DCG, Wintersemester 2024/25 Lecture 6 (6 pages)

Randomized Incremental Construction

Anne Driemel, Herman Haverkort, Elmar Langetepe updated: October 24, 2024

In the last lecture we saw that computing the convex hull of a set of n points in the plane takes $\Theta(n \log n)$ time. In this lecture we introduce the concept of the randomized incremental construction and we will apply it to analyze a randomized convex hull algorithm for points in the plane. Technically, this does not improve upon the deterministic algorithms in the previous lecture. However, we will see that the concept of the randomized incremental construction generalizes to a powerful framework for solving geometric problems.

1 Convex hulls in the plane, revisited

We start by looking at the second algorithm from the previous lecture and we fill in some implementation details. We assume general position as before. In the previous lecture, we did not specify

(i) how to check if the point p_i is contained in the current convex hull, and
(ii) how to find the common tangents of the point p_i with the current convex hull.

Consider maintaining the necessary information for solving these steps explicitly during the course of the algorithm. For this we define the conflict graph between edges of the convex hull and points to be added to the convex hull. Refer to Figures 1 and 2 for an example.

Definition 6.1 (Conflict-Graph)**.** *For some $3 < i < n$, let G be a bipartite graph defined by a set of vertices $V = V_1 \cup V_2$ and edges $E \subseteq V_1 \times V_2$. We define V_1 as the set of edges on the boundary of the convex hull of $p_1, \ldots p_{i-1}$. We define V_2 as the set of points that have not been processed $p_i, \ldots, p_n$. For a tuple (u, v) with $u \in V_1$ and a $v \in V_2$, let $(u, v) \in E$ if and only if the convex-hull edge u would be removed if v is added to the convex hull in the next step.*

For each $v \in V$, let $\mathcal{N}_G(v)$ denote the neighborhood of v in G. Formally,

$$\mathcal{N}_G(v) = \{\ u \in V \ \mid\ (u, v) \in E\ \}.$$

Now, our algorithm changes as follows. To solve the two subproblems (i) we simply check the neighborhood of p_i in G. If the neighborhood is empty, then p_i must be contained in the convex hull. To solve subproblem (ii), note that the set of edges that must be removed from the convex hull is exactly $\mathcal{N}_G(p_i)$. These edges must appear along the convex hull in sorted order with the vertices a and b appearing at the beginning, respectively, at the end.

It appears that the only expensive step that remains is updating the conflict graph G. We need to insert the new edges of the convex hull incident to p_i into V_1 and find the points that are in conflict with these two edges. For this, we look at the neighborhoods of the edges incident to a and b. Let C_{i-1} denote the convex hull before inserting p_i. Any point that is in conflict with the new edge (a, p_i), must have been in conflict with at least one of the two edges of C_{i-1} incident to a. Similarily, any point that is in conflict with the new edge (p_i, b), must have been in conflict with with at least one of the two edges of C_{i-1} incident to b. Indeed, a point is in conflict with an edge if an only if it lies to the opposite side of the line that supports the edge. The corresponding halfspace bounded by the line supporting the edge (a, p_i) is contained in the union of the two halfspaces corresponding to the two edges of C_{i-1} incident to a. See also the example in Figure 1 and 2. At the end of the update step we also delete the edges of $\mathcal{N}_G(p_i)$ from V_1, as well as the point p_i and their conflicts from G.

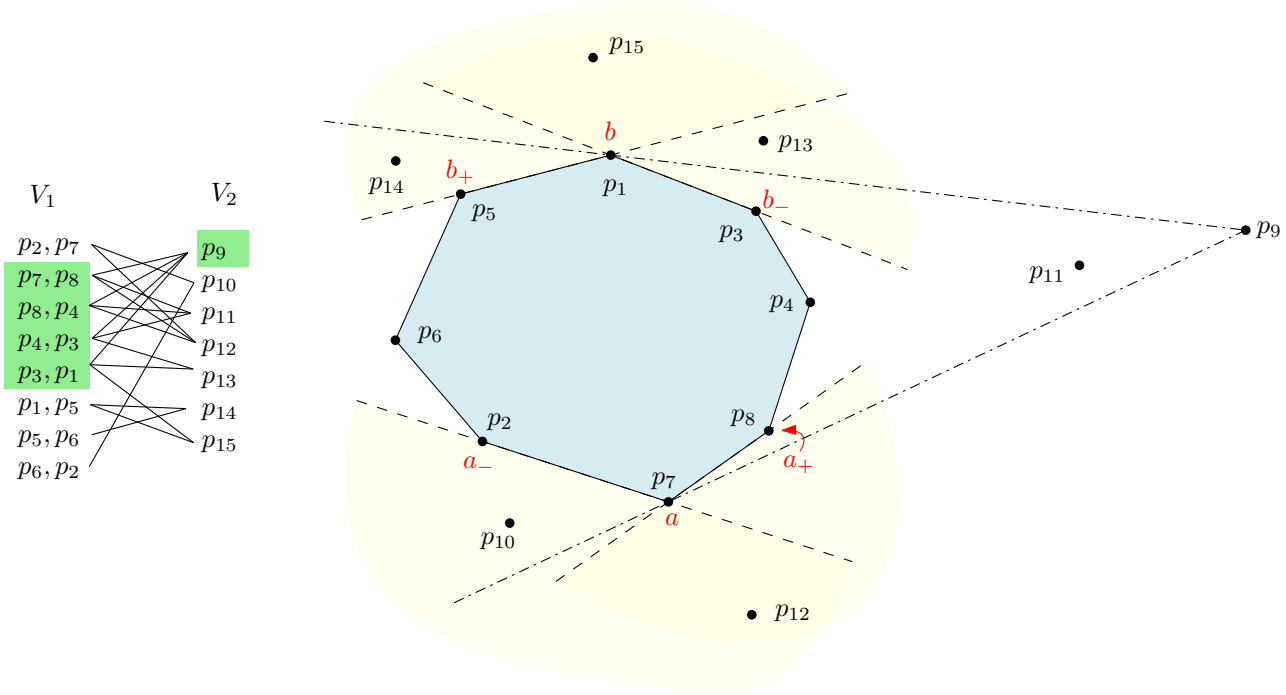

Figure 1: Before updating the conflict graph during insertion of p_9. The vertices of the conflict graph to be removed are shown with shaded green.

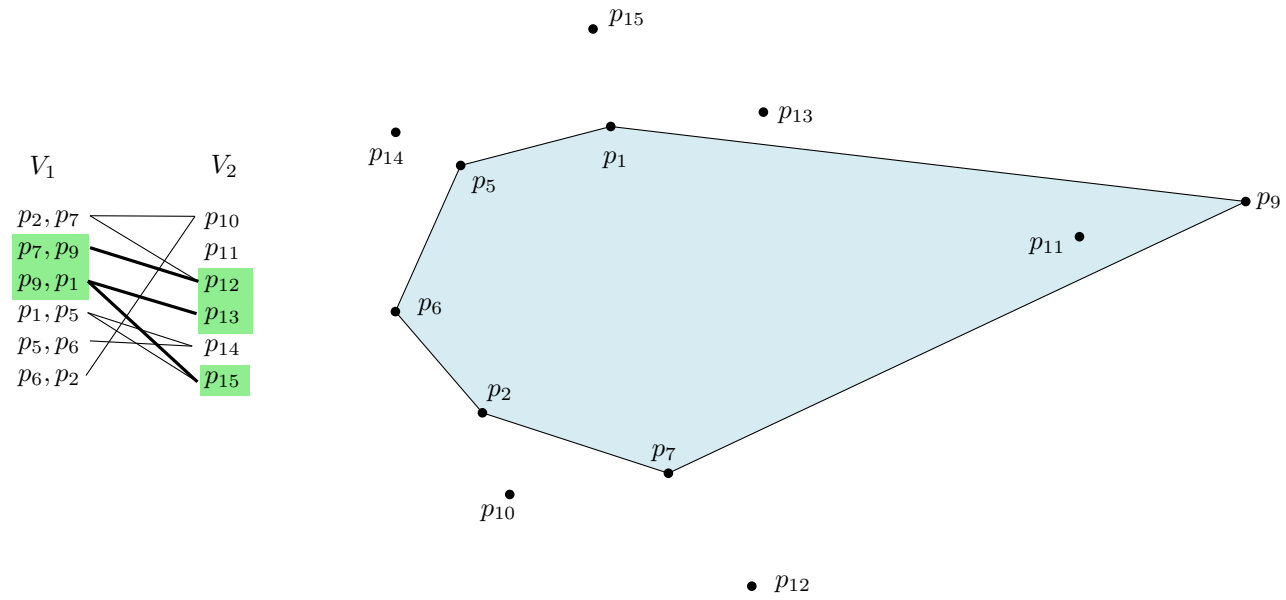

Figure 2: After updating the conflict graph during insertion of p_9. The edges of the conflict graph that were added are shown with thick lines.

Algorithm 6.1 Randomized incremental convex hulls

procedure CONVEX-HULL(Set of points P in $\mathbb{R}^2$)
  Let $p_1, \ldots, p_n$ be a random permutation of the points of P
  Let C be the (counter-clockwise, ccw) list of points on the convex hull of $\{p_1, p_2, p_3\}$
  Initialize G using the edges of C and the points $p_4, \ldots, p_n$
  for $i \leftarrow 4$ **to** n **do**
    if $\mathcal{N}_G(p_i) \neq \emptyset$ **then**
      // Consider the set of edges $\mathcal{N}_G(p_i)$ which form a chain in C
      Let a, b be first and the last point of the ccw chain of C formed by $\mathcal{N}_G(p_i)$
      UPDATECONFLICTGRAPH(G, p_i, a, b)
      Replace the chain $a, \ldots, b$ in C with a, p_i, b
    end if
  end for
  return C
end procedure

Algorithm 6.2 Subroutine to Algorithm 6.1

procedure UPDATECONFLICTGRAPH(conflict graph $G(V_1 \cup V_2, E)$, points p_i, a, b)
  $P_a \leftarrow \mathcal{N}_{\mathcal{G}}(a_-, a) \cup \mathcal{N}_{\mathcal{G}}(a, a_+)$
  Remove duplicate points in P_a
  Insert (a, p_i) into V_1
  for $p \in P_a$ **do**
    if (a, p_i) is in conflict with p **then**
      Add edge between (a, p_i) and p to E
    end if
  end for
  $P_b \leftarrow \mathcal{N}_{\mathcal{G}}(b_-, b) \cup \mathcal{N}_{\mathcal{G}}(b, b_+)$
  Remove duplicate points in P_b
  Insert (p_i, b) into V_1
  for $p \in P_b$ **do**
    if (p_i, b) is in conflict with p **then**
      Add edge between (p_i, b) and p to E
    end if
  end for
  Remove p_i from V_2
  Remove $\mathcal{N}_G(p_i)$ from V_1
  Remove all edges (u, v) from E with $u \in \mathcal{N}_G(p_i)$
end procedure

2 Analysis

We now want to analyze Algorithm 6.1. At first glance, the algorithm seems very inefficient. Note that it can happen that roughly half of the points are in conflict with roughly half of the edges of the convex hull. The algorithm stores all of these conflicts explicitly in the conflict graph G. In the worst case, this could lead to quadratic running time and space. However, we will see that in expectation the running time (and therefore also the space) is much lower, namely bounded by $O(n \log n)$. This is the average running time over all runs of the algorithm on the same input, averaged over all possible permutations chosen in line 1 of the algorithm. It is important to note that the bound, which we will show on the expected running time, will hold regardless of the distribution of input.

To analyze the algorithm, we introduce the notion of *configurations* and *killing sets*, which are an essential part of a general framework for analyzing randomized incremental constructions. Interestingly, the framework does not explicitly use any specific geometric properties of the algorithm, which is what makes it so generally applicable. It is important to note that the configurations and killing sets are not maintained explicitly by the algorithm. They are only used in the analysis. Let P be the set of n point which is the input of the algorithm.

Definition 6.2 (Configuration)**.** *A configuration Δ is a tuple (s, q, t) of three different points from P. We think of a configuration as three points appearing consecutively in the counter-clockwise order of points along the convex hull at some point in the algorithm.*

Definition 6.3 (Killing-Set)**.** *The killing-set $K(\Delta)$ of a configuration $\Delta = (s, q, t)$ is defined as follows. Let $h_{s,q}$ be the halfspace to the right of the directed line supporting the directed edge $\overline{sq}$, then*

$$K(\Delta) = (h_{s,q} \cup h_{q,t}) \cap (P \setminus \{s, q, t\})$$

We think of the killing-set as the set of points that would lead to one of the edges of the configuration being removed from the convex hull[1]. If one of the points of the killing set is added to the convex hull in the next step, the configuration is destroyed. Note that a configuration cannot coexist together with any of the points in its killing set in the convex hull.

Example 6.4. *Consider the example in Figure 1. For the configuration $\Delta = (p_2, p_7, p_8)$, the killing set is $K(\Delta) = \{p_9, p_{10}, p_{11}, p_{12}\}$. Since p_9 is added to the convex hull in the next step, the configuration Δ is destroyed. The killing-set should not be confused with the conflict graph.*

We can now proceed to analyze the randomized incremental construction for computing convex hulls in the plane.

Theorem 6.5. *Algorithm 6.1 with Algorithm 6.2 as a subroutine computes the convex hull of a set P of n points in the plane in $O(n \log n)$ expected time, where the expectation is with respect to the random permutation used by the algorithm.*

Proof. The algorithm maintains the invariant that at the end of each for-loop, the convex hull of the points processed so far is stored in sorted order in C. This can be shown by induction and this shows correctness of the algorithm.

The main part of the proof is to bound the running time. Line 2 of the algorithm can be done in $O(n)$ time. Line 3 takes constant time. Line 4 can be done in $O(n)$ time by scanning the input for each edge of the convex hull, since for any fixed edge and fixed point of the input we can evaluate in $O(1)$ time if they are in conflict with each other.

[1] This is where it is convenient that we assume general position: no three points of P are collinear. Without this assumption, we would have to be careful about points that are collinear with s and q, or with q and t.

The running time is dominated by the execution of the main for-loop. Assume that the point does not lie inside the convex hull (otherwise it takes $O(1)$ time). In order to determine the two vertices a and b at the end of the convex chain, we can walk along C from an arbitrary point in $\mathcal{N}_G(p_i)$. This can be done by storing a pointer from every edge in V_1 to its corresponding edge in C. If implemented in this way, lines 8 and 9 take time linear in the size of the set $\mathcal{N}_G(p_i)$, which is equal to the number of edges which are removed from the convex hull in the current iteration of the for-loop. Therefore, we can bound the total time spent on lines 8 and 9 over all executions of the for-loop by the number of edges *created* during the entire course of the algorithm. A single execution of the for-loop adds at most 2 edges to the convex hull, and there are less than n iterations of the main for-loop. Thus, this can be bounded by $O(n)$.

It remains to bound the time spent on executions of the line 10, which calls Algorithm 6.2 to update the conflict graph. We assume that the conflict graph stores an adjacency list for each edge that is sorted by the index of the points in P. This way, the union operation on two neighborhoods can be done in time linear in the total size of these neighborhoods. Also, if edges are added to the new adjacency list (in lines 7 and 15) in this sorted order, they are automatically sorted. If the union operation is implemented in this way, then the running time of this subroutine up to line 18 is dominated by the size of the sets P_a and P_b, which can be upper-bounded by the total size of the two killing sets $K(a_-, a, a_+)$ and $K(b_-, b, b_+)$. We can bound the expected total amount of time spent on this over all executions of the main for-loop in Algorithm 6.1 by the total expected size of the killing sets of configurations that appear on the convex hull at any stage of the algorithm. We prove below in Lemma 6.6 that this is in $O(n \log n)$.

The total number of vertices and edges removed from the conflict graph in lines 18-20 in calls to Algorithm 6.2 is bounded by the total number of vertices and edges created in this graph, which is also bounded by the total size of killing sets of configurations that are created during the course of the algorithm. Therefore, the expected time spent on deletions is also bounded by $O(n \log n)$ time by Lemma 6.6.

This proves the bound on the expected running time. □

Lemma 6.6. *Consider the execution of Algorithm 6.1 with Algorithm 6.2 as a subroutine for computing the convex hull of n points. Let $\mathcal{D}_i$ be the set of at most 3 configurations newly created in C in line 9 when adding point p_i, for $4 \leq i \leq n$, and let $\mathcal{D}_3$ be the set of 3 configurations in C created in line 3. It holds that*

$$\mathbf{E}\left[\sum_{i=3}^{n-1} \sum_{\Delta \in \mathcal{D}_i} \big|K(\Delta)\big|\right] \in O(n \log n)$$

where the summation is over all configurations that appear on the convex hull at any stage of the algorithm.

Proof. Let $\mathcal{W}_i$ denote the set of configurations that are destroyed in C in line 9 during the addition of point p_i. We start by determining the expected number of configurations that are created when adding a point p_i and are destroyed in the next round, when the next point p_{i+1} is added. Fix a configuration $\Delta \in \mathcal{D}_i$ and consider the probability that $\Delta \in \mathcal{W}_{i+1}$. This happens if and only if the next point p_{i+1} is contained in the killing set $K(\Delta)$. The probability of this event is therefore $\big|K(\Delta)\big|/(n-i)$, since we can think of the point p_{i+1} being chosen uniformly at random from the $n-i$ remaining points in P that have not been processed, yet. Therefore,

$$\mathbf{E}\left[\big|\mathcal{D}_i \cap \mathcal{W}_{i+1}\big|\right] = \mathbf{E}\left[\sum_{\Delta \in \mathcal{D}_i} \frac{\big|K(\Delta)\big|}{n-i}\right] = \frac{1}{(n-i)} \cdot \mathbf{E}\left[\sum_{\Delta \in \mathcal{D}_i} \big|K(\Delta)\big|\right] \tag{1}$$

where the last inequality follows from the linearity of expectation.

Now, fix a configuration $\Delta \in \mathcal{W}_{i+1}$ and look "backwards": What is the probability that Δ was only created in round i? This happens if and only if one of the 3 points defining the configuration Δ was the last point added to the convex hull, that is p_i. The probability of this event is $3/i$. Indeed, we can think of the random permutation being generated "backwards", by choosing the ith point uniformly at random among the i points that remain after having chosen the set $p_n, p_{n-1}, \ldots, p_{i+1}$. Now, we know that all three points defining the configuration Δ must be among the i points remaining when choosing p_i, since Δ must be present in C before it can be destroyed. Therefore, the probability is $3/i$. This implies

$$\mathbf{E}\left[\left|\mathcal{D}_i \cap \mathcal{W}_{i+1}\right|\right] = \mathbf{E}\left[\left|\mathcal{W}_{i+1}\right|\frac{3}{i}\right] = \frac{3}{i} \cdot \mathbf{E}\left[\left|\mathcal{W}_{i+1}\right|\right] \tag{2}$$

where, again, the last inequality follows from the linearity of expectation.

Since the left hand side is the same in the above equations, we obtain

$$\mathbf{E}\left[\sum_{\Delta \in \mathcal{D}_i} \left|K(\Delta)\right|\right] = \frac{3(n-i)}{i} \cdot \mathbf{E}\left[\left|\mathcal{W}_{i+1}\right|\right] \leq \frac{3n}{i} \cdot \mathbf{E}\left[\left|\mathcal{W}_{i+1}\right|\right] \tag{3}$$

By linearity of expectation, we can write

$$\mathbf{E}\left[\sum_{i=3}^{n-1}\sum_{\Delta \in \mathcal{D}_i} \left|K(\Delta)\right|\right] = \sum_{i=3}^{n-1}\mathbf{E}\left[\sum_{\Delta \in \mathcal{D}_i} \left|K(\Delta)\right|\right] \leq 3n\sum_{i=3}^{n-1}\frac{\mathbf{E}\left[\left|\mathcal{W}_{i+1}\right|\right]}{i} = 3n \cdot \mathbf{E}\left[\sum_{i=3}^{n-1}\frac{\left|\mathcal{W}_{i+1}\right|}{i}\right] \tag{4}$$

Now, consider the permutation of the input points fixed and let $\tau_-(\Delta) := i$ if $\Delta \in \mathcal{W}_i$, so that $\tau_-(\Delta)$ corresponds to the round of the algorithm at which a configuration is destroyed. Simlarly, define $\tau_+(\Delta) := i$ if $\Delta \in \mathcal{D}_i$, the round in which the configuration is created. Since $\tau_+(\Delta) \leq (\tau_-(\Delta) - 1)$ for any Δ, we can write

$$\sum_{i=3}^{n-1}\frac{\left|\mathcal{W}_{i+1}\right|}{i} = \sum_{i=3}^{n-1}\sum_{\Delta \in \mathcal{W}_{i+1}}\frac{1}{i} = \sum_{\Delta}\frac{1}{(\tau_-(\Delta)-1)} \leq \sum_{\Delta}\frac{1}{\tau_+(\Delta)} = \sum_{i=3}^{n-1}\sum_{\Delta \in \mathcal{D}_i}\frac{1}{i} = \sum_{i=3}^{n-1}\frac{\left|\mathcal{D}_i\right|}{i} \tag{5}$$

Now, since $\left|\mathcal{D}_i\right| \leq 3$, we can bound the right hand side above using the $(n-1)$th Harmonic number

$$\sum_{i=3}^{n-1}\frac{\left|\mathcal{D}_i\right|}{i} < \sum_{i=1}^{n-1}\frac{3}{i} = 3H_{n-1} \tag{6}$$

Since this is a worst-case bound over all permutations of the input, it is also an upper bound on the expectation. Therefore, we can now plug this into (5) obtaining

$$\mathbf{E}\left[\sum_{i=3}^{n-1}\sum_{\Delta \in \mathcal{D}_i} \left|K(\Delta)\right|\right] \leq 9nH_{n-1} \tag{7}$$

and this implies the claimed bound. □

References

- Mark de Berg, Otfried Cheong, Marc van Kreveld, Mark Overmars. Computational Geometry— Algorithms and Applications. Third Edition. Springer. Chapters 9 and 11.