume argument bounds the number of large sites in a grid cell.

▶ **Lemma 2.4.** *Let $\sigma \in \bigcup_{i=1}^4 G_i$ be a nonempty grid cell, and suppose that $D(S \cap \sigma)$ does not contain a triangle. Then σ contains at most 18 large sites.*

Proof. Suppose σ contains at least 19 large sites. We cover σ with 3×3 congruent squares of side length $\ell/3$. Then, at least one square τ contains at least $\lceil 19/9 \rceil = 3$ large sites. The associated radius of a large site is more than $\ell/4$ and each square has diameter $(\sqrt{2}/3)\ell < \ell/2$, so the large sites in τ form a triangle in $D(S \cap \sigma)$, a contradiction. ◀

Let $\sigma \in G_i$, $i \in \{1, \dots, 4\}$, be a grid cell. The *neighborhood* $N(\sigma)$ of σ is the 5×5 block of cells in G_i centered at σ. Since the diameter of a grid cell is $W/3$, any two sites $u, v \in S$ that form a triangle of perimeter at most W with a site $s \in S \cap \sigma$ must be in $N(\sigma)$. Let $S_\ell \subseteq S$ denote the large sites. At this stage, we know that any triangle in $D(S)$ has at least one vertex in S_ℓ. By Lemma 2.4, for any $\sigma \in \bigcup_{i=1}^4 G_i$, we have $|\cup_{\tau \in N(\sigma)} \tau \cap S_\ell| = O(1)$. Thus, to detect a triangle of perimeter at most W with at least two large vertices, we proceed as follows: for each non-empty cell $\sigma \in G_i$, iterate over all large sites s in σ, over all large sites t in $N(\sigma)$, and over all (not necessary large) sites u in $N(\sigma)$. Check whether stu is a triangle of perimeter at most W. If so, return YES. Since the sites in each grid cell are examined $O(1)$ times for $O(1)$ pairs of large sites, the total time is $O(n)$.

It remains to detect triangles of perimeter at most W with exactly one large vertex. We iterate over all grid cells $\sigma \in \bigcup_{i=1}^4 G_i$, and we compute $D(S \cap \sigma)$. Since $D(S \cap \sigma)$ contains no triangle, Lemma 2.1 shows that $D(S \cap \sigma)$ is plane, has $O(|S \cap \sigma|)$ edges and can be constructed in time $O(|S \cap \sigma| \log |S \cap \sigma|)$. For every edge $st \in D(S \cap \sigma)$ with both endpoints in $S \setminus S_\ell$, we iterate over all large sites u in $N(\sigma)$ and we test whether stu makes a triangle in $D(S)$ with perimeter at most W. If so, we return YES. By Lemma 2.4, this takes $O(|S \cap \sigma|)$ time, so the total running time is $O(n \log n)$. If there is a triangle of perimeter at most W with exactly one vertex in S_ℓ, the edge with both endpoints in $S \setminus S_\ell$ has length at most $\ell/2$ and thus must lie in a single grid cell $\sigma \in \bigcup_{i=1}^{4} G_i$. To summarize:

▶ **Lemma 2.5.** *Let $D(S)$ be a disk graph on n sites, and let $W > 0$. We can decide in $O(n \log n)$ worst-case time whether $D(S)$ contains a triangle of perimeter at most W.*

Now, Chan's framework [5] gives a randomized optimization algorithm with no additional overhead. The details can be found in the full version.

▶ **Theorem 2.6.** *Let $D(S)$ be a weighted disk graph on n sites. We can compute a shortest triangle in $D(S)$ in $O(n \log n)$ expected time, if one exists.*

3 Computing the Girth of a Disk Graph

We extend the results from Section 2 to the girth. The unweighted case is easy: if $D(S)$ is not plane, the girth is 3, by Lemma 2.1. If $D(S)$ is plane, we use the algorithm for planar graphs [7]. The weighted case is harder. If $D(S)$ is plane, we use the algorithm for planar graphs [15]. If not, Theorem 2.6 gives a shortest triangle Δ in $D(S)$. However, there could be cycles with at least four edges that are shorter than Δ. To address this, we use Δ to split $D(S)$ into sparse pieces where a shortest cycle can be found efficiently.

3.1 The Unweighted Case

Chang and Lu [7, Theorem 1.1] showed how to find the girth of an unweighted planar graph with n vertices in $O(n)$ time. Hence, we obtain a simple extension of Theorem 2.2.

▶ **Theorem 3.1.** *Let $D(S)$ be a disk graph for a set S of n sites. We can compute the unweighted girth of $D(S)$ in $O(n \log n)$ worst-case time.*

Proof. We proceed as in Theorem 2.2. If $D(S)$ is not plane, the girth is 3. If $D(S)$ is plane, we apply the algorithm of Chang and Lu [7, Theorem 1.1] to an explicit representation of $D(S)$. ◀

3.2 The Weighted Case

We describe how to find the shortest cycle through a given vertex in a weighted graph with certain properties. This is then used to compute the weighted girth of a disk graph.

Let G be a graph with nonnegative edge weights so that all shortest paths and cycles in G have pairwise distinct lengths and so that for all edges uv, the shortest path from u to v is the edge uv. We present a deterministic algorithm that, given G and a vertex s, computes the shortest cycle in G containing s, if it exists. A simple randomzied algorithm can also be found in Yuster [23, Section 2]. The next lemma states a structural property of the shortest cycle through s. It resembles Lemma 1 of Roditty and V. Williams [19] that deals with an overall shortest cycle in G.[2] For details on the proof see the full version.

[2] Even though this seems to be a simple fact, we could not locate a previous reference for this.

▶ **Lemma 3.2.** *The shortest cycle in G that contains s consists of two paths in the shortest path tree of s, and one additional edge.*

▶ **Theorem 3.3.** *Let $G = (V, E)$ be a weighted graph with n vertices and m edges that has the properties given at the beginning of this section. Let $s \in V$. We can compute the shortest cycle in G that contains s in $O(n \log n + m)$ time, if it exists.*

Proof (Sketch). We find the shortest path tree T for s, and we identify the edges that close a cycle in T containing s. We check all candidate cycles to find the shortest. Correctness follows from Lemma 3.2. The running time for finding the shortest path tree dominates the rest of the algorithm. ◀

Let $D(S)$ be a weighted disk graph on n sites. A careful combination of the tools developed so far gives an algorithm for the weighted girth of $D(S)$.

▶ **Theorem 3.4.** *Given a weighted disk graph $D(S)$ on n sites, we can compute the weighted girth of $D(S)$ in $O(n \log n)$ expected time.*

Proof (Sketch). We find the shortest triangle in $D(S)$ in $O(n \log n)$ expected time, if it exists (Theorem 2.6). If $D(S)$ has no triangle, it is plane by Lemma 2.1, and we construct $D(S)$ in $O(n \log n)$ time. We find the girth of $D(S)$ using the algorithm of Łącki and Sankowski [15, Section 5], in $O(n \log \log n)$ time. If $D(S)$ has a triangle, let W be the perimeter of the shortest triangle in $D(S)$. Then, W is an upper bound for the girth of $D(S)$. We set $\ell = W/(3\sqrt{2})$, and we call a site $s \in S$ *large*, if $r_s \geq \ell/4$, and we write $S_\ell \subseteq S$ for the set of large sites. Let G be the grid with side length ℓ and the origin $(0, 0)$ as a vertex.

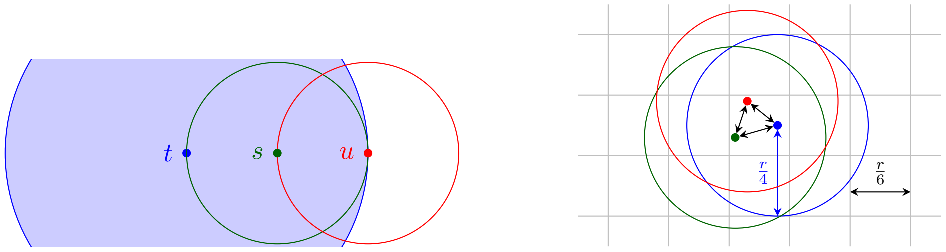
We must check whether $D(S)$ contains a cycle with more than three vertices and length less than W. By our choice of ℓ, the graph $D(S \setminus S_\ell)$ induced by $S \setminus S_\ell$ is plane. Thus, we can find a cycle in $D(S \setminus S_\ell)$ with the algorithm of Łącki and Sankowski [15, Section 5], in $O(n \log \log n)$ time. It remains to test cycles with at least one large site. The choice of ℓ implies that each cycle of length at most W is contained in a grid neighborhood of constant size. Since there are $O(1)$ large sites in each neighborhood, the induced graph in each neighborhood has linear size. We check the remaining cycles by applying Theorem 3.3 to all large sites in each neighborhood. ◀

4 Finding a Triangle in a Transmission Graph

Given a transmission graph $T(S)$ on n sites, we want to decide if $T(S)$ contains a directed triangle. We first describe an inefficient algorithm for this problem, and then we will explain how to implement it in $O(n \log n)$ expected time.

The algorithm iterates over each directed edge $e = st$ with $r_t \geq r_s$, and it performs two tests: first, for each directed edge tu with $r_u \geq r_t/2$, it checks if us is an edge in $T(S)$, i.e., if $s \in D_u$. If so, the algorithm reports the triangle stu. Second, the algorithm tests if there is a site u such that $r_u \in [r_s, r_t/2)$ and such that us is an edge in $T(S)$, i.e., such that $s \in D_u$. If such a u exists, it reports the triangle stu. If both tests fail for each edge e, the algorithm reports that $T(S)$ contains no triangle. The next lemma shows that the algorithm is correct.

▶ **Lemma 4.1.** *A triple stu reported by the algorithm is a triangle in $T(S)$. Furthermore, if $T(S)$ contains a triangle, the algorithm will find one.*



(a) We do not need to check $u \in D_t$.

(b) Three disks with radius at least $r/4$ in the same grid cell form a clique.

Proof (Sketch). It is easy to see that the algorithm finds a triangle if one exists. The algorithm is also sound: a triple reported by the first test is a triangle by construction, and for the second test, Figure 2a shows that $u \in D_t$, so tu is an edge of $T(S)$. ◀

There are several challenges for making the algorithm efficient. First of all, there might be many edges st with $r_t \geq r_s$. However, the following lemma shows that if there are $\omega(n)$ such edges, the transmission graph $T(S)$ must contain a triangle.

▶ **Lemma 4.2.** *There is an absolute constant α so that for any $r > 0$, if there is an $r \times r$ square σ that contains more than α sites $s \in S$ with $r_s \geq r/4$, then $T(S)$ has a directed triangle.*

Proof. We cover σ with a 6×6 grid of side length $r/6$; see Figure 2b. There are 36 grid cells. For every $s \in S \cap \sigma$ with $r_s \geq r/4$, the disk D_s completely covers the grid cell containing s. If σ contains more than $\alpha = 72$ sites s with $r_s \geq r/4$, then one grid cell contains at least three such sites. These sites form a directed triangle in $T(S)$. ◀

Thus, to implement the algorithm, we must solve two range searching problems.

(R1) EITHER determine that for every site $s \in S$, there are at most α outgoing edges st with $r_t \geq r_s/2$ and report all these edges; OR find a square σ of side length $r > 0$ that contains more than α sites $s \in S$ with $r_s \geq r/4$.

(R2) Given $O(n)$ query triples (s, r_1, r_2) with $s \in S$ and $0 < r_1 < r_2$, find a site $u \in S$ such that there is a query triple (s, r_1, r_2) with $u \neq s$, $r_u \in [r_1, r_2)$, and $s \in D_u$; or report that no such site exists.

The query (R1) indeed always has a valid outcome: suppose there is a site $s \in S$ with more than α outgoing edges st with $r_t \geq r_s/2$. Then, all the endpoints t lie in D_s, so the square σ centered at s with side length $r = 2r_s$ contains more than α sites with associated radius at least $r/4$. The next theorem shows that we can detect a triangle in $T(S)$ with linear overhead in addition to the time needed for answering (R1) and (R2). The proof is in the full version.

▶ **Theorem 4.3.** *If (R1) and (R2) can be solved in time $R(n)$ for input size n, we can find a directed triangle in a transmission graph $T(S)$ on n sites in time $R(n) + O(n)$, if it exists.*

In the next section, we will implement (R1) and (R2) in $O(n \log n)$ expected time.

▶ **Theorem 4.4.** *Let $T(S)$ be a transmission graph on n sites. We can find a directed triangle in $T(S)$ in expected time $O(n \log n)$, if it exists.*

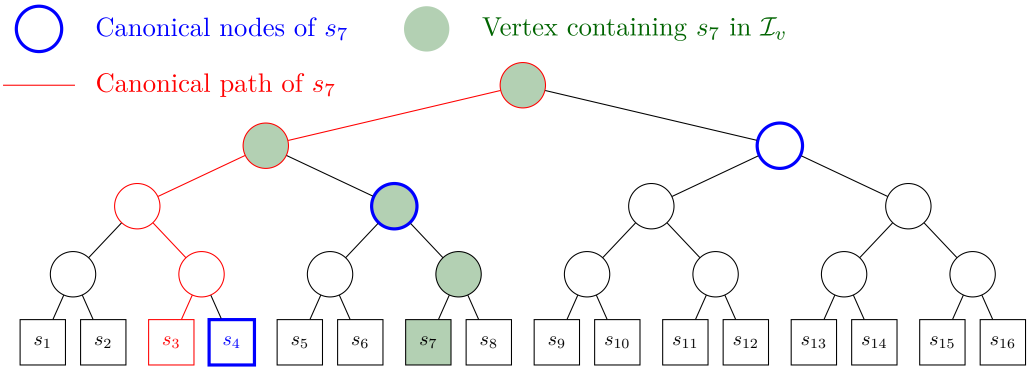


Figure 3 Example is for a query of type (R1), assuming that $r_{s_3} < r_{s_7}/2 \leq r_{s_4}$.

5 Batched Range Searching

The range queries must handle subsets of sites whose associated radii lie in certain intervals: a query s in (R1) concerns sites $t \in S$ such that $r_t \geq r_s/2$; and a query (s, r_1, r_2) in (R2) concerns sites t such that $r_t \in [r_1, r_2)$. Using a standard approach [2, 20], we subdivide each such *query interval* into $O(\log n)$ pieces from a set of *canonical intervals*. For this, we build a balanced binary tree B whose leaves are the sites of S, sorted by increasing associated radius. For each vertex $v \in B$, let the *canonical interval* $\mathcal{I}_v$ be the sorted list of sites in the subtree rooted at v. There are $O(n)$ canonical intervals.

Next, we define *canonical paths* and *canonical nodes*. For a radius $r > 0$, the (proper) *predecessor* of r is the site $s \in S$ with the largest radius $r_s \leq r$ ($r_s < r$). The (proper) *successor* of r is defined analogously. For a query s in (R1), we consider the path π in B from the root to the leaf with the proper predecessor t of $r_s/2$. If t does not exist (i.e., if $r_t \geq r_s/2$, for all $t \in S$), we let π be the left spine of B. We call π the *canonical path* for s. The *canonical nodes* for s are the right children of the nodes in π that are not in π themselves, plus possibly the last node of π, if $r_t \geq r_s/2$, for all $t \in S$, see Figure 3.

For a query (s, r_1, r_2) in (R2), 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 for the successor t_2 of r_2. Again, 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 the *canonical paths* for (s, r_1, r_2). The *canonical nodes* for (s, r_1, r_2) are defined as follows: for each vertex v in $\pi_1 \setminus \pi_2$, we take the right child of v if it is not in π_1, and for each v in $\pi_2 \setminus \pi_1$, we take the left child of v if it is not in π_1. Furthermore, we take 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. A standard argument bounds the number and total size of the canonical intervals, see the full version for the proof.

▶ **Lemma 5.1.** *The total size of the canonical intervals is $O(n \log n)$. The tree B and the canonical intervals can be built in $O(n \log n)$ time. For any query q in (R1) or (R2), 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.*

5.1 Queries of Type (R1)

We build a compressed quadtree on S, and we perform the range searches the compressed quadtree. It is possible to compute a compressed quadtree for each canonical interval without logarithmic overhead. Since Lemma 4.2 gives us plenty of freedom in choosing the squares

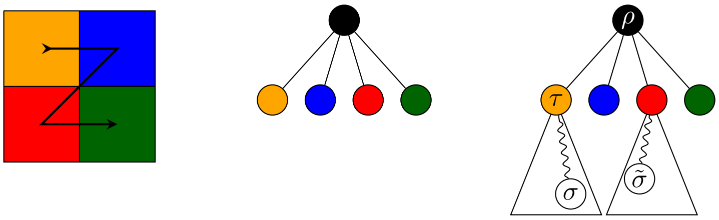


Figure 4 Z-Order. On the very right we have $\sigma \leq_Z \tau \leq_Z \tilde{\sigma}$.

for our range queries, we take squares from the grid that underlies the quadtree. This allows us to reduce the range searching problem to predecessor search in a linear list, a task that can be accomplished by one top-down traversal of B. Details follow.

Hierarchical grids, Z-order, compressed quadtrees. We translate and scale S (and the associated radii), so that S lies in the interior of the unit square $U = [0, 1]^2$ and so that all radii are at most $\sqrt{2}$. We define a sequence of hierarchical grids that subdivide U. The grid G_0 consists of the single cell U. The grid G_i, $i \geq 1$, consists of the 2^{2i} square cells with side length 2^{-i} and pairwise disjoint interiors that cover U. The hierarchical grids induce an infinite four-regular tree $\mathcal{T}$: the vertices are the cells of $\mathcal{G} = \bigcup_{i=0}^{\infty} G_i$. The unit square U is the root, and for $i = 1, \dots,$ a cell σ in G_i is the child of the cell in G_{i-1} that contains it. We make no explicit distinction between a vertex of $\mathcal{T}$ and its corresponding cell.

The *Z-order* $\leq_Z$ is a total order on the cells of $\mathcal{G}$; see [4] for more details. Let $\sigma, \tau \in \mathcal{G}$. If $\sigma \subseteq \tau$, then $\sigma \leq_Z \tau$: and if $\tau \subseteq \sigma$, then $\tau \leq_Z \sigma$, If σ and τ are unrelated in $\mathcal{T}$, let ρ be the lowest common ancestor of σ and τ in $\mathcal{T}$, and let σ' and τ' be the children of ρ with $\sigma \subseteq \sigma'$ and $\tau \subseteq \tau'$. We set $\sigma \leq_Z \tau$ if σ' is before τ' in the order shown in Figure 4; and $\tau \leq_Z \sigma$, otherwise. The next lemma shows that given $\sigma, \tau \in \mathcal{G}$, we can decide if $\sigma \leq_Z \tau$ in constant time.

▶ **Lemma 5.2** (Chapter 2 in Har-Peled [11]). *Suppose the floor function and the first differing bit in the binary representations of two given real numbers can be computed in $O(1)$ time. Then, we can decide in $O(1)$ time for two given cells $\sigma, \tau \in \mathcal{G}$ whether $\sigma \leq_Z \tau$ or $\tau \leq_Z \sigma$.*

For a site $s \in S$, let σ_s be the largest cell in $\mathcal{G}$ that contains only s. The *quadtree* for S is the smallest connected subtree of $\mathcal{T}$ that contains the root U and all cells σ_s, for $s \in S$. The *compressed quadtree* $\mathcal{C}$ for S is obtained from the quadtree by contracting any maximal path of vertices with only one child into a single edge. Vertices that were at the top of such a path are now called *compressed* vertices. The compressed quadtree for S has $O(n)$ vertices, and it can be constructed in $O(n \log n)$ time (see, e.g., [3, Appendix A] and [11]).

The *linearized compressed quadtree* $\mathcal{L}$ for S is the sorted sequence of cells obtained by listing the nodes of $\mathcal{C}$ according to a postorder traversal, were the children of a node $\sigma \in \mathcal{C}$ are visited according to the Z-order from Figure 4. The cells in $\mathcal{L}$ appear in increasing Z-order, and range searching for a given cell $\sigma \in \mathcal{G}$ reduces to a simple predecessor search in $\mathcal{L}$, as is made explicit in the following lemma.

▶ **Lemma 5.3.** *Let σ be a cell of $\mathcal{G}$, and let $\mathcal{L}$ be the linearized compressed quadtree on S. Let $\tau = \max_Z\{\rho \in \mathcal{L} \mid \rho \leq_Z \sigma\}$ be the Z-predecessor of σ in $\mathcal{L}$ ($\tau = \emptyset$, if the predecessor does not exist). Then, if $\sigma \cap \tau = \emptyset$, then also $\sigma \cap S = \emptyset$, and if $\sigma \cap \tau \neq \emptyset$, then $\sigma \cap S = \tau \cap S$.*

Proof. Let $\mathcal{C}$ be the compressed quadtree on S, and let $\mathcal{C}_\sigma = \{\tau \in \mathcal{C} \mid \tau \subseteq \sigma\}$ be the cells in $\mathcal{C}$ that are contained in σ. If $\mathcal{C}_\sigma$ is non-empty, then $\mathcal{C}_\sigma$ is a connected subtree of $\mathcal{C}$. Let τ be the root of this subtree. Then, $\tau = \max_Z\{\rho \in \mathcal{C}_\sigma\}$, and $\tau \leq_Z \sigma$. Furthermore, all other cells

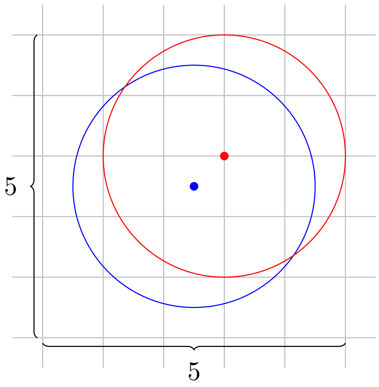


Figure 5 The neighborhood of a site has constant size.

in $\mathcal{C} \setminus \mathcal{C}_\sigma$ are either smaller than all cells in $\mathcal{C}_\sigma$ or larger than σ. Thus, τ is the Z-predecessor of σ in $\mathcal{L}$, and $\sigma \cap S = \tau \cap S \neq \emptyset$. Otherwise, if $\mathcal{C}_\sigma = \emptyset$, the Z-predecessor of σ in $\mathcal{L}$ either does not exist or is disjoint from σ. Thus, in this case, we have $\emptyset = \sigma \cap \tau = \sigma \cap S$. ◀

The search algorithm. For a site $s \in S$, we define the *neighborhood* $N(s)$ of s as all cells in $\mathcal{G}$ with side length $2^{\lfloor \log_2 r_s \rfloor}$ that intersect D_s. The neighborhood will be used to approximate D_s for the range search in the quadtrees.

▶ **Lemma 5.4.** *There is a constant β such that $|N(s)| \leq \beta$ for all $s \in S$.*

Proof. We have $r_s/2 < 2^{\lfloor \log_2 r_s \rfloor}$, and a 5×5 grid with cells of side length $r_s/2$ covers D_s, no matter where s lies; see Figure 5. Thus, the lemma holds with $\beta = 25$. ◀

We now show that a linearized compressed quadtree for each canonical interval can be found without logarithmic overhead.

▶ **Lemma 5.5.** *We can compute for each $v \in B$ the linearized quadtree $\mathcal{L}_v$ for the sites in $\mathcal{I}_v$ in $O(n \log n)$ time.*

Proof (Sketch). We traverse B and build the compressed quadtree $\mathcal{C}_v$ for $\mathcal{I}_v$, for each $v \in B$. For the root, this takes $O(n \log n)$ time. The compressed quadtree $\mathcal{C}_w$ for a child w of a node v can be found by traversing $\mathcal{C}_v$ in $O(|\mathcal{C}_v|)$ time. Then, we compute the linearized trees $\mathcal{L}_v$ by a postorder traversal of each $\mathcal{C}_v$. ◀

Using the linearized compressed quadtrees, the range searching problem can be solved by a batched predecessor search, using a single traversal of B.

▶ **Lemma 5.6.** *The range searching problem (R1) can be solved in $O(n \log n)$ time.*

Proof (Sketch). We apply Lemma 5.5 to find the linearized quadtrees for B. Let $\mathcal{Q}' = \bigcup_{s \in S} \{(\sigma, s) \mid \sigma \in N(s)\}$. We call $\mathcal{Q}'$ the set of *split queries*. They approximate the disks D_s by cells from the hierarchical grid. By Lemma 5.4, $|\mathcal{Q}'| = O(n)$. We now want to perform range queries for all the cells in the split queries. For this, we first sort the elements of $\mathcal{Q}'$ in the Z-order of their first components, in $O(n \log n)$ time. Now we store all split queries (σ, s) with all nodes $v \in B$ such that v is a canonical node of s. This can be done by one traversal over B. We call the resulting lists $\mathcal{Q}''_v$, for $v \in B$.

We iterate over all $v \in B$, and we merge the lists $\mathcal{Q}''_v$ with the lists $\mathcal{L}_v$, in Z-order. This takes $O\big(\sum_{v \in B} |\mathcal{L}_v| + |\mathcal{Q}''_v|\big) = O(n \log n)$ time. By Lemma 5.3, we obtain for each $(\sigma, s) \in \mathcal{Q}''_v$ a cell $\tau_v^{\sigma,s}$. If $\sigma \cap \tau_v^{\sigma,s} \neq \emptyset$, we know that $\sigma \cap \mathcal{I}_v = \tau_v^{\sigma,s} \cap \mathcal{I}_v$. Since these sites are all from $\mathcal{I}_v$ they all have radius at least $r_s/2$. We can find all these sites in $O(k)$ time, where k is the output size. If $k > \alpha$, we stop and report σ as a square with many sites of large radius.[3]

Otherwise, we use the sites in σ to accumulate the sites for the query disk D_s. This we can do by considering all canonical nodes of s and for each cell σ iterate over the sites contained in σ. In each such cell there are at most α sites. For each site $t \in \sigma$ we can check if $t \in D_s$. If we find a query disk D_s with more than α sites of large radius, we stop and report its enclosing square with many sites of large radius.[4] Otherwise, for each $s \in S$, we have found the at most α sites of radius at least $r_s/2$ in D_s. The whole algorithm takes $O(n \log n)$ time. ◀

5.2 Queries of Type (R2)

We use the tree structure of the canonical intervals (i) to construct quickly the search structures for each canonical interval; and (ii) to solve all queries for a canonical interval in one batch. We exploit the connection between planar disks and three-dimensional polytopes. Let $U = \big\{(x, y, z) \mid x^2 + y^2 = z\big\}$ be the three-dimensional unit paraboloid. For a site $s \in S$, the lifted site $\hat{s}$ is the vertical projection of s onto U. Each disk D_s is transformed into an upper halfspace $\widehat{D}_s$, so that the projection of $\widehat{D}_s \cap U$ onto the xy-plane is the set $\mathbb{R}^2 \setminus D_s$;[5] see Figure 6. The union of a set of disks in $\mathbb{R}^2$ corresponds to the intersection of the lifted upper halfspaces in $\mathbb{R}^3$.

▶ **Lemma 5.7.** *The range searching problem (R2) can be solved in $O(n \log n)$ expected time.*

Proof (Sketch). For each $v \in B$, we construct the intersection $\mathcal{E}_v$ of the $\widehat{D}_s$, $s \in \mathcal{I}_v$. We can find all the polyhedra $\mathcal{E}_v$, for $v \in B$, in overall $O(n \log n)$ time, by a single traversal of B. This traversal goes bottom up and uses that we can find the intersection of two convex polyhedra in $O(n)$ time [6]. For batched query processing, we need for each $v \in B$ the convex hull of the lifted query sites that have v as a canonical node. These convex hulls can be computed top-down by splitting the current convex hulls in each node in $O(n \log n)$ overall time, using linear time per node [8]. We call these hulls $\widehat{Q}_v$.

To answer all queries, we use the polyhedra $\widehat{Q}_v$ and $\mathcal{E}_v$. For each node $v \in B$, we compute the lifted sites in $\widehat{Q}_v$ that are not inside of $\mathcal{E}_v$. These sites correspond to the vertices of $\widehat{Q}_v$ that are not vertices of $\widehat{Q}_v \cap \mathcal{E}_v$. These intersections can be found in overall $O(n \log n)$ time, again by using the fact that we can intersect two polyhedra in $O(n)$ time. If for any such intersection $\widehat{Q}_v \cap \mathcal{E}_v$, there is a lifted site $\hat{s} \in \widehat{Q}_v$ that is not a vertex of $\widehat{Q}_v \cap \mathcal{E}_v$, we report s as the result of the range search. Otherwise, we report our range search to be unsuccessful. ◀

6 Finding the Shortest Triangle in a Transmission Graph

We extend Theorem 4.4 to find the shortest triangle in $T(S)$. As in Section 2.2, we solve the decision problem: given $W > 0$, does $T(S)$ have a directed triangle of perimeter at most W? We set $\ell = W/\sqrt{27}$, and call a site $s \in S$ *large* if $r_s > \ell$. We let $S_\ell \subseteq S$ be the set of all large sites.

[3] Note that here the radii are $\geq r_s/2$ inside of the cells σ. This might be larger than the value $2^{\lceil \log_2 r_s \rceil}/4$ needed by (R1). But still, if there are more than α sites in σ, we still have a triangle in a square. Otherwise we will later determine that each disk contains few sites of radius at least $r_s/2$.

[4] $r = 2r_s$ is the side length of the enclosing square, the radii are at least $r/4$ as desired.

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

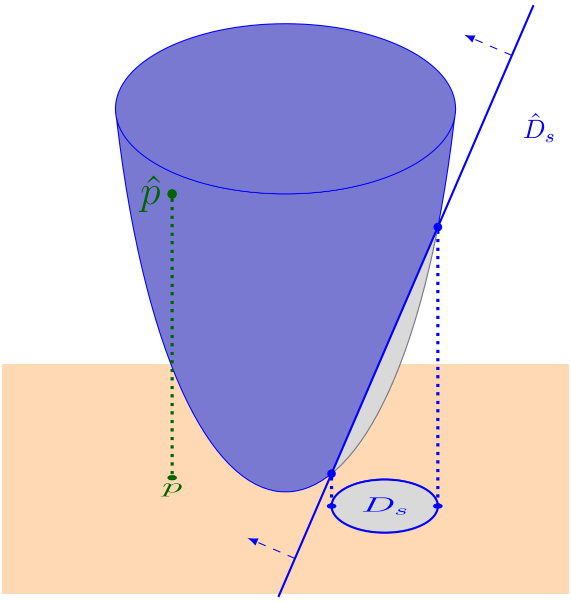


Figure 6 Lifting disks and points. For $\hat{D}$ only the bounding plane is shown.

▶ **Lemma 6.1.** *We can find a triangle in $T(S \setminus S_\ell)$ of perimeter at most W in $O(n \log n)$ time, if it exists.*

Proof. Any triangle in $T(S \setminus S_\ell)$ has perimeter at most W: consider a directed triangle stu in $T(S \setminus S_\ell)$ with $r_s \geq \max\{r_t, r_u\}$. Then we have $t, u \in D_s$, so the triangle stu lies in D_s. Elementary calculus shows that a triangle of maximum perimeter in D_s must be equilateral with its vertices on ∂D_s, so any triangle contained in D_s has perimeter at most $3 \cdot \sqrt{3} \cdot r_s \leq \sqrt{27} \cdot \ell = W$. We can find a triangle in T' in $O(n \log n)$ time by Theorem 4.4. ◀

It remains to check for triangles of perimeter at most W with at least one large vertex. Some such triangles have to be considered individually, while the others can be handled efficiently in batch mode. The following lemma shows that we may assume that there are few edges from $S \setminus S_\ell$ to S_ℓ.

▶ **Lemma 6.2.** *If $T(S)$ does not have a triangle of perimeter at most W, every site in S_ℓ has at most six incoming edges from $S \setminus S_\ell$. Furthermore, in $O(n \log n)$ time, we can either find a triangle of perimeter at most W in $T(S)$ or determine for each site in S_ℓ all incoming edges from $S \setminus S_\ell$.*

Proof (Sketch). By a suitable variant of (R1), we either find a triangle of perimeter at most W, or we restrict the overall number of edges from $S \setminus S_\ell$ to S_ℓ to $O(n)$ in $O(n \log n)$ time. To bound the indegree of the large sites, we observe that if a large site has 7 incoming edges from $S \setminus S_\ell$, then there is a triangle in $T(S)$ of perimeter at most W. ◀

Next, we want to limit the number of relevant edges between large sites. For this, we subdivide the plane with a grid G of side length $\ell/\sqrt{2}$. Then, we have the following:

▶ **Lemma 6.3.** *A triangle contained in a cell $\sigma \in G$ has perimeter at most W. If there is no triangle in σ, then σ contains $O(1)$ large sites. We can check for such triangles in $O(n \log n)$ overall expected time.*

Proof. The maximum perimeter of a triangle contained in σ is $(1+\sqrt{2})\ell < W$. Furthermore, if there are at least three large sites in σ, these large sites form a triangle, since the disk of a large site covers σ. By applying Theorem 4.4 to the induced subgraph in each cell of G, we can find such a triangle in $O(n \log n)$ total expected time. ◀

We define the neighborhood $N(\sigma)$ of a cell $\sigma \in G$ as the 5×5 block of cells centered at σ. Let t be a site and σ the cell containing t, then the neighborhood $N(t)$ of t are all sites contained in $N(\sigma)$. Since the side length of a grid cell is $W/3\sqrt{6}$, each triangle of perimeter at most W is completely contained in the neighborhood of some cell.

▶ **Lemma 6.4.** *We can check the remaining triangles in $O(n)$ overall time.*

Proof. Consider a remaining triangle sut with $r_t \geq \max\{r_u, t_s\}$. Then, $t \in S_\ell$, and s, t, u all lie in $N(t)$. By Lemma 6.3, there are $O(1)$ large candidates for u in $N(t)$, and by Lemma 6.2, there are $O(1)$ small candidates for u. Having fixed a t and a possible candidate u, we iterate over all $s \in N(t)$ and check if s, u, and t form a triangle with weight at most W. Every site s is contained in $O(1)$ grid neighborhoods, and since there are $O(1)$ candidate pairs in each grid neighborhood, s participates in $O(1)$ explicit checks. The result follows. ◀

The following theorem summarizes the considerations in this section.

▶ **Theorem 6.5.** *It takes $O(n \log n)$ expected time to find the shortest triangle in a transmission graph.*

Proof. We already saw that there is an $O(n \log n)$ time decision algorithm for the problem. As in Theorem 2.6, the result follows from an application of Chan's randomized optimization technique [5]. ◀

7 Conclusion

Once again, disk graphs and transmission graphs prove to be a simple yet powerful graph model where difficult algorithmic problems admit faster solutions. It would be interesting to find a *deterministic* $O(n \log n)$ time algorithm for finding a shortest triangle in a disk graph. Currently, we are working on extending our results to the girth problem in transmission graphs; can we find an equally simple and efficient algorithm as for disk graphs?

References

1 Noga Alon, Raphael Yuster, and Uri Zwick. Finding and Counting Given Length Cycles. *Algorithmica*, 17(3):209–223, 1997.

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

3 Kevin Buchin, Maarten Löffler, Pat Morin, and Wolfgang Mulzer. Preprocessing Imprecise Points for Delaunay Triangulation: Simplified and Extended. *Algorithmica*, 61(3):674–693, 2011.

4 Kevin Buchin and Wolfgang Mulzer. Delaunay triangulations in $O(\text{sort}(n))$ time and more. *J. ACM*, 58(2):6:1–6:27, 2011.

5 Timothy M. Chan. Geometric Applications of a Randomized Optimization Technique. *Discrete Comput. Geom.*, 22(4):547–567, 1999.

6 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.

7 Hsien-Chih Chang and Hsueh-I Lu. Computing the Girth of a Planar Graph in Linear Time. *SIAM J. Comput.*, 42(3):1077–1094, 2013.

8 Bernard Chazelle and Wolfgang Mulzer. Computing Hereditary Convex Structures. *Discrete Comput. Geom.*, 45(4):796–823, 2011.

9 William S. Evans, Mereke van Garderen, Maarten Löffler, and Valentin Polishchuk. Recognizing a DOG is Hard, But Not When It is Thin and Unit. In *Proc. 8th FUN*, pages 16:1–16:12, 2016.

10 François Le Gall. Powers of tensors and fast matrix multiplication. In *Proc. 39th Internat. Symp. Symbolic and Algebraic Comput. (ISSAC)*, pages 296–303, 2014.

11 Sariel Har-Peled. *Geometric approximation algorithms.* American Mathematical Society, 2008.

12 Alon Itai and Michael Rodeh. Finding a Minimum Circuit in a Graph. *SIAM J. Comput.*, 7(4):413–423, 1978.

13 Haim Kaplan, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. Spanners and Reachability Oracles for Directed Transmission Graphs. In *Proc. 31st Int. Sympos. Comput. Geom. (SoCG)*, pages 156–170, 2015.

14 Haim Kaplan, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. Spanners for Directed Transmission Graphs. *SIAM J. Comput.*, 47(4):1585–1609, 2018.

15 Jakub Łącki and Piotr Sankowski. Min-cuts and shortest cycles in planar graphs in $O(n \log \log n)$ time. In *Proc. 19th Annu. European Sympos. Algorithms (ESA)*, pages 155–166, 2011.

16 Shay Mozes, Kirill Nikolaev, Yahav Nussbaum, and Oren Weimann. Minimum Cut of Directed Planar Graphs in $O(n \log \log n)$ Time. In *Proc. 29th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 477–494, 2018.

17 Christos H. Papadimitriou and Mihalis Yannakakis. The Clique Problem for Planar Graphs. *Inform. Process. Lett.*, 13(4/5):131–133, 1981.

18 Valentin Polishchuk. Personal communication, 2017.

19 Liam Roditty and Virginia Vassilevska Williams. Minimum Weight Cycles and Triangles: Equivalences and Algorithms. In *Proc. 52nd Annu. IEEE Sympos. Found. Comput. Sci. (FOCS)*, pages 180–189, 2011.

20 Dan E. Willard and George S. Lueker. Adding Range Restriction Capability to Dynamic Data Structures. *J. ACM*, 32(3):597–617, 1985.

21 Virginia Vassilevska Williams and R. Ryan Williams. Subcubic Equivalences Between Path, Matrix, and Triangle Problems. *J. ACM*, 65(5):27:1–27:38, 2018.

22 Huacheng Yu. An Improved Combinatorial Algorithm for Boolean Matrix Multiplication. In *Proc. 42nd Internat. Colloq. Automata Lang. Program. (ICALP)*, pages 1094–1105, 2015.

23 Raphael Yuster. A shortest cycle for each vertex of a graph. *Inform. Process. Lett.*, 111(21-22):1057–1061, 2011.