An $O(n \log n)$ Algorithm for Single-Source Shortest Paths in Disk Graphs

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

Sergio Cabello
Faculty of Mathematics and Physics, University of Ljubljana, Slovenia
Institute of Mathematics, Physics and Mechanics, Slovenia

Abstract

We prove that the single-source shortest-path problem on disk graphs can be solved in $O(n \log n)$ expected time, and that it can be solved on intersection graphs of fat triangles in $O(n \log^3 n)$ time.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases shortest path, geometric intersection graph, disk graph, fat triangles

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.81

Related Version *Full Version*: `arXiv:2506.07571` [11]

Funding *Mark de Berg*: Supported by the Dutch Research Council (NWO) through Gravitation-grant NETWORKS-024.002.003.
Sergio Cabello: Funded in part by the Slovenian Research and Innovation Agency (P1-0297, N1-0218, N1-0285). Funded in part by the European Union (ERC, KARST, project number 101071836). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

1 Introduction

Finding shortest paths in graphs is a classic topic covered in all basic courses and textbooks on algorithms. In the most basic setting, which is the setting we consider here, the graph is unweighted and the length of a path refers to the number of edges in the path. In the single-source shortest-path (SSSP) problem, the task is to compute shortest paths from a given vertex, the *source*, to all other vertices of the graph. The solution is represented using a so-called *shortest-path tree*.

The SSSP problem for a graph with n vertices and m edges can be solved in $O(n + m)$ time by breadth-first search, which is optimal if the graph is given as a collection of vertices and edges. In this paper we will consider the SSSP problem for implicitly defined graphs. In particular, we consider (planar) *intersection graphs*: graphs whose node set corresponds to a set $\mathcal{D}$ of n objects in the plane and that contain an edge (D, D') between two distinct objects $D, D' \in \mathcal{D}$ iff D and D' intersect. If the objects in $\mathcal{D}$ are disks then the intersection graph, which we denote by $\mathcal{G}^\times[\mathcal{D}]$, is called a *disk graph* – see Figure 1 – and if all disks have unit radius then it is called a *unit-disk graph*. Disk graphs are arguably the most popular and widely studied intersection graphs. One can solve the SSSP problem on intersection graphs by first constructing $\mathcal{G}^\times[\mathcal{D}]$ explicitly and then running breadth-first search. In the worst case, however, this requires $\Omega(n^2)$ time. This raises the question: is it possible to solve the SSSP problem on intersection graphs in subquadratic time in the worst case?

Given the basic nature of the SSSP problem and the prominence of (unit-)disk graphs, it is not surprising that this question has already been considered for such graphs. For unit disks, Roditty and Segal [24] noticed that the dynamic data structure of Chan [6] for nearest-neighbor queries can be used to solve the problem in $O(n \operatorname{polylog} n)$ expected time. Cabello and Jejčič [4] gave an $O(n \log n)$ algorithm and showed that this is asymptotically optimal in the algebraic decision-tree model. They remarked that Efrat noted that the semi-dynamic data structure of Efrat, Itai and Katz [14] also gives an algorithm with running time $O(n \log n)$. Finally, Chan and Skrepetos [8] provided a simpler $O(n \log n)$ algorithm. The above algorithms either use Delaunay triangulations [4] or fixed-resolution grids [8, 14]; these approaches are not applicable for disks of arbitrarily different sizes.

For arbitrary disks graphs, Kaplan et al. [20] and Liu [22] presented algorithms running in $O(n \log^4 n)$ time. This was recently improved by Klost [21], who presented a general framework for solving SSSP problems in intersection graphs. She used the framework to obtain an $O(n \log^2 n)$ algorithm for the SSSP problem on disk graphs, and to obtain an $O(n \log n)$ algorithm for intersection graphs of axis-aligned squares.

Our results. Our main result in this paper is an algorithm for the SSSP problem in disk graphs that runs in $O(n \log n)$ expected time. We obtain this result using a version of the framework proposed by Klost [21]; our specialized framework and its relation to Klost's framework are discussed in detail in Section 2. A core ingredient in the framework is a *clique-based contraction* of the intersection graph $\mathcal{G}^\times[\mathcal{D}]$: a graph obtained from $\mathcal{G}^\times[\mathcal{D}]$ by contracting certain cliques to single nodes. To compute such a clique-based contraction in $O(n \log n)$ time, we combine shifted quadtrees [5] and skip quadtrees [15] – powerful and beautiful concepts that we believe deserve more attention – with standard techniques for stabbing fat objects.

Our algorithm for computing a clique-based contraction not only applies to disk graphs but also to intersection graphs of *fat triangles*, that is, triangles all of whose internal angles are lower bounded by a fixed constant $\alpha > 0$ [2]. By combining this with a novel intersecting-detection data structure for fat triangles, we obtain an $O(n \log^3 n)$ algorithm for the SSSP problem on intersection graphs of fat triangles. We are not aware of previous work on the SSSP problem for fat triangles. The all-pairs shortest-path (APSP) problem, however, has been considered for fat triangles, by Chan and Skrepetos [9]. The algorithm they present runs in $O(n^2 \log^4 n)$ time, under the condition that the fat triangles have roughly the same size. (They also present an $O(n^2 \log n)$ algorithm for the APSP problem for arbitrary disk graphs, and an $O(n^2)$ algorithm for unit-disk graphs [8].) Running our new SSSP algorithm n times, once for each input triangle as the source, we obtain an APSP algorithm for arbitrarily-sized fat triangles that runs in $O(n^2 \log^3 n)$ time. This is faster and more general than the algorithm of Chan and Skrepetos. (The improvement for the APSP problem is mainly because of our new intersection-detection data structure for fat triangles; using this data structures in the existing algorithm [9] would also give an $O(n^2 \log^3 n)$ algorithm.)

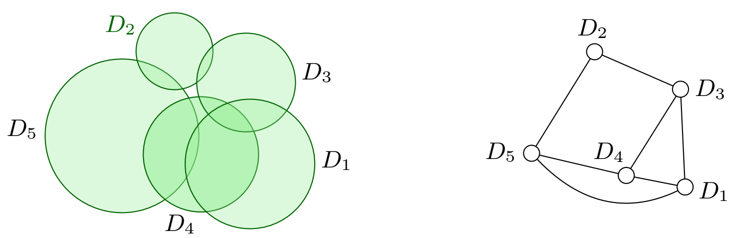

Figure 1 A set $\mathcal{D}$ of five disks (left) and their intersection graph (right).

In the full version of this work [11] we provide two enhancements: we obtain a *deterministic* $O(n \log n)$ algorithm for the SSSP problem in disk graphs and we reduce the running time for the SSSP problem on intersection graphs of fat triangles to $O(n \log^2 n)$. The basic structure of the algorithms is the same as here, but we improve some of the subroutines that we use.

On the model of computation. We use the real-RAM model – this is the standard model in computational geometry, which allows us for example to check in $O(1)$ time if two disks intersect – extended with an additional operation that allows us to compute a compressed quadtree on a set of n points in $O(n \log n)$ time. This extension is common when working with quadtrees; see the book by Har-Peled [19].

2 The framework

Our framework for computing a shortest-path tree on an intersection graph $\mathcal{G}^\times[\mathcal{D}]$ is an instantiation of the framework of Klost [21]. We describe it in detail to keep the paper self-contained and make the required subroutines explicit; at the end of this section we discuss the correspondence of our approach to that of Klost. For convenience, we will from now on not distinguish between the objects in the set $\mathcal{D}$ and the corresponding nodes in $\mathcal{G}^\times[\mathcal{D}]$.

Let $D_{\mathrm{src}} \in \mathcal{D}$ be the given source node, and let $\mathcal{T}_{\mathrm{sp}}$ be the shortest-path tree of $\mathcal{G}^\times[\mathcal{D}]$ that we want to compute for the given source. For an object $D \in \mathcal{D}$, let $\mathrm{dist}[D]$ be the distance from D_{src} to D in $\mathcal{G}^\times[\mathcal{D}]$, and let $L_\ell := \{D \in \mathcal{D} : \mathrm{dist}[\mathcal{D}] = \ell\}$. Thus, L_ℓ is the set of nodes at level ℓ in $\mathcal{T}_{\mathrm{sp}}$. We follow the natural approach of computing $\mathcal{T}_{\mathrm{sp}}$ level by level. To be able to go from one level to the next, we need a subroutine for the following problem.

> Bichromatic Intersection Testing. Given a set B of n_B blue objects and a set R of n_R red objects, report the blue objects that intersect at least one red object, and for each reported blue object, report a witness (a red object intersecting it).

A subroutine *BIT-Subroutine* that solves Bichromatic Intersection Testing allows us to compute $L_{\ell+1}$ by setting $R := L_\ell$ and $B := \mathcal{D}_{\mathrm{cand}}$, where $\mathcal{D}_{\mathrm{cand}}$ is a set of *candidate objects* that should contain all objects from $L_{\ell+1}$ and no objects from $L_{\leqslant \ell}$, where $L_{\leqslant \ell} := \bigcup_{i=0}^{\ell} L_i$. We could simply set $\mathcal{D}_{\mathrm{cand}} := \mathcal{D} \setminus L_{\leqslant \ell}$, but this will not be efficient; we must keep the total size of the candidates sets over all levels ℓ under control. We do this using a so-called clique-based contraction, as defined next.

Let $\mathcal{G} = (V, E_\mathcal{G})$ be an (undirected) graph. We say that a graph $\mathcal{H} = (\mathcal{C}, E_\mathcal{H})$ is a *clique-based contraction* of $\mathcal{G}$ if

- each node $C \in \mathcal{C}$ corresponds to a clique in $\mathcal{G}$ and the cliques in $\mathcal{C}$ form a partition[1] of the nodes in V;
- there is an edge $(C, C') \in E_\mathcal{H}$ iff there are nodes $v \in C$ and $v' \in C'$ such that $(v, v') \in E_\mathcal{G}$.

Note that for each edge $(v, v') \in E_\mathcal{G}$ there exists a clique $C \in \mathcal{C}$ that contains both v and v', or there are cliques $C \ni v$ and $C' \ni v'$ such that $(C, C') \in E_\mathcal{H}$.

A clique-based contraction $\mathcal{H}$ of the intersection graph $\mathcal{G}^\times[\mathcal{D}]$ helps us to find good candidate sets. To see this, let $\mathcal{C}_\ell \subset \mathcal{C}$ denote the set of cliques[2] that contain at least one object from L_ℓ. Then any object $D \in L_{\ell+1}$ must be part of a clique C that is either in $\mathcal{C}_\ell$

[1] In our application, it would also be sufficient to require that each node in V appears in $O(1)$ cliques, but we do not need this extra flexibility.

[2] With a slight abuse of terminology, we not only use the term *clique* to refer a complete graph, but also to refer to a set of geometric objects whose intersection graph is a clique.

itself or that is a neighbor in $\mathcal{H}$ of a clique in $\mathcal{C}_\ell$. In other words, if $D \in L_{\ell+1}$ then $D \in C$ for some clique $C \in N_{\mathcal{H}}[\mathcal{C}_\ell]$, where $N_{\mathcal{H}}[\mathcal{C}_\ell]$ denotes the closed neighborhood of $\mathcal{C}_\ell$ in $\mathcal{H}$. Moreover, the objects in a clique are in at most two levels in the shortest-path three $\mathcal{T}_{\text{sp}}$, which helps to control the total size of the candidate sets. The following pseudocode describes our framework in detail. It does not explicitly construct the shortest-path tree itself (it only constructs the levels), but this is easy to do using the witnesses reported by *BIT-Subroutine*.

Algorithm 1 *SSSP-for-Geometric-Intersection-Graphs*$(\mathcal{D}, D_{\text{src}})$.

1: Construct a clique-based contraction $\mathcal{H} = (\mathcal{C}, E_{\mathcal{H}})$ of $\mathcal{G}^{\times}[\mathcal{D}]$
2: $L_0 \leftarrow \{D_{\text{src}}\}$; $\mathcal{C}_0 \leftarrow \{$the clique $C \in \mathcal{C}$ that contains $D_{\text{src}}\}$ ▷ D_{src} is the source
3: Label D_{src} as *ready*; $\ell \leftarrow 0$; *done* $\leftarrow$ FALSE.
4: **while** not *done* **do**
5: $\quad \mathcal{D}_{\text{cand}} \leftarrow \{D \in \mathcal{D} : D \in C$ for a clique $C \in N_{\mathcal{H}}[\mathcal{C}_\ell]$ and D is not *ready*$\}$
6: $\quad L_{\ell+1} \leftarrow$ *BIT-Subroutine*$(\mathcal{D}_{\text{cand}}, L_\ell)$
7: $\quad$ **if** $L_{\ell+1} = \emptyset$ **then**
8: $\quad\quad L_\infty \leftarrow \mathcal{D} \setminus L_{\leqslant \ell}$; *done* $\leftarrow$ TRUE ▷ nodes in L_∞ are unreachable from D_{src}
9: $\quad$ **else**
10: $\quad\quad \mathcal{C}_{\ell+1} \leftarrow \{C \in \mathcal{C} : C$ contains an object $D \in L_{\ell+1}\}$
11: $\quad\quad$ Label all objects in $L_{\ell+1}$ as *ready*; $\ell \leftarrow \ell + 1$

We obtain the following theorem.

▶ **Theorem 1.** *Let $\mathcal{D}$ be a set of n constant-complexity objects in the plane. Suppose that*
(i) *we can compute a clique-based contraction for $\mathcal{G}^{\times}[\mathcal{D}]$ in $T_{\text{ccg}}(n)$ time,*
(ii) *for any two subsets $B, R \subset \mathcal{D}$ we can solve* Bichromatic Intersection Testing *in $T_{\text{bit}}(n_B, n_R)$ time, where $n_B := |B|$ and $n_R := |R|$.*

Then we can compute a shortest-path tree in $\mathcal{G}^{\times}[\mathcal{D}]$ for a given source node $D_{\text{src}} \in \mathcal{D}$ in $O(T_{\text{ccg}}(n) + T_{\text{bit}}(n, n))$ time.

Proof. It is straightforward to prove by induction on ℓ that our algorithm correctly computes the levels L_ℓ of the shortest-path tree. To prove the bound on the running time, observe that Steps 1–3 take $O(T_{\text{ccg}}(n))$ time. It remains to bound the runtime of the while-loop.

For a clique $C \in \mathcal{C}$, let $\text{dist}[C] := \min_{D \in C} \text{dist}[D]$. Note that for objects D, D' in cliques $C, C' \in \mathcal{C}$ that are neighbors in $\mathcal{H}$, we have $|\,\text{dist}[D] - \text{dist}[D']\,| \leqslant 3$. Hence, D can only be in the candidate set $\mathcal{D}_{\text{cand}}$ in iterations of the while-loop where $\text{dist}[D] - 3 \leqslant \ell \leqslant \text{dist}[C] - 1$. Thus, if we denote the size of $\mathcal{D}_{\text{cand}}$ in iteration ℓ by n_ℓ, then the total time needed for Step 6 over all iterations is $\sum_\ell T_{\text{bit}}(n_\ell, |L_\ell|)$, where $\sum_\ell n_\ell \leqslant 3n$ and $\sum_\ell |L_\ell| \leqslant n$. Since $T_{\text{bit}}(n_B, n_R)$ is at least linear in n_B and n_R (and at most quadratic), we have $\sum_\ell T_{\text{bit}}(n_\ell, |L_\ell|) = O(T_{\text{bit}}(n, n))$. This also bounds the total runtime of Steps 7–11 over all iterations.

To bound the total time for Step 5, note that a clique C is only considered in iterations where $\text{dist}[C] - 2 \leqslant \ell \leqslant \text{dist}[C] + 2$. This implies that the total time to find the relevant cliques in Step 5 is $O(|E_{\mathcal{H}}|)$. The total time to inspect those cliques in order to find the candidate sets, is $O(\sum_{C \in \mathcal{C}} |C|)$. Thus, the total time for Step 5 over all iterations is $O(|E_{\mathcal{H}}| + \sum_{C \in \mathcal{C}} |C|)$, which can be bounded by $O(T_{\text{ccg}}(n))$. ◀

▶ Remark (relation to the framework of Klost). The framework presented above is an instantiation of the framework of Klost [21]: she also computes the shortest-path tree level by level as described above – this natural approach was also taken by earlier papers on the SSSP problem on (unit-)disk graphs [4, 8, 9, 21] – and she also keeps the size of the candidate sets under control using an auxiliary graph. Klost uses a so-called *shortcut graph* $\mathcal{G}_{\mathrm{sc}}$ as auxiliary graph. The nodes in $\mathcal{G}_{\mathrm{sc}}$ correspond to the objects in $\mathcal{D}$, and the edge set E_{sc} is a superset of the edge set of $\mathcal{G}^{\times}[\mathcal{D}]$ such that for any edge $(D, D') \in E_{\mathrm{sc}}$ we have that the distance between D and D' in $\mathcal{G}^{\times}[\mathcal{D}]$ is $O(1)$. As E_{sc} is a superset of the edge set of $\mathcal{G}^{\times}[\mathcal{D}]$, the size of E_{sc} can be quadratic, so $\mathcal{G}_{\mathrm{sc}}$ needs to be constructed implicitly. Our clique-based contraction $\mathcal{H} = (\mathcal{C}, E_{\mathcal{H}})$ provides such an implicit shortcut graph, by defining $E_{\mathrm{sc}} := \{\, (D, D') : D$ and D' are part of the same clique in $\mathcal{C}$ or of neighboring cliques$\,\}$.

3 The algorithm for disks

In this section we implement the framework presented above for the case where the input $\mathcal{D}$ is a set of n disks in the plane. We first explain how to create the cliques in our clique-based contraction $\mathcal{H}$ of $\mathcal{G}^{\times}[\mathcal{D}]$, and how to compute the edge set of $\mathcal{H}$. Using known results on Bichromatic Intersection Testing for disks we then obtain our final result.

Finding the cliques. Our algorithm to create the set of cliques is related to a recent construction by Chan and Huang [7, Section 3] of 3-hop spanners for disk graphs; we comment more on the similarities and differences later. The construction is based on so-called shifted quadtrees, introduced by Chan [5], which we describe next.

We start by defining hierarchical grids, a concept closely related to quadtrees; the connection will be made clear below. Let $o = (o_x, o_y)$ be any point in the plane. A *cell* in the hierarchical grid centered at o is any square of the form

$$\sigma(o, \ell, i, j) := \left[o_x + i \cdot 2^{\ell}, o_x + (i+1) \cdot 2^{\ell}\right) \times \left[o_y + j \cdot 2^{\ell}, o_y + (j+1) \cdot 2^{\ell}\right)$$

for integers $i, j, \ell \in \mathbb{Z}$. Note that $\sigma(o, \ell, i, j)$ is a translation of the square $[0, 2^{\ell}) \times [0, 2^{\ell})$ by the vector $o + (i \cdot 2^{\ell}, j \cdot 2^{\ell})$, and that the set $\Sigma(o, \ell) := \{\sigma(o, \ell, i, j) : i, j \in \mathbb{Z}\}$ forms a regular grid. Moreover, the grid $\Sigma(o, \ell - 1)$ is a refinement of the grid $\Sigma(o, \ell)$, obtained by partitioning each cell of $\Sigma(o, \ell)$ into four quadrants. We define the *hierarchical grid centered at* o, denoted by $\Gamma(o)$, to be the (infinite) collection $\{\Sigma(o, \ell) : \ell \in \mathbb{Z}\}$ of nested grids.

Define the *size* of any object D, denoted by $\mathrm{size}(D)$, to be the side length of a smallest enclosing (axis-parallel) square of D. Thus $\mathrm{size}(\sigma) = 2^{\ell}$ for any cell $\sigma \in \Sigma(o, \ell)$. We say that an object D is *c-aligned* with a hierarchical grid Γ, for a given constant $c > 0$, if there exists a cell σ in Γ such that $D \subset \sigma$ and $\mathrm{size}(\sigma) \leqslant c \cdot \mathrm{size}(D)$. Note that whether or not D is c-aligned with Γ only depends on the choice of the center o of the hierarchical grid. The definition of being c-aligned trivially implies the following.

▶ **Observation 2.** *Let D be an object that is c-aligned with a hierarchical grid Γ, and let σ be a cell of Γ such that D intersects the boundary $\partial\sigma$ of σ. Then $\mathrm{size}(D) \geqslant (1/c) \cdot \mathrm{size}(\sigma)$.*

Let $\mathcal{D}$ be a finite set of objects. In general, it is impossible to choose the center of a hierarchical grid Γ such that each object $D \in \mathcal{D}$ is c-aligned with Γ. However, Chan [5] has shown that, surprisingly, it *is* possible to pick a small number of different centers such that each object $D \in \mathcal{D}$ is c-aligned with at least one of the resulting hierarchical grids, for a suitable constant c. Even more surprisingly, we can select these centers independently of $\mathcal{D}$. The following lemma follows directly from Lemma 3.2 in Chan's paper.[3]

[3] Chan uses the term *shifted quadtree* in his lemma. To stress the fact that the shifts are independent of any input set on which one might build a quadtree, we prefer to use the term *hierarchical grid*.

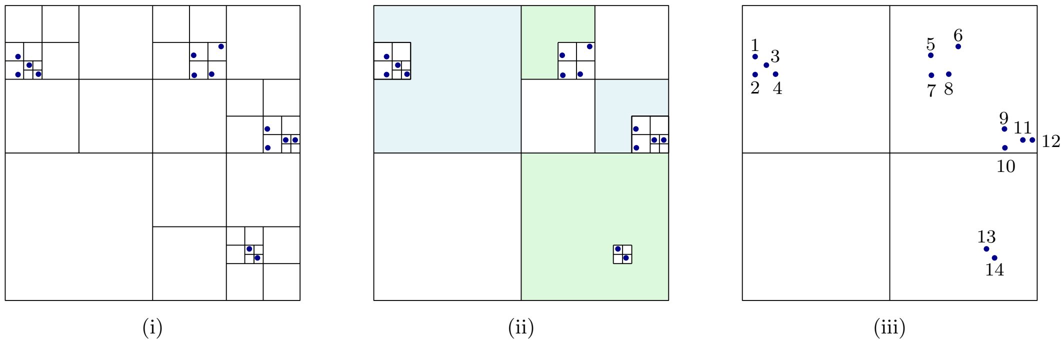

Figure 2 (i) A quadtree subdivision for a set P of points. (ii) The compressed-quadtree subdivision for P, with its donut cells marked. (iii) The ordering of the points to construct a skip quadtree.

▶ **Lemma 3 ([5]).** *There is a collection Ξ of three hierarchical grids, each centered at a different point, with the following property: for any object D contained in the square $[0, 1) \times [0, 1)$ there is a hierarchical grid $\Gamma \in \Xi$ such that D is 6-aligned with Γ.*

Let P be a set of points, let Γ be a hierarchical grid centered at a given center o, and let σ_0 be a cell of Γ that contains P. Then we can construct a *quadtree* on P – see Figure 2(i) – whose root node corresponds to σ_0. Any node in this quadtree corresponds to a cell of Γ. Let $\mathcal{Q} = \mathcal{Q}(\sigma_0, P)$ be the corresponding *compressed quadtree* [19]. Each internal node ν of $\mathcal{Q}$ corresponds to a cell σ_ν of Γ and each leaf node μ corresponds to a region σ_μ that is either a cell of Γ or a donut cell; see Figure 2(ii). (A *donut cell* is the difference $\sigma_1 \setminus \sigma_2$ of two cells σ_1 and σ_2 of Γ with $\sigma_2 \subset \sigma_1$.) The *subdivision* defined by the compressed quadtree $\mathcal{Q}$ is the set of regions corresponding to the leaves of $\mathcal{Q}$; this subdivision is a decomposition of σ_0. Our algorithm to efficiently compute the cliques in the clique-based contraction combines the hierarchical grids of Lemma 3 with skip quadtrees [15], as explained next.

Skip quadtrees, defined by Eppstein, Goodrich, and Sun [15] are defined as follows. Let P be a set of n points inside a given starting cell σ_0 of a hierarchical grid Γ. We start by sorting the points in P in Z-order, as follows. First, order the points according to the quadrants from σ_0 they fall in: put the points in the NW-quadrant first, then those in the NE-quadrant, then those in the SW-quadrant, and finally those in the SE-quadrant. Next, recursively sort the points in each quadrant. Let $p_1, \ldots, p_n$ be the resulting sorted list of points; see Figure 2(iii). We now create a sequence $P = P_0 \supset P_1 \supset \cdots \supset P_t$, where P_i is obtained from P_{i-1} by deleting every other element (starting with the first element) and $|P_t| = 1$. From the sequence $P_0 \supset P_1 \supset \cdots \supset P_t$ we construct a sequence of compressed-quadtree *subdivisions* $\mathcal{Q}_0, \mathcal{Q}_1, \ldots, \mathcal{Q}_t$, each based on the same starting cell σ_0. The sequence $\mathcal{Q}_0, \mathcal{Q}_1, \ldots, \mathcal{Q}_t$ has the following properties, as illustrated in Figure 3:

- $\mathcal{Q}_t$ is equal to the square σ_0.
- $\mathcal{Q}_i$ is a refinement of $\mathcal{Q}_{i+1}$ for all $0 \leqslant i < t$, that is, each region in $\mathcal{Q}_i$ is contained in a unique region in $\mathcal{Q}_{i+1}$. Moreover, each region in $\mathcal{Q}_{i+1}$ contains $O(1)$ regions from $\mathcal{Q}_i$.
- Each region in $\mathcal{Q}_0$ contains at most one point from P.

We can view $\mathcal{Q}_t, \ldots, \mathcal{Q}_1, \mathcal{Q}_0$ as the levels of a tree $\mathcal{T}$, where a node ν at level i corresponds to a region R_ν of the subdivision $\mathcal{Q}_i$. In particular, the root of $\mathcal{T}$ corresponds to the starting square σ_0 and the leaves of $\mathcal{T}$ correspond to the regions in $\mathcal{Q}_0$ (which is the compressed-quadtree subdivision for the whole point set P with root σ_0). We call this tree structure[4] a *skip quadtree* for P with root σ_0. It can be constructed in $O(n \log n)$ time [15].

[4] The skip quadtree defined by Eppstein, Goodrich, and Sun is slightly more complicated, as it stores

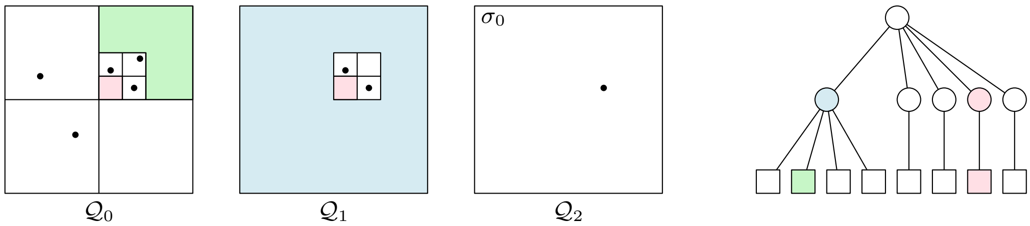

Figure 3 Example of the three compressed-quadtree subdivisions for a set of five points, and the resulting skip quadtree. Colors indicate the correspondence between regions and nodes.

Without loss of generality, we will assume henceforth that all the disks of $\mathcal{D}$ are fully contained in the unit square $[0,1) \times [0,1)$. Let Ξ be the collection of three hierarchical grids given by Lemma 3. Each disk D of $\mathcal{D}$ is 6-aligned with some hierarchical grid $\Gamma \in \Xi$, and we can find such a hierarchical grid for D in constant time.

Now consider a hierarchical grid Γ from the collection Ξ. Let $\mathcal{D}(\Gamma) \subset \mathcal{D}$ be the set of disks that are 6-aligned with Γ, where we put a disk that is 6-aligned with multiple grids into an arbitrary one of the corresponding sets $\mathcal{D}(\Gamma)$. We create a set $\mathcal{C}(\Gamma)$ of cliques, as follows.

1. Construct a skip quadtree $\mathcal{T}(\Gamma)$ on the set $P(\Gamma)$ of centers of the disks in $\mathcal{D}(\Gamma)$, where the starting square σ_0 of the skip quadtree is a cell of Γ that contains all the disks in $\mathcal{D}(\Gamma)$.
2. For each disk $D \in \mathcal{D}(\Gamma)$, follow a path in $\mathcal{T}(\Gamma)$ consisting of nodes μ whose region R_μ contains the center of D, until a node ν is reached such that D intersects ∂R_ν or a leaf is reached. Assign D to that node or leaf.
3. Let $\mathcal{D}_\nu \subset \mathcal{D}(\Gamma)$ be the set of disks assigned to an (internal or leaf) node ν in $\mathcal{T}(\Gamma)$. For each node ν, partition $\mathcal{D}_\nu$ into a set $\mathcal{C}_\nu$ of $O(1)$ cliques, as follows. If ν is a leaf and R_ν contains a disk – note that a leaf can contain at most one disk – then we create a singleton clique for that disk. Now suppose ν is an internal node. First, suppose R_ν is a square region. Because the disks in $\mathcal{D}_\nu$ are 6-aligned with Γ, we know that $\text{size}(D) \geqslant \frac{1}{6} \cdot \text{size}(R_\nu)$ for all $D \in \mathcal{D}_\nu$ intersecting ∂R_ν. Hence, we can create a set of $O(1)$ points such that each disk D is stabbed by at least one of them, which we use to create $\mathcal{C}_\nu$. If R_ν is a donut we proceed similarly, where we treat the disks intersecting the boundary of the hole of the donut – this hole can be much smaller than the donut – separately from the disks that only intersect the outer boundary of the donut. Finally, set $\mathcal{C}(\Gamma) := \bigcup_{\nu \in \mathcal{T}(\Gamma)} \mathcal{C}_\nu$.

Each clique that is generated by our algorithm is a so-called *stabbed clique*, that is, for each clique $C \in \mathcal{C}(\Gamma)$ there is a point stabbing all disks in C. We refer to the union of all the disks in C as a *flower*, which we denote by $F(C)$, and we let $\mathcal{F}(\Gamma) := \{F(C) : C \in \mathcal{C}(\Gamma)\}$ be the set of flowers corresponding to the cliques in $\mathcal{C}(\Gamma)$. Define $\text{ply}(S)$, the *ply* of a set S of objects in the plane, to be the maximum, over all points $q \in \mathbb{R}^2$, of the number of objects in S containing q. The following lemma states a property of our algorithm that will be crucial to efficiently compute the edges in the clique-based contraction.

▶ **Lemma 4.** *The algorithm above constructs in $O(n \log n)$ time a partition of $\mathcal{D}(\Gamma)$ into a set $\mathcal{C}(\Gamma)$ of stabbed cliques such that $\text{ply}(\mathcal{F}(\Gamma)) = O(\log n)$.*

---

more information than just the compressed-quadtree subdivisions. Since we do not need our structure to be dynamic, the simple hierarchy described above suffices.

Proof. Step 1 of the algorithm, constructing the skip quadtree $\mathcal{T}(\Gamma)$ on the set $P(\Gamma)$ of the centers of the at most n disks in $\mathcal{D}(\Gamma)$, can be done in $O(n \log n)$ time [15]. The depth of the skip quadtree is $O(\log n)$, and so Step 2 takes $O(\log n)$ time per disk and $O(n \log n)$ time in total. Step 3 takes $\sum_{v \in \mathcal{T}(\Gamma)} O(1 + |\mathcal{D}_v|)$ time, which is $O(n)$ because each disk is assigned to exactly one node. Hence, the total time for the algorithm is $O(n \log n)$.

To bound $\text{ply}(\mathcal{F}(\Gamma))$, consider an arbitrary point $q \in \mathbb{R}^2$. Recall that for each node ν in $\mathcal{T}(\Gamma)$ we created $O(1)$ stabbed cliques and, hence, $O(1)$ flowers. Thus, it suffices to prove the following claim: at each level of the skip quadtree $\mathcal{T}(\Gamma)$ there are only $O(1)$ nodes ν such that $q \in D$ for some disk D assigned to ν. To prove this claim, follow the search path of q in $\mathcal{T}(\Gamma)$, that is, the path consisting of nodes μ such that $q \in R_\mu$. Let $D \in \mathcal{D}$ be a disk that contains q, let ν be the node to which D is assigned, and let μ be the parent of ν. Then μ must lie on the search path of q. Indeed, D is fully contained in R_μ – otherwise D would have been assigned to μ – and so $q \in D \subset R_\mu$. Since any node in a skip quadtree has $O(1)$ children, this implies the claim. ◀

▶ Remark (relation to the spanner construction of Chan and Huang). In their algorithm for constructing a 3-hop spanner for $\mathcal{G}^\times[\mathcal{D}]$, Chan and Huang [7] create a collection of cliques in a similar way as we create our cliques. As they are only concerned with the size of the spanner, however, they do not analyze the time needed to construct it. The differences between our approach and theirs mainly stem from the necessity to construct the clique-based contraction $\mathcal{H}$ in $O(n \log n)$ time, as discussed next.

A first difference is that Chan and Huang use a centroid decomposition of a compressed quadtree instead of a skip quadtree. This is not a crucial difference – possibly we could have used a centroid decomposition as well, although working with a skip quadtree is more convenient. Another difference is that they work with a collection Ξ of hierarchical grids – they call them shifted quadtrees – such that, for each pair $D, D' \in \mathcal{D}$, there is at least one hierarchical grid $\Gamma \in \Xi$ such that both D and D' are aligned with Γ. This allows them to construct their spanner by applying the following procedure to each quadtree $\mathcal{Q}$ defined by the hierarchical grid Γ: (i) take a centroid node ν in a compressed quadtree $\mathcal{Q}$ and partition the set $\mathcal{D}_\nu$ of disks intersecting $\partial\sigma_\nu$ into $O(1)$ cliques, (ii) for each clique C, add a star graph on that clique to the spanner, (iii) for each disk D and each clique C intersected by D, add an edge (D, D') to the spanner, where $D' \in C$ is an arbitrary disk intersected by D, and (iv) recursively construct 3-hop spanners for the disks inside and outside σ_ν. The crucial difference lies in step (iii), where they add an edge from every disk D to a neighboring disk in each clique (if such a neighboring disk exists). It seems difficult to do this in $O(n \log n)$ time in total over all recursive calls. We therefore proceed differently: we first recursively construct cliques on the disks not intersecting $\partial\sigma_\nu$, and then we only add a single edge between two cliques if the corresponding flowers intersect. Such an approach would increase the spanning ratio if we were to use it to construct a spanner, but this is of no concern to us. The advantage of being more conservative is that we can compute the edges in our clique-based contraction in $O(n \log n)$ time, as explained next.

Computing the edges in the clique-based contraction. Let $\mathcal{C} := \bigcup_{\Gamma \in \Xi} \mathcal{C}(\Gamma)$ be the set of cliques generated by applying the procedure above to each hierarchical grid Γ in the collection Ξ given by Lemma 3. Let $\mathcal{F} := \bigcup_{\Gamma \in \Xi} \mathcal{F}(\Gamma)$ be the corresponding flower set. Note that $\text{ply}(\mathcal{F}) = O(\log n)$ by Lemma 4. To construct the edge set $E_\mathcal{H}$ of the clique-based contraction $\mathcal{H} = (\mathcal{C}, E_\mathcal{H})$, we must find all pairs $F, F' \in \mathcal{F}$ that intersect. Next we show how to do this in $O(n \log n)$ time.

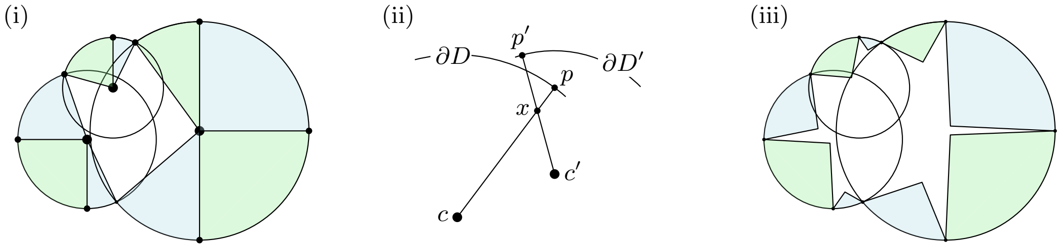

Figure 4 (i) The set Z_F of circular sectors defined by the boundary arcs of a flower F. Some boundary arcs have been cut into sub-arcs to ensure they span an angle of at most $\pi/2$. (ii) If $|px| < |p'x|$ then $p \in D'$; otherwise $p' \in D$. (iii) The modified sectors.

Consider a flower F defined by some clique C. The boundary ∂F of F is comprised of maximal pieces of the boundaries of the disks in C that show up on ∂F. We call these pieces *boundary arcs* and we denote the set of boundary arcs of a flower F by $\mathcal{B}(F)$. For a set $\mathcal{F}$ of flowers, we define $\mathcal{B}(\mathcal{F}) := \bigcup_{F \in \mathcal{F}} \mathcal{B}(F)$ to be the set of boundary arcs of the flowers in $\mathcal{F}$.

▶ **Lemma 5.** *Let $\mathcal{F}$ be a set of flowers that consist of n disks in total. Then the number of intersections between the boundary arcs in $\mathcal{B}(\mathcal{F})$ is $O(n \cdot \text{ply}(\mathcal{F}))$.*

Proof. First, observe that $|\mathcal{B}(F)| = O(n_F)$, where n_F is the number of disks defining a flower $F \in \mathcal{F}$, since the union of a set of disks has linear complexity. Hence, $|B(\mathcal{F})| = O(n)$.

Now consider a boundary arc $\beta \in \mathcal{B}(F)$ of some flower $F \in \mathcal{F}$. We assume for simplicity that the angle spanned by the circular arc β is at most $\pi/2$; if this is not the case we can split β into at most four sub-arcs, and work with the sub-arcs instead. Let D be the disk that contributes the arc β, let c be the center of D, and let s_1 and s_2 be the segments that connect c to the endpoints of β. Together with β, these two segments bound a (convex) circular sector. Let Z_F be the set of such circular sectors created for the arcs in $\mathcal{B}(F)$; see Figure 4(i). We say that two sectors z, z' are *non-overlapping* if their interiors are disjoint.

▷ Claim. The sectors in the set Z_F are pairwise non-overlapping.

Proof. Consider two sectors $z, z' \in Z_F$. Let D, D' be the disks whose boundaries define z and z', respectively, and let c and c' be their centers. If $D = D'$ then obviously z and z' do not overlap, so assume that $D \neq D'$. Since a sector in Z_F cannot be fully contained in another sector in Z_F by construction, the sectors z, z' can only overlap if there is a proper intersection between their boundaries. Such an intersection can only happen between a segment s connecting c to an endpoint p of some boundary arc $\beta \subset \partial D$ and a segment s' connecting c' to an endpoint p' of some boundary arc $\beta' \subset \partial D'$. But then, by the triangular inequality, either $p \in D'$ or $p' \in D$ – see Figure 4(ii) for an illustration – contradicting that p and p' are both points on ∂F. ◁

Now consider a circular sector z, defined by some arc β. Let x be the apex of z (which coincides with the center of the disk D contributing β). If the angle at apex x is large enough then z is fat, but if the angle is very small then this is not the case. We therefore proceed as follows: we move the apex x towards the midpoint of β until the angle at x is slightly larger than $\pi/2$; see Figure 4(iii). It is easy to see that the modified region (which is no longer a circular sector) is fat and convex. Moreover, no two modified regions share an apex anymore. With a slight abuse of notation, from now on we use Z_F to denote the set of modified regions created for a flower $F \in \mathcal{F}$.

Let $Z(\mathcal{F}) := \bigcup_{F \in \mathcal{F}} Z_F$. Note that $|Z(\mathcal{F})| \leqslant 4 \cdot |\mathcal{B}(\mathcal{F})| = O(n)$ and that $\text{ply}(Z(\mathcal{F})) \leqslant 2 \cdot \text{ply}(\mathcal{F}) = O(\log n)$; the latter is true since the regions created for a single flower R are pairwise disjoint except at shared endpoints of boundary arcs. For each arc $\beta \in \mathcal{B}(\mathcal{F})$, there is a region $z \in Z(\mathcal{F})$ such that $\beta \subset \partial z$. Hence, the number of intersections between the arcs in $\mathcal{B}(\mathcal{F})$ is upper bounded by the number of intersections between the regions in $Z(\mathcal{F})$.

▷ Claim. The number of intersections between the regions in $Z(\mathcal{F})$ is $O(|Z(\mathcal{F})| \cdot \text{ply}(Z(\mathcal{F})))$.

Proof. The proof is standard, but we sketch it for completeness. We charge the intersection between two regions z, z' to the smaller of the two regions, and claim that each region is charged $O(\text{ply}(Z(\mathcal{F})))$ times. To see this, consider a region $z \in Z(\mathcal{F})$ and let b be the smallest enclosing disk of Z. Let b' be the disk with the same center as b such that $\text{radius}(b') = 2 \cdot \text{radius}(b)$. Due to the fatness of the regions in $Z(\mathcal{F})$, any region z' that intersects z and is at least as large as z, will cover a constant fraction of the area of b'. Hence, there can be at most $O(\text{ply}(Z(\mathcal{F})))$ such regions z'. ◁

This finishes the proof. ◀

Observe that two flowers F, F' intersect iff the following holds: a boundary arc of F intersects a boundary arc of F', or $F \subset F'$, or $F' \subset F$. Thus we can find all intersecting pairs of flowers – and, hence, all edges of the clique-based contraction $\mathcal{H}$ – as follows. (The algorithm below may report a pair of intersecting flowers multiple times, but this is not a problem.)

First, we construct the arrangement $\mathcal{A}(\mathcal{F})$ induced by the flower set $\mathcal{F}$. Next, for each vertex of $\mathcal{A}(\mathcal{F})$ that is the intersection of the boundaries of some pair of flowers F, F, we report that pair. Finally, we need to find the pairs of flowers F, F' such that $F \subset F'$. To this end, we traverse the dual graph of $\mathcal{A}(\mathcal{F})$, maintaining a list $\mathcal{L}$ of flowers containing the face we are currently in. Whenever we enter a flower F for the first time, we report the pairs (F, F') for the faces F' that are currently in the list $\mathcal{L}$.

We obtain the following lemma.

▶ **Lemma 6.** *Let $\mathcal{D}$ be a set of n disks in the plane. Then we can compute a clique-based contraction for $\mathcal{G}^{\times}[\mathcal{D}]$ in $O(n \log n)$ expected time.*

Proof. Computing the collection $\mathcal{C}$ of cliques takes $O(n \log n)$ time by Lemma 4. It remains to analyze the time needed to compute the edge set $E_{\mathcal{H}}$ of the clique-based contraction.

For each clique $C \in \mathcal{C}$, we can compute the corresponding flower $F(C)$ in $O(|C| \log |C|)$ time using a simple divide-and-conquer algorithm. Since each disk is part of only clique, this implies that the flower set $\mathcal{F}$ can be computed in $O(n \log n)$ time. Moreover, the total number of boundary arcs of the flowers is $O(n)$ because the complexity of a single flower is linear. Hence, we can construct the arrangement $\mathcal{A}(\mathcal{F})$ in $O(n \log n + k)$ expected[5] time, where k is the number of intersections between the boundary arcs, using a randomized incremental algorithm by Mulmuley [23]. Since $k = O(n \log n)$ by Lemmas 4 and 5, constructing the arrangement thus takes $O(n \log n)$ expected time. The traversal of the dual graph, including the maintenance of $\mathcal{L}$ takes $O(n \log n)$ time. Since the size of the list $\mathcal{L}$ is $O(\log n)$ at any time – this is because the ply of the flower set is $O(\log n)$ – we report at most $O(n \log n)$ pairs in total during the traversal. Thus, the total time to compute $E_{\mathcal{H}}$ is $O(n \log n)$. ◀

[5] We cannot use the deterministic algorithm for line-segment intersection by Chazelle and Edelsbrunner [10] because it does not work for curves. The deterministic algorithm for line-segment intersection by Balaban [3] also works for curves, but that algorithm does not seem to give enough information to construct the arrangement.

Putting it all together. Lemma 6 gives us the first ingredient we need to apply Theorem 1. The second ingredient is an algorithm that solves BICHROMATIC INTERSECTION TESTING for disks. As observed in previous papers [9, 21] this can be done in $O((n_B + n_R) \log n_R)$ time, as follows. First, compute the additively weighted Voronoi diagram $\mathrm{Vor}(R)$ on the centers of the set R of red disks, where the weight of a center is equal to the radius of its corresponding disk. Next, query with the center of each blue disk D in $\mathrm{Vor}(R)$ to find the red disk D' closest to D. Now D intersects $\bigcup R$, the union of the red disks, iff D intersects D'. Since $\mathrm{Vor}(R)$ can be computed in $O(n_R \log n_R)$ time [17], after which it can be preprocessed for logarithmic-time point location in $O(n_R \log n_R)$ time [13], we can solve BICHROMATIC INTERSECTION TESTING in $O((n_B + n_R) \log n_R)$ time. We thus obtain our main result.

▶ **Theorem 7.** *Let $\mathcal{D}$ be a set of n disks in the plane. Then we can compute a shortest-path tree in $\mathcal{G}^\times[\mathcal{D}]$ for a given source disk $D_{\mathrm{src}} \in \mathcal{D}$ in expected $O(n \log n)$ time.*

In the full version [11] we present a (slightly more complicated) deterministic algorithm to compute the edges of the clique-based contraction for $\mathcal{G}^\times[\mathcal{D}]$ in $O(n \log n)$ time. Using that algorithm instead of Lemma 6, we obtain an algorithm to compute shortest-path trees on disks graphs deterministically in $O(n \log n)$ time.

4 Extension to fat triangles

A triangle Δ is called *α-fat* if its minimum angle is at least α. Let $\mathcal{D} = \{\Delta_1, \dots, \Delta_n\}$ be a set of α-fat triangles, where $\alpha > 0$ is some fixed absolute constant. From now on, we simply refer to the triangles in $\mathcal{D}$ as *fat triangles* and we refer to α as the *fatness constant.*

Constructing the clique-based contraction. Our approach to construct a clique-based contraction for fat triangles is similar to the algorithm for disks. We now go over the various ingredients and explain how to adapt them, if necessary.

We first observe that Lemma 3.2 from Chan's paper [5] actually holds for any type of objects. Hence, our Lemma 3 also holds for any type of objects. To construct the skip quadtree, the set $P(\Gamma)$ of disk centers is replaced by a set $P(\Gamma)$ that contains an arbitrary point in each object. Constructing the cliques can therefore be done in exactly the same way as before. Indeed, the crucial property was as follows: any set of disks intersecting the boundary of a region R_ν and whose size is at least $\frac{1}{6} \cdot \mathrm{size}(R_\nu)$, can be stabbed by $O(1)$ points. This property also holds for fat objects.

Now let C be a stabbed clique of fat triangles. We refer to the union of the triangles in C as a *spiky flower*, and we denote it by $F(C)$. As before, we let $\mathcal{C}(\Gamma)$ denote the set of cliques created by our algorithm, and we define $\mathcal{F}(\Gamma) := \{F(C) : C \in \mathcal{C}(\Gamma)\}$ to be the corresponding set of spiky flowers. Then Lemma 4 also holds for fat triangles – we can follow the proof verbatim, only replacing occurrences of "disk" by "fat triangle."

It remains to prove the equivalent of Lemma 5 for fat triangles. To keep the terminology similar to the case of disks, we will refer to the edges of a spiky flower F as *boundary segments*, and we denote the set of boundary segments of a flower F by $\mathcal{B}(F)$. Furthermore, we let $\mathcal{F} := \bigcup_{\Gamma \in \Xi} \mathcal{F}(\Gamma)$ denote the set of spiky flowers created by our algorithm, and we define $\mathcal{B}(\mathcal{F}) := \bigcup_{F \in \mathcal{F}} \mathcal{B}(F)$.

▶ **Lemma 8.** *Let $\mathcal{F}$ be a set of spiky flowers that consist of n fat triangles in total. Then the number of intersections between the boundary segments in $\mathcal{B}(\mathcal{F})$ is $O(n \cdot \mathrm{ply}(\mathcal{F}))$.*

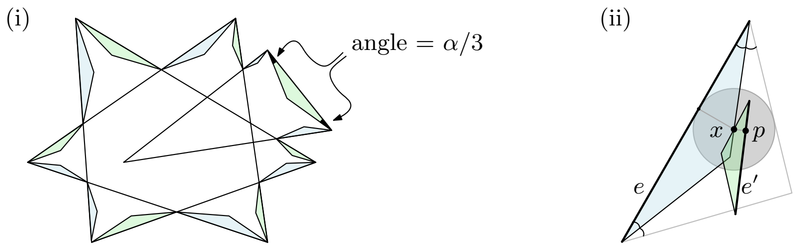

Figure 5 (i) The set Z_F of sectors defined by the boundary segments of a spiky flower F. (ii) The point p lies in D, which in turn lies inside the expanded sector.

Proof. We apply the same proof technique as in the proof of Lemma 5: we cover the boundary segments of each spiky flower F by a set Z_F of non-overlapping fat "sectors" contained in the flower, as explained below, from which the lemma follows.

The number of boundary segments of a spiky flower is $O(n_F)$, where n_F is the number of triangles defining F [1, Section 3.2]. We create the set Z_F for a spiky flower F as follows. For each boundary segment β of F, we create an isosceles triangle $z \subset F$ whose angles at the endpoints of β are $\alpha/3$, where α is the fatness constant of the triangles; see Figure 5. With a slight abuse of terminology, we refer to these isosceles triangles as sectors.

▷ Claim. The sectors in the set Z_F are pairwise non-overlapping.

Proof. Each sector z has one edge that corresponds to a boundary segment of F; we call this the *external edge* of z. The two other edges, which make an angle $\alpha/3$ with the external edge, are called *internal edges*.

Now assume for a contradiction that two sectors z, z' overlap. Since the external edge of any sector cannot properly intersect an (external or internal) edge of any other sector, there must be a crossing between an internal edge of z and an internal edge of z'. Let x be this crossing, and let e and e' be the external edges of z and z', respectively; see Figure 5(ii). Assume without loss of generality that the distance from x to e is at least the distance from x to e'. Let p be the point on e' nearest to x, and let D be the disk centered at x and touching e. Then p is contained in D. Moreover, D is contained in the triangle $\Delta \in \mathcal{D}$ contributing e to the boundary of the spiky flower F. Indeed, if we were to expand the sector z by making the angles at the endpoints of e equal to $2\alpha/3$ then $D \subset z$, and this expanded sector is contained in Δ. Thus, $p \in \Delta$, which contradicts that p lies on an external edge. ◁

As in the proof of Lemma 5, it now follows that the number of intersections between the sectors in $Z(\mathcal{F})$ is $O(|Z(\mathcal{F})| \cdot \text{ply}(Z(\mathcal{F})))$, which proves the lemma. ◀

As before, we have the following corollary. Its proof is the same as the proof of Lemma 6; the only difference is that we can now use the algorithm by Chazelle and Edelsbrunner [10] to construct the arrangement defined by the flowers, thus avoiding the use of randomization.

▶ **Corollary 9.** *Let $\mathcal{D}$ be a set of n fat triangles in the plane. Then we can compute a clique-based contraction for $\mathcal{G}^{\times}[\mathcal{D}]$ in $O(n \log n)$ time.*

Bichromatic Intersection Testing for fat triangles. The final ingredient that we need is an efficient algorithm for Bichromatic Intersection Testing for a set B of n_B blue triangles and a set R of n_R red triangles, where all triangles in $B \cup R$ are fat. Such an algorithm can be developed using standard machinery, as described next.

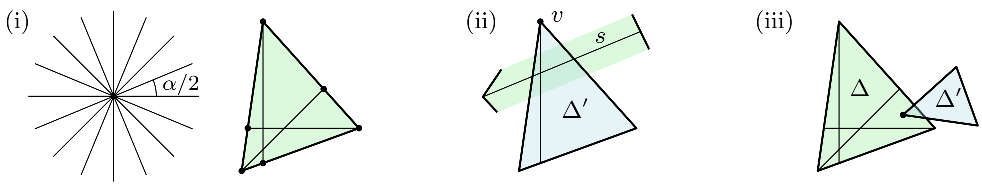

Figure 6 (i) The set A of canonical directions and the canonical chords of a triangle. (ii) Chord s intersects two sides of Δ' so it intersects a canonical chord of Δ'. (iii) All canonical chords of Δ miss Δ', so Δ' must have a vertex inside Δ.

Let $A := \{i \cdot (\alpha/2) : 0 \leqslant i \leqslant \lfloor 4\pi/\alpha \rfloor\}$ be a set of $O(1/\alpha) = O(1)$ *canonical directions*, where α is the fatness constant of the triangles. Define a *canonical segment* to be a segment whose direction is canonical, and define a *canonical chord* of a fat triangle Δ to be a canonical segment that connects a vertex of Δ to its opposite side; see Figure 6(i). Note that each vertex of Δ admits a canonical chord. Let $S(\Delta)$ be a set of three canonical chords of Δ, one per vertex of Δ, and let $P(\Delta)$ be the endpoints of these chords (including the vertices of Δ).

▶ **Observation 10.** *If a fat triangle Δ intersects a fat triangle Δ' then at least one of the following conditions holds:*
- **(i)** *Δ contains a vertex of Δ';*
- **(ii)** *a point from $P(\Delta)$ lies inside Δ';*
- **(iii)** *a canonical chord from $S(\Delta)$ intersects a canonical chord from $S(\Delta')$.*

Proof. Suppose that condition (ii) does not hold. Then there is a canonical chord $s \in S(\Delta)$ that intersects two edges of Δ', or all canonical chords in $S(\Delta)$ are disjoint from Δ'. In the former case, s separates a vertex v of Δ' from its opposite side, which implies that s intersects the canonical chord of v; see Figure 6(ii). Thus, condition (iii) holds in this case. In the latter case, Δ' must have a vertex inside Δ and condition (i) holds; see Figure 6(iii). ◀

Observation 10 implies the following lemma.

▶ **Lemma 11.** *Let R be a set of n fat triangles. There exists a data structure of $O(n \log^2 n)$ size such that we can decide in $O(\log^3 n)$ time if a fat query triangle Δ intersects any triangle in R and, if so, report a witness triangle. The data structure can be built in $O(n \log^3 n)$ time.*

Proof. The data structure consists of three separate data structures, each handling one of the conditions in Observation 10.

For condition (i) we need a data structure for range searching with fat triangles in a set V of $3n$ points, namely the vertices of the triangles in R. As observed by Gray [18, page 14], such queries can be answered in $O(\log^3 n)$ time with a structure of size $O(n \log^2 n)$ that can be built in $O(n \log^3 n)$ time.

For condition (ii) we use a *union tree* on the triangles in R. This is a binary tree $\mathcal{T}$ whose leaves correspond to the triangles in R, and whose internal nodes store the union of the triangles in their subtree, preprocessed for point-location queries. Checking if a query point q is contained in any triangle in R can be done in $O(\log n)$ by performing a point location at the root of $\mathcal{T}$. If this is the case, then we can find a witness triangle in $O(\log^2 n)$ time by walking down $\mathcal{T}$, always proceeding to a child whose union contains q. Since the union complexity of n fat triangles is $O(n \log^* n)$ [2], the total amount of storage is $O(n \log^* n \log n)$. Note that the union at a node ν can be computed in $O(n_\nu \log^* n_\nu \log n_\nu)$ time from the unions of its two children by a simple plane-sweep algorithm, where n_ν is the number of triangles in the subtree of ν. Hence, the total preprocessing time is $O(n \log^* n \log^2 n)$.

To handle condition (iii) we proceed as follows. Let $S_R : \{s \in S(\Delta) : \Delta \in R\}$ be the set of canonical chords of the triangles in R. We will build $O(1/\alpha)$ different data structures on S_R, one for each of the $O(1/\alpha)$ possible directions of the query chord. To construct a data structure for a fixed direction of the query chord, we partition S_R into $O(1/\alpha)$ classes, one for each canonical direction in A, and we build a separate substructure for each class A_i. Assume wlog that the query chord is vertical and that the chords in A_i are horizontal. Then we can use an interval tree [12, Section 10.1] to answer queries in $O(\log n)$, using $O(n)$ space and $O(n \log n)$ preprocessing. (Using a point-location structure on the vertical decomposition of the chords in A_i is another option.) Thus the total amount of storage is $O(n/\alpha) = O(n)$, the total preprocessing is $O((n/\alpha)\log n) = O(n \log n)$, and the total query time is $O((1/\alpha)\log n) = O(\log n)$.

Since all data structure work within the claimed bounds, the lemma follows. ◀

We can now solve Bichromatic Intersection Testing for fat triangles in $O((n_B + n_R)\log^3(n_B + n_R))$ time, by querying with each triangle $\Delta \in B$ in the data structure provided by Lemma 11. This leads to the following theorem.

▶ **Theorem 12.** *Let $\mathcal{D}$ be a set of n fat triangles in the plane. Then we can compute a shortest-path tree in $\mathcal{G}^{\times}[\mathcal{D}]$ for a given source triangle $\Delta_{\text{src}} \in \mathcal{D}$ in $O(n \log^3 n)$ time.*

In the full version [11] we provide a better data structure for the range-query problem considered in Lemma 11: its construction time is $O(n \log^2 n)$ and each query is then answered in $O(\log^2 n)$ time. Using this data structure, we obtain an $O(n \log^2 n)$ algorithm to compute shortest-path trees on the intersection graph of fat triangles.

5 Concluding remarks

We presented the first algorithm for the SSSP problem in disk graphs that runs in $O(n \log n)$ (expected) time. We extended the algorithm to intersection graphs of fat triangles, where we obtain a worst-case running time of $O(n \log^3 n)$. In the full version [11] we provide a deterministic $O(n \log n)$ algorithm for the SSSP problem in disk graphs and improve the running time for intersection graphs of fat triangles to $O(n \log^2 n)$.

A natural question is whether the algorithm for fat triangles can be further improved. Another natural question is whether an $O(n \operatorname{polylog} n)$ algorithm is possible for intersection graphs of non-fat objects such as line segments. This is unlikely, however, because Hopcroft's problem – given a set L of n lines and a set P of n points, decide if any of the points lies on any of the lines – can be reduced to the SSSP problem for segments, and Hopcroft's problem has an $\Omega(n^{4/3})$ lower bound [16] in a somewhat restricted model of computation.

References

1. Pankaj K. Agarwal, Matthew J. Katz, and Micha Sharir. Computing depth orders for fat objects and related problems. *Comput. Geom.*, 5:187–206, 1995. doi:10.1016/0925-7721(95)00005-8.
2. Boris Aronov, Mark de Berg, Esther Ezra, and Micha Sharir. Improved bounds for the union of locally fat objects in the plane. *SIAM J. Comput.*, 43(2):543–572, 2014. doi:10.1137/120891241.
3. Ivan J. Balaban. An optimal algorithm for finding segments intersections. In *Proc. 11th Symposium on Computational Geometry (SoCG)*, pages 211–219, 1995. doi:10.1145/220279.220302.

4 Sergio Cabello and Miha Jejčič. Shortest paths in intersection graphs of unit disks. *Comput. Geom.*, 48(4):360–367, 2015. doi:10.1016/j.comgeo.2014.12.003.

5 Timothy M. Chan. Polynomial-time approximation schemes for packing and piercing fat objects. *J. Algorithms*, 46(2):178–189, 2003. doi:10.1016/S0196-6774(02)00294-8.

6 Timothy M. Chan. A dynamic data structure for 3-d convex hulls and 2-d nearest neighbor queries. *J. ACM*, 57(3):16:1–16:15, 2010. doi:10.1145/1706591.1706596.

7 Timothy M. Chan and Zhengcheng Huang. Constant-hop spanners for more geometric intersection graphs, with even smaller size. In *Proc. 39th International Symposium on Computational Geometry (SoCG)*, volume 258 of *LIPIcs*, pages 23:1–23:16, 2023. doi:10.4230/LIPICS.SoCG.2023.23.

8 Timothy M. Chan and Dimitrios Skrepetos. All-pairs shortest paths in unit-disk graphs in slightly subquadratic time. In Seok-Hee Hong, editor, *Proc. 27th International Symposium on Algorithms and Computation (ISAAC)*, volume 64 of *LIPIcs*, pages 24:1–24:13, 2016. doi:10.4230/LIPICS.ISAAC.2016.24.

9 Timothy M. Chan and Dimitrios Skrepetos. All-pairs shortest paths in geometric intersection graphs. *J. Comput. Geom.*, 10(1):27–41, 2019. doi:10.20382/JOCG.V10I1A2.

10 Bernard Chazelle and Herbert Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *J. ACM*, 39(1):1–54, 1992. doi:10.1145/147508.147511.

11 Mark de Berg and Sergio Cabello. An $O(n \log n)$ algorithm for single-source shortest paths in disk graphs, 2025. doi:10.48550/arXiv.2506.07571.

12 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational Geometry: Algorithms and Applications (3rd Edition)*. Springer, 2008. doi:10.1007/978-3-540-77974-2.

13 Herbert Edelsbrunner, Leonidas J. Guibas, and Jorge Stolfi. Optimal point location in a monotone subdivision. *SIAM J. Comput.*, 15(2):317–340, 1986. doi:10.1137/0215023.

14 Alon Efrat, Alon Itai, and Matthew J. Katz. Geometry helps in bottleneck matching and related problems. *Algorithmica*, 31(1):1–28, 2001. doi:10.1007/S00453-001-0016-8.

15 David Eppstein, Michael T. Goodrich, and Jonathan Z. Sun. Skip quadtrees: Dynamic data structures for multidimensional point sets. *Int. J. Comput. Geom. Appl.*, 18:131–160, 2008. doi:10.1142/S0218195908002568.

16 Jeff Erickson. New lower bounds for Hopcroft's problem. *Discret. Comput. Geom.*, 16(4):389–418, 1996. doi:10.1007/BF02712875.

17 Steven Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2:153–174, 1987. doi:10.1007/BF01840357.

18 Chris Gray. *Algorithms for Fat Objects: Decompositions and Applications*. PhD thesis, TU Eindhoven, 2008. URL: https://pure.tue.nl/ws/portalfiles/portal/3256691/200811208.pdf.

19 Sariel Har-Peled. *Geometric approximation algorithms*, volume 173 of *Mathematical Surveys and Monographs*. American Mathematical Society, 2011. doi:10.1090/surv/173.

20 Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. Dynamic planar Voronoi diagrams for general distance functions and their algorithmic applications. *Discret. Comput. Geom.*, 64(3):838–904, 2020. doi:10.1007/S00454-020-00243-7.

21 Katharina Klost. An algorithmic framework for the single source shortest path problem with applications to disk graphs. *Comput. Geom.*, 111:101979, 2023. doi:10.1016/j.comgeo.2022.101979.

22 Chih-Hung Liu. Nearly optimal planar k nearest neighbors queries under general distance functions. *SIAM Journal on Computing*, 51(3):723–765, 2022. doi:10.1137/20M1388371.

23 Ketan Mulmuley. A fast planar partition algorithm, II. *J. ACM*, 38(1):74–103, 1991. doi:10.1145/102782.102785.

24 Liam Roditty and Michael Segal. On bounded leg shortest paths problems. *Algorithmica*, 59(4):583–600, 2011. doi:10.1007/S00453-009-9322-3.