Randomization in Computational Geometry

Mahmoodreza Jahanseir

1 Introduction

Computational Geometry problems deal with geometric objects such as points, lines, planes, etc. In many computational geometry problems, the use of randomization yields algorithms that are faster than that known deterministic counterparts. Also, these randomized algorithms are simpler. However, analysis of these algorithms can be complex in many cases. One of the well known techniques to analyze randomized algorithms is *backwards analysis*. In this survey, we use the *backwards analysis* to analyze the expected performance of some randomized algorithms. In backwards analysis, we pretend the algorithm is running backwards in time, from output to input. Backwards analysis was popularized for the first time by Seidel [5].

There are many approaches to design randomized algorithm. In this paper, we use two of them, namely *divide and conquer* and *randomized incremental construction*. The divide and conquer approach is a well know in the area of designing algorithms. In this survey, for solving three problems including Delaunay triangulation of a convex polygon, trapezoidal decomposition, and planar convex hulls we use the divide and conquer approach. In randomized incremental construction, we only use randomization to generate a random permutation for input objects. Then, each object will be processed according to its occurrence in the permutation. In fact, a geometric structure will be built incrementally. In the last part of this survey, we present a randomized incremental construction for the d-dimensional convex hull problem. Note that for many problems, we can easily change the approach of designing from one to another.

2 Delaunay Triangulation of a Convex Polygon

Let S be a set of n points in the plane in general position, i.e. no four points are co-circular and not three points lie on a straight line. The Delaunay triangulation of S or $Del(S)$ is a triangulation of S such that for each triangle in $Del(S)$, the circumscribed disk of the triangle does not have any points of S in its interior. In this section, we describe a randomized algorithm proposed by Chew [1] for the case that the points are vertices of a convex polygon P. This algorithm is recursive and it is as follows.

1. If S has only three points, then $Del(S)$ is the triangle formed by those points and quit.
2. Choose a random point $q \in S$, and Let p and r be the two points adjacent to q in convex polygon P. Recursively compute the Delaunay triangulation of $S' = S \setminus \{q\}$.
3. Identify all bad triangles, namely the triangle p, q, r and all triangles of $Del(S')$ whose circumscribed disk contains q.

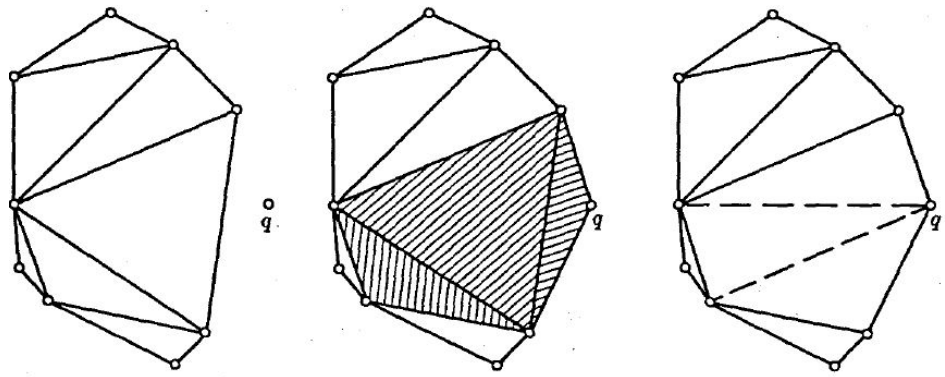

Figure 1: Constructing Delaunay triangulation of a convex polygon starting at a random point q. (left) $Del(S')$. (center) The bad triangles. (right) $Del(S)$.

4. Remove those edges of bad triangles which are shared between two of them. This results some faces combined into one face f containing q.

5. Retriangulate face f by adding all diagonals that have q as an endpoint.

Figure 1 illustrate an example for this algorithm. For the third step of the algorithm, we create a dual graph G for $Del(S')$, and we search G in a depth-first manner. In the dual graph G, nodes are triangles and if two triangles share an edge, we create an edge in G between the two corresponding nodes. Note that the maximum degree of nodes in G is 3, and the set of bad triangles forms a connected subgraph of G. Therefore, we can identify all bad triangles in time proportional to their number.

Now, we want to find the expected running time of this algorithm. Note that the expected running time of this algorithm is the expected time without recursive call. It is not hard to see that this cost is proportional to the number of bad triangles. Therefore, our goal is finding the expected number of bad triangles.

Finding bad triangles in the above algorithm seems difficult, especially when we fix our attention on a specific point q. The trick is expressing this cost in terms of the resulting structure $Del(S)$, not in terms of $Del(S')$ and q. So, it is easy to see that the number of bad triangles is exactly one more than the number of diagonals incident to q and introduced in the algorithm. Equivalently, this number is proportional to the degree of q in $Del(S)$. Because q is chosen randomly from S, the expected degree of q is $Del(S)$ is the sum of the degrees of all vertices in $Del(S)$ divided by n, which is twice the number of edges if $Del(S)$ divided by n. According to the Euler's formula for planar graphs, $|F| + |V| - |E| = 2$. In this formula, F, V and E are the set of faces, vertices and edges of a planar graph, respectively. Note that in a triangulation of a convex polygon, $|F| = |V| - 1$. Therefore, $|E| = 2|V| - 3$, and the expected degree of q is $4 - 6/n$. Thus, the expected time of the algorithm without recursive call is constant and it follows that overall expected time of the algorithm is $O(n)$.

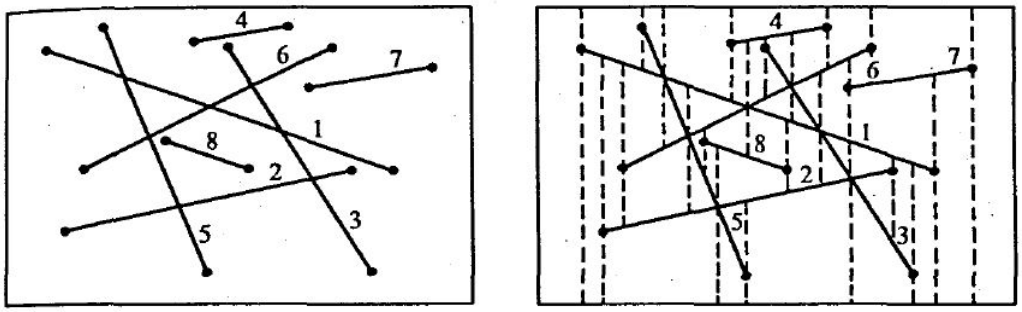

Figure 2: (left) A set of line segments S in a bounding rectangle F. (right) $T(S)$.

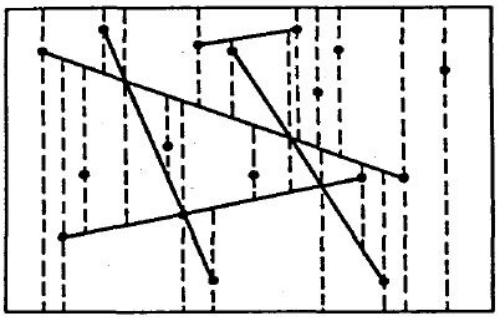

Figure 3: Trapezoidal decomposition $T_S(R)$ for a subset $R = \{1, 2, 3, 4, 5\}$ of S.

3 Line Segments Intersection

Let S be a set of n straight line segments in the plane. The goal is finding all intersecting pairs of segments in S. We also assume that S is non-degenerate. By non-degenerate we mean endpoints are unique, endpoints do not share the same x-coordinate, endpoints do not lie on a line segment, and no three segments intersect in a common point.

In this section we present an algorithm proposed by Mulmuley [4]. Mulmuley's algorithm not only determine pairs of intersecting segments, but also it constructs a *trapezoidal decomposition* induced by S. This decomposition $T(S)$ is defined as follows. First draw an axis-aligned rectangle F that contains S in its interior. Then, for each point in the set of endpoints and intersection points of S, draw its *vertical extension*. A vertical extension is two vertical rays, one going up and the other going down, that extend until they hit a segment of S or the boundary F. Thus, F is decomposed into trapezoids that each have two vertical side. Note that one of the two vertical sides can have length zero. Figure 2 illustrates an example for the trapezoidal diagram. Let K be the number of intersecting pairs of segments in S. Using a sweep argument it is not hard to prove that $T(S)$ contains exactly $3(n + K) + 1$ trapezoids. So, we can view $T(S)$ as a planar graph with $O(n + K)$ faces, edges and vertices.

Similar to definition of $T(S)$, we define a trapezoidal decomposition for a subset $R \subset S$ of size r. In the trapezoidal decomposition $T_S(R)$, we only keep line segments R and the endpoints of segments of $S \setminus R$ in the rectangle F. Then for all endpoints, we draw their vertical extension and we extend them until they hit a line segment of R or the boundary of F. Figure 3 shows an example

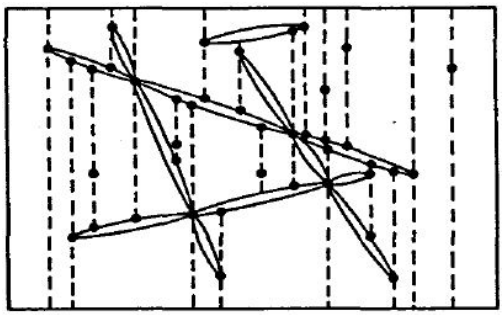

Figure 4: $G_S(R)$.

of $T_S(R)$. The decomposition $T_S(R)$ partitions F into $2n + r + 3K_R + 1$ trapezoids, where K_R is the number of pairs of intersecting segments in R. So, $T_S(R)$ is a planar graph with $O(n + K_R)$ faces, edges, and vertices.

Mulmuley's algorithm is based on the notion of $T_S(R)$ and it is as follows.

1. If $R = \emptyset$, then compute $T_S(R)$ directly by sorting the endpoints of S by their x-coordinates.
2. Otherwise, randomly pick a segment $s \in R$ and recursively compute $T_S(R \setminus \{s\})$ and introduce s into this trapezoidation to obtain $T_S(R)$.

In order to introduce s into $T_S(R \setminus \{s\})$, Mulmuley used a specific representation of $T_S(R)$ that we describe in the following. For a segment $s \in R$, if s is intersected by i other segments of R, then s is partitioned into $i + 1$ pieces. We represent each piece of each segment in R as a narrow zero width face. This results another planar graph $G_S(R)$. Figure 4 shows an example of $G_S(R)$. Note that the number of vertices in $G_S(R)$ is the same as the number of vertices in $T_S(R)$, which is $O(n + K_R)$. Each non zero width face of $G_S(R)$ is incident with at most six edges (one for each top and bottom segment and at most two for each of the left and the right vertical extensions). Note that only the zero width faces can have more than a constant number of edges.

To introduce the segment s into $G_S(R')$, where $R' = R \setminus \{s\}$, first we should determine the faces and edges of $G_S(R')$ that are intersected by s. Then, we create new faces of $G_S(R)$ that includes splitting the faces of $G_S(R')$ that are intersected by s, creating the zero width faces for the pieces of s, adding the vertical extensions from intersection points of s with the other segments of R, and merging faces that are separated by vertical edges that are not part of vertical extensions any more.

The first step can be done as follows. By the definition, the two endpoints of s are in $G_S(R')$. So, in constant time we can determine the leftmost face of $G_S(R')$ that is intersected by s. Then, we walk along segment s until we reach the right endpoint of s. Assume that we just entered a face f through some edge e. We require to find an edge of f that s leaves f through that edge and enters another face. To do this end, we examine all edges of face f. Now, we have reached a new face and we repeat this process.

To analyze the time complexity of introducing segment s, we need to compute the time for the two previous steps. It is not hard to see that cost of step two is more than step one. So, we only need to consider the cost of step one. This cost is clearly the sum of the degrees of the faces of $G_S(R')$ that are intersected by s. For a face f, let $deg(f, G_S(R'))$ be the degree of face f, i.e. the number of edges incident to f.

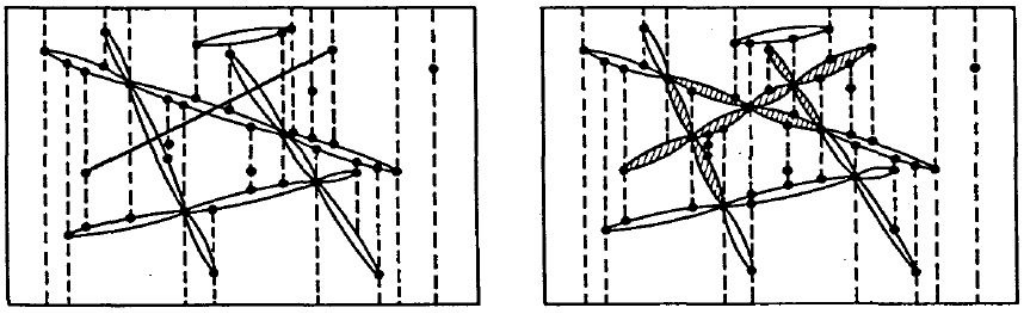

Figure 5: (left) Inserting segment 6 into $G_S(R)$. (right) $Z_R(6)$ is shaded.

In order to apply backwards analysis, we are required to express the cost of step one of introducing a segment s into $G_S(R')$ in terms of result graph $G_S(R)$ and not in terms of $G_S(R')$. It is not hard to see that this cost is given by $\sum_{f \in Z_R(s)} deg(f, G_S(R))$, where $Z_R(s)$ is the set of all zero width faces of $G_S(R)$ that either derive from pieces of s in $G_S(R)$ or from those pieces of other segments that are incident to intersection points with s, see Figure 5.

If s is chosen randomly from segments of R, then the expected cost of adding s to $G_S(R')$ is proportional to

$$\frac{1}{r} \sum_{s \in R} \sum_{f \in Z_R(s)} deg(f, G_S(R)).$$

Note that in this double sum, every piece of a segment in R contributes at most three times, because each piece of a segment is incident to at most two other segment. Therefore, if $Z(R)$ denotes the set of all faces formed by pieces of segments in R, then this double sum is at most

$$\frac{3}{r} \sum_{f \in Z(R)} deg(f, G_S(R)).$$

Because $G_S(R)$ is a planar graph, this sum is proportional to the complexity of the graph, which is $O(n + K_R)$. Therefore, the expected cost of introducing a segment is $O(\frac{n}{r} + \frac{K_R}{r})$. Note that in the algorithm, R is a sample subset of S of size r. Now, if $\{s, t\}$ is one of the K pairs of intersecting segments in S, then the probability of selecting both of them in R is $\frac{r(r-1)}{n(n-1)}$. So, the expected number of pairs of intersecting segments in R is $\frac{r(r-1)}{n(n-1)}K$. It follows that the expected cost of introducing the last segment into $G_S(R)$ is $O(\frac{n}{r} + \frac{r-1}{n(n-1)}K)$. Therefore, the expected cost for all recursive call of the entire algorithm is

$$\sum_{r=1}^{n} O(\frac{n}{r} + \frac{r-1}{n(n-1)}K) = O(n \log n + K).$$

Note that to compute $G_S(\emptyset)$, we only require to sort $2n$ numbers. Thus, the expected running time of the entire algorithm is $O(K + n \log n)$.

4 Planar Convex Hulls

Given a set S of n points, the convex hull $CH(S)$ is the smallest convex set that contains all of the n points. For a set of points in the plane, the boundary of $CH(S)$ forms a convex polygon whose

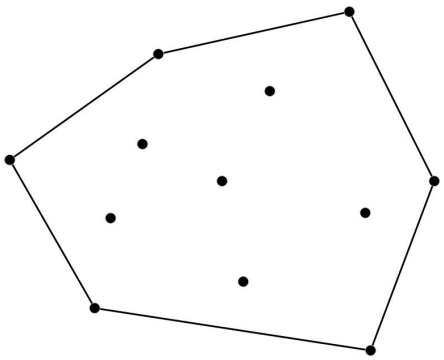

Figure 6: The convex hull of a set S of 12 points in the plane.

vertices are a subset of S. So, the problem of computing the convex hull of S in the plane is finding the smallest convex polygon that contains all of the points in S. The output of this problem is a list of points in S that appear as vertices of $CH(S)$. In this problem, we assume that points are in general position, i.e. no three points are colinear. Figure 6 shows an example of planar convex hulls. In this section, we present an algorithm proposed by Clarkson and Shor [3] to find the convex hull of a point set in the plane.

The algorithm to compute $CH(R)$ for $R \subset S$ is as follows.

1. If $|R| = 3$, then $CH(R)$ is the triangle formed by the three points of R, and quit.
2. Otherwise, randomly choose a point q from R, and let $R' = R \setminus \{q\}$.
3. Recursively compute $CH(R')$.
4. If q is contained in $CH(R')$, then $CH(R) = CH(R')$ and quit.
5. Otherwise, let e be an edge of $CH(R')$ visible from q.
6. Determine all visible edges of $CH(R')$ from q by doing a search starting at e.
7. Replace all the visible edges by two edges that have q as a common endpoint.

Figure 7 illustrates the last two steps of the algorithm. In summary, this algorithm has two main steps. The first step is finding all visible edges. Then, in the second step it removes all of visible edges and create two new edges. To find the cost of second step, we use an amortization argument. Note that when a visible edge is deleted, it does not appear in the convex hull any more. So, we can charge the cost of deletion to the cost of creation of an edge. Furthermore, whenever a point is added, at most two new edges are created. Therefore, the overall cost of creations and deletions for the entire algorithm is $O(n)$.

To implement the first step, for each point $p \in S \setminus R$ we keep a canonical conflict edge e_p of $CH(R)$ which is visible from p. A canonical conflict edge e_p is a unique edge of $CH(R)$ that intersects a straight line segment that connects p to a fixed point c in the interior of $CH(R)$ (not necessarily in S). If for $p \in S \setminus R$, the conflict edge e_p is not visible from q, then this edge is also an edge of $CH(R)$ and remains to be the conflict edge for p. Otherwise, e_p is visible from q and needs to be deleted. Now, the new conflict edge of p is one of the two new added edges incident to q.

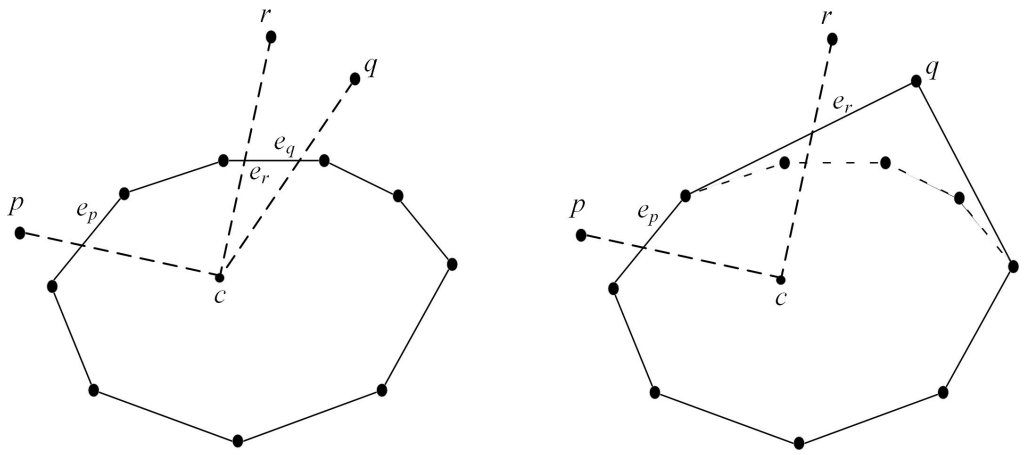

Figure 7: (left) before inserting q. (right) after inserting q.

To estimate the expected cost of maintaining the conflict information, we consider each point $p \in S$ individually. If conflict edge of p is changed, we can find the new conflict edge in constant time. So, we only require to estimate the expected number of changes for p. Now, we find the probability of changing a conflict edge e_p when computing $CH(R)$ from $CH(R')$. Backwards analysis suggests that we should express this probability in terms of the output $CH(R)$. Clearly, e_p is new for p iff q is one of the endpoints of e_p. Because q is a random element of R, the probability that q be one of the two endpoints of e_p is $2/r$. Therefore, the expected number of conflict edge changes for a point p when computing $CH(R)$ from $CH(R')$ is at most $2/r$. Summing over all $r \leq n$ yields that the expected total number of conflict edge changes for a point $p \in S$ is $O(\log n)$. Note that creating initial conflict information in the case $|R| = 3$ takes $O(n)$ time. Thus, the entire maintenance of the conflict information and also the entire algorithm takes expected time $O(n \log n)$.

5 d-dimensional Convex Hulls

In Section 4 we consider a special case of the convex hull problem, when the given point set is in the plane. Here, we present an algorithm given by Clarkson et al. [2] to compute the convex hull of a point set S in d-dimension. Similar to Section 4, assume that the input points are in general position, i.e. no $d+1$ points are coplanar. If $S \subset E^3$, then the boundary of $CH(S)$ is a collection of triangles, which are called *facets*. Each facet has a *supporting plane* containing it. The *supporting halfspace* of a facet is a halfspace of the supporting plane that contains $CH(S)$, see Figure 8. In fact, $CH(S)$ is the intersection of supporting halfspaces of its facets. The open complement of a supporting halfspace of a facet an *empty halfspace* of S.

Now consider the problem of maintaining the convex hull under insertions. In fact, we want to know that for a new point $s \in S \setminus R$, the difference between $CH(R')$, $R' = R \cup \{s\}$, and $CH(R)$. Similar to the planar case, we are interested to find visible facets from point s. So, the algorithm has two main steps. In the first step, search step, we want to find the facets of $CH(R)$ that are visible from s. In the next step, update step, we should delete the visible facets and determine the new facets incident to s.

In order to make the search and the update steps convenient, we maintain a triangulation T of

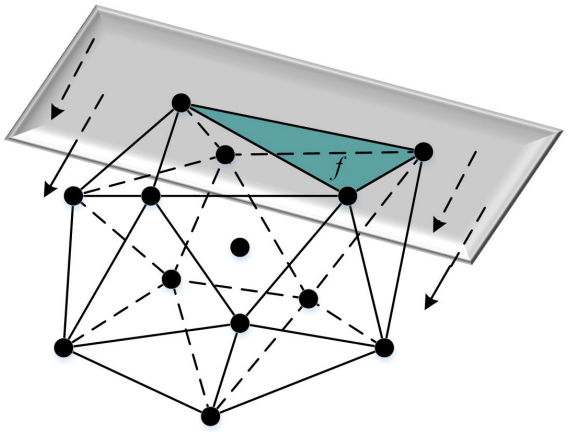

Figure 8: A convex hull for input points $S \subset E^3$. a facet f and its supporting plane is shown. Arrows indicate the supporting halfspace of f

the convex hull. This triangulation is a collection of tetrahedra. When a new point is inserted into T, for each visible facet we create the tetrahedron with that facet and a vertex which is the new point. We call the visible facet the *base* facet of the tetrahedron, and the inserted point its *peak* vertex, see Figure 9. The facets of $CH(R') \setminus CH(R)$ will be among the facets of the new tetrahedra. We also maintain adjacencies among these tetrahedra such that each tetrahedron knows the four others that it shares a facet. We can represent adjacencies for current facets by including tetrahedra with current base facets and with a dummy peak vertex that may later be replaced by a point visible to the base facet.

If less than $d+1$ points are added, we simply maintain R. When R has $d+1$ points, we initialize T with an origin tetrahedron whose vertices are all points and it has $d+1$ adjacent tetrahedra, which all have the dummy point as a peak vertex.

In order to find current visible facets of $CH(R)$, we use visibility of old deleted facets. After insertion of new point, we mark its visible facets deleted. These deleted facets live on as base facets of tetrahedra in T, and they help with searches for other points. To use these deleted facets, we need to extend the notion of visibility. A facet is visible to a point if the point is in the halfspace that was empty halfspace for the facet when it was created. With this definition of visibility, the search procedure starts with the origin tetrahedron and search the neighbors until it finds all of visible facets. The reason that the search procedure works is if we draw a line segment from a point in the origin tetrahedron to the new point, every base facet of a tetrahedron that meets the segment is visible to the new point. Other visible facets can also be found using the adjacency information.

To update the convex hull, we need to create new facets incident to s. Each new facet incident to s has an edge called *horizon* that is contained in a facet visible to s, and also in another facet not visible to s. Figure 10 shows horizon edges. Since each visible facet is adjacent to at most three facets that are not visible, the time needed to create the new facets is proportional to the number of old facets.

The required time to insert a new point is dominated by the search time and this is proportional to the number of deleted and current visible facets. We assume that the points are inserted randomly

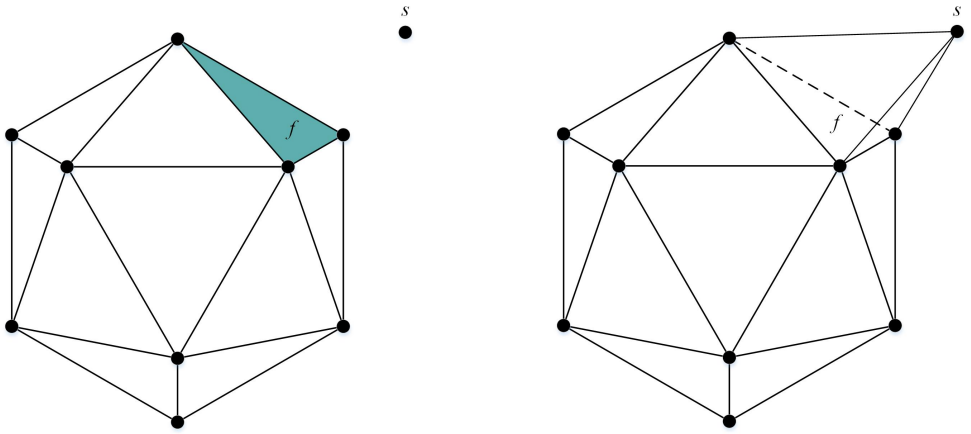

Figure 9: (left) $CH(R)$ before inserting s. The face f is one the visible facets from s. (right) The tetrahedron with base f and peak s.

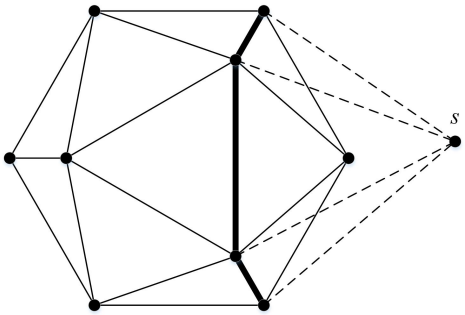

Figure 10: Horizon edges for a new point s.

in the order $x_1, x_2, \cdots, x_n$. This implies that x_i is a random element of $S \setminus R_{i-1}$. Let T_i denote the triangulation T after i insertions. The analysis will express the expected time in terms of f_j, which is the expected number of facets of $CH(R_j)$. Note that when $d = 3$, we have $f_j < 2r - 4$, and in general $f_j = O(j^{\lfloor d/2 \rfloor})$. Let $f_d = 2$ and $f_i = 0$ for $i < d$.

Theorem 1. *The expected size of the triangulation T_r of $CH(R_r)$ is $p_r = \sum_{j \le r} df_j/j$.*

Proof. After $d+1$ points have been added, the triangulation T_{d+1} has $d+2$ simplices. For $j \ge d+2$, the base facets of T_j that are not base facets of T_{j-1} are those incident to x_j and facets of the hull of R_j. Since each facet is incident to d vertices, the expected number of vertex-facet incidences is df_j. Since x_j is a random element of R_j, it is incident to an expected df_j/j facets. □

Theorem 2. *The expected number of base facets of T_{r-1} that see x_r is*

$$-\frac{df_r}{r} + \sum_{j \le r} \frac{d(d-1)f_j}{j(j-1)}.$$

Proof. Let $T = T_{r-1} = T(x_1, \cdots, x_{r-1})$ and $T' = T(x_r, x_1, \cdots, x_{r-1})$. In fact, in T' we pretend that x_r was put in first. Let X be the number of base facets of T_{r-1} that see x_r. Now, $X = |T \setminus T'|$, since $T \setminus T'$ is the set of base facets in T which see x_r, and hence not in T'. Note that for finite sets A and B,

$$|A| + |B \setminus A| = |A \cup B| = |B| + |A \setminus B|.$$

So, we have

$$X = |T \setminus T'| = |T| - |T'| + |T' \setminus T|.$$

By the linearity of expectation, and using Theorem 1,

$$E[X] = E[|T \setminus T'|] = p_{r-1} - p_r + E[|T' \setminus T|].$$

It remains to estimate $E[|T' \setminus T|]$. Note that $T' \setminus T$ is a set of base facets incident to x_r. It is also a collection of facets of the hull with x_r as a vertex over the insertion sequence $x_r, x_1, \cdots, x_{r-1}$. To count these base facets, we count the expected number that appear when x_j is inserted, for $j \le r$. Let $R'_j = R_j \cup \{x_r\}$. For each base facet $f \in T'$, either f is a facet of $CH(R'_{d-1})$, or there is exactly one $j \ge d$ such that f is a facet of $CH(R'_j)$ and x_j is incident to f. A base facet in $T' \setminus T$ is also incident to x_r, and so for given j the number of base facets we count is the expected number of facets of $CH(R'_j)$ incident to x_r and x_j. Since each facet of $CH(R'_j)$ is incident to $\binom{d}{2}$ ordered pair of vertices, the expected number of facets incident to random pair of vertices in R'_j like x_r and x_j is $\binom{d}{2} E[CH(R'_j)] / \binom{j+1}{2}$. Therefore,

$$E[|T' \setminus T|] = f_d + \sum_{d \le j \le r} \binom{d}{2} E[CH(R'_j)] / \binom{j+1}{2}.$$

Using this value of $E[|T' \setminus T|]$ in the previous expression, we have

$$E[X] = p_{r-1} - p_r + f_d + \sum_{d \le j \le r} \binom{d}{2} E[CH(R'_j)] / \binom{j+1}{2}.$$

Since R'_j is a random subset of S, $E[CH(R'_j)] = f_{j+1}$, and a bit of rearranging gives the result. □

When $d = 3$, the expected search time is $O(1) \sum_{1 \le j \le r} 1/j = O(\log r)$. This means the expected total time to build $CH(S)$ is $O(n \log n)$, which is optimal for arbitrary point sets.

References

[1] L. P. Chew. Building voronoi diagrams for convex polygons in linear expected time. Technical report, Hanover, NH, USA, 1990.

[2] K. L. Clarkson, K. Mehlhorn, and R. Seidel. Four results on randomized incremental constructions. *Computational Geometry*, 3(4):185 – 212, 1993.

[3] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, ii. *Discrete & Computational Geometry*, 4(5):387–421, 1989.

[4] K. Mulmuley. A fast planar partition algorithm, i. *Journal of Symbolic Computation*, 10(3):253 – 280, 1990.

[5] R. Seidel. *New Trends in Discrete and Computational Geometry*, chapter Backwards Analysis of Randomized Geometric Algorithms, pages 37–67. Springer Berlin Heidelberg, Berlin, Heidelberg, 1993.