CS 388R: Randomized Algorithms, Fall 2021 November 4, 2021

Lecture 20: Computational Geometry II

Prof. Eric Price *Scribe: Jiaxun Cui, Rojin Rezvan*

NOTE: THESE NOTES HAVE NOT BEEN EDITED OR CHECKED FOR CORRECTNESS

1 Overview

In the last lecture we finished up Markov Chains and introduced closest pair problem. In this lecture we study two geometry problem (1) Find a *convex hull* for a set of points and (2) Given a point and intersecting lines, find which region is it in.

2 Convex Hull

Problem Setting

We are given a set of n points in the plane, and we are looking for the convex hull of the points, i.e. the smallest convex polygon that contains all the points (in its interior and on its edges, shown in Figure 1). For this algorithm, we will assume that no three points are co-linear.

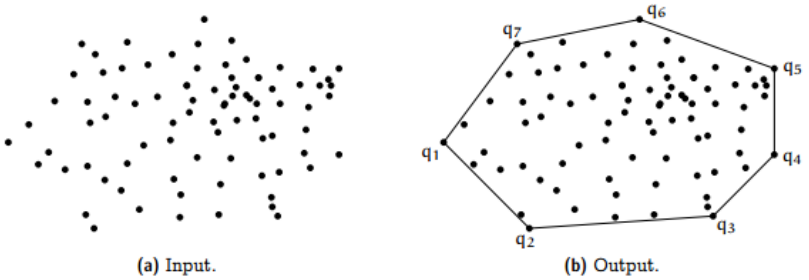

Figure 1: A demonstration of convex hull

Deterministic algorithms

- **Gift-wrapping** The algorithm starts by selecting the point p_0 of minimal y-coordinate as a vertex of the convex hull. Then the algorithm scans in a counterclockwise fashion to find a point p_{i+1} of minimal angle from the half-plane formed by $p_{i-1}p_i$. This algorithm takes $O(nk)$ time for k points of the convex hull, or $O(n^2)$ in the worst case.
- **Graham Scan** achieves $O(n \log n)$ time complexity.

Randomized algorithm

We present a randomized algorithm that achieves $O(n \log n)$ time complexity in expectation.

1. Start with three points as your convex hull.

2. When you get a previously unseen point p, check whether it is in your convex hull. If yes, skip it; If not, update the convex hull by removing all edges that are "visible" to the new point and replace them with two new edges. Formally, a point q is visible to p if the line segment $\overline{pq}$ does not intersect the convex hull. All of the visible edges will be adjacent to each other, as shown in Figure 1(a). The updated convex hull is shown in Figure 1(b).

The key idea is to randomized the order of points to be added so as to reduce the insertion time.

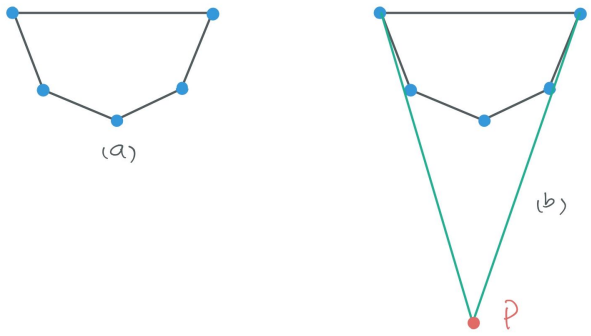

Figure 2: An update step of convex hull. Time = T(Finding an edge to delete if any) + O(n)

Time Complexity

At each step, we must find a visible edge (if any). The other visible edges, if any, are adjacent, and visibility of adjacent edges can be checked in $O(1)$ time. We remove the visible edges and add two new edges.

Since at most two new edges are added to the convex hull at each step, edge removal takes $O(n)$ total time over all steps. What remains to discuss is how quickly we can find a visible edge at each step of the algorithm.

How to maintain the knowledge As Figure 3 shows, we pick some point c inside the initial triangle. For every point p, use some data structure to keep track of the points p such that $\overline{pc}$ intersects the convex hull and the edge that $\overline{pc}$ intersects.

$$a[p] \coloneqq \text{ edge in hull intersecting } \overline{pc}, \text{ if any}$$

which takes $O(n)$ to initialize. To maintain this data structure, when an edge is removed from the convex hull, we must update all pointers to this edge, which takes $O(1)$ time per point that referred to this edge. Therefore, the time complexity is $O(n) + O(\#$ updates to pointers).

Randomized Order

If we add points to the convex hull in some unlucky sequence, as shown in Figure 4, we will end up with $\Theta(n^2)$ changes for each insertion. However, we hope the expected number of updates to pointers is small ($O(n \log n)$) if the points are added in a **random order**.

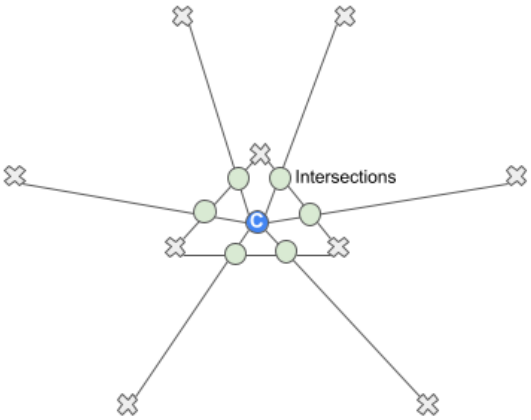

Figure 3: Find the intersections of the point-center and convex hull. If an edge on convex hull has intersection, then this edge is to be deleted

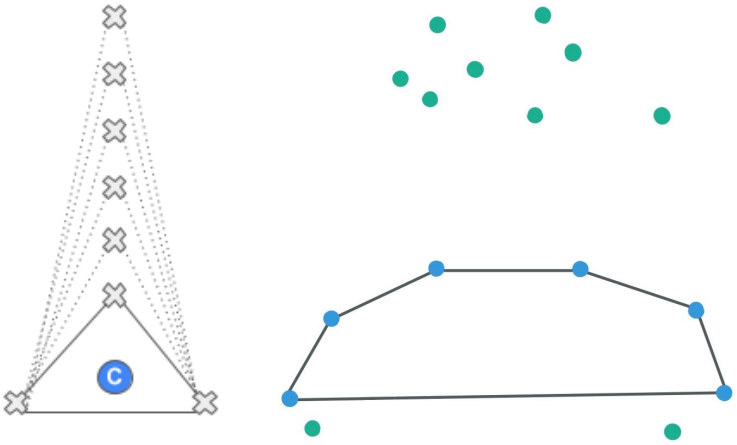

Figure 4: Bad cases of sequence of random points, the number of changes can be as bad as $\Theta(n^2)$

The idea of proof is as follows

$$\begin{aligned} \mathbb{E}[\ \#\text{ of changes }] &\leq \max_p \mathbb{E}[\#\text{ of changes to one } p] \\ \mathbb{E}[\ \#\text{ of changes to } p\ ] &= \sum_{i=1}^{n} Pr[\ p \text{ changes in } i^{th} \text{ insertion }] \end{aligned} \tag{1}$$

At step i, i points have been inserted. It is hard to know the probability that the next update modifies $a[p]$. So we use a trick called **Backwards Analysis**. We know that the probability that last update modified $a[p]$ is $\frac{2}{i}$, because when a point x is removed from the convex hull, we update a point y if $\overline{yc}$ intersects one of the two edges that involve x.

$$Pr[\text{ Last update modified } a[p]\ ] = \frac{2}{i}$$

So the expectation of number of updates for a pointer is

$$\mathbb{E}[\ \#\text{ updates }] = \sum_i \frac{2}{i} = O(\log n)$$

Since the total number of pointers in our data structure is at most n, we conclude that the total expected runtime of the algorithm is

$$\mathbb{E}[\text{Time}] = O(n) + n\sum_{i=1}^{n}\frac{2}{i} = O(n\log n).$$

3 Planar Point Location

Problem setting

For the 2D point location problem, consider n lines, $V = \{l_i\}_{i\in[n]}$ in a plane. Given a query point $x \in \mathbb{R}^2$, find its enclosing region out of the $\Theta(n^2)$ possible regions cut by the n lines in V, and more generally, find the enclosing triangle in the triangulation of the plane based on V. Our goal is to answer this query in $O(\log n)$ time.

A Randomized Algorithm

First Define Triangulation

$$T(L) \coloneqq \text{ Triangulation of regions}$$

We build a tree structure via the recursive triangulation of plane using the idea of divide-and-conquer as following:

1. Sample a set $R \subset V$ that contains $r = O(1)$ lines randomly
2. Compute the triangulation of R, denoted as $T(R)$;
3. For each triangle, find intersecting lines and build recursive solution to those lines; if not **good**, resample R.

Definition 1. *We say that a region R is **good** if the number of intersecting lines in $T(R)$ is at most $O(\frac{n\log r}{r})$*

Lemma 2. *R is good with probability $\frac{3}{4}$*

Run time analysis

Query Time Assuming that all R's selected by the algorithm are "good" (which can be achieved with good probability as Lemma 2 indicates), then a query will take

$$Q(n) = r^2 + Q\left(O\left(\frac{n}{r}\log r\right)\right) \le O(1) + Q\left(O\left(\frac{n}{2}\right)\right) = O(\log n)$$

time. That is, on each level of triangulation formed by R, one needs to check up to r^2 possible regions to find an enclosing region for the query point. Each level of triangulation decreases the number of lines from $|V| = n$ to $|V'| = n \cdot \frac{\ln r}{r}$.

Construction Time Meanwhile, building the data structure requires

$$T(n) = Poly(r) + r^2 n + r^2 T\left(O\left(n\frac{\ln r}{r}\right)\right)$$

time, where $r^2 n$ time is required to check the intersections of the n lines in V within the r^2 regions cut by R. Then the algorithm recurs on the r^2 regions cut by R where $|V'| = O\left(n\frac{\ln r}{r}\right)$ for each region. To express $T(n)$ explicitly in terms of n, we first expand the recursion term,

$$\begin{aligned}
T(n) =& r^2\left[n + r^2 \cdot \left(\frac{n\ln r}{r} + T\left(n\left(\frac{\ln r}{r}\right)^2\right)\right)\right] \\
& \left(n(r\log r) \text{ is bigger than } n\right) \\
\asymp & \left(r^2\right)^{\log_{\frac{\log r}{r}} n} \\
=& \, n^{\log_{\frac{\log r}{r}} r^2} \\
=& \, n^{2+O_r(1)} \text{ where } O_r(1) = \Theta\left(\frac{\log\log r}{\log r}\right)
\end{aligned}$$

Space Similarly, we cam find the space complexity

$$\begin{aligned}
S(n) &= r^2 + r^2 S(\frac{n\log r}{r}) \\
&= \Theta((r^2)^{\log_{\frac{r}{\log r}} n}) \\
&= n^{2+O_r(1)}
\end{aligned}$$

Next we prove the lemma 2

Proposition 3. *R is good with probability $\frac{3}{4}$*

Proof. Given V, R with $|V| = n$, $|R| = r = O(1)$, denote S as all intersections of V such that $|S| = O(n^2)$, and denote Δ as a set of all triplets of S (n^6 triplets), δ is one of the triplet, (s_1, s_2, s_3) where $s_i \in S$. The idea is to show that if many lines in V intersects in the region defined by a given δ, then the probability that δ forms a triangle in $T(R)$ is small. To show this, we consider the case where at least $k = |\delta \cap V|$ lines intersect in the region enclosed by δ,

$$\begin{aligned}
&\Pr[\delta \text{ lies in } T(R)] \\
=&\Pr[\text{end points formed in } R] \cdot \Pr[\text{no interior lines of } \delta \text{ is in } R \mid \text{end points lie in } R]
\end{aligned}$$

where for a fixed triplet δ in S, each of the three end points is formed by two lines selected by R. Thus, the probability that none of the k lines intersecting in δ is in R for a fixed δ is the probability that all selections of lines in R, except for those forming the end points (at most 6 lines), avoid the k intersecting lines in δ. That is,

$$\begin{aligned}
&\Pr[\text{none of the lines intersecting in } \delta \text{ is in } R \mid \text{end points of } \delta \text{ lie in } R] \\
\le& \left(1 - \frac{k}{n}\right)^{r-6} \le \exp\left(-\frac{k(r-6)}{n}\right) \\
\le& \exp\left(-\frac{kr}{2n}\right) \quad \forall\, r \ge 12.
\end{aligned}$$

Therefore, by the union bound

$$
\begin{aligned}
&\Pr[R \text{ is not "good"}] \\
&= \Pr[\text{ any } \delta \text{ with at lest } k \text{ lines intersecting in the enclosed region forms a triangle in } T(R)] \\
&\leq \exp\left(-\frac{kr}{2n}\right)(\ \#\ \text{triplets of end points in } R) \\
&\leq \exp\left(-\frac{kr}{2n}\right) \cdot \sum_{\delta \in \Delta} Pr[\text{ end points of } \delta \text{ lie in } R\ ] \\
&= \exp\left(-\frac{kr}{2n}\right) \cdot \mathbb{E}[\ \#\ \text{of triplets in } R\ ] \\
&= \exp\left(-\frac{kr}{2n}\right) \cdot \binom{\binom{r}{2}}{3} \\
&\quad \Big(\text{Each intersection in } S \text{ is formed by 2 out of } r \text{ lines in } R\Big) \\
&\quad \Big(\text{Each triplet is formed by 3 intersection points}\Big) \\
&\lesssim r^6 \cdot \exp\left(-\frac{kr}{2n}\right) \leq \frac{1}{r} < \frac{1}{4} \text{ if } k = \tfrac{n}{r} 6 \log(47),\ r \geq 12
\end{aligned}
$$

□