Delaunay Triangulation and Convex Polygons with Predictions*†

Sergio Cabello[1] and Panos Giannopoulos[2]

1 **Faculty of Mathematics and Physics, University of Ljubljana, Slovenia**
Institute of Mathematics, Physics and Mechanics, Slovenia
sergio.cabello@fmf.uni-lj.si

2 **Department of Computer Science, City, University of London, UK**
Panos.Giannopoulos@city.ac.uk

Abstract

We show that given a triangulation T of a set P of n points in the plane, the Delaunay triangulation $DT(P)$ of P can be built in $O(n + k \log^4 k)$ time, where k is the number of edges of $DT(P)$ not present in T. We also show that several problems about convex polygons can be solved in $O(\log k)$ time, when the vertices of the polygons are given in an array and we are given a pointer to vertices that in the array are at distance at most k from the vertices defining the solution.

1 Introduction

We give algorithms for some fundamental geometric problems under the so-called 'predictions' paradigm; see Mitzenmacher and Vassilvitskii [9] for an overview. Given a problem Π and a prediction P for a solution of an instance of Π, we would like to devise an algorithm that computes a solution S in time at most $f(n, d(P, S))$, where n the size of the instance and d is an appropriate measure of how close the prediction is to the actual solution. Roughly speaking, we would like f to be such that when d is small (good prediction) the running time is better than the best known time complexity, and when d is large (bad prediction) the running time approaches that of the best known algorithms, or at least it does not get much worse. A usual example here is binary search in a sorted array with a given starting position; the binary search can be done in $O(1 + \log k)$ steps using exponential search, where k is the distance in the array between the starting and the final position of the search.

Our purpose is to initiate the study of problems in Computational Geometry under the lens of algorithms with predictions. As an example, consider the classical problem of computing the closest pair in a set P of n points in $\mathbb{R}^d$, where d is constant. Assume that the prediction is a pair of points $(p, q) \in P^2$, $p \neq q$. We can then decompose the space into cube cells of diagonal length $|pq|$ and assign each point of P to the cube cell that contains it. If δ^* is the distance of the closest pair, then inside each cube cell there are at most $O((|pq|/\delta^*)^d)$ points. We then iterate over the points in P and, for each point $p_i \in P$, check the distance from p_i to the points that are in the $O(1)$ cube cells that may have some point at distance at most $|pq|$ from p_i. In total, it takes $O(n \cdot (|pq|/\delta^*)^d)$ time, assuming the floor function is

* Research funded in part by the Slovenian Research and Innovation Agency (P1-0297, J1-2452, N1-0218, N1-0285). Research funded in part by the European Union (ERC, KARST, project number 101071836). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

† The authors would like to thank Kostas Tsakalidis for suggesting the area of algorithmic problems with predictions.

40th European Workshop on Computational Geometry, Ioannina, Greece, March 13–15, 2024.
This is an extended abstract of a presentation given at EuroCG'24. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

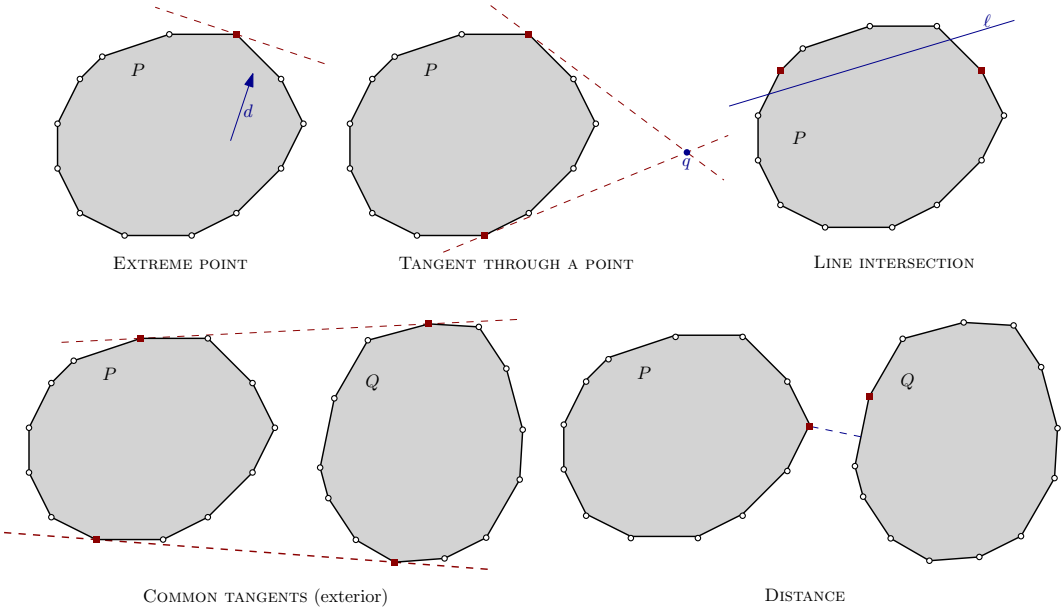

Figure 1 Problems considered in Section 2. In all cases the vertices defining the optimum are marked with red squares.

available. When the prediction is good, i.e., $|pq|/\delta^*$ is constant, the running time will be linear. As the prediction deteriorates, the running time may become quadratic and it is better to turn to other algorithms.

A geometric result that falls in this paradigm is the data structure of Iacono and Langerman [8] for point location in the plane. They consider the problem of storing a plane triangulation for point location with a finger: preprocess the triangulation such that, for a given point p and a finger to a triangle Δ of the triangulation, we can locate the triangle Δ_p containing p in $O(\log d(\Delta, \Delta_p))$ time, where d is a distance-like metric in the subdivision.

In this paper, we provide the following results:

- Several problems for convex polygons described by an array of vertices and with some vertices as prediction can be solved in $O(1 + \log k)$, where k is the distance in the array of the prediction to the vertices describing the solution.
- Given a triangulation T of a planar n-point set in the plane, we can compute its Delaunay triangulation in $O(n + k \log^4 k)$ time, where k is the number of non-Delaunay edges in T.
- Computing the Delaunay triangulation of P from a given triangulation of P requires $\Omega(n \log n)$ operations.

2 Convex polygons

We reconsider early computational geometry problems on convex polygons given by an array under the lens of predictions. Here, predictions are indices of the array that, in the case of a good prediction, are close to the indices of the elements that define the solution object.

Let P and Q be two polygons with m and n vertices respectively, described by the arrays $P[0, \ldots, m-1]$ and $Q[0, \ldots, n-1]$ containing their vertices in counterclockwise order. For any two indices $0 \leqslant i, j \leqslant m-1$ for P, let $d_m(i, j) = \min\{|i-j|, |i+m-j|, |i-j-m|\}$ be their cyclic distance. The cyclic distance $d_n(\cdot, \cdot)$ for Q is defined similarly.

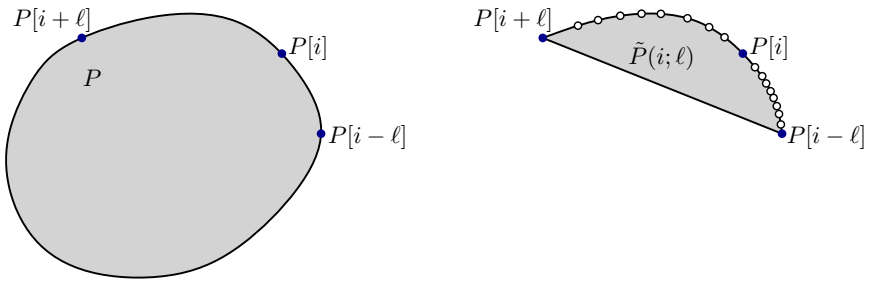

Figure 2 The polygon $\tilde{P}(i, \ell)$.

We consider the following problems with predictions; see Figure 1. In all cases we use k to measure the distance between the prediction and the vertices defining the optimal solution.

- Extreme point: Given P and a direction d, find a extremal vertex of P in the direction d. The prediction is an index i and the measure is

$$k \;=\; \min\{d_m(i, i^*) \mid P[i^*] \text{ extreme point of } P \text{ in direction } d\}.$$

- Tangent through a point: Given P and a point q external to P, find the tangents from q to P. The prediction is two indices i_1 and i_2 and the measure is

$$k \;=\; \min\{d_m(i_1, i_1^*) + d_m(i_2, i_2^*) \mid qP[i_1^*] \text{ and } qP[i_2^*] \text{ define the tangents from } q \text{ to } P\}.$$

- Line intersection: Given P and a line ℓ, find the intersections of ℓ with the boundary of P. The prediction is two indices i_1 and i_2 and the measure is

$$k \;=\; \min\{d_m(i_1, i_1^*)+d_m(i_2, i_2^*) \mid i_1^* \neq i_2^*,\ \ell \text{ intersects } P[i_1^*]P[i_1^*+1] \text{ and } P[i_2^*]P[i_2^*+1]\}.$$

- Common tangents: Given P and Q that are disjoint, find the common exterior/interior tangents of P and Q. For the exterior tangents, the prediction is two indices $i_1, i_2 \in \{0, \ldots, m-1\}$ and two indices $j_1, j_2 \in \{0, \ldots, n-1\}$ and the measure is

$$k \;=\; \min\{ \sum_{t=1,2} (d_m(i_t, i_t^*) + d_n(j_t, j_t^*)) \mid P[i_1^*]Q[j_1^*] \text{ and } P[i_2^*]Q[j_2^*] \text{ define the exterior tangents of } P \text{ and } Q\}.$$

  For the interior tangents the definition is similar.
- Distance: Given P and Q, find the minimum distance between P and Q. The prediction is two indices $i_1 \in \{0, \ldots, m-1\}$ and $j_1 \in \{0, \ldots, n-1\}$ and the measure is

$$k \;=\; \min\{d_m(i_1, i_1^*)+d_n(j_1, j_1^*) \mid \text{the distance between } P[i_1^*]P[i_1^*+1] \text{ and } Q[j_1^*]Q[j_1^*+1] \text{ defines the distance between } P \text{ and } Q\}.$$

▶ **Theorem 2.1.** Extreme point, Tangent through a point, Line intersection, Common tangents *and* Distance *with predictions can be solved in* $O(\log k)$ *time.*

Proof sketch. The idea is to perform a doubly exponential search on the indices. For index i and value ℓ, let $\tilde{P}(i;\ell)$ be the portion of the polygon with vertices $P[i-\ell, \ldots, i+\ell]$, where indices are taken cyclically; see Figure 2. Let $\tilde{P}(i_1, i_2; \ell)$ be the polygon obtained by joining $\tilde{P}(i_1;\ell)$ and $\tilde{P}(i_2;\ell)$. Each of $\tilde{P}(i_1;\ell)$ and $\tilde{P}(i_1, i_2;\ell)$ have at most $O(\ell)$ vertices.

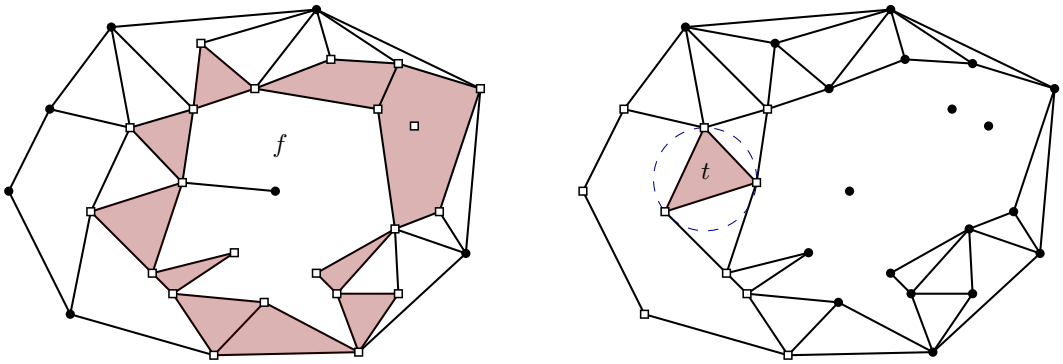

Figure 3 Left: $N_G(f)$ are shaded for the face f, and the points $V(N_G(f))$ are marked with squares. Right: $V(N_G(t))$ are marked with squares for the shaded triangle t.

We start with $\ell = 2$ and use the standard algorithm for the subpolygons $\tilde{P}(i_1;\ell)$ or $\tilde{P}(i_1,i_2;\ell)$, and/or $\tilde{Q}(j_1;\ell)$ or $\tilde{Q}(j_1,j_2;\ell)$, depending on the problem [2, 6, 7, 10]. We get a candidate solution for the subpolygons in $O(\log \ell)$ time, which we can test for optimality for the whole P and Q in constant time. If it is optimal, we have finished, otherwise we square the value of ℓ and repeat. The values of ℓ follow the sequence 2^{2^i} for $i = 0, 1, 2, \ldots, \lceil \log_2 \log_2 k \rceil$ and therefore the total running time is $\sum_{i=0}^{\lceil \log_2 \log_2 k \rceil} O\left(\log(2^{2^i})\right) = O(\log k)$. ◀

3 Delaunay triangulation

A *plane straight-line graph*, shortened to *PSLG*, is a planar embedding of a graph where all edges are drawn as straight-line edges. 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 $CH(P)$. We remark that a triangle with an extra vertex inside, possibly isolated, is not a triangular face of a PSLG.

Let P be a set of n points in the plane. For simplicity, we assume that the points in P are in general position, i.e., no four points are co-circular and no three points are collinear. For a triangle t, let $C(t)$ be its circumcircle, and for $\{p,q,r\} \in \binom{P}{3}$, let $C(p,q,r) = C(\triangle(p,q,r))$.

The Delaunay triangulation of P, denoted by $DT(P)$, can be defined as follows: for any three distinct points $p,q,r \in P$, the triangle $\triangle(p,q,r)$ belongs to $DT(P)$ if and only if the circle $C(p,q,r)$ has no point of P in its interior. The Delaunay triangulation of a point set in general position is unique. We assume familiarity with $DT(P)$; see [4, Chapter 9].

Let G be a PSLG and f be a face of G. Let $N_G(f)$ be the set of neighbor faces of f in the dual graph of G and $V(N_G(f))$ be the vertex set of the faces of $N_G(f)$. Any isolated vertices in the interior of a face of $N_G(f)$ belong also to $V(N_G(f))$; see Figure 3.

A standard property of $DT(P)$ is that it can be tested locally: a triangulation T of P is $DT(P)$ if and only if, for each triangle $t = \triangle(p_i,p_j,p_k)$ of T it holds that no point $p \in V(N_G(t))$ is in the interior of the circle $C(p_i,p_j,p_k)$. We will need the following extension to identify subgraphs of $DT(P)$. See Figure 3, right, for the hypothesis.

▶ **Lemma 3.1.** *Let G be a PSLG with vertex set P and such that each edge of G is on the boundary of $CH(P)$ or adjacent to a triangular face. Assume that, for each triangle t of G, all the points of $V(N_G(t))$ are outside $C(t)$. Then each edge of G is an edge of $DT(P)$.*

Our algorithm will identify and delete an appropriate subset of the edges of a given triangulation of P. For this, we will need the following bounds.

▶ **Lemma 3.2.** *Let T be a triangulation of P and let E' be a subset of $E(T)$ with k edges. Remove from T the edges of E' and then remove also all edges that have bounded, non-triangular faces on both sides. In total, we remove $O(k)$ edges.*

We will use the following data structures that readily follow from Chan [1]. For the circles, one uses the lift to the paraboloid and point-plane duality in $\mathbb{R}^3$.

▶ **Lemma 3.3.** *We can maintain a dynamic set P of n points in the plane such that:*

- *insertion or deletion of a point takes $O(\log^4 n)$ amortized time;*
- *for a query circle C, we can detect whether some point of P is inside C in $O(\log^2 n)$ time.*

We can maintain a dynamic set $\mathcal{C}$ of n circles in the plane such that:

- *insertion or deletion of a circle takes $O(\log^4 n)$ amortized time;*
- *for a query point p, we can detect whether some circle of $\mathcal{C}$ contains p in $O(\log^2 n)$ time.*

We next show our main result, where we assume that the prediction is an arbitrary PSLG.

▶ **Theorem 3.4.** *Assume that we have a set P of n points in general position in the plane and we are given a PSLG G with vertex set P. Let k be the (unknown) number of edges of $DT(P)$ that are not in G. In $O(n + k \log^4 k)$ time we can compute $DT(P)$.*

Proof sketch. The first step is identifying a subset E' of $O(k)$ edges of G in $O(n + k \log^4 k)$ time such that all edges of $G - E'$ are from the Delaunay triangulation $DT(P)$. The subset E' may contain also some edges of $DT(P)$; thus, we compute a superset of the "wrong" edges. The criteria to identify E' is that $G - E'$ should satisfy the assumptions of Lemma 3.1. There are two basic ideas that are being used for this:

- Test circumcircles of triangular faces to see whether they contain some point of a neighbor face. The same triangle may have to be checked multiple times because its neighbor faces may change, and we employ data structures to do this efficiently.
- If an edge is on the boundary of two non-triangular faces, we have no triangle to test the edge. Then, we just add the edge to E'; by Lemma 3.2, this can happen $O(k)$ times.

For any subgraph G' of G, let $F_{\geqslant 4}(G')$ be the faces of G' that are non-triangular and bounded. At the start, $F_{\geqslant 4}(G)$ has $O(k)$ vertices and edges; even if G would be a subgraph of $DT(P)$, we still have to add k edges to G to obtain $DT(P)$.

We start with $E' = \emptyset$ and the PSLG $G' = G$, and maintain the invariant that $G = G' + E'$. Thus, E' is the set of edges that are being removed. Let Q be the set of points that lie in the faces of $F_{\geqslant 4}(G')$. We maintain Q in the first data structure of Lemma 3.3. We also maintain the set τ of triangles that share an edge with a face of $F_{\geqslant 4}(G')$. For each triangle of τ, its circumcircle is stored in the second data structure of Lemma 3.3.

We iterate over the triangles t of G' and check whether $C(t)$ contains some point of $V(N_{G'}(t))$. The adjacent triangles are checked explicitly, while the adjacent non-triangular faces are checked by querying Q with $C(t)$. If $C(t)$ contains some point of $V(N_{G'}(t))$, we delete *all* edges of t from G' and add them to E'. This makes a new non-triangular face of G'. We have to update Q, τ and the data structures that store them; there are $O(1)$ new elements. The circumcircles of the old triangles of τ may contain some new point of Q, and the new triangles of τ (adjacent to the new face of G') have to be retested. See Figure 4. This can be done efficiently using $O(1)$ queries and editions in the data structures. More precisely, for each of the $O(1)$ new triangles in τ we check whether they contain some point

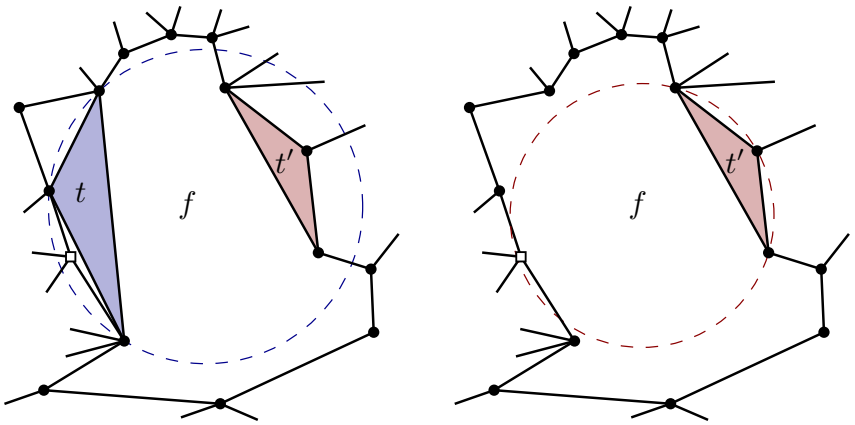

Figure 4 When we check t, we decide to delete $E(t)$ because $C(t)$ contains some point of $V(f) \subseteq Q$. The point marked with an empty square becomes a new point of Q that is in the interior of $C(t')$.

of Q, and for each of the $O(1)$ new points in Q we check whether it is contained in some circumcircle of $\{C(t') \mid t' \in \tau\}$.

For each triangle we delete, at least one of its edges is not in $DT(P)$. Therefore, we delete at most $3k$ edges. Each triangle that is deleted triggers $O(1)$ additional operations in the data structures of Lemma 3.3 storing Q and τ and triggers that $O(1)$ new triangles of τ have to be tested again. Finally, we remove from the final G' all the edges that have on both sides non-triangular faces, and add them to E'. This does not change the set Q nor τ, and therefore we keep having that, for each triangle t of G', $C(t)$ contains no point of $V(N_{G'}(t))$. Because of Lemma 3.2, we have deleted $O(3k) = O(k)$ additional edges.

We have obtained $E' \subseteq E(G)$ such that $|E'| = O(k)$ and Lemma 3.1 applies to $G - E'$. Thus all edges of $G - E'$ are edges of $DT(P)$. We then compute the constrained Delaunay triangulation for the faces of $G - E'$ using the algorithm of Chew [3] in $O(k \log k)$ time. ◀

▶ **Corollary 3.5.** *Assume that we have a set P of n points in general position in the plane and we are given a triangulation T of P. Let k be the (unknown) number of edges of T that are not in $DT(P)$. In $O(n + k \log^4 k)$ time we can compute $DT(P)$.*

Finally we can easily show the following lower bound.

▶ **Theorem 3.6.** *Computing $DT(P)$ for a set P of n points in the plane from a triangulation of P takes $\Omega(n \log n)$ time in the decision tree model of computation.*

4 Conclusions

We conjecture that the Delaunay triangulation $DT(P)$ can be computed from any other triangulation T of P in $O(n + k \log k)$ time, where k is the number of edges of $DT(P)$ that are not in T.

References

1 Timothy M. Chan. Dynamic geometric data structures via shallow cuttings. *Discret. Comput. Geom.*, 64(4):1235–1252, 2020. doi:10.1007/s00454-020-00229-5.

2 Bernard Chazelle and David P. Dobkin. Intersection of convex objects in two and three dimensions. *J. ACM*, 34(1):1–27, 1987. doi:10.1145/7531.24036.

3 L. Paul Chew. Constrained Delaunay triangulations. *Algorithmica*, 4(1):97–108, 1989. doi:10.1007/BF01553881.

4 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational geometry: algorithms and applications, 3rd Edition*. Springer, 2008.

5 Hristo N. Djidjev and Andrzej Lingas. On computing Voronoi diagrams for sorted point sets. *Int. J. Comput. Geom. Appl.*, 5(3):327–337, 1995. doi:10.1142/S0218195995000192.

6 Herbert Edelsbrunner. Computing the extreme distances between two convex polygons. *J. Algorithms*, 6(2):213–224, 1985. doi:10.1016/0196-6774(85)90039-2.

7 Leonidas J. Guibas, John Hershberger, and Jack Snoeyink. Compact interval trees: a data structure for convex hulls. *Int. J. Comput. Geom. Appl.*, 1(1):1–22, 1991. doi:10.1142/S0218195991000025.

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

9 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.

10 Mark H. Overmars and Jan van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, 23(2):166–204, 1981. doi:10.1016/0022-0000(81)90012-X.