Quickly Placing a Point to Maximize Angles

Boris Aronov* boris.aronov@nyu.edu
Mark Yagnatinsky† myag@cis.poly.edu

Polytechnic School of Engineering, NYU, Brooklyn, New York

Abstract

Given a set P of n points in the plane in general position, and a set of non-crossing segments with endpoints in P, we seek to place a new point q such that the constrained Delaunay triangulation of $P \cup \{q\}$ has the largest possible minimum angle. The expected running time of our (randomized) algorithm is $O(n^2 \log n)$ on any input, improving the near-cubic time of the best previously known algorithm. Our algorithm is somewhat complex, and along the way we develop a simpler cubic-time algorithm quite different from the ones already known.

1 Introduction

Often in applications one needs a triangulation (or more generally a mesh) of a point set in the plane, and the triangulation should be as "nice" as possible. What counts as "nice" varies with the application, but one common desire is for the triangles produced to be fat, instead of long and skinny. The ideal in this case is equilateral triangles, but this is usually impossible. However, it is also overkill, since it often suffices that all angles are in a Goldilocks range of "not too big and not too small"; say between 30 and 120 degrees. One common way to formalize the wish for fat triangles is to ask for the smallest angle in the triangulation to be as big as possible. It is well known that the Delaunay triangulation of a planar point set has precisely this property.

Unfortunately, the smallest angle in a Delaunay triangulation can be arbitrarily small. Sometimes, it is acceptable to introduce extra points, known as Steiner points, so as to get a better triangulation. However, it is desirable to avoid introducing too many, because they increase the memory and time requirements of all algorithms that operate on the triangulation.

There are two natural approaches to this problem. One is: given that we want all angles to measure at least x degrees, how many additional points do we need? The other is: given a budget of k points, how large can we force the smallest angle to be?

*Research supported by NSF grants CCF-11-17336 and CCF-12-18791.

†Research supported by GAANN Grant P200A090157 from the US Department of Education and by NSF grant CCF-11-17336.

The fixed-budget question was actually addressed in [1], which presented an algorithm that, given a point set P, finds the best placement of k additional points $q_1, \ldots, q_k$, so that the minimum angle in the Delaunay triangulation of $P \cup \{q_1, \ldots, q_k\}$ is maximized. In fact, the problem they solved was slightly more general, in that the input also included a set S of non-crossing line segments connecting points of P, which must be in the final triangulation. (These are sometimes called *constrained edges*, or simply *constraints*, and the resulting triangulation is called a *constrained Delaunay triangulation*.) This generalization allows one to triangulate a simple polygon, by specifying the polygon boundary as the set of mandatory edges, and more generally handle real-world applications with boundary conditions.

Unfortunately, the running time of the algorithm in [1] is $n^{O(k)}$, because it relies on explicit construction of high-dimensional arrangements. They also present an algorithm for the case $k = 1$, which runs in time $O(n^{4+\varepsilon})$. In [2], we present a slightly super-cubic algorithm for the same problem. In [3], a very simple algorithm was presented for the case where S is empty, running in $O(n^{2+\varepsilon})$ time. A slight tweak handles the non-empty case, slowing the running time down to $O(n^{3+\varepsilon})$.

In this paper we present yet another near-cubic time algorithm for this problem, and then improve it to $O(n^2 \log n)$ by removing the main bottleneck.

2 The algorithm

We first describe a *decision procedure*, which takes as input a set P of n points, an edge set S, and a number z representing an angle measurement. It determines whether there exists a placement of a new point q such that all angles in T_q, the constrained Delaunay triangulation of $P \cup \{q\}$, have measure at least z. It does this by actually computing the locus V_z of all such placements, and reporting whether it is empty. This decision procedure is used as a black box by a search procedure, which uses it to implicitly search a space of $O(n^3)$ critical values of z to find the largest value z^* of z such that V_z is not empty.

2.1 Decision procedure

We begin with some terminology. The circumcircle of a Delaunay triangle is a *Delaunay circle*. The interior of a Delaunay circle is an (open) *Delaunay disk*. If two Delaunay triangles share an edge, the intersection of the two corresponding disks is a *Delaunay lune*, or just *lune* for short. Every edge of a Delaunay triangulation has an associated Delaunay lune. For edges of the hull, this is not obvious, since there is no triangle on the outer side. However, we imagine the third point in that case to be the "point at infinity," and thus the circumdisk has infinite radius and degenerates to a "circumhalf-plane"; that is, the half-plane which lies on the outside of the hull, whose line goes through the edge in question.

Lastly, we say that two points *see* each other, if the line segment connecting them does not cross any edge of S. Note that the two endpoints of an edge in S see each other.

We now review the characterization of Delaunay triangulations. Let P be a point set (in general position) with $r, s, t \in P$. Then $\triangle rst$ is part of the Delaunay triangulation T of P if and only if the triangle's circumdisk contains no other points of P.

The characterization of constrained Delaunay triangulations is more complicated than that of unconstrained ones. Let P again be a point set in general position with $r, s, t \in P$, and let S be the set of constraints. Then $\triangle rst$ is part of the Delaunay triangulation if and only if the following two conditions are met. First, the vertices of the triangle must see each other. Second, the interior of the triangle's circumcircle must contain no points of P that can be seen from inside the triangle.

Let T be the constrained Delaunay triangulation of P, and let T_q be the triangulation after point q is added. The decision procedure works by computing two types of regions: "bad" regions, and "good" regions. All angles in T_q have measure at least z if and only if q is in all good regions and no bad regions. Thus V_z is the intersection of the good regions minus the union of the bad regions. The decision procedure computes V_z by constructing the good and bad regions, building the arrangement induced by their boundaries, and then traversing this arrangement to label each arrangement feature (face, edge, vertex) as "in V_z" or "not in V_z," in quadratic time. We now fill in the details of the plan above.

2.1.1 Bad regions

Each edge $e = rs$ of T is associated with an *existence region*. This is the locus E_e of points such that placing the new point q there will result in $\triangle qrs$ being a part of T_q. In the absence of constrained edges, E_e would be the symmetric difference of e's two Delaunay disks (that is, their union minus their intersection). If e itself is constrained, then E_e is their union instead of their symmetric difference.

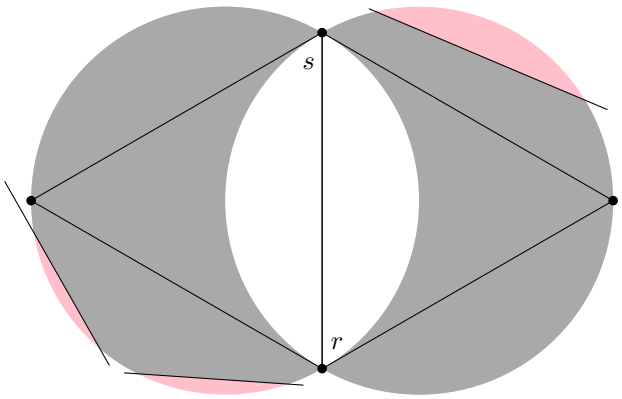

Figure 1: E_{rs}, clipped by three constraints

What happens to E_e in the presence of other constrained edges? Let R be what E_e would look like if there were no constraints other than possibly e. That is, R is either the symmetric difference or union of e's two Delaunay disks. Any constraint that does not intersect R does not affect visibilities inside it. A constraint with an endpoint inside R must not be visible to e, so we can disregard those too. That leaves segments that "clip" R. Such segments divide R into two parts, and e is visible from only one of them. Placing q in the part that does not see e would not cause $\triangle qrs$ to appear in T_q, hence, that part of R doesn't belong to E_e. So, in the presence of constraints, an existence region is the symmetric difference or union of two clipped disks, see Figure 1. Note that due to clipping, the complexity of an existence region may be linear in n.

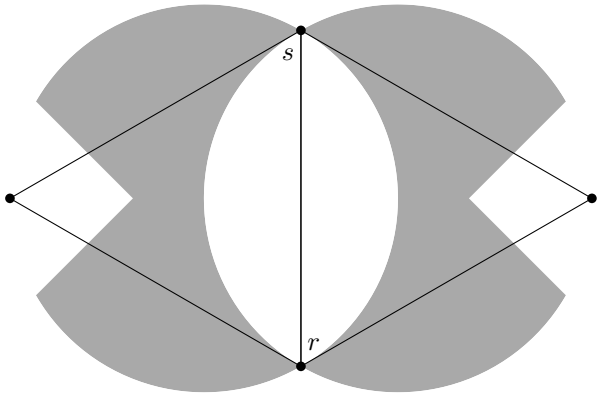

Figure 2: B_{rs} (unclipped), with $z = 45°$

For a fixed value of z, the *bad region* B_e of e is the subset of E_e where some angle of $\triangle qrs$ has measure less than z (see Figure 2). We now describe what the bad region B_e for $e = rs$ looks like. Consider the locus L_e of points that see e at an angle with measure at least z. Since z is less than 90 degrees, it is the union

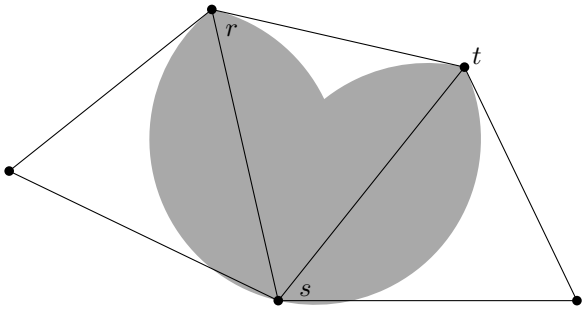

Figure 3: $G_{\angle rst}$, for $z > 52°$

of two congruent disks, both of whose boundaries go through the endpoints of e. Define L'_e to be the locus of points q such that the measure of $\angle qrs$ is at least z; L'_e is the union of two half-planes. Define L''_e analogously for $\angle qsr$. Then $B_e = E_e - (L_e \cap L'_e \cap L''_e)$ is the bad region associated with e: if q is in B_e then triangle qrs exists and at least one of its three angles has measure less than z. Note that we need not analyze the effect of constraints on a bad region, since it is a subset of its existence region, which we have analyzed. Like existence regions, bad regions can have linear complexity due to clipping by constraints.

2.1.2 Good regions

Intuitively, bad regions are our mechanism to avoid placements of q that create new small angles. To ensure that we eliminate all preexisting small angles, each angle α of T is associated with a *good region* G_α. If the measure of α is at least z, then G_α is the entire plane. Otherwise, it is the locus of points such that placing q there will eliminate at least one of α's two bounding edges from T_q; see Figure 3. (Intuitively, if α is small, we want to merge it with a neighboring angle.) In the absence of constraints, if α measures less than z, then G_α is the union of the Delaunay lunes of α's bounding segments. If one of those segments is constrained, then G_α is simply the lune of the other segment. If both segments are constrained, then G_α is the empty set. Good regions are affected by clipping in the same way as existence regions, for the same reasons.

2.1.3 Wrap-up

So, to conclude, in order to have all angles of T_q larger than z, we must ensure q lies in no bad region, and in every good region. Letting $\alpha(T)$ denote the set of angles of T, we have:

Lemma 1 *No angle of T_q is smaller than z if and only if q is in V_z, where $V_z = \bigcap_{\beta\in\alpha(T)} G_\beta - \bigcup_{e\in T} B_e$.*

Proof. One direction is immediate: if $q \notin V_z$, then there will be some angle smaller than z in T_q, for if q fails to be in some good region, there is a preexisting small angle that remains, and if q ends up in a bad region, a new small angle is created.

It remains to show the other direction: if $q \in V_z$, can there be small angles? There are two possible sources of small angles. The first source is a new small angle created by q: either $\angle qrs$ or maybe $\angle rqs$, for some $rs \in T$. Then q is in B_{rs} and thus not in V_z.

The other source is a small angle already present in T. This also can't happen, because then q fails to be in that angle's good region. This last statement seemingly sweeps a bit under the rug. Suppose two adjacent angles are both so small that their sum measures less than z. The good region requirement is satisfied if we merely merge the two angles by eliminating their shared edge, but we are still left with an angle whose measure is less than z. We are saved by the bad regions.

To see how, we first take a brief detour to ask: how does T change when q is added to P? This actually becomes obvious once we look at the situation backwards: starting from T_q, remove q to obtain T. Clearly, all edges (and triangles) which were incident to q are gone, leaving a star-shaped polygonal hole. The certificate that it is star-shaped comes from q itself, since of course q was able to see all of its neighbors! Edges that were not incident to q remain, since if they were Delaunay edges before, they were associated with some Delaunay disk, and removing q can not cause a disk that used to be empty to suddenly become non-empty. Thus, in the forward direction, inserting q makes some edges go away, and the resulting polygonal hole is triangulated by connecting q to all vertices of the hole.

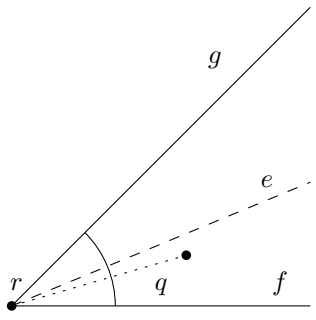

Figure 4: Merging two small angles does not allow q to sneak into V_z.

We can now answer our original question: what prevents us from merging two adjacent small angles into a new angle that is still too small, and then mistakenly declaring victory? Let one of the small angles be bounded by edges e and f, and the other by e and g; see Figure 4. Now, suppose insertion of q eliminates e, but f and g remain. By assumption, f and g bound an angle α measuring less than z. Clearly, f and g are two consecutive edges of the star-shaped hole created by q,

and thus T_q has an edge connecting q to the common endpoint r of f and g. But qr splits α into two smaller angles! These are both *new* angles, and thus are protected from being too small by the bad regions of f and g. (The case of three or more consecutive small angles is handled identically.) □

2.1.4 Building V_z

We begin by taking care of some technicalities. Let the universe U be the portion of the plane strictly inside the convex hull of P, minus all constraints of S and all points of P. From now on, all the calculations will be confined to U, unless otherwise explicitly stated. The notions of "open" and "closed" for subsets of U will be interpreted relative to U.

It is not difficult to see that good regions are closed (relative to U) and bad regions are open, intuitively, because bad regions were defined using strict inequality in terms of the angles of T_q, and good regions use non-strict. By Lemma 1, we conclude that V_z is closed.

We now explain how to compute the overlay of all the good and bad regions. The boundaries of such regions come from a collection $\mathcal{C}_z$ of $O(n)$ rays, lines, circles, and segments. Let $\mathcal{A}_z$ be the arrangement of $\mathcal{C}_z$. Note that $\mathcal{A}_z$ is a refinement of the arrangement of the boundaries of the good and bad regions. It can be constructed in $O(n^2 \log n)$ time by a standard sweep-line algorithm, as any two objects of $\mathcal{C}_z$ intersect only a constant number of times. Since this would introduce a bottleneck, we instead use a randomized incremental construction from [5, Theorem 6.20] to get expected quadratic time. We would now like to augment this arrangement with information related to the good/bad regions that will be useful for computing V_z. To do that, we must first identify which pieces of $\mathcal{A}_z$ correspond to which regions. To this end, we explain how to trace the relative boundary ∂R of a fixed good/bad region R through $\mathcal{A}_z$ in linear time. In the absence of constraints, ∂R consists of a constant number of connected portions of objects from $\mathcal{C}_z$ and hence is a union of $O(n)$ edges of $\mathcal{A}_z$. We start with a point on ∂R in $\mathcal{A}_z$ and just walk along the edges of $\mathcal{A}_z$ tracing out the boundary, in time proportional to the number of edges and vertices of $\mathcal{A}_z$ visited.

With constraints present, we need a mechanism for clipping R by the constraints. We first identify a point of ∂R that does not lie on a constraint, by picking a point on the unclipped boundary and adding constraints one by one. Each constraint either clips off the chosen point or not. If it does, there is an unclipped point adjacent to the new constraint. Once all constraints have been processed, we have a point on the boundary that has not been clipped off.

We then proceed as before, tracing the boundary of the unclipped R until the current curve crosses a constraint c, necessarily at a vertex v of $\mathcal{A}_z$. As we know the arcs defining the unclipped R and we know c, we can compute the other intersection point w of c with the curve we are tracing. When we hit v, we set a flag saying we have left ∂R, and continue tracing the unclipped boundary. When we hit w, we unset the flag. As long as the flag is not set, all edges and vertices of ∂R that we trace through $\mathcal{A}_z$ will be marked as "this vertex/edge lies on ∂R relative to U (and hence in R if R is a good region and not in R if R is a bad region)," and for edges, we also record which side of the edge lies in R and which outside of it.

We repeat this process for each good/bad region and obtain, in $O(n^2)$ time, an arrangement $\mathcal{A}_z$ decorated with $O(n^2)$ pieces of information marking the relative boundary of each good/bad region.

By construction, $\mathcal{A}_z$ is (a refinement of) the overlay of the boundaries of all the good and bad regions, and hence, V_z, being a Boolean combination of said regions, is the union of some faces in $\mathcal{A}_z$ and, being relatively closed, their relative boundaries as well. Now recall that we view $\mathcal{A}_z$ as contained in U, so S and P are removed. In general, U consists of several connected components we call *rooms*, separated from each other by the constraints (which can be considered as walls). We will compute V_z by computing its part in each room separately, as follows.

Consider the following version of the adjacency graph G of $\mathcal{A}$: there is a node for every feature of $\mathcal{A}$ (vertex, edge, and face), and each node corresponding to an edge of $\mathcal{A}$ is connected to the two nodes corresponding to the edge's adjacent faces and also to its (at most) two adjacent vertices (recall that some vertices and edges are considered removed from $\mathcal{A}$ as they are not in U). Each room of U has a connected component of G associated with it. We build a spanning forest F of G, in which all vertices of $\mathcal{A}$ appear as leaves. We traverse each tree of F in depth-first order, starting at a non-leaf node, as follows.

First, we generate a generic point q in a face f of a room. By construction, either f is completely in V_z or completely outside of it. By running a standard constrained Delaunay triangulation algorithm on $P \cup \{q\}$, we could construct T_q in $O(n \log n)$ time. Instead, by taking advantage of the fact that the old and new point sets are almost identical, we obtain T_q from T in linear time [4]. We then compare the angles of T_q against z to decide whether $f \subset V_z$ or not. In fact, we get more information—we can determine if f is contained in or disjoint from every good/bad region and we record this information in an array indexed by the regions. We also keep track of the number of good and bad regions containing f: recall f is in V_z if and only if it is contained in *all* good regions and *none* of the bad ones.

An edge of F corresponds to either a face/edge in-

cidence or an edge/vertex incidence in $\mathcal{A}_z$. At each face/edge incidence, we have recorded what good/bad regions contain the edge on the boundary and whether the region lies to the left or the right of the edge, which is sufficient to modify the good/bad region counts and the current region array when transitioning between a face and an incident edge, in either direction.

As for an edge/vertex incidence, since vertices of $\mathcal{A}_z$ appear as leaves in F, we will enter a vertex v of $\mathcal{A}_z$ from an adjacent edge e and immediately leave it, again by e, so we have to explain only how to handle an "edge TO vertex" traversal, since the next step is just reversing the previous one and can be implemented by recording the changes made and explicitly undoing them. Since a vertex is decorated with the list of regions it is adjacent to, we simply use this information to update the array.

To summarize, for every one of $O(n)$ rooms of U, we spend linear time initializing T_q and then traverse the corresponding component of F at cost proportional to the complexity of the room (that is, the number of features and decorations in the room). Since the combined complexity of all rooms is $O(n^2)$, we have:

Lemma 2 *Given a set P of n points, a set S of non-crossing edges with endpoints in P, and a number z representing an angle measurement, the decision procedure computes V_z in quadratic time.*

2.2 Search procedure

We already have enough to numerically approximate the best achievable angle z^*. We know that it must be between zero and sixty degrees, so we simply do a binary search on that interval, looking for the largest value of z for which V_z is not empty. Hence we can find z^* with additive error ε in time $O(n^2 \log \frac{1}{\varepsilon})$, or relative error δ in time $O(n^2(\log \frac{1}{\delta} + \log \frac{1}{z^*}))$. In this section we describe how to find the exact answer.

Recall that z^* is the highest value of z such that V_z contains at least one point. To find it, we must somehow reduce the set of candidate z values to finite size. Clearly, if z is the measure of some angle in T, then it is a candidate, but there are also many others. To identify them observe that as z changes, the curves of $\mathcal{C}_z$ (and hence the bounding curves of V_z) defining good and bad regions also change: some lines rotate, some circles grow, etc. We visualize the dependence on z by letting it represent the third dimension. In particular, a moving curve sweeps a surface in three dimensions. Let $\mathcal{A}$ be the arrangement of the set $\mathcal{C}$ of the $O(n)$ surfaces that arise in this way, and let $V = \bigcup_z V_z \times \{z\}$.

Viewed in this manner, the curves defining V_z become unions of two-dimensional faces of $\mathcal{A}$ bounding V; V is the union of some three-dimensional cells in $\mathcal{A}$. Note that our original problem is equivalent to locating a topmost point of V; its z-coordinate is z^* and its xy-projection is an optimal location for q. This point must be the topmost point of some cell of $\mathcal{A}$, and not at an interior point. There are only three ways to be the topmost point of a cell: you must either be at the interior maximum of some face bounding the cell (and thus at the local maximum of a surface from $\mathcal{C}$), or at the interior maximum of an edge of the cell (and thus at the local maximum of the intersection curve of two surfaces from $\mathcal{C}$), or at a vertex of the cell formed by intersecting three surfaces. Since these are bounded-degree algebraic surfaces (after suitable re-parametrization), intersecting three of them yields a set of vertices with a size which is upper-bounded by a constant. This is enough to achieve a running time of $O(n^3 \log n)$: compute all vertices and local maxima by brute force, and then sort them by z coordinate. After that, perform binary search using the decision procedure. In fact, we can speed this up to cubic, using linear-time median-find in order to avoid sorting: find the median z value, invoke the decision procedure, and eliminate half the z values. Repeat $O(\log n)$ times.

To speed this up, we need to avoid enumerating all vertices of $\mathcal{A}$, which means we can't afford to find all intersections. (We can afford to find all local maxima, since there are only $O(n^2)$ of those.) Instead, we wish to do a binary-like search on the z values that we would obtain from the enumeration of triples of surfaces of $\mathcal{C}$, without actually enumerating them all. We first pick a random subset of n^2 triples. For each chosen triple, we compute the relevant arrangement vertices and collect their z values. This gives us a set Z of $O(n^2)$ different z values. Thus, Z partitions the set of $O(n^3)$ candidate z values into $O(n^2)$ intervals, with the largest interval having expected size $O(n \log n)$. We can afford to do a binary search on Z, which will narrow it down to a single interval $[z_0, z_1]$, and then explicitly enumerate all vertices within the slab $I = R^2 \times [z_0, z_1]$ defined by that interval. We then conclude by performing a binary search on the vertices of I using the decision procedure.

We now explain how to enumerate every vertex in I. Fix a surface σ from $\mathcal{C}$. Intersecting σ with all other surfaces in $\mathcal{C}$ produces a collection of $O(n)$ curves in σ forming an arrangement. We find the vertices of this arrangement lying in I by using a "plane" sweep from z_0 to z_1, in $O((k_\sigma + n) \log n)$ time, where k_σ is the number of vertices found. Repeating this for each surface $\sigma \in \mathcal{C}$ produces the list of all the vertices of $\mathcal{A}$ lying in I in time $O((n^2 + \sum k_\sigma) \log n)$, where we already observed that $\sum k_\sigma$ is $O(n \log n)$, in expectation. Performing a binary search on this set identifies the critical value z^* of z in time $O(n^2 \log n)$, as claimed. This finally gives us:

Theorem 3 *Given a set P of n points in the plane in general position, and a set of non-crossing segments with endpoints in P, the algorithm described finds a point*

q such that the constrained Delaunay triangulation of $P \cup \{q\}$ has the largest possible minimum angle. The expected running time of our algorithm is $O(n^2 \log n)$ on any input.

3 Unfinished business and open problems

Most likely, a deterministic algorithm with comparable running time exists. We postpone its discussion until a full version of this paper.

Does there exist a significantly faster approximation algorithm for our problem?

Finally, it would be nice to have a non-trivial lower bound for this problem. Is it 3SUM-hard to determine whether V_z is empty, for a given value z?

References

[1] B. Aronov, T. Asano, and S. Funke. Optimal triangulations of points and segments with Steiner points. *Int. J. Comput. Geom. Appl.*, 20(1) 89–104, 2010.

[2] B. Aronov and M. Yagnatinsky. How to place a point to maximize angles. *Canadian Conference on Computational Geometry* 2013, pages 259–263; see also `http://arxiv.org/abs/1310.6413`.

[3] B. Aronov and M. Yagnatinsky. A simple way to place a point to maximize angles; presented at *Fall Workshop on Computational Geometry* 2013; `http://www-cs.engr.ccny.cuny.edu/~peter/fwcg13/abstracts/m_yagnatinski.pdf`.

[4] L. De Floriani and E Puppo. An on-line algorithm for constrained Delaunay triangulation. *CVGIP: Graphical Models and Image Processing*, 54(4) 290–300, 1992.

[5] M. Sharir and P. K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications*. 1995. Cambridge University Press.