Efficiently Computing All Delaunay Triangles Occurring over All Contiguous Subsequences

Stefan Funke
Universität Stuttgart, Germany
funke@fmi.uni-stuttgart.de

Felix Weitbrecht
Universität Stuttgart, Germany
weitbrecht@fmi.uni-stuttgart.de

Abstract

Given an ordered sequence of points $P = \{p_1, p_2, \ldots, p_n\}$, we are interested in computing T, the set of distinct triangles occurring over all Delaunay triangulations of contiguous subsequences within P. We present a deterministic algorithm for this purpose with near-optimal time complexity $O(|T| \log n)$. Additionally, we prove that for an arbitrary point set in random order, the expected number of Delaunay triangles occurring over all contiguous subsequences is $\Theta(n \log n)$.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Computational Geometry, Delaunay Triangulation, Randomized Analysis

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2020.28

1 Introduction

For an ordered sequence of points $P = \{p_1, p_2, \ldots, p_n\}$, we consider for $1 \leq i < j \leq n$ the contiguous subsequences $P_{i,j} := \{p_i, p_{i+1}, \ldots, p_j\}$ and their Delaunay triangulations $T_{i,j} := DT(P_{i,j})$. We are interested in the set $T := \bigcup_{i<j} \{t \mid t \in T_{i,j}\}$ of distinct Delaunay triangles occurring over all contiguous subsequences. Figure 1 shows a sequence of points $p_1, \ldots, p_n$ where $|T| = \Omega(n^2)$ (collinearities could be perturbed away). For $j > \frac{n}{2}$, any point p_j will be connected to all points $\{p_1, \ldots, p_{\frac{n}{2}}\}$ in $T_{1,j}$, so any such $T_{1,j}$ contains $\Theta(n)$ Delaunay triangles not contained in any $T_{1,j'}$ with $j' < j$, hence $|T| = \Omega(n^2)$. Note that for this argument we only used linearly many contiguous subsequences. It is conceivable that the quadratically many contiguous subsequences create even a superquadratic number of distinct Delaunay triangles. We will show, though, that if P is in *random order*, $E[|T|] = \Theta(n \log n)$, and $|T| = O(n^2)$ for any order. Then we design a deterministic algorithm to compute T with asymptotically near-optimal time complexity $O(|T| \log n)$.

1.1 Motivation and Related Work

Subcomplexes of the Delaunay triangulation have proven to be very useful for representing the shape of objects from a discrete sample in many contexts, see for example α-shapes [4], the β-skeleton [10], or the crust [2]. If the samples are acquired over time, a subcomplex of the Delaunay triangulation of the samples within a contiguous time interval might allow for interesting insights into the data, see for example [3], where the authors use α-shapes to visualize the regions of storm event data within the United States between 1991 and 2000.

While samples do not occur in truly random order in real world scenarios, it has been observed in [3] that the potentially huge size of T seems more like a pathological setting. Our first result provides some sort of theoretical explanation for this observation. Our second result shows that T can also be computed in time $\widetilde{O}(|T|)$. This suggests the possibility of precomputing all Delaunay triangles of all contiguous subsequences and indexing them with

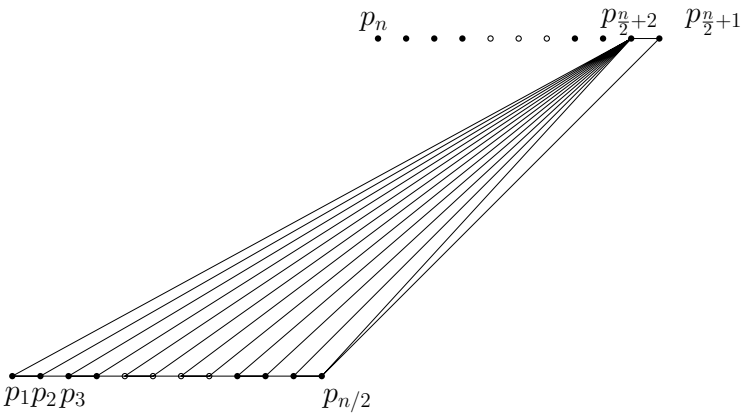

Figure 1 A sequence of points with $|T| = \Theta(n^2)$, as in [7].

respect to time and possibly some other parameter (e.g., the α value for α-shapes, or the β value for the β-skeleton). Then a time interval query, possibly with an α/β parameter, could be answered in an output sensitive manner, which might be considerably faster than computing the structures from scratch for the interval of interest.

The incremental construction of the Delaunay triangulation seems like a promising starting point for construction of T. Here the Delaunay triangulation of a point set is computed by inserting the points one-by-one, always updating an accompanying point location data structure and the triangulation itself via a sequence of flip operations. With the analysis in [7], one can show in expectation for random point orderings that only $O(1)$ new Delaunay triangles are created upon a point insertion, and hence there are $O(n)$ Delaunay triangles alive during the process. The expected location costs are $O(n \log n)$ overall, but for arbitrary orderings the running time might be quadratic in the number of created Delaunay triangles due to the point location costs. Furthermore, while this algorithm on the way constructs $T_{1,1}, T_{1,2}, \ldots, T_{1,n}$, triangulations $T_{>1,j}$ are not considered at all.

Other approaches for constructing Delaunay triangulations or Voronoi diagrams like sweepline [5], or divide and conquer [6] – at least at first sight – do not appear to be promising starting points for enumerating T.

Along the lines of our work, Kaplan et al. in [9] have investigated the complexity of the overlay of the features (including transient ones) constructed during the randomized incremental construction of minimization diagrams. While the incremental construction of Delaunay triangles can indeed be expressed as a suitable minimization diagram, Kaplan et al. only consider prefixes in contrast to contiguous subsequences in our case.

1.2 Contribution

This paper has two main contributions. First, we show that for arbitrary point sets in random order, the expected size of T is $\Theta(n \log n)$ and in the worst case $O(n^2)$. Secondly, we present an output sensitive algorithm to compute T in time $O(|T| \log n)$. Our results lay the theoretical foundation for faster algorithms that make use of subcomplexes of the Delaunay triangulation for time series data. In Section 2 we analyze the worst case as well as the expected size of T, Section 3 develops a deterministic output sensitive algorithm for computing T. Section 4 presents some preliminary results of a prototypical implementation which support our theoretical findings. We conclude with an outlook and directions for future research.

2 Counting Delaunay Edges and Triangles

Our proof will proceed in two steps. We first show that the expected number of Delaunay edges encountered when considering all contiguous subsequences in a randomly ordered sequence of points is $\Theta(n \log n)$. Then we show that, for an arbitrary order, there is a linear dependence between the number of Delaunay triangles and Delaunay edges.

Assuming non-degeneracy of P, i.e., absence of four co-circular or three co-linear points, we define the set of edges used in triangles of T as:

$$E_T := \{e \mid \exists\ t \in T : e \text{ edge of } t\}$$

We first bound the expected size of E_T. The following lemma is a simple observation that helps to focus on a smaller subsequence when considering a potential edge $\{p_i, p_j\}$.

▶ **Lemma 1.** *For any order of P, if an edge $e = \{p_i, p_j\} \in E_T$ $(i < j)$, then e also appears in $T_{i,j}$.*

Proof. For suitable $a \le i, b \ge j$, e is a Delaunay edge in $T_{a,b}$, i.e. there exists a disk with p_i, p_j on its boundary and its interior free of points from $P_{a,b}$. As $P_{i,j} \subseteq P_{a,b}$ this disk is also free of points from $P_{i,j}$, hence $e \in T_{i,j}$. ◀

Lemma 1 states that we only need to consider the minimal contiguous subsequence containing p_i and p_j to argue about the probability of an edge $e = \{p_i, p_j\}$ being present in E_T. We bound the probability that e is a Delaunay edge in $T_{i,j}$:

▶ **Lemma 2.** *For a potential edge $e = \{p_i, p_j\}$, $i < j$, we have $Pr[e \in T_{i,j}] < \frac{6}{j-i}$.*

Proof. The claim holds trivially for $j = i + 1$. For point sets $P_{i,j}$ with $j > i + 1$, observe that clearly $T_{i,j}$ will be the same regardless of how the points in $P_{i,j}$ are ordered. All points in $P_{i,j}$ are equally likely to be p_i, or p_j. The triangulation $T_{i,j}$ is a planar graph with more than 2 nodes, and hence per Euler's formula contains at most $3(j - i + 1) - 6$ edges. So $Pr[e \in T_{i,j}]$ is bounded by the probability of two randomly chosen nodes in a graph with $j - i + 1$ nodes and $\le 3(j - i + 1) - 6$ edges to be connected with an edge. By randomly choosing two nodes, we randomly choose one edge among all potential $\binom{j-i+1}{2}$ edges. The probability of that edge to be one of the $\le 3(j - i + 1) - 6$ edges of $T_{i,j}$ is $< \frac{6}{j-i}$. ◀

Due to Lemma 1 we have that $Pr[e \in E_T] = Pr[e \in T_{i,j}]$, hence:

▶ **Lemma 3.** *The expected size of E_T is $\Theta(n \log n)$.*

Proof. Linearity of expectation allows us to sum all potential edges' probability of existence, bounded using Lemma 2, to upper bound the expected size of E_T.

$$\begin{aligned} E[|E_T|] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} Pr[\{p_i, p_j\} \in E_T] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} Pr[\{p_i, p_j\} \in T_{i,j}] \\ &\le \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{6}{j-i} = 6 \sum_{i=1}^{n-1} \sum_{j=1}^{n-i} \frac{1}{j} \le 6 \sum_{i=1}^{n-1} H_n = O(n \log n) \end{aligned}$$

For the lower bound, consider the nearest neighbor of point p_1 in $P_{2,i}$ as i grows from 2 to n. It is well known that for random order of P, the nearest neighbor changes $\Theta(\log n)$ times in expectation. It is also well known that the nearest neighbor graph of a point set is a subgraph of its Delaunay triangulation, so p_1 in expectation is involved in the creation of $\Omega(\log n)$ distinct Delaunay edges. The same argument applies to all other points, yielding a $\Omega(n \log n)$ bound. ◀

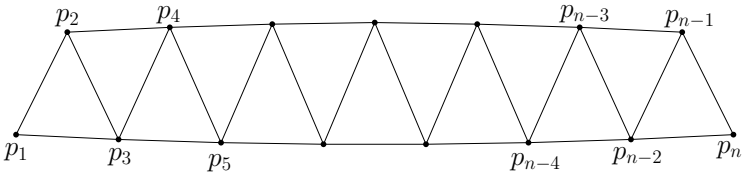

Figure 2 A sequence of points with $|T| = O(n)$.

In general, an edge might be used by up to $\Omega(n)$ different Delaunay triangles, so a linear dependence between the number of Delaunay edges and triangles is not immediately obvious. Yet, the following lemma shows why this is the case.

▶ **Lemma 4.** *For any order of P, $|T| \in \Theta(|E_T|)$.*

Proof. Consider some Delaunay triangle $t = p_a p_b p_c \in T$ (w.l.o.g. $a < b < c$). Due to Lemma 1, we have $t \in T_{a,c}$. Apart from t, there can exist at most one other triangle $t' \in T_{a,c}$ which uses the edge $\{p_a, p_c\}$. This way, we can charge every Delaunay triangle of T to some Delaunay edge of E_T, charging at most 2 Delaunay triangles to any Delaunay edge. So the overall number of Delaunay triangles is at most twice the overall number of Delaunay edges, hence $|T| \in O(|E_T|)$. Any triangle creates at most 3 edges, so $|E_T| \in O(|T|)$. ◀

As a corollary, Lemma 4 implies that $O(n^2)$ is also an upper bound for $|T|$ for arbitrary orderings of n points, as there are only $O(n^2)$ possible edges.

Finally we can state our main theorem which follows from Lemmas 3 and 4.

▶ **Theorem 5.** *The expected number of different Delaunay triangles occurring over all contiguous subsequences of a (uniformly) randomly ordered point set of size n is $\Theta(n \log n)$.*

3 Constructing Delaunay Triangles

Given an arbitrary (but non-degenerate) point set P in arbitrary order, the most naive way of deterministically constructing all Delaunay triangles is the execution of some $O(n \log n)$ algorithm for each sequence $P_{i,j}$, which results in worst-case $O(n^3 \log n)$ running time. Unfortunately, this approach also takes at least cubic time if $|T| = O(n)$, like it is the case in Figure 2 where removing any number of points from either end of the interval of interest creates no new triangles. Another approach could be to execute the incremental algorithm (without randomization) from [7] for the sequences $P_{1,n}$, $P_{2,n}$, …, $P_{n-2,n}$, which produces all $T_{i,j}$ with $j < n$ as intermediate results. It does not seem obvious, though, how to show that a single run on $T_{i,n}$ can be performed in subquadratic time in the worst case; flipping costs can be charged to $|T_{i,n}|$, but point location costs could be quadratic despite a small $|T|$. So a runtime better than $O(n^3)$ in the worst case seems difficult to achieve even if $|T| = O(n)$. Our goal in the following will be the development of a deterministic algorithm for constructing T in time proportional to $|T|$, up to a logarithmic factor.

The resulting algorithm is similar to the incremental algorithm in that points are inserted one-by-one. The incremental algorithm maintains one triangulation of the full point set $P_{1,j}$ which is expanded as points are inserted. Our algorithm computes the same full triangulations $T_{1,3}$, …, $T_{1,n}$, but in reverse order. Additionally, our algorithm maintains many partial triangulations of usually very small subsets of $P_{i,j}$. These triangulations are expanded as points are inserted, but at the same time they are also pruned to avoid doing the same triangulation work multiple times for different partial triangulations.

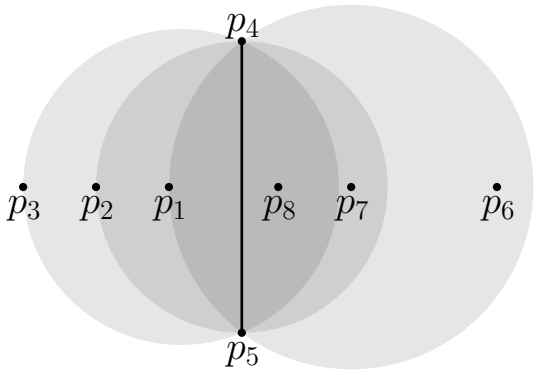

Figure 3 An edge which requires information of size $\Omega(n)$ to encode its lifetime. The edge $\{p_4, p_5\}$ is part of $T_{1,5}, T_{2,6}, T_{3,7}$, but not $T_{1,6}, T_{2,7}, T_{3,8}$. Other edges of these triangulations are omitted to keep the figure comprehensible.

3.1 Preliminary Considerations

In our construction algorithm we will be concerned with the construction of Delaunay *triangles* and only indirectly with Delaunay *edges*, since the former provide a significant advantage for data structures indexing the result set. We will show that the lifetime of a Delaunay triangle can be represented compactly via two indices, whereas for a Delaunay edge this might require additional information of size $\Omega(n)$. This is because an edge is Delaunay if there exists an open disk free of other points through the edge's two endpoints. As demonstrated in Figure 3, an edge e could be Delaunay in some $T_{a,b}$, not Delaunay in $T_{a,b+1}$, Delaunay in $T_{a+1,b+1}$, not Delaunay in $T_{a+1,b+2}$, etc. by repeatedly inserting a point close to e on one side, and then removing the closest point on the other side. During this process, inserting a close point on one side prevents the existence of such an empty disk on e, and removing the closest point on the other side allows an empty disk to exist again.

Conceptually, we care about all triangulations $T_{i,j}, i < j$, as depicted here:

$$\begin{array}{ccccccccccc}
T_{1,2} & T_{1,3} & \dots & \dots & \dots & T_{1,j} & \dots & \dots & T_{1,n} \\
 & T_{2,3} & \dots & \dots & \dots & T_{2,j} & \dots & \dots & T_{2,n} \\
 & & & & & \vdots & & & \\
 & & & T_{i,i+1} & \dots & T_{i,j} & \dots & \dots & T_{i,n} \\
 & & & & & \vdots & & & \\
 & & & & & & & T_{n-2,n-1} & T_{n-2,n} \\
 & & & & & & & & T_{n-1,n}
\end{array}$$

This matrix allows us to use terms such as "above" or "row i" to describe the relationship between triangulations and other data structures with indices corresponding to a triangulation's first and last points. As ordered point sets exist with $|T| = O(n)$, we cannot construct all $\Theta(n^2)$ triangulations in the matrix explicitly. We have to use the fact that Delaunay triangulations of similar point sets (i.e., a small Manhattan distance in the matrix) tend to share many edges and triangles, with differences being local to removed and added points' neighborhoods.

To that end, let us first define an ordering on the triangulations in the matrix.

$$T_{a,b} < T_{c,d} \iff b < d \vee (b = d \wedge a < c)$$

That is, "smaller" means all columns to the left, and that part of the column that is above the current triangulation. The contribution set $C_{i,j}$ is the set of triangles for which $T_{i,j}$ is the smallest triangulation they appear in:

$$C_{i,j} = \{t \in T_{i,j} \mid \forall\ T_{a,b} < T_{i,j} : t \notin T_{a,b}\}$$

Since any triangle appears "first" in exactly one triangulation, the number of non-empty $C_{i,j}$ is $O(|T|)$, so we can at least afford to consider all cells of the matrix with non-empty contribution sets. To that end, let us characterize $C_{i,j}$ in a different way. Inserting a point p_j into $T_{i,j-1}$ destroys all triangles that have p_j in their circumcircle, and, in their place, creates a star-shaped set of triangles incident to p_j. As $T_{i,j-1} < T_{i,j}$, we have:

$$C_{i,j} \subseteq \{t \in T_{i,j} \mid t \text{ incident to } p_j\} \tag{1}$$

Removing p_{i-1} from $T_{i-1,j}$ does the opposite, destroying all triangles incident to p_{i-1} and leaving a star-shaped hole in $T_{i-1,j}$. Re-triangulating this hole's interior yields $T_{i,j}$ without any further changes. For $i > 1$ we have $T_{i-1,j} < T_{i,j}$, so:

$$C_{i,j} \subseteq \{t \in T_{i,j} \mid p_{i-1} \text{ in } t\text{'s circumcircle}\} \tag{2}$$

In fact, these two conditions define $C_{i,j}$ sufficiently:

▶ **Lemma 6.** *$C_{i,j} = \{t \in T_{i,j} \mid p_{i-1}$ in t's circumcircle $\wedge$ t incident to $p_j\}$ for $i > 1$.*

Proof. The $\subseteq$ inclusion follows directly from (1) and (2). To prove the reverse inclusion, consider some triangle $t \in T_{i,j}$ incident to p_j which has p_{i-1} in its circumcircle, and some $T_{i',j'}$ with $t \in T_{i',j'}$. If $i' < i$, $t \notin T_{i',j'}$ because p_{i-1} is inside t's circumcircle. If $j' < j$, $t \notin T_{i',j'}$ because t requires p_j. Thus $T_{i,j} \leq T_{i',j'}$, so $T_{i,j}$ is the first triangulation containing t and $t \in C_{i,j}$. ◀

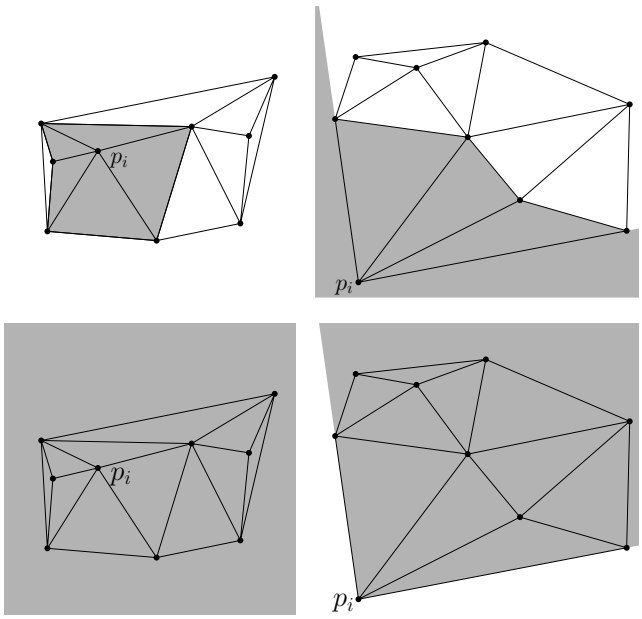

Figure 4 Examples of a point's star shape (top) and populated sector (bottom) for interior points (left) and points on the convex hull (right).

We define the *star shape* of point p_i (w.r.t. $T_{i,j}$) as the area of the plane visible from p_i without edges obstructing the view. We define the *populated sector* of point p_i (w.r.t. $T_{i,j}$) as the smallest circle sector of infinite radius centered on p_i which contains triangles incident to p_i. Figure 4 illustrates these concepts.

Maintaining the entire Delaunay triangulation of $P_{i,j}$ for every row of the matrix is prohibitively expensive. As noted above, the changes resulting from the removal of p_{i-1} from $T_{i-1,j}$ are local to p_{i-1}. We would like to exploit that fact by maintaining only the difference between $T_{i-1,j}$ and $T_{i,j}$ for all rows $i > 1$, and we do that using a *Hole Triangulation* $H_{i,j}$, which contains the triangles resulting from the removal of p_{i-1} from $T_{i-1,j}$. When speaking of star shape or populated sector in the context of some hole triangulation $H_{i,j}$, we mean the respective star shape or populated sector of p_{i-1} w.r.t. $T_{i-1,j}$.

Figure 5 shows how a hole triangulation may evolve as points are inserted. The removal of p_{i-1} from $T_{i-1,j}$ replaces p_{i-1} and all its incident triangles with a new set of triangles. We call those edges of p_{i-1}'s incident triangles that are opposite p_{i-1} *the boundary*, as they bound the star shape. We say the boundary is *complete* once it surrounds p_{i-1}, i.e. once the populated sector is the entire plane. The new set of triangles is also adjacent to the boundary, but until the boundary is complete, some boundary edges may not have an adjacent triangle in $H_{i,j}$, as is the case in Figure 5, left. We also maintain these boundary edges in hole triangulations so they're available once a point is inserted on p_{i-1}'s boundary that creates triangles against these edges, as in Figure 5, middle right. So, technically, a hole triangulation is not necessarily a proper triangulation because it can contain edges without adjacent triangles even when more than two points are present. This is not only because of the additional boundary edges it maintains, but also because it only maintains triangles inside the star shape, so concave boundaries are not triangulated behind the boundary – that area is handled by other triangulations.

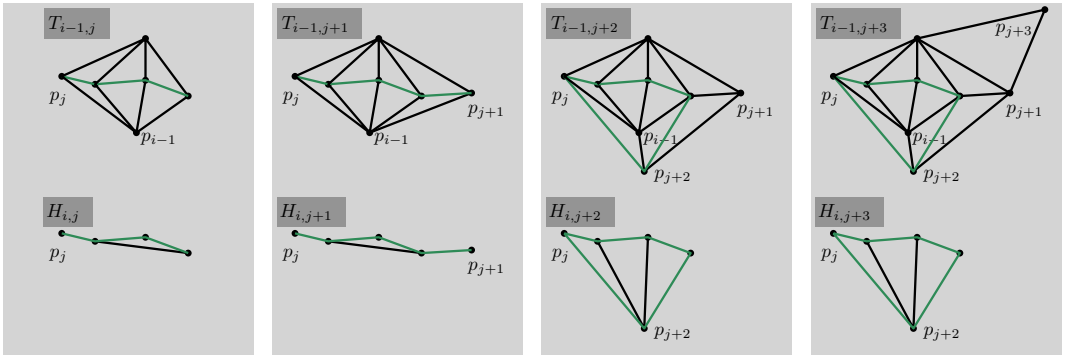

Figure 5 On the bottom, the evolution of a hole triangulation is shown as points are inserted. Above, the full triangulations depicting the situation before the removal of p_{i-1} are shown. They are only shown for reference, the algorithm does not maintain these full triangulations. Boundary edges, which are present both in the original triangulation and in the hole triangulation, are shown in green. **Left:** The point p_{i-1} is on the convex hull of $T_{i-1,j}$, so its boundary is incomplete. The hole triangulation contains an edge without an adjacent triangle. **Middle left:** Another point is inserted, adding a second boundary edge without an adjacent triangle to the hole triangulation. **Middle right:** The insertion of p_{j+2} tightens the boundary around p_{i-1}. The right-most boundary edge from before is no longer a boundary edge, so it is removed from the hole triangulation, but it is still a Delaunay edge in $T_{i-1,j+2}$ and $T_{i,j+2}$. The boundary is now complete and the left-most edge, which previously did not have an adjacent triangle in the hole triangulation, has formed a triangle with the newest point. **Right:** A point is inserted without creating an edge to p_{i-1}, so the hole triangulation does not change.

3.2 Main Algorithm Overview

Here we describe the main part of the algorithm that computes all Delaunay triangles. Later subsections elaborate on data structures and update procedures.

Our algorithm will work through the matrix column-by-column from left to right and conceptually each column is considered top to bottom. The first row of the matrix is precomputed in a way similar to an incremental construction as in [7], but with additional logic to avoid point location operations. As a result, the main part of the algorithm does not compute anything for the first row, instead it only works with precomputed contribution sets $C_{1,j}$. After work for column j is completed, we have for each row i with $1 < i < j$ a current hole triangulation $H_{i,j}$ of the inside of p_{i-1}'s star shape. As $H_{i,j}$ is a subgraph of $T_{i,j}$ which mainly describes the difference between $T_{i-1,j}$ and $T_{i,j}$, we could – in principle – recover $T_{i,j}$ by computing $T_{1,j}$ from $C_{1,3}$, $C_{1,4}$, …, $C_{1,j}$ and then successively applying the "differences" $H_{2,j}$, $H_{3,j}$, …, $H_{i,j}$ to $T_{1,j}$.

Once all columns up to $j-1$ have been computed, we compute all necessary updates for column j. For the first row, we simply retrieve the precomputed $C_{1,j}$. For rows $i > 1$, $C_{i,j}$ can be constructed according to Lemma 6 by updating the corresponding $H_{i,j-1}$ to $H_{i,j}$. If p_j is not adjacent to p_{i-1} in $T_{i-1,j}$, p_{i-1}'s star shape does not change and $H_{i,j} = H_{i,j-1}$, $C_{i,j} = \emptyset$. Fortunately, if p_{i-1} is adjacent to p_j in $T_{i-1,j}$, this is discovered by some triangulation above row i that was already updated. That is because Lemma 1 guarantees that the edge $\{p_{i-1}, p_j\}$ exists, at the latest, in $T_{i-1,j}$.

So the necessary updates of $H_{i,j}$ can always be triggered from above, allowing us to avoid inspecting hole triangulations that do not change. In fact, any newly found Delaunay edge $\{p_a, p_b\}$, $a < b$, triggers exactly one hole triangulation update, namely the update from $H_{a+1,b-1}$ to $H_{a+1,b}$. These update triggers actually form a partial order over the updated hole triangulations, potentially leading to a computation order that deviates from the order defined in subsection 3.1. However, the definition of hole triangulations does not depend on update order, so the $C_{i,j}$ are still computed correctly.

▶ **Lemma 7.** *We have* $\sum_{i<j} |C_{i,j}| = |T|$ *and* $\sum_{i<j} |H_{i,j} \setminus H_{i,j-1}| = O(|T|)$.

Proof. Any triangle appears in exactly one contribution set, which yields the first part of the statement. For the second part, observe that all triangles of $H_{i,j} \setminus H_{i,j-1}$ appear in $C_{i,j}$. The boundary edges in $H_{i,j} \setminus H_{i,j-1}$ may not be used in triangles of $C_{i,j}$, but there are at most two of them. They can be charged to the edge triggering the hole triangulation update, and any of the $O(|T|)$ edges triggers exactly one update. ◀

In the following we fill in the details of how to update the hole triangulations, and in particular how to perform the point location operations necessary in hole triangulations and how to avoid them in the incremental construction.

3.3 Treatment of the first row of the matrix

The first row of the matrix is precomputed to avoid complicated or expensive point location logic in the main part of the algorithm. We first construct $T_{1,n}$ and then transform it into $T_{1,n-1}$, then into $T_{1,n-2,}$, …, then into $T_{1,2}$. Essentially, we perform the steps of an incremental construction of $T_{1,n}$ in reverse order. The triangles destroyed by the transformation of $T_{1,i}$ into $T_{1,i-1}$ are thus exactly $C_{1,i}$. We store these contribution sets on a stack so that the main part of the algorithm can access them efficiently as it works its way through the matrix column-by-column.

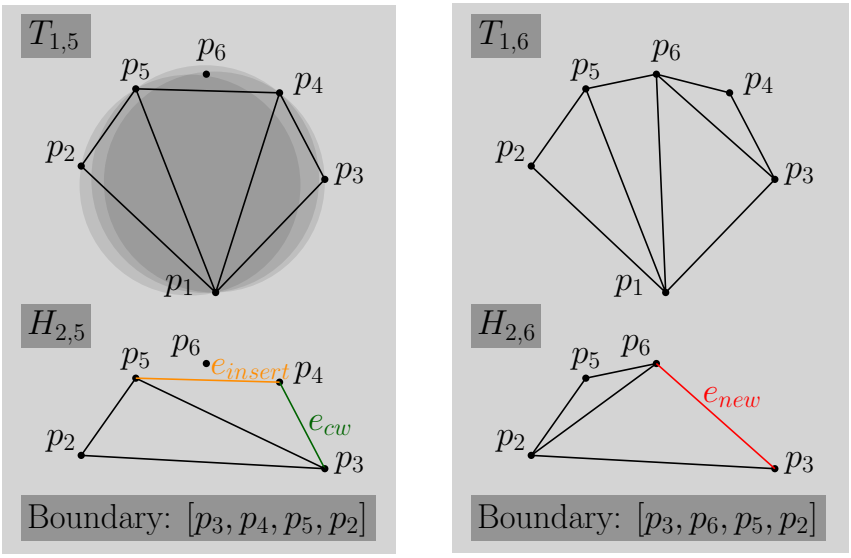

Figure 6 The sorted sequence representing the boundary of p_1 is updated as p_6 is inserted, $T_{1,5}$ and $T_{1,6}$ are shown for reference. The point p_6 is of course not part of $T_{1,5}$ or $H_{2,5}$, its position within these triangulations is only shown for reference.

▶ **Lemma 8.** *In time $O(n \log n + |T|)$ we can precompute all $C_{1,i}$.*

Proof. $T_{1,n}$ can be computed in time $O(n \log n)$ using one of many $O(n \log n)$ algorithms for computing Delaunay triangulations or Voronoi diagrams, for example [5]. We then transform $T_{1,n}$ step-by-step to obtain $T_{1,n-1}$, $T_{1,n-2,}$, …, $T_{1,2}$. Transforming $T_{1,i}$ into $T_{1,i-1}$ means removing p_i with its incident edges and retriangulating the possibly emerging hole with Delaunay triangles. Retriangulation can be accomplished in time linear in the number of points on p_i's boundary using [1] or [11]. That number is linear in $|C_{1,i}|$, so per Lemma 7, overall costs for retriangulation are $O(|T|)$. Immediately before p_i is removed, we store its incident triangles as $C_{1,i}$. ◀

3.4 Updating Hole Triangulations

During the course of our algorithm we have to update $H_{i,j-1}$ to $H_{i,j}$. To that end we maintain a hole triangulation's boundary in counter-clockwise order in a sorted sequence, as seen in Figure 6 where the sorted sequence representing the boundary of p_1 is updated upon insertion of p_6. The sorted sequence provides sufficient information to reconstruct p_{i-1}'s boundary, and thus also its incident triangles in $T_{i-1,j}$ as well as its star shape and populated sector. By using a $(2, 4)$-tree [8] to maintain the sorted sequence, we achieve logarithmic update and lookup times with space linear in the size of the boundary.

To update the sorted sequence, we find between which two points of the old boundary p_j is visible to p_{i-1} using a lookup in the sorted sequence. We call the boundary edge connecting these points e_{insert}. From there, we explore the sorted sequence in both directions to find the two points of the old boundary to which p_j will connect to form the new boundary. The two new boundary edges extend from p_j as far as the corresponding triangles of $T_{i-1,j-1}$ around p_{i-1} contain p_j in their circumcircle. So starting at p_j's insertion point in the sorted sequence, we explore boundary points in both directions as long as the respective triangles of p_{i-1}'s star shape contain p_j in their circumcircle. Of the corresponding explored boundary

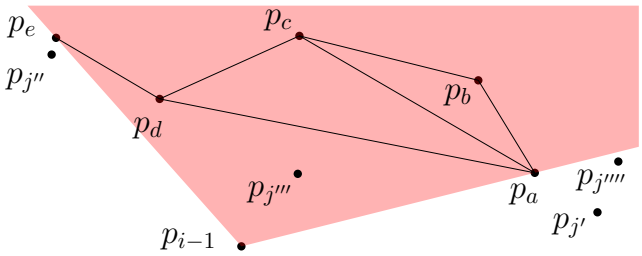

Figure 7 Potential positions of p_j in $H_{i,j-1}$, populated sector in red. The point p_{i-1} is not actually part of $H_{i,j}$, its position is only shown for reference.

edges, which will no longer be boundary edges after the insertion of p_j, we remember the clockwise-most one as e_{cw}. In Figure 6, only the two right-most shown circumcircles contain p_6, so the new boundary edges extend to p_5 on the left and to p_3 on the right.

We now know the two new boundary edges, but we do not immediately insert them into the hole triangulation because existing edges might be in the way. Knowledge of the new boundary edges will be useful in the remaining steps of the hole triangulation update, during which the new boundary edges and all other new edges will be created in a systematic way. Note that while the boundary is incomplete, we may only get one new boundary edge when the insertion point corresponds to an end of the sorted sequence rather than some edge e_{insert}. The remaining steps necessary to actually update the hole triangulation to match the new boundary are to locate p_j within $H_{i,j-1}$, and to insert p_j while keeping the hole triangulation Delaunay and restricted to the star shape. The following subsubsections describe these steps.

3.4.1 Locating a new point

Locating a new point p_j typically means finding the triangle that contains p_j, or, if p_j is outside the old triangulation, finding an edge visible to p_j. However, as hole triangulations always contain all boundary edges and triangles are never created behind the boundary, there is a third option in our case. When p_j is outside the populated sector and no edge of the old hole triangulation is visible to p_j from a side facing the star shape, like $p_{j''''}$ in Figure 7, we need to find the point to which p_j attaches. The following describes the logic necessary to distinguish these cases.

If p_j is not in the old populated sector, like most examples in Figure 7, it cannot be contained in any triangle of $H_{i,j-1}$. Of the hull edges covering $H_{i,j-1}$ on the inside of the star shape ($\{p_a, p_d\}$ and $\{p_d, p_e\}$ in Figure 7), we then consider that edge which is incident to that end of the boundary on whose side p_j is being inserted. In Figure 7, $p_{j''}$ would be inserted on the side of the boundary ending at p_e, so its covering edge would be $\{p_d, p_e\}$. The points $p_{j'}$ and $p_{j''''}$ would be inserted on the side of the boundary ending at p_a, so their covering edge would be $\{p_a, p_d\}$. If that side of the covering edge which faces the star shape is visible to p_j, like it is the case for $p_{j'}$ and $p_{j''}$, we return it as a visible edge. Otherwise, like it is the case for $p_{j''''}$, no edge can be visible from inside of the star shape and the point simply connects to the outermost point on the boundary, in this case p_a.

Things are more complicated if p_j falls inside the old populated sector. Then, p_j is inside the old star shape if and only if p_j and p_{i-1} are on the same side of e_{insert}. If p_j is outside the old star shape, i.e. behind the boundary, no triangle of $H_{i,j-1}$ can contain p_j because those are inside the old star shape, so e_{insert} is a visible edge for p_j. Note that this is the only case in which a new point creates a triangle against that side of an edge which faces outside the old star shape.

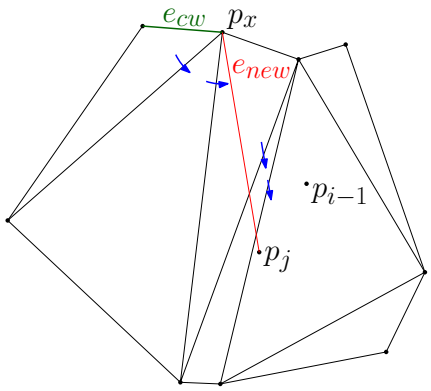

Figure 8 The search algorithm locating p_j in $H_{i,j-1}$. The position of p_{i-1} is only shown for reference.

Otherwise, p_j is inside the old star shape. If the boundary is incomplete, p_j might not be contained in any triangle, as is the case for $p_{j'''}$ in Figure 7. Regardless of boundary completion state, the search algorithm proceeds as follows. By definition, one of the new boundary edges in $H_{i,j}$, e_{new}, will connect p_j to one of e_{cw}'s points, say p_x. In Figure 6, p_x is p_3. Note that e_{new} is between e_{cw} and p_{i-1}. We search for p_j starting from e_{cw}. Figure 8 shows how the search algorithm traverses $H_{i,j-1}$. First, we walk along triangles incident to p_x until we find the triangle that is intersected by e_{new}. From there, we walk along the triangles that e_{new} intersects. If p_j is contained in some triangle, this process will find it. Otherwise, the search terminates at an edge visible to p_j.

3.4.2 Inserting a new point

Having located p_j on the old star shape's boundary, we need to update the hole triangulation by inserting p_j. This is done in three steps. First, we *create initial edges*, potentially extending the area covered by the old hole triangulation such that the area covered by the new hole triangulation is contained within the extended triangulation. Extending the covered area is not only necessary when the populated sector grows. Once the boundary is complete, new points may still effect an additive change in the star shape area, see Figure 9. The second step, *Delaunay Flipping*, ensures the Delaunay property of created edges and triangles, and finally the *Pruning* step cuts off any leftover triangles and edges caused by star shape shrinkage and by the first step extending the hole's area too much.

Create initial edges: If the location algorithm returns a containing triangle, we simply split it by inserting three edges between its points and p_j. If the location algorithm returns a visible edge $e_{visible}$ and p_j is inside the old star shape, more edges may have their side facing the star shape visible to p_j. We explore $H_{i,j-1}$'s hull starting at $e_{visible}$ to find all visible edges inside the old star shape, and create a triangle between p_j and each visible edge. We must stop exploring when we reach the outside of the hole triangulation, i.e. the "backside" of boundary edges. Otherwise p_j might create some triangles that are actually outside the star shape, see for example $p_{j'}$ and the edge $\{p_a, p_b\}$ in Figure 7.

If p_j is behind the boundary, i.e. in the populated sector but outside the star shape, the location algorithm returns e_{insert} as a visible edge. The points of e_{insert} are not necessarily those to which p_j will connect on the new boundary. Yet it is sufficient to create only two initial edges between p_j and e_{insert}'s points, for the following reason. If a new boundary edge is not incident to either of e_{insert}'s points, that new boundary edge will be between the

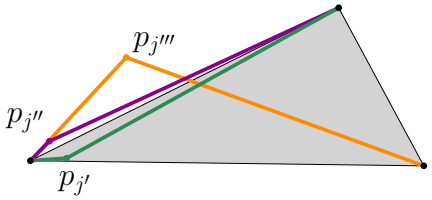

Figure 9 Potential positions of p_j with different effects on the area change between $H_{i,j-1}$, which is just one triangle, and $H_{i,j}$. The position of p_{i-1} is only shown for reference. Compared to $H_{i,j-1}$, $p_{j'}$ would reduce the area (green edges), $p_{j''}$ would extend the area (purple edges), and $p_{j'''}$ would extend the area in one direction, but reduce it in another (orange edges).

two initial edges, i.e. inside the extended triangulation area, and the flipping step will create them. This is the case for the right orange edge of $p_{j'''}$ in Figure 9. As the initial edges give a bound on the new boundary, further edges or triangles created behind the initial two edges would not be part of $H_{i,j}$.

Delaunay Flipping: The Delaunay Flipping algorithm is used to restore the Delaunay property of this intermediate triangulation, starting with the edges p_j attached to in the first step. In the first step we extended $H_{i,j-1}$ such that the area it now covers has all the edges the same area in $T_{i,j-1}$ would have after initial edge creation using the incremental construction algorithm. The star shape is not always convex, so we must also show that we never need to create edges outside the intermediate triangulation. This never happens because the circumcircles of triangles along the boundary (and, by a circle-shrinking argument, all other triangles) cannot extend back into the star shape from outside the star shape. But to create an edge outside the intermediate triangulation, p_j would need to lie inside the star shape (otherwise the necessary edges behind the old boundary would be created during initial edge creation) and inside the circumcircle of a triangle for which the resulting flipped edge would lie behind the old boundary.

So the flipping algorithm will have exactly the same effect on the area inside the new star shape as it would in a full incremental construction of $T_{i,n}$. There is actually one small difference: hole triangulations do not maintain points behind the boundary, so old boundary edges outside the new star shape cannot be flipped even if they are not Delaunay anymore. The next step discards that part of the intermediate triangulation, so this is not a real issue.

Pruning: Pruning is necessary to restrict hole triangulations to the new star shape. We cut off the triangles that are behind the two new boundary edges. Other, i.e. pre-existing, boundary edges do not have any triangles attached behind them. That is because any triangle created in previous steps uses one pre-existing and two newly created edges. The only pre-existing boundary edge p_j may have attached to from outside the star shape in the first step, e_{insert}, is no longer part of the boundary. The flipping algorithm never changes the area taken up by the triangulation, so it also cannot create triangles behind boundary edges.

Finally, $C_{i,j}$ are the newly created triangles which were not cut off in the last step. Even if a hole triangulation is updated, we may still get an empty $C_{i,j}$ if the boundary is incomplete and concave. But the following subsection shows that we can afford even these update costs.

3.5 Runtime analysis

▶ **Lemma 9.** *Point location costs in hole triangulations are* $O(|T| \log n)$.

Proof. Recall how points are located in hole triangulations. We first update the sorted sequence that represents the boundary, paying $O(\log n)$ per insertion and deletion. One insertion is triggered for all $O(|T|)$ Delaunay edges, and at most one deletion is possible for each insertion. So per Lemma 4, total costs to maintain the sorted sequences are $O(|T| \log n)$.

To actually locate p_j in a hole triangulation, we walk along that part of the old boundary whose points were just deleted from the sorted sequence by the insertion of p_j. This causes total costs linear in the total number of deletions from the sorted sequence, $O(|T|)$. We then walk along triangles of the old hole triangulation, only visiting triangles that contain p_j in their circumcircle: in the first step, we only visit triangles that lie behind the new boundary, and in the second step, we only visit triangles that intersect one of the edges created by p_j's insertion into the hole triangulation. All these Delaunay triangles are destroyed once p_j is inserted, so any triangle is only ever visited once by this search, hence total costs for hole triangulation traversal are $O(|T|)$. ◀

▶ **Lemma 10.** *Triangle and edge creation costs in hole triangulations are* $O(|T|)$.

Proof. During a hole triangulation update, initial triangles are created, each using two initial edges and one pre-existing edge, so it is sufficient to show that only $O(|T|)$ initial edges are created.

Any initial edge on the new boundary, i.e. any of the up to two new boundary edges created during initial edge creation, can be charged to the edge which triggered the hole triangulation update, and any of the $O(|T|)$ Delaunay edges triggers only one update. Any initial edge outside the new star shape can be charged to an adjacent pre-existing edge e with which it formed an initial triangle, so at most two initial edges are charged to e. As e is at least partially outside the new star shape, it will not be part of the resulting hole triangulation, so e is only ever charged for one hole triangulation update[1]. Any initial edge inside the new star shape is a new Delaunay edge, of which only $O(|T|)$ exist. Note that while the boundary edges are also Delaunay edges, they may already have appeared in many triangulations above, so we had to use a charging argument for them.

Lastly, the Delaunay Flipping algorithm causes costs linear in the number of initial triangles plus costs linear in the number of destroyed pre-existing (Delaunay) triangles. No triangle can be destroyed twice, so these costs are also $O(|T|)$. ◀

Finally, we can combine the previous lemmas to get a $O(|T| \log n)$ total runtime.

▶ **Theorem 11.** *The runtime of this algorithm to compute all Delaunay triangles occurring over all contiguous subsequences is* $O(|T| \log n)$.

Proof. For the first row of the matrix, the claim follows from Lemma 8 and $|T| \in \Omega(n)$. For all other rows, the claim follows from Lemmas 9 and 10. ◀

[1] Strictly speaking, e may also exist in other hole triangulations where it could also be charged. However, any instance of e in a fixed hole triangulation is only ever charged once like this.

Table 1 Average runtime and triangle counts for random point sets of size n.

n	$\lvert T\rvert$	T construction time	$\lvert \cup_{j\le n} T_{1,j}\rvert$	$T_{1,n}$ construction time
16,384	1,294,633	2,439 ms	97,830	103 ms
32,768	2,860,956	6,309 ms	196,168	260 ms
65,536	6,267,247	14,229 ms	392,592	745 ms
131,072	13,622,094	32,817 ms	785,879	1,779 ms
262,144	29,425,885	70,545 ms	1,572,292	4,068 ms
524,288	63,210,634	155,370 ms	3,144,770	9,008 ms
1,048,576	135,134,028	347,186 ms	6,290,562	20,374 ms
2,097,152	287,719,166	771,705 ms	12,581,989	44,082 ms

3.6 Lifetime of Delaunay Triangles for Indexing

As a byproduct of our algorithm run, we can compute the *lifetime* of Delaunay triangles: for any triangle $t = p_a p_b p_c$, $a < b < c$, we store the index k_{right} of the smallest index point in t's circumcircle with $k_{right} > c$, and the index k_{left} of the largest index point in t's circumcircle with $k_{left} < a$. We then have $t \in T_{i,j} \iff (k_{left} < i \le a) \wedge (c \le j < k_{right})$. These indices are computed as follows. The index k_{right} is always the index of that point which destroys t upon insertion. If no such point exists, k_{right} is ∞. The index k_{left} is $-\infty$ for triangles found in the first row of the matrix. For triangles found in rows $i > 1$, i.e. in a hole triangulation $H_{i,j}$, k_{left} is $i - 1$.

In applications where we have time series data as in [3], we could precompute T and store it in a suitable index structure for fast retrieval with respect to time.

4 Experimental Results

We benchmarked a simplified prototypical implementation of our construction algorithm (e.g., no precomputation for the first row of the matrix) implemented in Java. Points were sampled uniformly at random from the unit square. The results, averaged over 20 runs, can be found in Table 1. For example, for 2^{19} points, our implementation of the incremental algorithm from [7] took about 9 seconds to construct roughly 3 million triangles during the construction of $T_{1,n}$; our algorithm for constructing all Delaunay triangles over all contiguous subsequences took about 155 seconds to construct the roughly 63 million triangles of T. The table indicates that our algorithm is slightly faster per triangle than the randomized incremental algorithm. Looking at the ratio between $|T|$ and the construction time for T, we observe near-linear behavior. The measurements show that our algorithm works well in practice.

5 Outlook

Delaunay subcomplexes are not only of interest in $\mathbb{R}^2$ but maybe even more so in $\mathbb{R}^3$. So the generalization of both our complexity result as well as the enumeration algorithm are of interest. Another challenge could be to improve the running time of our algorithm, possibly shaving off the logarithmic factor to achieve $O(|T|)$ running time. This does not seem hopeless, as we could exploit the relationship between edges that appear in multiple triangulations.

References

1 Alok Aggarwal, Leonidas J. Guibas, James Saxe, and Peter W. Shor. A linear-time algorithm for computing the voronoi diagram of a convex polygon. *Discrete & Computational Geometry*, 4(6):591–604, December 1989. doi:10.1007/BF02187749.

2 Nina Amenta, Marshall W. Bern, and David Eppstein. The crust and the beta-skeleton: Combinatorial curve reconstruction. *Graphical Models and Image Processing*, 60(2):125–135, 1998. doi:10.1006/gmip.1998.0465.

3 Annika Bonerath, Benjamin Niedermann, and Jan-Henrik Haunert. Retrieving α-shapes and schematic polygonal approximations for sets of points within queried temporal ranges. In *SIGSPATIAL/GIS*, pages 249–258. ACM, 2019. doi:10.1145/3347146.3359087.

4 Herbert Edelsbrunner, David G. Kirkpatrick, and Raimund Seidel. On the shape of a set of points in the plane. *IEEE Trans. Information Theory*, 29(4):551–558, 1983. doi:10.1109/TIT.1983.1056714.

5 Steven Fortune. A sweepline algorithm for voronoi diagrams. *Algorithmica*, 2(1-4):153, 1987. doi:10.1007/BF01840357.

6 Leonidas Guibas and Jorge Stolfi. Primitives for the manipulation of general subdivisions and the computation of voronoi diagrams. *ACM Transactions On Graphics (TOG)*, 4(2):74–123, 1985. doi:10.1145/282918.282923.

7 Leonidas J. Guibas, Donald E. Knuth, and Micha Sharir. Randomized incremental construction of delaunay and voronoi diagrams. *Algorithmica*, 7(1):381–413, 1992. doi:10.1007/BF01758770.

8 Scott Huddleston and Kurt Mehlhorn. A new data structure for representing sorted lists. *Acta Informatica*, 17:157–184, 1982. doi:10.1007/BF00288968.

9 Haim Kaplan, Edgar Ramos, and Micha Sharir. The overlay of minimization diagrams in a randomized incremental construction. *Discrete & Computational Geometry*, 45(3):371–382, 2011. doi:10.1007/s00454-010-9324-6.

10 David G. Kirkpatrick and John D. Radke. A framework for computational morphology. In Godfried Toussaint, editor, *Computational Geometry*, volume 2 of *Machine Intelligence and Pattern Recognition*, pages 217–248. North-Holland, 1985. doi:10.1016/B978-0-444-87806-9.50013-X.

11 Cao An Wang and Francis Chin. Finding the constrained delaunay triangulation and constrained voronoi diagram of a simple polygon in linear-time. In Paul Spirakis, editor, *Algorithms – ESA '95*, pages 280–294. Springer, 1995. doi:10.1007/3-540-60313-1_150.