Delaunay Triangulations with Predictions

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

Timothy M. Chan ✉ 📷
Siebel School of Computing and Data Science, University of Illinois at Urbana-Champaign, USA

Panos Giannopoulos ✉ 📷
Department of Computer Science, City St George's, University of London, UK

── **Abstract** ──────────────────

We investigate *algorithms with predictions* in computational geometry, specifically focusing on the basic problem of computing 2D Delaunay triangulations. Given a set P of n points in the plane and a triangulation G that serves as a "prediction" of the Delaunay triangulation, we would like to use G to compute the correct Delaunay triangulation $DT(P)$ more quickly when G is "close" to $DT(P)$. We obtain a variety of results of this type, under different deterministic and probabilistic settings, including the following:

1. Define D to be the number of edges in G that are not in $DT(P)$. We present a deterministic algorithm to compute $DT(P)$ from G in $O(n + D\log^3 n)$ time, and a randomized algorithm in $O(n + D\log n)$ expected time, the latter of which is optimal in terms of D.

2. Let R be a random subset of the edges of $DT(P)$, where each edge is chosen independently with probability ρ. Suppose G is any triangulation of P that contains R. We present an algorithm to compute $DT(P)$ from G in $O(n\log\log n + n\log(1/\rho))$ time with high probability.

3. Define d_{vio} to be the maximum number of points of P strictly inside the circumcircle of a triangle in G (the number is 0 if G is equal to $DT(P)$). We present a deterministic algorithm to compute $DT(P)$ from G in $O(n\log^* n + n\log d_{\mathrm{vio}})$ time.

We also obtain results in similar settings for related problems such as 2D Euclidean minimum spanning trees, and hope that our work will open up a fruitful line of future research.

1   Introduction

1.1   Towards computational geometry with predictions

Algorithms with predictions have received considerable attention in recent years. Given a problem instance and a prediction for the solution, we would like to devise an algorithm that solves the problem in such a way that the algorithm performs "better" when the prediction is "close" to the correct solution. "Better" may be in terms of running time, or approximation factor (for optimization problems), or competitive ratio (for on-line problems). Different measures of "closeness" may be used depending on the problem, or alternatively we may assume that predictions obey some probabilistic models. The paradigm has been applied to problems in diverse areas, such as scheduling, classical graph problems, resource allocation, mechanism design, facility location, and graph searching to name a few; see the survey in [52] and the regularly updated online list of papers in [1].

In contrast, work on algorithms with predictions in *computational geometry* have been few and far between. The goal of this paper is to promote this direction of research. Primarily, we concentrate on the use of predictions to improve the running time of geometric algorithms.

As many fundamental problems in classical computational geometry may be viewed as two- or higher-dimensional generalizations of standard one-dimensional problems like sorting and searching, we start our discussion with the 1D sorting problem. Here, algorithms with predictions are related to the historically well-explored topic of *adaptive sorting* algorithms [32]—if we are given a sequence that is already close to sorted, we can sort faster. There are different ways to measure distance to sortedness. For example:

- Consider D_{inv}, the number of inversions in the given sequence (i.e., the number of out-of-order pairs). In terms of this parameter, good old insertion-sort runs in $O(n + D_{inv})$ time and already beats $O(n \log n)$ algorithms when $D_{inv} = O(n)$. Better algorithms are known running in $\Theta(n \log(1 + D_{inv}/n))$ time, which is optimal in comparison-based models.
- Or consider D_{rem}, the minimum number of elements to remove (the "wrong" elements) so that the remaining list is sorted. Here, the optimal time bound is known to be $\Theta(n + D_{rem} \log n)$.
- Or consider D_{runs}, the number of runs (i.e., the number of increasing contiguous subsequence that the input can be partitioned into). Here, the optimal time bound is known to be $\Theta(n \log D_{runs})$. Unfortunately, even when D_{runs} is as low as $n^{0.1}$, it is not possible to beat $n \log n$ under this distance measure.

The survey in [32] gives an extensive background and a (long) list of other parameters that have been considered in the algorithms literature. Adaptive sorting has continued to be studied (some random recent examples include [7, 53, 4]), and some recent papers [6, 47] examine newer models for sorting with predictions.

1.2   The 2D Delaunay triangulation problem

Can we obtain similar results for geometric problems in 2D? In this paper, we pick arguably one of the most basic problems in computational geometry to start this study: namely, the computation of the *Delaunay triangulation*, denoted $DT(P)$, of a set P of n points in the plane [27]. One way to define $DT(P)$ is via the *empty circle property*: three points $p, p', p'' \in P$ form a Delaunay triangle iff the circumcircle through p, p', p'' does not have any points of P strictly inside. (Throughout this paper, we assume for simplicity that the input is in general position, e.g., no 4 input points are co-circular.)
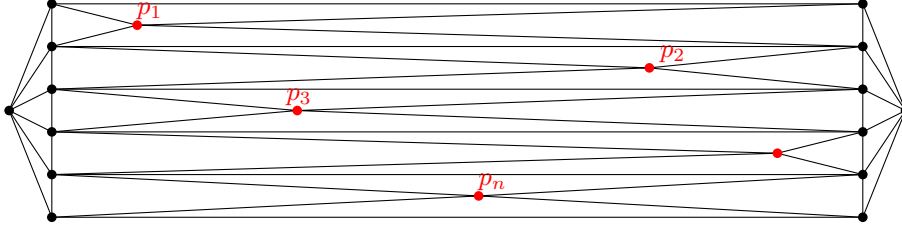
Figure 1 Sorting the x-coordinates of $p_1, \ldots, p_n$ reduces to computing the Delaunay triangulation of the $O(n)$ vertices of the triangulation shown (assuming a rescaling to compress the y-coordinates).

Delaunay triangulations are combinatorially equivalent to *Voronoi diagrams* (under the Euclidean metric) by duality. (It is a little more convenient to work with Delaunay triangulations than Voronoi diagrams when specifying the prediction G or defining closeness, and so all our results will be stated in terms of the former.) Delaunay triangulations and Voronoi diagrams are among the most ubiquitous geometric structures, with countless applications, and lie at the very core of computational geometry.

In the prediction setting, we are given not just the input point set P, but also a triangulation G of P (i.e., a decomposition of the convex hull of P into nonoverlapping triangles, using all the points of P as vertices, but no Steiner points). The goal is to devise an algorithm to compute $DT(P)$ faster, when the "distance" between G and $DT(P)$ is small. In the traditional setting, several standard textbook algorithms are known for computing the Delaunay triangulation in $O(n \log n)$ time, e.g., via divide-and-conquer, plane sweep, and randomized incremental construction [8, 27, 5, 34], which is optimal in the worst case. Without closeness, an arbitrary triangulation G of P still provides some extra information about the input, but not enough to beat $n \log n$: it is not difficult to see that $\Omega(n \log n)$ remains a lower bound by reduction from 1D sorting (see Figure 1). We want to do better than $n \log n$ when G is close. As we will see, designing faster adaptive algorithms for 2D Delaunay triangulations is technically more difficult, and so more interesting, than for 1D sorting.

1.3 Two specific questions

There are a number of natural parameters to measure distance to the Delaunay triangulation, as we define in Table 1. Some of these definitions (e.g., $D, D_{\mathrm{flip}}, D_{\mathrm{cross}}$) may be used to compare any two triangulations, while others (e.g., $D_{\mathrm{vio}}, D_{\mathrm{local}}$) are specific to the Delaunay triangulation. For all these parameters, the value is 0 if and only if $G = DT(P)$. Some relationships between these parameters are obvious, e.g., $D_{\mathrm{cross}} \leq O(nd_{\mathrm{cross}})$ and $D_{\mathrm{vio}} \leq O(nd_{\mathrm{vio}})$, and it is known that

$$D_{\mathrm{local}} \leq D \leq D_{\mathrm{flip}} \leq D_{\mathrm{cross}}.$$

The inequality $D_{\mathrm{local}} \leq D$ holds because all non-locally Delaunay edges can't be actual Delaunay edges, while the last inequality $D_{\mathrm{flip}} \leq D_{\mathrm{cross}}$ is nontrivial and was proved by Hanke et al. [36].

There are examples when D_{local} is a constant and D is $\Omega(n)$. In a very vague sense, D_{local} is more similar to D_{runs} (the number of runs) for sorting, while D is more similar to D_{rem} and D_{flip} and D_{cross} are more similar to D_{inv}. Indeed, it is not difficult to prove an $\Omega(n \log D_{\mathrm{local}})$ lower bound for computing the Delaunay triangulation in terms of D_{local}, by reduction from sorting with $\Theta(D_{\mathrm{local}})$ runs, or equivalently, merging $\Theta(D_{\mathrm{local}})$ sorted lists

parameter	definition
D	number of edges in G that are not in $DT(P)$ (or equivalently, number of edges in $DT(P)$ that are not in G)
D_{flip}	minimum number of *edge-flips* to transform G into $DT(P)$ (in a flip, an edge pp' incident to triangles $\triangle pp'p''$ and $\triangle pp'p'''$ is replaced by a new edge $p''p'''$, assuming that $pp''p'p'''$ is a convex quadrilateral)
D_{cross}	number of edge crossings between G and $DT(P)$
d_{cross}	maximum number of edges of $DT(P)$ crossed by an edge of G
D_{vio}	number of *violations*, i.e., $(p''', \triangle pp'p'')$ with $p''' \in P$ and $\triangle pp'p''$ in $DT(P)$ such that p''' lies strictly inside the circumcircle through p, p', p''
d_{vio}	maximum number of points of P strictly inside the circumcircle through a triangle in G
D_{local}	number of *non-locally-Delaunay* edges in G (i.e., edges pp' incident to triangles $\triangle pp'p''$ and $\triangle pp'p'''$ in G such that p''' is inside the circumcircle through p, p', p'')

Table 1 Different measures of closeness between a given triangulation G and the Delaunay triangulation $DT(P)$.
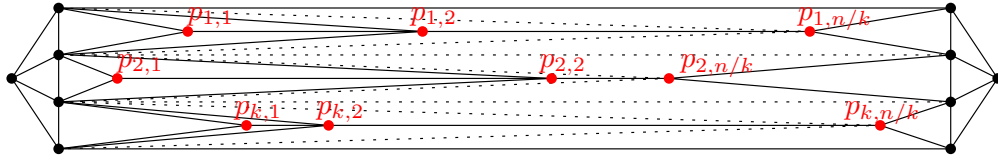
Figure 2 Merging k x-sorted lists $\langle p_{1,1}, \ldots, p_{1,n/k} \rangle, \ldots, \langle p_{k,1}, \ldots, p_{k,n/k} \rangle$ reduces to computing the Delaunay triangulation of the $O(n)$ vertices of the triangulation shown (assuming a rescaling to compress the y-coordinates). This triangulation has $D_{\text{local}} = \Theta(k)$ non-locally-Delaunay edges shown in dotted lines.

(see Figure 2). So, D_{local} may not be an ideal parameter for the problem (even when D_{local} is as small as $n^{0.1}$, we can't hope to beat $n \log n$).

Arguably the simplest and most natural of all these parameters is D, the number of "wrong" edges in G that are not in $DT(P)$ (which is the same as the number of edges in $DT(P)$ that are in G, since all triangulations of the same point set have the same number of edges). To initiate the study of algorithms with predictions for the Delaunay triangulation, we thus begin with the following simple question:

Question 1: what is the best 2D Delaunay triangulation algorithm with running time analyzed as a function of n and the parameter D?

This simple question already turns out to be quite challenging. An upper bound on time complexity in terms of D would automatically imply an upper bound in terms of D_{flip} and D_{cross} by the above inequalities. We can ask a similar question for other parameters such as d_{cross} and d_{vio}.

Although Question 1 provides one way to model predictions, often errors are probabilistic in nature. Thus, for various problems, algorithms with predictions have sometimes been studied under probabilistic models. To define a simple probabilistic model of prediction for

the Delaunay triangulation, one could just consider tossing an independent coin to determine whether an edge of $DT(P)$ should be placed in G (similar to a model studied by Cohen-Addad et al. [26] for the max-cut problem). But there is one subtlety for our problem: the input G is supposed to be a triangulation, containing wrong edges (but not containing self-crossings). We propose the following 2-stage model: first, each edge of $DT(P)$ is placed in a set R independently with a given probability ρ; second, R is completed to a triangulation of P, but we make no assumption about how this second step is done, i.e., R can be triangulated *adversarially* (this makes results in this model a bit more general).

Question 2: under the above probabilistic model, what is the best 2D Delaunay triangulation algorithm with running time analyzed as a function of n and ρ?

In this model, the expected number of "wrong" edges D is upper-bounded by $(1-\rho)m$, where $m \leq 3n-6$ is the number of edges in any triangulation of P. The hope is that the problem may be solved faster under the probabilistic model than by applying a worst-case algorithm in terms of the parameter D.

We could consider other probabilistic models too. For example, one variant is to pick a random edge in $DT(P)$, perform a flip if its two incident triangles form a convex quadrilateral (or do nothing otherwise), and repeat for $(1-\rho)m$ steps. In this model, the flip distance D_{flip} is guaranteed to be at most $(1-\rho)m$; in contrast, the above model allows for potentially larger flip distances and so is in some sense stronger.

1.4 Our contributions

We obtain a number of results addressing Questions 1 and 2 and related questions:

1. We present several different algorithms for computing $DT(P)$ from G in time sensitive to the parameter D. One deterministic algorithm runs in $O(n + D\log^3 n)$ time. A randomized algorithm runs in $O((n + D\log n)\log^* n)$ time, where $\log^*$ is the (very slow-growing) iterated logarithmic function. Our final randomized algorithm eliminates the $\log^* n$ factor and runs in $O(n + D\log n)$ expected time, which is optimal since an $\Omega(\max\{n, D\log D\}) = \Omega(n + D\log n)$ lower bound follows by reduction from sorting D numbers (we can just embed the configuration in Figure 1 on $\Theta(D)$ points into a larger Delaunay triangulation). Thus, we have completely answered Question 1 (modulo the use of randomization). All of our algorithms do not require knowing the value of D a priori. It is interesting to note that our optimal randomized algorithm runs in *linear* time even when the number D of "wrong" edges is as large as $n/\log n$.
   An immediate consequence is that the problem can be solved in $O(n + D_{\text{flip}}\log n)$ expected time—we get linear running time as long as the flip distance is at most $n/\log n$.

2. On the probabilistic model in Question 2, we obtain another algorithm for computing $DT(P)$ from G in $O(n \log\log n + n\log(1/\rho))$ time with high probability (the algorithm is always correct). For any constant ρ, the running time is thus almost linear except for the small $\log\log n$ factor. This is notable when compared against our $O(n + D\log n)$ algorithm, since in this probabilistic setting, D is expected to be near $(1-\rho)m$. For example, if we set $\rho = 0.1$ (or even $\rho = 1/\log n$), we can still substantially beat $n\log n$ even though a majority of the edges are wrong!
   We can also get a similar result for the random flip model mentioned earlier.

3. We also obtain results sensitive to other parameters. For example, in terms of d_{vio}, we describe a deterministic algorithm for computing $DT(P)$ from G in $O(n\log^* n + n\log d_{\text{vio}})$ time. Note that there are examples where $d_{\text{vio}} = 1$ but $D = \Omega(n)$ (i.e., a fraction of the

edges are wrong!), so our D-sensitive algorithms can't be used directly. The parameter d_{vio} is related to a notion called *order-k Delaunay triangulation*[1] in the literature [35] (with motivation very different from ours): a triangulation G is an "order-k Delaunay triangulation" precisely when $d_{\mathrm{vio}} \leq k$. Our result implies the following interesting consequence: given an order-$O(1)$ Delaunay triangulation, we can recover the actual Delaunay triangulation in almost linear ($O(n \log^* n)$) time!

4. We similarly describe an $O(n \log^* n + n \log d_{\mathrm{cross}})$-time deterministic algorithm for computing $DT(P)$ from G. Again, there are examples where $d_{\mathrm{cross}} = 1$ but $D = \Omega(n)$. Although the parameter d_{cross} may not appear to be the most natural at first, it turns out to be the key that allows us to obtain items 2 and 3 above in an elegant, unified way: it is not difficult to show that $d_{\mathrm{cross}} = O((1/\rho) \log n)$ with high probability in the probabilistic model, and we are also able to show that $d_{\mathrm{cross}} \leq O(d_{\mathrm{vio}}^2)$ always holds.

5. Although we have focused on the Delaunay triangulation problem, we can obtain similar results for a few related problems. For example, consider the problem of computing the Euclidean minimum spanning tree $EMST(P)$ of a set P of n points in 2D. Given a spanning tree T, we can compute $EMST(P)$ from T in $O((n+D_{\mathrm{emst}} \log n) \log^* n)$ expected time, where D_{emst} is the number of "wrong" edges in T that are not in $EMST(P)$. In fact, this result, and the idea of using a spanning tree in the first place, will be instrumental to the eventual design of our $O(n + D \log n)$-time randomized algorithm for Delaunay triangulations.

   We get similar results for EMST in the probabilistic model or in terms of other parameters, and also for computing other related geometric graphs such as the relative neighborhood graph and the Gabriel graph in the plane [38]. We also get similar results for the problem of computing the convex hull of n points in 3D that are in convex position (by a standard lifting map, Delaunay triangulations in 2D maps to convex hulls in 3D)—the convex position assumption is necessary since one could show an $\Omega(n \log n)$ lower bound for checking whether $n/2$ points are all inside the convex hull of another $n/2$ points, even when we are given the correct convex hull.

1.5   Related work in computational geometry

We are not aware of any prior work in computational geometry that explicitly addresses the paradigm of algorithms with predictions, except for a recent paper by Cabello and Giannopoulos [10] on an on-line geometric search problem (where predictions are used to obtain competitive ratio bounds). Implicitly related are *adaptive* data structures, and these have been studied in computational geometry; as one example, see Iacono and Langerman's work [37] on "proximate" planar point location (where a query can be answered faster when given a finger near the answer).

   Our work is related to, and may be regarded as a generalizaton of, work on the *verification* of geometric structures: how to check whether a given answer is indeed correct, using a simpler or faster algorithm than an algorithm for computing the answer. This line of work has received much attention from computational geometers in the 1990s [29, 50, 51], motivated by algorithm engineering. For the Delaunay triangulation problem, it is easy to verify whether G is equal to $DT(G)$ in linear time, by checking that all edges are locally Delaunay. Verifying

---

[1] The name is somewhat misleading as it is not equivalent to the *order-k Voronoi diagram* in the dual. The set of all candidate triangles for order-k Delaunay triangulations does correspond to the vertices of the order-k Voronoi diagram, but an order-k Delaunay triangulation is not unique, comprising of a subset of the candidate triangles.

correctness corresponds to the case $D = 0$, and our results provides more powerful extensions: we can correct errors (as many as $n/\log n$ wrong edges) in linear time. Viewing our results in the lens of *correcting errors in the output* provides further motivation, even for readers not necessarily interested in learning-augmented algorithms.

Some recent work such as [31] studies geometric algorithms in models that tolerate errors in the computation process itself (e.g., noisy or faulty comparisons), but our work seems orthogonal to this research direction. On the other hand, there has been considerable prior work in computational geometry on models that capture *errors or imprecisions in the input*. For example, in one result by Löffler and Snoeyink [46], an imprecise input point is modelled as a range, specifically, a unit disk (whose center represents its estimated position); it is shown that n disjoint unit disks in the plane can be preprocessed in $O(n \log n)$ time, so that given n actual points lying in these n disks, their Delaunay triangulation can be computed in $O(n)$ time. Similar results have been obtained for different types of regions and different problems. Our model provides an alternative way to address imprecise input: in the preprocessing phase, we can just compute the Delaunay triangulation G of the estimated points; when the actual points P are revealed, we can correct G to obtain $DT(P)$ using our algorithms.[2] An advantage of our model is its simplicity: we don't need to be given any geometric range or distribution for each input point; rather, accuracy is encapsulated into a single parameter D.

Even without relevance to algorithms with paradigms or coping with errors in the input/output, Question 1 is appealing purely as a classical computational geometry problem. In the literature, a number of papers have explored conditions under which Delaunay triangulations can be computed in linear or almost linear time. For example, Devillers [28] showed that given the Euclidean minimum spanning tree of a 2D point set, one can compute the Delaunay triangulation in $O(n \log^* n)$ expected time (this was later improved to deterministic linear time [22]); two Delaunay triangulations $DT(P_1)$ and $DT(P_2)$ can be merged into $DT(P_1 \cup P_2)$ in linear time [40]; conversely, a Delaunay triangulation $DT(P_1 \cup P_2)$ can be split into $DT(P_1)$ and $DT(P_2)$ in linear time (by Chazelle et al. [15]); the Delaunay triangulation of the vertices of a convex polygon can be computed in linear time (by Aggarwal et al. [2, 21]); the Delaunay triangulation for 2D points with integer coordinates can be computed in the same time complexity as sorting in the word RAM model (by Buchin and Mulzer [9]); etc. A powerful work by Löffler and Mulzer [45] unifies many of these results by showing *equivalence* of 2D Delaunay triangulations with quadtree construction and related problems. Although some of these tools will be useful later, it isn't obvious how they can used to resolve Questions 1 and 2.

1.6 Our techniques

Although we view our main contributions to be conceptual (introducing a new research direction in computational geometry), our algorithms are also technically interesting in our opinion. The solutions turn out to be a lot more challenging than 1D adaptive sorting algorithms or the standard textbook Delaunay triangulation algorithms:

- Our deterministic $O(n + D \log^3 n)$-time algorithm (given in Section 2) uses planar graph separators or r-divisions in the given graph G (for classical computational geometry problems, Clarkson–Shor-style sampling-based divide-and-conquer [23, 24, 19] is more commonly used than planar graph separators).

---

[2] One technical issue is that G may have self-crossings when re-embedded into the actual point set P, but at least one of our algorithms works even when G has self-crossings (see Corollary 6).

- Our randomized $O((n + D \log n) \log^* n)$-time algorithm (given in Section 3) is obtained via an adaptation of the abovementioned randomized $O(n \log^* n)$-time algorithm for computing the Delaunay triangulation from the EMST, by Devillers [28] (which in turn was based on the randomized $O(n \log^* n)$-time algorithms by Seidel [54] and Clarkson, Tarjan, and Van Wyck [25] for the polygon triangulation problem). The adaptation is more or less straightforward, but our contribution is in realizing that these randomized $O(n \log^* n)$-time algorithms are relevant in the first place.
- Our final randomized $O(n + D \log n)$-time algorithm (given in Section 4) eliminates the extra $\log^* n$ factor by incorporating a more complicated divide-and-conquer involving planar graph separators or r-divisions, this time applied to actual Delaunay triangulations, not the input graph G. (Curiously, ideas from certain output-sensitive convex hull algorithms [30, 12] turn out to be useful.)
- For our deterministic $O(n \log^* n + n \log d_{\mathrm{cross}})$-time algorithm (given in Section 5), which as mentioned is the key to our result in the probabilistic model as well as our d_{vio}-sensitive result, we return to using r-divisions in the given graph G. The way the $\log^* n$ factor comes up is interesting—it arises not from the use of known randomized $O(n \log^* n)$ algorithms like Seidel's, but from an original recursion. Although our D-sensitive algorithm does not directly yield a d_{cross}-sensitive result, it will turn out to be an important ingredient in the final algorithm.

1.7 Open questions

Many interesting questions remain from our work (which we see as a positive, as our goal is to inspire further work in this research direction):

- Could the extra $\log^* n$ factors be eliminated in some of our results?
- Could the extra $\log \log n$ factor in our result in the probabilistic model be removed?
- Could we get a time bound in terms of D_{flip} better than $O(n + D_{\mathrm{flip}} \log n)$? For example, is there a Delaunay triangulation algorithm that runs in $O(n + D_{\mathrm{flip}})$ time?
- Similarly, is there a Delaunay triangulation algorithm that runs in $O(n + D_{\mathrm{cross}})$ time? Or, better still, $O(n \log(1 + D_{\mathrm{cross}}/n))$ time? Or $O(n + D_{\mathrm{vio}})$ or $O(n \log(1 + D_{\mathrm{vio}}/n))$ time?

More generally, we hope that our work will prompt other models of prediction to be studied for the Delaunay triangulation problem, or similar models for other fundamental problems in computational geometry and beyond.

2 A deterministic $O(n + D \log^3 n)$ algorithm

We begin by presenting a deterministic algorithm for computing the Delaunay triangulation in $O(n + D \log^3 n)$ time. The algorithm uses planar graph separators, or more generally, r-divisions, on the input triangulation G. (A different deterministic algorithm is given in the full version of the paper; it is slightly slower but does not require separators.)

An *r-division* of a planar graph G with n vertices is a partition of G into $O(n/r)$ regions (edge-induced subgraphs) such that each region has at most $O(r)$ vertices and at most $O(\sqrt{r})$ boundary vertices, i.e., vertices shared by more than one region. An r-division can be computed in $O(n)$ time; see, for example, the classical result by Federickson [33] or the more recent, improved result by Klein, Mozes, and Sommer [42]. Note that the regions may have holes (i.e., faces that are not faces of G). We may also assume that each region is connected.

▶ **Theorem 1.** *Given a triangulation G of a set P of n points in the plane, the Delaunay triangulation $DT(P)$ can be constructed in $O(n + D \log^3 n)$ deterministic time, where D is the (unknown) number of edges in G that are not in $DT(P)$.*

Proof. Our algorithm proceeds in the following steps:

1. First we compute, in $O(n)$ time, a t-division Γ of the dual graph of the input triangulation G, for some t to be chosen later. Each region γ of Γ is a connected set of $O(t)$ triangles. Let V_γ denote the set of vertices of all the triangles in γ and B_γ denote the set of all the triangles surrounding γ (here, it is more convenient to work with boundary triangles instead of boundary edges). Then $|V_\gamma| = O(t)$ and $|B_\gamma| = O(\sqrt{t})$. Also, let B be the set of all boundary triangles in Γ and V_B be the set of the vertices of the edges in B. Then $|V_B| = O(n/\sqrt{t})$.
2. We compute the Delaunay triangulation $DT(V_B)$ of V_B, in $O(|V_B| \log |V_B|) \leq O((n/\sqrt{t}) \log n)$ time.
3. We call a region γ of Γ *good* if (i) all boundary triangles in B_γ are in $DT(V_B)$, and (ii) the triangles of γ and their immediate neighborhood are locally Delaunay (i.e., for every neighboring pair of triangles $\triangle p_1 p_2 p_3$ and $\triangle p_1 p_2 p_4$ in G with at least one inside γ (the other may be inside or be a boundary triangle), we have p_4 outside the circumcircle through p_1, p_2, p_3); we call it *bad* otherwise. Let

$$V_{good} = \left(\bigcup_{\gamma \text{ is good}} V_\gamma \right) \setminus V_B \quad \text{and} \quad V_{bad} = \left(\bigcup_{\gamma \text{ is bad}} V_\gamma \right) \setminus V_B.$$

   Note that condition (i) can be tested using standard data structures, while condition (ii) can be tested in linear total time.
4. We compute $DT(V_B \cup V_{good})$. This can be done by modifying $DT(V_B)$. Namely, for each good region γ, we replace what is inside B_γ (by definition, B_γ belongs to $DT(V_B)$), with the triangles of γ. Then the resulting triangulation is locally Delaunay, and thus must coincide with $DT(V_B \cup V_{good})$ (since the local Delaunay property holding for the entire triangulation implies that it is the Delaunay triangulation [27]). This step takes linear time.
5. We compute $DT(V_{bad})$. We will do this step from scratch, in $O(|V_{bad}| \log |V_{bad}|)$ time. For each bad region γ, there is at least one non-Delaunay edge of G inside γ or on its boundary triangles. So, the number of bad regions is at most $O(D)$, implying that $|V_{bad}| \leq O(Dt)$. So, this step takes $O(Dt \log n)$ time.
6. Finally, we merge $DT(V_B \cup V_{good})$ and $DT(V_{bad})$, in linear time by standard algorithms for merging Delaunay triangulations (Chazelle [14], and later Chan [11], showed more generally that two convex hulls in $\mathbb{R}^3$ can be merged in linear time, but an earlier result by Kirkpatrick [40] suffices for the case of Delaunay triangulations in $\mathbb{R}^2$).

The total running time is $O(n + (n/\sqrt{t} + Dt) \log n)$. Choosing $t = \log^2 n$ gives the desired $O(n + D \log^3 n)$ bound. ◀

3 A randomized $O((n + D \log n) \log^* n)$ algorithm

In this section, we present a completely different, and faster, randomized algorithm for computing the Delaunay triangulation in terms of the parameter D, which is just a $\log^* n$ factor from optimal. The algorithm is an adaptation of an algorithm by Devillers [28], which uses the Euclidean minimum spanning tree to construct the Delaunay triangulation in almost

linear ($O(n \log^* n)$) expected time. Slightly more generally, the Delaunay triangulation can be constructed from any given spanning tree of $DT(P)$ of maximum degree $O(1)$ in $O(n \log^* n)$ expected time. We generalize his result further to show that the same can be achieved even if the given spanning tree has some wrong edges, and even if the maximum degree requirement is dropped.

Devillers' algorithm is similar to the more well-known $O(n \log^* n)$ randomized algorithms for polygon triangulation by Seidel [54] and Clarkson, Tarjan, and Van Wyk [25], which are variants of standard $O(n \log n)$ randomized incremental or random sampling algorithms [23, 24, 19], but with the point location steps sped up. We will describe our algorithm based on random sampling.

In this and subsequent sections, we will sometimes work with the more general notion of a *plane straight-line graph*, shortened to *PSLG*, i.e., a planar embedding of a graph where all edges are drawn as straight-line edges. Thus, a triangulation of a planar point set P is a connected PSLG such that: (i) its vertex set is precisely P, (ii) all its bounded faces are triangles, and (iii) the unbounded face is the complement of the convex hull $CH(P)$.

It will also be more convenient to work in 3D, where 2D Delaunay triangulations and Voronoi diagrams are mapped to 3D convex hulls and lower envelopes of planes. We begin with some standard terminologies:

For a set of planes Π in $\mathbb{R}^3$, let $LE(\Pi)$ denote the ***lower envelope of*** Π, that is, the set of points lying on some plane of Π but with no plane of Π below it. The ***vertical decomposition of*** $LE(\Pi)$, denoted $Decomp(\Pi)$, is a partition of the space below $LE(\Pi)$ into vertical, truncated prisms that are unbounded from below and are bounded from above by a triangular face contained in $LE(\Pi)$. The triangles bounding the prisms have vertices on $LE(\Pi)$ and are defined by some canonical rule, such as triangulating each face of $LE(\Pi)$ with diagonals from the point in the face with the smallest x-coordinate. (Since the faces are convex, such a triangulation is possible.) We also have to "triangulate" the infinite faces of $LE(\Pi)$, that is, to split them into regions defined by three edges. We will refer to these prisms as ***cells***.

For each point $p = (\alpha, \beta) \in \mathbb{R}^2$, let p^* denote the plane in $\mathbb{R}^3$ defined by $z = -2\alpha x - 2\beta y + \alpha^2 + \beta^2$. The plane p^* is tangent to the paraboloid $z = -x^2 - y^2$ at the point p. For any set P of points, define $P^* = \{p^* \mid p \in P\}$.

For a set P of points in the plane, the xy-projection of the lower envelope of P^* is the Voronoi diagram of P (dual to the Delaunay triangulation $DT(P)$). The vertices and edges of $LE(P^*)$ correspond to the vertices and edges of the Voronoi diagram of P in the plane. Since we assume general position, each vertex of $LE(P^*)$ is contained in three planes of P^*. The xy-projection of a cell in $Decomp(P^*)$ is a triangle contained in a Voronoi cell and defined by three Voronoi vertices.

For a vertex $v = p_1^* \cap p_2^* \cap p_3^* \in \mathbb{R}^3$, the ***conflict list of*** v, denoted P_v^*, refers to the list of all planes in P^* strictly below v. Observe that p^* is in the conflict list P_v^* iff p is strictly inside the circumcircle through p_1, p_2, p_3.

▶ **Lemma 2.** *Let p_1, p_2, p_3 be the vertices of the triangle $\triangle$, and let p, q be points such that pq intersects the interior of the triangle $\triangle$. If pq is an edge of $DT(\{p_1, p_2, p_3, p, q\})$, then p^* or q^* must be in the conflict list of $p_1^* \cap p_2^* \cap p_3^*$.*

Proof. See Figure 3. If p and q are not inside the circumcircle of $\triangle$, then the edges of $\triangle$ are edges of $DT(\{p_1, p_2, p_3, p, q\})$. Since the endpoints of pq are not inside $\triangle$, pq crosses some edge of $\triangle$. This would contradict the planarity of $DT(\{p_1, p_2, p_3, p, q\})$.   ◀

For a cell $\tau \subset \mathbb{R}^3$, the ***conflict list of*** τ, denoted P_τ^*, refers to the list of all planes in
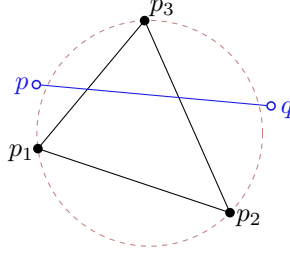
Figure 3 Proof of Lemma 2.

P^* intersecting τ. Observe that the conflict list of τ is just the union of the conflict lists of the three vertices of τ.

Before describing our main algorithm, we first present the following lemma, which uses a given spanning tree T to speed up conflict list computations. To deal with the scenario when the maximum degree of T is not bounded, we will work with weighted samples, or equivalently, samples of multisets.

▶ **Lemma 3.** *Let P be a set of n points, and T be a spanning tree of P, where the edges around each vertex have been angularly sorted. Let $\widehat{P}$ be the multiset where each point $p \in P$ has multiplicity $deg_T(p)$. Let $\widehat{R}$ be a random (multi)subset of $\widehat{P}$ of a given size r, and let R be the set $\widehat{R}$ with duplicates removed. Suppose $LE(R^*)$ is given. Then we can compute the conflict lists of all the cells in $Decomp(R^*)$ in $O(n + D_T \log n)$ expected time, where D_T is the number of edges of T that are not in $DT(P)$.*

Proof. We preprocess $DT(R)$ for point location [41]. The preprocessing takes $O(r)$ time and a point location query can be answered in $O(\log r) \leq O(\log n)$ time.

For each point $p \in P$, let γ_p be the number of vertices of $Decomp(R^*)$ whose conflict lists contain p^*.

We first describe how to compute, for each point $p \in P$, the triangle $\triangle_p \in DT(R)$ containing p. (Technically, we should add 3 vertices of a sufficiently large bounding triangle to both R and P, to ensure that $\triangle_p$ always exists, and to deal with vertices of $Decomp(R^*)$ "at infinity".)

To this end, we traverse the edges of the spanning tree T. For each edge pq of T, knowing $\triangle_p$ we can get to $\triangle_q$ by performing a linear walk in $DT(R)$. However, if we cross more than $\log n$ triangles of $DT(R)$, we switch to performing a point location query for q. This takes $O(\min\{c_{pq}, \log n\})$ time, where c_{pq} is the number of triangles of $DT(R)$ crossed by pq. Iterating over all edges in T, we get $\triangle_p$ for all $p \in P$. The total time is

$$O\left(\sum_{pq \in T} \min\{c_{pq}, \log n\}\right). \tag{1}$$

If pq is an edge of $DT(P)$, then Lemma 2 implies that c_{pq} is bounded by $\gamma_p + \gamma_q$. Therefore

$$\sum_{pq \in T \cap DT(P)} c_{pq} \;\leq\; \sum_{p \in P} (\gamma_p \cdot \deg_T(p)),$$

which is proportional to the total conflict list size, with multiplicities included. By Clarkson and Shor's analysis [24], the expected total conflict list size, with multiplicities included, is $O(r \cdot |\widehat{P}|/r) = O(n)$, where we are using the fact that $|\widehat{P}| = O(\sum_p \deg_T(p)) = O(n)$.

On the other hand,

$$\sum_{pq \in T \setminus DT(P)} \min\{c_{pq}, \log n\} \ \leq \ O(D_T \log n).$$

Plugging these in Equation (1), we get the claimed expected time bound.

One case not considered above is when $p \in R$. To begin the walk from p to q for a given edge pq in T, we need to find the triangle $\triangle_{pq} \in DT(R)$ incident to p that is crossed by the ray $\overrightarrow{pq}$. To this end, during preprocessing, we can merge the sorted list of the edges of T around p with the sorted list of edges of $DT(R)$ around p, for every $p \in R$; the total cost of merging is linear. Afterwards, each triangle $\triangle_{pq}$ can be identified in constant time.

Now, $\triangle_p$ maps to a vertex of $LE(P^*)$ whose conflict list contains p^*. By a graph search, we can generate all vertices of $LE(P^*)$ whose conflict lists contain p^*, for each $p \in P$. By inversion, we obtain the conflict lists of all vertices of $LE(P^*)$, from which we get the conflict lists of all cells in $Decomp(P^*)$. All this takes additional time linear in the total conflict list size, which is $O(n)$ in expectation as mentioned by Clarkson and Shor's bound. ◄

▶ **Theorem 4.** *Given a spanning tree T of a set P of n points in the plane, the Delaunay triangulation $DT(P)$ can be constructed in $O((n + D_T \log n) \log^* n)$ expected time, where D_T is the number of edges of T that are not in $DT(P)$.*

Proof. Let $\widehat{P}$ be the multiset where each point $p \in P$ has multiplicity $\deg_T(p)$; note that $|\widehat{P}| = O(n)$. Define a sequence of random samples $\widehat{R}_1 \subset \widehat{R}_2 \subset \cdots \subset \widehat{R}_\ell \subset \widehat{P}$, where $\widehat{R}_i$ has size $|\widehat{P}|/s_i$ for a sequence $s_1, s_2, \ldots, s_\ell$ to be chosen later. Let R_i be the set $\widehat{R}_i$ with duplicates removed.

Given $LE(R_i^*)$, we describe how to compute $LE(R_{i+1}^*)$. First, we compute the conflict lists of all cells in $Decomp(R_i^*)$ by Lemma 3 in $O(n + D_T \log n)$ expected time. For each cell $\tau \in Decomp(R_i^*)$, we compute $LE((R_{i+1}^*)_\tau) \cap \tau$ by a standard lower envelope algorithm in $O(|(R_{i+1}^*)_\tau| \log |(R_{i+1}^*)_\tau|)$ time. Finally, we glue all these envelopes to obtain $LE(R_{i+1}^*)$. The running time is

$$O\left(\sum_{\tau \in Decomp(R_i^*)} |(R_{i+1}^*)_\tau| \log |(R_{i+1}^*)_\tau|\right),$$

which by Clarkson and Shor's bound [24] has expected value at most

$$O(|\widehat{R}_i| \cdot (|\widehat{R}_{i+1}|/|\widehat{R}_i|) \log(|\widehat{R}_{i+1}|/|\widehat{R}_i|)) \ = \ O((n/s_{i+1}) \log(s_i/s_{i+1})).$$

The total expected time is upper-bounded by $O(\ell(n + D_T \log n) + \sum_{i=1}^{\ell-1} (n/s_{i+1}) \log s_i)$. Setting $s_1 = n$, $s_{i+1} = \lfloor \log s_i \rfloor$, ..., $s_\ell = 1$, with $\ell = O(\log^* n)$ yields the theorem. ◄

▶ **Corollary 5.** *Given a triangulation G of a set P of n points in the plane, the Delaunay triangulation $DT(P)$ can be constructed in $O((n + D \log n) \log^* n)$ expected time, where D is the (unknown) number of edges in G that are not in $DT(P)$.*

Proof. Let T be any spanning tree of G. We just apply Theorem 4, noting that $D_T \leq D$ (since $T \subseteq G$, and edges in G around each vertex are already sorted). ◄

(Chazelle and Mulzer [17] obtained a similar expected time bound $O(n \log^* n + k \log n)$ for a different problem, on computing the Delaunay triangulation of a subset of n edges with k connected components, given the Delaunay triangulation of a possibly much larger point set.)

Since Theorem 4 holds even if T has self-crossings, our result holds even if the given G is a combinatorial triangulation that may have self-crossings.

▶ **Corollary 6.** *Given a set P of n points in the plane and a combinatorial triangulation G with vertex set P that may have self-crossings, the Delaunay triangulation $DT(P)$ can be constructed in $O((n + D \log n) \log^* n)$ expected time, where D is the (unknown) number of edges in G that are not in $DT(P)$.*

Proof. Let d_p be the number of triangular faces in G incident to p that are not triangles in $DT(P)$. The neighbors of each vertex p in G are almost sorted correctly except possibly for $O(d_p)$ displaced elements; they can be sorted in $O(\deg_G(p) + d_p \log d_p)$ time by known adaptive sorting algorithms [32]. The total sorting time is thus $O(n + \sum_p d_p \log n) = O(n + D \log n)$.

Let T be any spanning tree of G. We can now apply Theorem 4 to T as before. ◀

Incidentally, we also obtain an algorithm for EMST sensitive to the number of wrong edges:

▶ **Corollary 7.** *Given a spanning tree T of a set P of n points in the plane, the Euclidean minimum spanning tree $EMST(P)$ can be constructed in $O((n + D_{\mathrm{emst}} \log n) \log^* n)$ expected time, where D_{emst} is the (unknown) number of edges of T that are not in $EMST(P)$.*

Proof. Since the maximum degree of $EMST(P)$ is at most 6, every vertex p of degree at least 7 in T is adjacent to at least $\deg_T(p) - 6$ edges not in $EMST(P)$. This implies that $\sum_p \max\{\deg_T(p) - 6, 0\} \le O(D_{\mathrm{emst}})$. We can sort the edges of T around each vertex, in total time $O(\sum_p \deg_T(p) \log \deg_T(p)) \le O(n + \sum_p \max\{\deg_T(p) - 6, 0\} \log n) = O(n + D_{\mathrm{emst}} \log n)$. We can now apply Theorem 4, noting that $D_T \le D_{\mathrm{emst}}$ (since $EMST(P) \subseteq DT(P)$).

Having computed $DT(P)$, we can then obtain $EMST(P)$ by computing a minimum spanning tree of $DT(P)$. For this, we can use the linear-time algorithms for planar graphs by Matsui [49] or by Cheriton and Tarjan [20] (the general, linear-time randomized algorithm of Karger, Klein and Tarjan [39] is not necessary). ◀

We can also obtain a similar result for the *relative neighborhood graph* [38]: pq forms an edge iff the intersection of the two disks centered at p and q of radius $\|p - q\|$ does not have any points of P inside. The relative neighborhood graph contains the Euclidean minimum spanning tree and is thus connected.

▶ **Corollary 8.** *Given a set P of n points in the plane and a connected PSLG G with vertex set P, the relative neighborhood graph $RNG(P)$ can be constructed in $O((n + D_{\mathrm{rng}} \log n) \log^* n)$ expected time, where D_{rng} is the (unknown) number of edges of G that are not in $RNG(P)$.*

Proof. Let T be any spanning tree of G. We just apply Theorem 4, noting that $D_T \le D_{\mathrm{rng}}$ (since $T \subseteq G$ and $RNG(G) \subseteq DT(G)$, and edges in G around each vertex are already sorted). Having computed $DT(P)$, we can then obtain $RNG(G)$ in linear time [44]. ◀

A similar result holds for the Gabriel graph or, more generally, the β-skeleton for any $\beta \in [1, 2]$ [38].

4 An optimal randomized $O(n + D \log n)$ algorithm

In this section, we present our optimal randomized algorithm for computing the Delaunay triangulation in terms of the parameter D, eliminating the extra $\log^* n$ factor from the algorithm in the previous section. This is interesting since for related problems such as polygon triangulation, removing the extra $\log^* n$ factor from Clarkson, Tarjan, and Van Wyk's or Seidel's algorithm [25, 54] is difficult to do (cf. Chazelle's linear-time polygon triangulation algorithm [13]).

Our new approach is to stop the iterative algorithm in the proof of Theorem 4 earlier after a constant number of rounds, when the conflict list sizes gets sufficiently small (say, $O(\log \log n)$), and then switch to a different strategy. For this part, we use planar graph separators (t-divisions) on actual Delaunay triangulations (of random subsets).

▶ **Theorem 9.** *Given a triangulation G of a set P of n points in the plane, the Delaunay triangulation $DT(P)$ can be constructed in $O(n + D \log n)$ expected time, where D is the (unknown) number of edges in G that are not in $DT(P)$.*

Proof. Note that at most $O(D)$ of the triangles of G are not in $DT(P)$, or equivalently, at most $O(D)$ of the triangles of $DT(P)$ are not in G (since G and $DT(P)$ have the same number of triangles).

Call a triangle of $DT(P)$ **good** if it is in G, and **bad** otherwise.

Let T be any spanning tree of G. Since T has at most D edges not in $DT(P)$, we can apply the algorithm in Theorem 4. However, we will stop the algorithm after ℓ iterations with a nonconstant choice of s_ℓ instead of $s_\ell = 1$. Once we have computed $LE(R_\ell^*)$, we will switch to a different strategy to finish the computation of $LE(P^*)$, as described below.

First, we construct the conflict lists of $Decomp(R_\ell^*)$ by Lemma 3 in $O(n + D \log n)$ expected time.

Call a cell **exceptional** if it has conflict list size more than bs_ℓ for some parameter b to be set later. We directly construct the lower envelope $LE(P_\tau^*) \cap \tau$ in $O(|P_\tau^*| \log |P_\tau^*|)$ time for all exceptional cells $\tau \in Decomp(R_\ell^*)$. By Chazelle and Friedman's "Exponential Decay Lemma" [16], the expected number of cells with conflict list size more than is_ℓ is $O(2^{-\Omega(i)} \cdot n/s_\ell)$, and thus the expected total cost of this step is at most $O(\sum_{i \geq b} 2^{-\Omega(i)} \cdot (n/s_\ell) \cdot (is_\ell) \log(is_\ell)) = O(2^{-\Omega(b)} \cdot n \log s_\ell)$.

We compute a t-division of the dual planar graph of the xy-projection of $Decomp(R_\ell^*)$ in $O(n)$ time for some parameter t. This yields $O(n/(s_\ell t))$ regions, which we will call **supercells**, each of which consists of $O(t)$ cells in $Decomp(R_\ell^*)$ and has $O(\sqrt{t})$ boundary walls.

We compute $LE(P_\beta^*) \cap \beta$ for each boundary wall β. This costs $O(n/(s_\ell t) \cdot \sqrt{t} \cdot (bs_\ell) \log(bs_\ell))$ time (since exceptional cells have already been taken care of).

The vertices of $LE(P^*)$ at its boundary walls map to chains of edges in $DT(P)$ when returning to $\mathbb{R}^2$. (Some past output-sensitive convex hull algorithms also used these kinds of chains for divide-and-conquer [30, 12, 3], but we will use them in a different way.) These edges divide $DT(P)$ into regions, where each region is associated to one supercell (although one supercell could be associated with multiple regions). We can remove "spikes" (i.e., an edge where both sides bound the same region). For each such region, we check whether all its boundary edges appear in G, and if so, check whether all triangles in G inside the region are locally Delaunay (i.e., for every neighboring pair of triangles $\triangle p_1 p_2 p_3$ and $\triangle p_1 p_2 p_4$ in G inside the region, we have p_4 outside the circumcenter through p_1, p_2, p_3). If the checks are true, then we know that all triangles in G inside the region are correct Delaunay triangles,[3] and will mark the region as such. All these checks take linear total time.

For each supercell γ, if its corresponding regions are marked correct, then we have all the vertices in $LE(P^*) \cap \gamma$ for free, and we say that the supercell γ is **good**. Otherwise, the supercell γ is **bad**, and we just compute the vertices in $LE(P^*) \cap \gamma$ by computing $LE(P_\tau^*) \cap \tau$

---

[3] If a triangulation G satisfies the local Delaunay property inside a region γ, and all the boundary edges of γ are correct Delaunay edges, then the part of G inside γ are correct Delaunay triangles. This fact follows immediately, for example, from known properties on constrained Delaunay triangulations [43], with $\partial \gamma$ as constrained edges.

for each cell τ of γ, in time $O(t \cdot (bs_\ell) \log(bs_\ell))$ (again, exceptional cells have already been taken care of).

Observe that if the Delaunay triangles corresponding to all vertices in $LE(P^*) \cap \gamma$ are all good, then the supercell γ must be good. Thus, each bad supercell contains at least one vertex in $LE(P^*)$ whose Delaunay triangle is bad. So, the number of bad supercells is at most $O(D)$.

We conclude that from $LE(P_\ell^*)$, we can compute $LE(P^*)$ in expected time

$$O(n + D \log n + n/(s_\ell t) \cdot \sqrt{t} \cdot (bs_\ell) \log(bs_\ell) + D \cdot t \cdot (bs_\ell) \log(bs_\ell) + 2^{-\Omega(b)} \cdot n \log s_\ell).$$

For example, we can choose $b = s_\ell$ and $t = s_\ell^4$ and obtain the time bound $O(n + D \log n + Ds_\ell^{O(1)})$.

Recall that the algorithm from the proof of Theorem 4 has expected time bound $O(\ell(n + D \log n) + \sum_{i=1}^{\ell-1}(n/s_{i+1}) \log s_i)$. Setting $s_1 = n$, $s_2 = \log n$, $s_3 = \log \log n$ with $\ell = 3$ yields overall time bound $O(n + D \log n + D(\log \log n)^{O(1)}) = O(n + D \log n)$. ◀

The same result holds also when the given graph G is not a triangulation but just a PSLG (possibly disconnected), with missing edges.

▶ **Corollary 10.** *Given a set P of n points in the plane and a PSLG G with vertex set P, the Delaunay triangulation $DT(P)$ can be constructed in $O(n + D \log n)$ expected time, where D is redefined as the (unknown) number of edges in $DT(G)$ that are not in G.*

Proof. The number of missing edges is at most D, and one can easily first triangulate the non-triangle faces of G in $O(D \log D)$ time and then run our algorithm in Theorem 9. ◀

5 An $O(n \log^* n + n \log d_{\mathrm{cross}})$ algorithm

In this section, we present an algorithm for computing the Delaunay triangulation which is sensitive to the parameter d_{cross}. The algorithm uses planar graph separators or r-divisions in the input graph G, like in the deterministic algorithm in Section 2, but the divide-and-conquer is different. The difficulty is that even when d_{cross} is small, the number D of wrong edges could be large. Our idea is to recursively compute the correct Delaunay triangulation of each region, and combine them to obtain a new triangulation where the number of wrong edges is smaller ($O(n/\mathrm{polylog}\, n)$); afterwards, we can apply our earlier D-sensitive algorithm to fix this triangulation. The depth of the recursion is $O(\log^* n)$ (but this is not related to the $\log^* n$ factor in the randomized algorithm in Section 3).

We begin with known subroutines on ray shooting:

▶ **Lemma 11.** *Given h disjoint simple polygons with a total of n vertices in the plane, we can build a data structure in $O(n + h^2 \log^{O(1)} h)$ time so that a ray shooting query can be answered in $O(\log n)$ time.*

More generally, given a parameter $\delta \in [0, 1]$, we can build a data structure in $O(n + h^{2-\delta} \log^{O(1)} h)$ time so that a ray shooting query can be answered in $O(h^\delta \log n)$ time.

Proof. The first part is due to Chen and Wang [18]. The second part follows by dividing the input into h^δ subcollection of $h^{1-\delta}$ polygons each, and applying the data structure to each subcollection: the preprocessing time becomes $O(n + h^\delta \cdot (h^{1-\delta})^2 \log^{O(1)} h)$ and the query time becomes $O(h^\delta \log n)$ (since ray shooting is a "decomposable" problem—knowing the answers to a query for each of the h^δ subcollections, we would know the overall answer). ◀

▶ **Lemma 12.** *Given a connected PSLG G with n vertices and a set Q of line segments, we can compute the subset A of all edges of G that cross Q, in $O(n + (|Q| + |A|)^{2-\delta} \log^{O(1)} n)$ time for a sufficiently small constant $\delta > 0$.*

Proof. Let $r \in [1, |Q|]$ be a parameter to be chosen later. Let R be a $(1/r)$-***net*** of Q of size $O(r \log r)$, i.e., a subset $R \subset Q$ such that every line segment crossing at least $|Q|/r$ segments of Q must cross at least one segment of R. By known results on $(1/r)$-nets [48], this takes $O(|Q| r^{O(1)})$ deterministic time (if randomization is allowed, a random sample already satisfies the property with good probability).

We preprocess G in $O(n)$ time so that a ray shooting query can be answered in $O(\log n)$ time by Lemma 11 (since G can be turned into a tree by introducing cuts, and a tree can be viewed as a simple polygon by traversing each edge twice). For each segment $q \in R$, we find all edges of G crossing q in time $O(\log n)$ times the number of crossings, by repeated ray shooting. The total time of this step is at most $O(n + (r \log r) \cdot |A| \cdot \log n)$.

Let A_R denote the subset of all edges of G that cross R. We next preprocess $G - A_R$ in $O(n + |A|^{2-\delta} \log^{O(1)} n)$ time so that a ray shooting query can be answered in $O(|A|^\delta \log n)$ time by Lemma 11 (since $G - A_R$ has at most $O(|A_R|) \leq O(|A|)$ components and can be viewed as $O(|A|)$ simple polygons). For each segment $q \in Q$, we find all edges of $G - A_R$ crossing q in time $O(|A|^\delta \log n)$ times the number of crossings, by repeated ray shooting. Each edge in $G - A_R$ can cross at most $|Q|/r$ segments of Q because R is a $(1/r)$-net; thus, the total number of crossings between $G - A_R$ and Q is at most $O(|A| \cdot |Q|/r)$. The total time of this step is at most $O(n + |A|^{2-\delta} \log^{O(1)} n + |A| \cdot |Q|/r \cdot |A|^\delta \log n)$. The lemma follows by setting $r = |Q|^{2\delta}$ and choosing a sufficiently small $\delta > 0$ (in our application, it is not important to optimize the value of δ). ◀

▶ **Lemma 13.** *Every (not necessarily connected) planar graph with n vertices has at most $3n - 6$ edges and at least $3n - 6 - O(z)$ edges, where z is the number of edges appearing in non-triangular faces plus the number of isolated vertices.*

Proof. By adding $O(z)$ edges, we can turn the graph into a fully triangulated (i.e., maximal planar) graph, which has exactly $3n - 6$ edges. ◀

▶ **Theorem 14.** *Given a triangulation G of a set P of n points in the plane, the Delaunay triangulation $DT(P)$ can be constructed in $O(n \log^* n + n \log d_{\mathrm{cross}})$ deterministic time, where d_{cross} is the (unknown) maximum number of edges of $DT(P)$ that an edge of G may cross.*

Proof. Consider a region γ_0, which is the union of a connected set of at most μ triangles of G, with at most $c\sqrt{\mu}$ boundary edges, for a sufficiently large constant c. (Initially, γ_0 is the entire convex hull of P.) Let V_{γ_0} denote the set of vertices of all triangles in γ_0. We describe a recursive algorithm to compute $DT(V_{\gamma_0})$:

1. First we compute, in $O(\mu)$ time, a t-division Γ of the dual graph of the part of G inside γ_0, for some t to be chosen later. Each region γ of Γ is the union of a connected set of at most t triangles, with at most $c\sqrt{t}$ boundary edges. Let V_γ denote the set of vertices of all triangles in γ.
2. For each region γ, we compute $DT(V_\gamma)$ recursively.
3. For each region γ, let X_γ be the set of all edges in $DT(V_\gamma)$ that cross the boundary $\partial \gamma$, and let Y_γ be the set of all edges in $DT(V_\gamma)$ that are contained in γ.
   We bound the number of "wrong" edges in Y_γ via an indirect argument, by considering $DT(P)$: We know that the number of edges and triangles of $DT(P)$ crossing $\partial \gamma$ is at most $O(d_{\mathrm{cross}} \sqrt{t})$. The number of edges of $DT(P)$ contained in γ must then be at least

$3|V_\gamma| - O(d_{\text{cross}}\sqrt{t})$ by Lemma 13. Since all edges of $DT(P)$ contained in γ are in both Y_γ and in $DT(V_{\gamma_0})$, there are at least $3|V_\gamma| - O(d_{\text{cross}}\sqrt{t})$ edges of Y_γ that are in $DT(V_{\gamma_0})$. This implies that there are at most $3|V_\gamma| - (3|V_\gamma| - O(d_{\text{cross}}\sqrt{t})) = O(d_{\text{cross}}\sqrt{t})$ "wrong" edges of Y_γ that are not in $DT(V_{\gamma_0})$.

In particular, we also have $|Y_\gamma| \geq 3|V_\gamma| - O(d_{\text{cross}}\sqrt{t})$, and $|X_\gamma| \leq 3|V_\gamma| - |Y_\gamma| = O(d_{\text{cross}}\sqrt{t})$.

We can compute X_γ by applying Lemma 12 to the triangulation $DT(V_\gamma)$ and the line segments in $\partial\gamma$, in $O(t + (d_{\text{cross}}\sqrt{t})^{2-\delta}\log^{O(1)} t)$ time. We can then generate Y_γ in $O(t)$ additional time by a graph traversal. The total running time for this step is $O(\mu/t \cdot (t + (d_{\text{cross}}\sqrt{t})^{2-\delta}\log^{O(1)} t)) = O(\mu + (d_{\text{cross}}^{2-\delta}\mu/t^{\delta/2})\log^{O(1)} t)$.

4. For each region γ, we extend Y_γ to obtain a complete triangulation G'_γ of γ using all the vertices in V_γ (but no Steiner points), by triangulating the non-triangular faces in $Y_\gamma \cup \partial\gamma$ inside γ. Since $\partial\gamma$ has size $O(\sqrt{t})$ and $|Y_\gamma| \geq 3|V_\gamma| - O(d_{\text{cross}}\sqrt{t})$, the non-triangular faces have complexity $O(d_{\text{cross}}\sqrt{t})$ and can be triangulated in $O(d_{\text{cross}}\sqrt{t}\log(d_{\text{cross}}\sqrt{t}))$ time. This cost is absorbed by the cost of other steps.

5. We combine the triangulations G'_γ for all γ in Γ to get a complete triangulation G' of $DT(V_{\gamma_0})$, by triangulating the non-triangular faces in $Y_{\gamma_0} \cup \partial CH(V_{\gamma_0})$. These non-triangular faces have total complexity $O(\sqrt{\mu})$, and so can be triangulated in $O(\sqrt{\mu}\log\mu)$ time.

6. Finally, apply our previous algorithm from Theorem 1 to obtain $DT(V_{\gamma_0})$ from G'. The total number of "wrong" edges of G' that are not in $DT(V_{\gamma_0})$ is at most $O(\mu/t \cdot d_{\text{cross}}\sqrt{t} + \sqrt{\mu}) = O(d_{\text{cross}}\mu/\sqrt{t} + \sqrt{\mu})$. So, this step takes $O(\mu + (d_{\text{cross}}\mu/\sqrt{t} + \sqrt{\mu})\log^3\mu)$ deterministic time.

The total time satisfies the following recurrence:

$$T(\mu) = \max_{\substack{\mu_1, \mu_2, \ldots \leq t: \\ \mu_1 + \mu_2 + \cdots \leq \mu}} \left(\sum_i T(\mu_i) + O(\mu + (d_{\text{cross}}^{2-\delta}/t^{\delta/2} + d_{\text{cross}}/\sqrt{t})\mu\log^{O(1)}\mu) \right).$$

Let b be a parameter to be chosen later. Setting $t = \max\{\log^c \mu, b^c\}$ for a sufficiently large constant c (depending on δ) gives

$$T(\mu) = \max_{\substack{\mu_1, \mu_2, \ldots \leq \max\{\log^c \mu, b^c\}: \\ \mu_1 + \mu_2 + \cdots \leq \mu}} \left(\sum_i T(\mu_i) + O((1 + d_{\text{cross}}^2/b^2)\mu) \right).$$

Combined with the naive bound $T(\mu) \leq O(\mu\log\mu) \leq O(\mu\log b)$ for the base case when $\mu \leq b^c$, the recurrence solves to $T(\mu) = O((1 + d_{\text{cross}}^2/b^2)\mu\log^*\mu + \mu\log b)$.

If we know d_{cross}, we can simply set $b = d_{\text{cross}}$. If not, we can "guess" its value using an increasing sequence. More precisely, for $j = \lceil\log(\log^*\mu)\rceil, \lceil\log(\log^*\mu)\rceil + 1, \ldots$, we set $b = 2^{2^j}$ and run the algorithm with running time capped at $O(\mu 2^j)$. The algorithm succeeds at the latest when j reaches $\max\{\lceil\log(\log^*\mu)\rceil, \lceil\log\log d_{\text{cross}}\rceil\}$, and the total running time is dominated by that of the last iteration, which is $O(\mu\log^*\mu + \mu\log d_{\text{cross}})$. ◄

The same approach can be applied to EMSTs:

▶ **Theorem 15.** *Given a triangulation G of a set P of n points in the plane, the Euclidean minimum spanning tree $EMST(P)$ and the Delaunay triangulation $DT(P)$ can be constructed in $O(n(\log^* n)^2 + n\log d_{\text{cross-emst}})$ expected time, where $d_{\text{cross-emst}}$ is the (unknown) maximum number of edges of $EMST(P)$ that an edge of G may cross.*

Proof. We modify the algorithm in the proof of Theorem 14, replacing $DT(\cdot)$ with $EMST(\cdot)$ and "triangulations" with "spanning trees". All the coefficients 3 are replaced with 1. In step 6, we invoke Corollary 7 instead (which causes the extra $\log^* n$ factor). When guessing $d_{\text{cross-emst}}$, if we cap the running time at $C\mu 2^j$ for a sufficiently large constant C, the probability that j needs to reach $\max\{\lceil \log(\log^* \mu) \rceil, \lceil \log \log d_{\text{cross-emst}} \rceil + s\}$ for a given s is $O(1/C)^s$, so the expected total running time is

$$\sum_{s \geq 0} O(1/C)^s \cdot C(\mu \log^* \mu + 2^s \mu \log d_{\text{cross-emst}}) = O(\mu \log^* \mu + \mu \log d_{\text{cross-emst}}).$$

After computing $EMST(P)$, we can obtain $DT(P)$ e.g. by Devillers' algorithm [28] in $O(n \log^* n)$ additional expected time. ◀

We can also obtain results involving multiple parameters; for example, it is not difficult to adapt the proof of Theorem 14 to get a Delaunay triangulation algorithm with running time $O(n \log^* n + D \log d_{\text{cross}})$.

6 An $O(n \log \log n + n \log(1/\rho))$ algorithm in the probabilistic model

Using our d_{cross}-sensitive algorithm, we can immediately obtain an efficient algorithm for Delaunay triangulations in the probabilistic model.

▶ **Lemma 16.** *Given a set P of n points in the plane and $\rho \in (0,1)$, let R be a subset of the edges of $DT(P)$ where each edge is selected independently with probability ρ. Let G be any triangulation of P that contains R. Let d_{cross} be the maximum number of edges of $DT(P)$ that an edge of G may cross. Then $d_{\text{cross}} \leq O((1/\rho) \log n)$ with high probability.*

Proof. This follows from a standard sampling-based hitting set or epsilon-net argument: None of the edges of G cross R. For a fixed $p, q \in P$ such that pq crosses more than $(c/\rho) \ln n$ edges of $DT(P)$, the probability that pq does not cross R is at most $(1 - \rho)^{(c/\rho) \ln n} \leq (e^{-\rho})^{(c/\rho) \ln n} = n^{-c}$. By a union bound, the probability that $d_{\text{cross}} > (c/\rho) \ln n$ is at most n^{-c+2}. ◀

▶ **Corollary 17.** *Given a set P of n points in the plane and $\rho \in (0,1)$, let R be a subset of the edges of $DT(P)$ where each edge is selected independently with probability ρ. Let G be any triangulation of P that contains R. Given G, the Delaunay triangulation $DT(P)$ can be constructed in $O(n \log \log n + n \log(1/\rho))$ time with high probability.*

Proof. We just apply Theorem 14 in combination with the preceding lemma. ◀

A similar approach works for EMST:

▶ **Lemma 18.** *Given a set P of n points in the plane and $\rho \in (0,1)$, let R be a subset of the edges of $EMST(P)$ where each edge is selected independently with probability ρ. Let T be any spanning tree that contains R and does not have any crossings. Let $d_{\text{cross-emst}}$ be the maximum number of edges of $EMST(P)$ that an edge of T may cross. Then $d_{\text{cross-emst}} \leq O((1/\rho) \log n)$ with high probability.*

▶ **Corollary 19.** *Given a set P of n points in the plane and $\rho \in (0,1)$, let R be a subset of the edges of $EMST(P)$ where each edge is selected independently with probability ρ. Let T be any spanning tree that contains R and does not have any crossings. Given T, the Euclidean minimum spanning tree $EMST(P)$ and the Delaunay triangulation $DT(P)$ can be constructed in $O(n \log \log n + n \log(1/\rho))$ time with high probability.*

Proof. We can compute a triangulation G of P from T, e.g., by Chazelle's algorithm in linear time [13] or Seidel's algorithm in $O(n \log^* n)$ expected time [54]. We can then apply Theorem 15 in combination with the preceding lemma. ◄

A similar result holds in a random flip model:

▶ **Lemma 20.** *Given a set P of n points in the plane and $\rho \in (0, 1)$, consider the following process:*

(∗) Initialize $G = DT(P)$. Let m be the number of edges in G. For $i = 1, \dots, (1-\rho)m$: pick a random edge e_i from G, and if the two triangles incident to e_i in G forms a convex quadrilateral, flip e_i in G.

At the end of the process, let d_{cross} be the maximum number of edges of $DT(P)$ that an edge of G may cross. Then $d_{\mathrm{cross}} \leq O((1/\rho) \log n)$ with high probability.

Proof. Let $t = (1-\rho)m$. For the analysis, it is helpful to assign a distinct label $\ell(e) \in \{1, \dots, m\}$ to each edge e in G. When an edge is flipped, we give the new edge the same label as the old edge, to maintain the invariant. Then $\ell(e_1), \dots, \ell(e_t)$ are independent, uniformly distributed random variables from $\{1, \dots, m\}$. Consider a fixed $p, q \in P$ such that pq crosses more than $b := (c/\rho) \ln n$ edges of $DT(P)$. Let B be the labels of a fixed subset of b of these edges. The probability that pq is in the final G is at most the probability that $\{\ell(e_1), \dots, \ell(e_t)\}$ contains B, which is equal to $\binom{t}{b} \cdot \frac{b!}{m^b} = \frac{t!}{(t-b)! m^b} \leq \frac{t^b}{m^b} = (1-\rho)^b \leq e^{-\rho b} = n^{-c}$. By a union bound, the probability that $d_{\mathrm{cross}} > (c/\rho) \ln n$ is at most n^{-c+2}. ◄

▶ **Corollary 21.** *Given a set P of n points in the plane and $\rho \in (0, 1)$, let G be a triangulation of P generated by the process (∗) above. Given G, the Delaunay triangulation $DT(P)$ can be constructed in $O(n \log \log n + n \log(1/\rho))$ time with high probability.*

7 An $O(n \log^* n + n \log d_{\mathrm{vio}})$ algorithm

Using our d_{cross}-sensitive algorithm, we can also immediately obtain a d_{vio}-sensitive algorithm for Delaunay triangulations, after proving that d_{cross} is always upper-bounded by $O(d_{\mathrm{vio}}^2)$.

▶ **Lemma 22.** *Given a triangulation G of a set P of n points in the plane, let d_{vio} be the maximum number of points of P that the circumcircle of a triangle of G may contain. Consider a triangle $\triangle qq''q'''$ in G. The number of edges of $DT(P)$ incident to q that cross $q''q'''$ is at most d_{vio}.*

Proof. Suppose qq' is an edge of $DT(P)$ that crosses $q''q'''$. Then q must be inside the circumcircle of $\triangle qq''q'''$ (because otherwise, qq' and $q''q'''$ would both be edges of $DT(\{q, q', q'', q'''\})$, contradicting planarity of Delaunay triangulations). So, there are at most d_{vio} choices of q'. ◄

▶ **Lemma 23.** *Given a triangulation G of a set P of n points in the plane, let d_{cross} be the maximum number of edges of $DT(P)$ that an edge of G may cross, and let d_{vio} be the maximum number of points of P that the circumcircle of a triangle of G may contain. Then $d_{\mathrm{cross}} \leq O(d_{\mathrm{vio}}^2)$.*

Proof. Consider an edge pp' of G. Let $\triangle pp'p''$ be a triangle of G incident to this edge. Let C be the circumcircle of $pp'p''$. Let H be the subgraph of G induced by the points inside C.

Suppose qq' is an edge of $DT(P)$ that crosses pp'. Then at least one of q and q' must be inside C (because otherwise, pp' and qq' would both be edges of $DT(\{p, p', q, q'\})$,
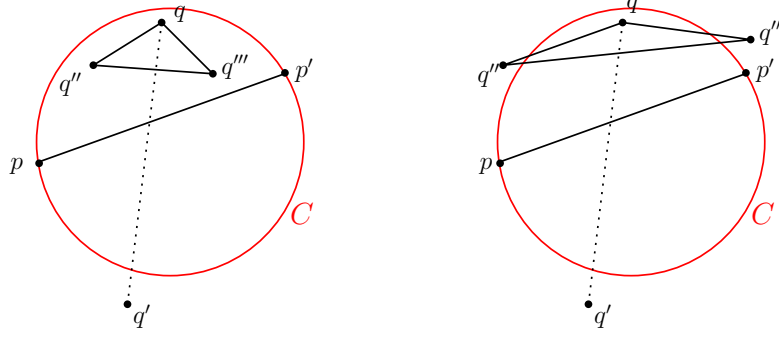
Figure 4 Proof of Lemma 23. The left example is in case (i); the right example is in case (ii).

contradicting planarity of Delaunay triangulations). W.l.o.g., suppose q is inside C. Let $\triangle qq''q'''$ be the triangle in G incident to q such that q' lies in the wedge between $\overrightarrow{qq''}$ and $\overrightarrow{qq'''}$. See Figure 4. Note that $\triangle qq''q'''$ does not intersect pp' (since G is planar). Thus, qq' must intersect $q''q'''$. By Lemma 22, there are at most d_{vio} choices of q' for each fixed $\triangle qq''q'''$.

Furthermore, we must have one of the following: (i) q'' or q''' is inside C, or (ii) p and p' lies in the wedge between $\overrightarrow{qq''}$ and $\overrightarrow{qq'''}$. In case (i), there are at most $2 \cdot \deg_H(q)$ choices of $\triangle qq''q'''$ for each fixed q. In case (ii), there is only one choice of $\triangle qq''q'''$ per q. Thus, the total number of choices of $\triangle qq''q'''$ is $O(\sum_q 2\deg_H(q) + 1))$, which is $O(d_{\mathrm{vio}})$ (since H is a planar graph with at most d_{vio} vertices). The total number of choices of qq' is $O(d_{\mathrm{vio}}^2)$.   ◀

▶ **Corollary 24.** *Given a triangulation G of a set P of n points in the plane, the Delaunay triangulation $DT(P)$ can be constructed in $O(n\log^* n + n\log d_{\mathrm{vio}})$ deterministic time, where d_{vio} is the (unknown) maximum number of points of P that the circumcircle of a triangle of G may contain.*

Alternatively, it is easier to prove the inequality $d_{\text{cross-emst}} \leq O(d_{\mathrm{vio}})$ (since the EMST has constant maximum degree); we can then apply Theorem 15 to obtain the almost same result, except with one more $\log^* n$ factor.

───── **References** ─────

1   Algorithms with predictions. `https://algorithms-with-predictions.github.io/`. Led and managed by Alexander Lindermayr and Nicole Megow.

2   Alok Aggarwal, Leonidas J. Guibas, James B. Saxe, and Peter W. Shor. A linear-time algorithm for computing the Voronoi diagram of a convex polygon. *Discret. Comput. Geom.*, 4:591–604, 1989. `doi:10.1007/BF02187749`.

3   Nancy M. Amato, Michael T. Goodrich, and Edgar A. Ramos. Parallel algorithms for higher-dimensional convex hulls. In *Proc. 35th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 683–694, 1994. `doi:10.1109/SFCS.1994.365724`.

4   Nicolas Auger, Vincent Jugé, Cyril Nicaud, and Carine Pivoteau. On the worst-case complexity of TimSort. In *Proc. 26th Annual European Symposium on Algorithms (ESA)*, pages 4:1–4:13, 2018. `doi:10.4230/LIPICS.ESA.2018.4`.

5   Franz Aurenhammer and Rolf Klein. Voronoi diagrams. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, pages 201–290. North Holland / Elsevier, 2000. `doi:10.1016/b978-044482537-7/50006-1`.

6 Xingjian Bai and Christian Coester. Sorting with predictions. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*, 2023. URL: `http://papers.nips.cc/paper_files/paper/2023/hash/544696ef4847c903376ed6ec58f3a703-Abstract-Conference.html`.

7 Jérémy Barbay and Gonzalo Navarro. On compressing permutations and adaptive sorting. *Theor. Comput. Sci.*, 513:109–123, 2013. `doi:10.1016/J.TCS.2013.10.019`.

8 Jean-Daniel Boissonnat and Mariette Yvinec. *Algorithmic Geometry*. Cambridge University Press, 1998.

9 Kevin Buchin and Wolfgang Mulzer. Delaunay triangulations in $O(\mathrm{sort}(n))$ time and more. *J. ACM*, 58(2):6:1–6:27, 2011. `doi:10.1145/1944345.1944347`.

10 Sergio Cabello and Panos Giannopoulos. Searching in Euclidean spaces with predictions. In *Proceedings of the 22nd Workshop on Approximation and Online Algorithms (WAOA)*, volume 15269 of *Lecture Notes in Computer Science*, pages 31–45, 2024. `doi:10.1007/978-3-031-81396-2\_3`.

11 Timothy M. Chan. A simpler linear-time algorithm for intersecting two convex polyhedra in three dimensions. *Discret. Comput. Geom.*, 56(4):860–865, 2016. `doi:10.1007/s00454-016-9785-3`.

12 Timothy M. Chan, Jack Snoeyink, and Chee-Keng Yap. Primal dividing and dual pruning: Output-sensitive construction of four-dimensional polytopes and three-dimensional Voronoi diagrams. *Discret. Comput. Geom.*, 18(4):433–454, 1997. `doi:10.1007/PL00009327`.

13 Bernard Chazelle. Triangulating a simple polygon in linear time. *Discret. Comput. Geom.*, 6:485–524, 1991. `doi:10.1007/BF02574703`.

14 Bernard Chazelle. An optimal algorithm for intersecting three-dimensional convex polyhedra. *SIAM J. Comput.*, 21(4):671–696, 1992. `doi:10.1137/0221041`.

15 Bernard Chazelle, Olivier Devillers, Ferran Hurtado, Mercè Mora, Vera Sacristán, and Monique Teillaud. Splitting a Delaunay triangulation in linear time. *Algorithmica*, 34(1):39–46, 2002. `doi:10.1007/S00453-002-0939-8`.

16 Bernard Chazelle and Joel Friedman. A deterministic view of random sampling and its use in geometry. *Comb.*, 10(3):229–249, 1990. `doi:10.1007/BF02122778`.

17 Bernard Chazelle and Wolfgang Mulzer. Computing hereditary convex structures. *Discret. Comput. Geom.*, 45(4):796–823, 2011. `doi:10.1007/S00454-011-9346-8`.

18 Danny Z. Chen and Haitao Wang. Visibility and ray shooting queries in polygonal domains. *Comput. Geom.*, 48(2):31–41, 2015. `doi:10.1016/J.COMGEO.2014.08.003`.

19 Otfried Cheong, Ketan Mulmuley, and Edgar Ramos. Randomization and derandomization. In Jacob E. Goodman, Joseph O'Rourke, and Csaba D. Tóth, editors, *Handbook of Discrete and Computational Geometry*. CRC Press, 2017. URL: `https://www.csun.edu/~ctoth/Handbook/chap44.pdf`.

20 David R. Cheriton and Robert Endre Tarjan. Finding minimum spanning trees. *SIAM J. Comput.*, 5(4):724–742, 1976. `doi:10.1137/0205051`.

21 L. Paul Chew. Building Voronoi diagrams for convex polygons in linear expected time. Technical Report PCS-TR90-147, Dept. Math. Comput. Sci., Darmouth College, Hanover, NH, 1986.

22 Francis Y. L. Chin and Cao An Wang. Finding the constrained Delaunay triangulation and constrained Voronoi diagram of a simple polygon in linear time. *SIAM J. Comput.*, 28(2):471–486, 1998. `doi:10.1137/S0097539795285916`.

23 Kenneth L. Clarkson. Randomized geometric algorithms. In D.-Z. Du and F. K. Hwang, editors, *Computing in Euclidean Geometry*, pages 117–162. World Scientific, 1992.

24 Kenneth L. Clarkson and Peter W. Shor. Application of random sampling in computational geometry, II. *Discret. Comput. Geom.*, 4:387–421, 1989. `doi:10.1007/BF02187740`.

25 Kenneth L. Clarkson, Robert Endre Tarjan, and Christopher J. Van Wyk. A fast Las Vegas algorithm for triangulating a simple polygon. *Discret. Comput. Geom.*, 4:423–432, 1989. `doi:10.1007/BF02187741`.

26   Vincent Cohen-Addad, Tommaso d'Orsi, Anupam Gupta, Euiwoong Lee, and Debmalya Panigrahi. Learning-augmented approximation algorithms for maximum cut and related problems. In *Advances in Neural Information Processing Systems 38 (NeurIPS)*, 2024. URL: `http://papers.nips.cc/paper_files/paper/2024/hash/2db08b94565c0d582cc53de6cee5fd47-Abstract-Conference.html`.

27   Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational geometry: Algorithms and Applications*. Springer, 3rd edition, 2008.

28   Olivier Devillers. Randomization yields simple $O(n \log^* n)$ algorithms for difficult $\Omega(n)$ problems. *Int. J. Comput. Geom. Appl.*, 2(1):97–111, 1992. `doi:10.1142/S021819599200007X`.

29   Olivier Devillers, Giuseppe Liotta, Franco P. Preparata, and Roberto Tamassia. Checking the convexity of polytopes and the planarity of subdivisions. *Comput. Geom.*, 11(3-4):187–208, 1998. `doi:10.1016/S0925-7721(98)00039-X`.

30   Herbert Edelsbrunner and Weiping Shi. An $O(n \log^2 h)$ time algorithm for the three-dimensional convex hull problem. *SIAM J. Comput.*, 20(2):259–269, 1991. `doi:10.1137/0220016`.

31   David Eppstein, Michael T. Goodrich, and Vinesh Sridhar. Computational geometry with probabilistically noisy primitive operations. In *Proc. 19th International Symposium on Algorithms and Data Structures (WADS)*, pages 24:1–24:20, 2025. `doi:10.4230/LIPIcs.WADS.2025.24`.

32   Vladimir Estivill-Castro and Derick Wood. A survey of adaptive sorting algorithms. *ACM Comput. Surv.*, 24(4):441–476, 1992. `doi:10.1145/146370.146381`.

33   Greg N. Federickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM Journal on Computing*, 16(6):1004–1022, 1987. `doi:10.1137/0216064`.

34   Steven Fortune. Voronoi diagrams and Delaunay triangulations. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, pages 705–721. Chapman and Hall/CRC, 3rd edition, 2017. URL: `https://www.csun.edu/~ctoth/Handbook/chap27.pdf`.

35   Joachim Gudmundsson, Mikael Hammar, and Marc J. van Kreveld. Higher order Delaunay triangulations. *Comput. Geom.*, 23(1):85–98, 2002. `doi:10.1016/S0925-7721(01)00027-X`.

36   Sabine Hanke, Thomas Ottmann, and Sven Schuierer. The edge-flipping distance of triangulations. *J. Univers. Comput. Sci.*, 2(8):570–579, 1996. `doi:10.3217/JUCS-002-08-0570`.

37   John Iacono and Stefan Langerman. Proximate planar point location. In *Proceedings of the 19th ACM Symposium on Computational Geometry (SoCG)*, pages 220–226, 2003. `doi:10.1145/777792.777826`.

38   Jerzy W. Jaromczyk and Godfried T. Toussaint. Relative neighborhood graphs and their relatives. *Proc. IEEE*, 80(9):1502–1517, 1992. `doi:10.1109/5.163414`.

39   David R. Karger, Philip N. Klein, and Robert Endre Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *J. ACM*, 42(2):321–328, 1995. `doi:10.1145/201019.201022`.

40   David G. Kirkpatrick. Efficient computation of continuous skeletons. In *Proc. 20th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 18–27, 1979. `doi:10.1109/SFCS.1979.15`.

41   David G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM J. Comput.*, 12(1):28–35, 1983. `doi:10.1137/0212002`.

42   Philip N. Klein, Shay Mozes, and Christian Sommer. Structured recursive separator decompositions for planar graphs in linear time. In *Proceedings of the 45th ACM Symposium on Theory of Computing (STOC)*, pages 505–514, 2013. `doi:10.1145/2488608.2488672`.

43   D. T. Lee and A. K. Lin. Generalized Delaunay triangulation for planar graphs. *Discret. Comput. Geom.*, 1:201–217, 1986. `doi:10.1007/BF02187695`.

44   Andrzej Lingas. A linear-time construction of the relative neighborhood graph from the Delaunay triangulation. *Comput. Geom.*, 4:199–208, 1994. `doi:10.1016/0925-7721(94)90018-3`.

45 Maarten Löffler and Wolfgang Mulzer. Triangulating the square and squaring the triangle: Quadtrees and Delaunay triangulations are equivalent. *SIAM J. Comput.*, 41(4):941–974, 2012. `doi:10.1137/110825698`.

46 Maarten Löffler and Jack Snoeyink. Delaunay triangulation of imprecise points in linear time after preprocessing. *Comput. Geom.*, 43(3):234–242, 2010. `doi:10.1016/J.COMGEO.2008.12.007`.

47 Pinyan Lu, Xuandi Ren, Enze Sun, and Yubo Zhang. Generalized sorting with predictions. In *Proc. 4th Symposium on Simplicity in Algorithms (SOSA)*, pages 111–117, 2021.

48 Jiří Matoušek. Epsilon-nets and computational geometry. In János Pach, editor, *New Trends in Discrete and Computational Geometry*, pages 69–89. Springer, 1993. `doi:10.1007/978-3-642-58043-7_4`.

49 Tomomi Matsui. The minimum spanning tree problem on a planar graph. *Discret. Appl. Math.*, 58(1):91–94, 1995. `doi:10.1016/0166-218X(94)00095-U`.

50 Ross M. McConnell, Kurt Mehlhorn, Stefan Näher, and Pascal Schweitzer. Certifying algorithms. *Comput. Sci. Rev.*, 5(2):119–161, 2011. `doi:10.1016/j.cosrev.2010.09.009`.

51 Kurt Mehlhorn, Stefan Näher, Michael Seel, Raimund Seidel, Thomas Schilz, Stefan Schirra, and Christian Uhrig. Checking geometric programs or verification of geometric structures. *Comput. Geom.*, 12(1-2):85–103, 1999. Preliminary version in SoCG 1996. `doi:10.1016/S0925-7721(98)00036-4`.

52 Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. In Tim Roughgarden, editor, *Beyond the Worst-Case Analysis of Algorithms*, pages 646–662. Cambridge University Press, 2020. `doi:10.1017/9781108637435.037`.

53 J. Ian Munro and Sebastian Wild. Nearly-optimal mergesorts: Fast, practical sorting methods that optimally adapt to existing runs. In *Proc. 26th Annual European Symposium on Algorithms (ESA)*, pages 63:1–63:16, 2018. `doi:10.4230/LIPICS.ESA.2018.63`.

54 Raimund Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Comput. Geom.*, 1:51–64, 1991. `doi:10.1016/0925-7721(91)90012-4`.