CS 388R: Randomized Algorithms, Fall 2019 November 26, 2019

Lecture 26: Computational Geometry

Prof. Eric Price *Scribes: Anna Yesypenko, Yijun Dong*

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

1 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). For this algorithm, we will assume that no three points are colinear.

Deterministic algorithms A deterministic algorithm is known as the "gift-wrapping algorithm". 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 in the interior of the convex hull, or $O(n^2)$ in the worst case. There is another deterministic algorithm known as the Graham scan which achieves $O(n \log n)$ time.

Randomized algorithm We present a randomized incremental algorithm that achieves $O(n \log n)$ runtime.

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 not, update the convex hull by removing all edges that are "visible" to the new point and replacing 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).

Runtime 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.

We will pick some point c inside the initial triangle. At all times, we keep track of the points x such that $\overline{xc}$ intersects the convex hull and the edge that $\overline{xc}$ intersects. 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 runtime is $O(n) + O(\#$ updates to pointers$)$.

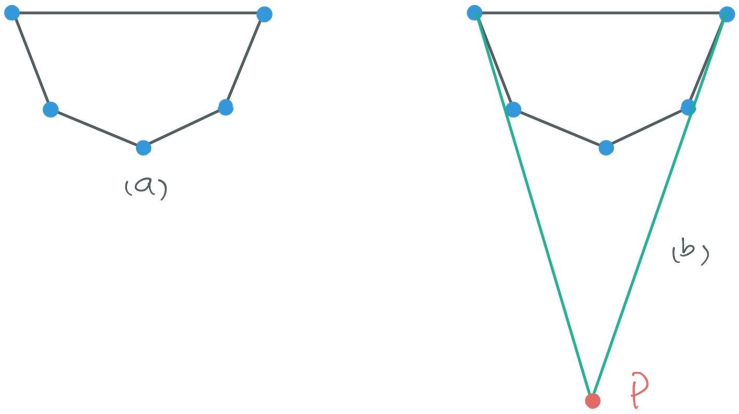

Figure 1: An update to the convex hull, after adding point p

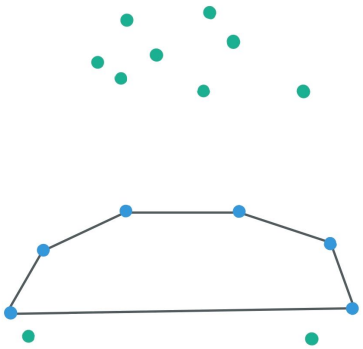

Figure 2: An unlucky sequence of random points, where with large probability, a randomly chosen green point will cause many pointer updates to the data structure

Backwards Analysis We would like to say that the expected number of updates to pointers is small if the points are added in a random order. In particular, we would like to say that the expected number of updates in step i is n/i, but that may not happen if we add points to the convex hull in some unlucky sequence, as shown in Figure 2.

We instead use a different technique called backwards analysis. At step i, we have seen points $S = \{p_1, \ldots, p_i\}$ and generated a convex hull H. For any set S,

$$\mathbb{E}_{x \in S}\left[\#\text{ updates to turn } S \setminus \{x\} \text{ into } S\right] = O\left(\frac{n}{i}\right).$$

When $x \in S$ is removed from the convex hull, we update a point y if $\overline{yc}$ intersects one of the two edges that involve x. Let $l(e)$ for $e \in H$ be the number of pointers associated with an edge in the convex hull, i.e. the number of points y such that $\overline{yc}$ intersects $e \in H$. The point $x \in S$ is chosen with equal probability. Therefore, the expected number of updates is

$$\mathbb{E}_{x \in S}\left[\#\text{ updates to turn } S \setminus \{x\} \text{ into } S\right] = \frac{1}{i}\sum_{e \in H} 2l(e) \leq \frac{2n}{i},$$

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

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

2 Point Location

Problem setting For the 2D point location problem, consider n lines, $V = \{l_i\}_{i\in[n]}$ in a plane. We want to build a data structure such that given a query point $x \in \mathbb{R}^2$, one can find its enclosing region out of the $O(n^2)$ 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.

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

1. picking a set $R \subset V$ of $|R| = r = O(1)$ lines randomly and build the triangulation of R, denoted as $T(R)$;
2. triangulate each region (i.e., triangle) $A \in T(R)$ recursively: find all lines in V that intersect in A, denoted as V'; randomly pick $R' \subset V'$ with $|R'| = O(1)$ and triangulate A with respect to R'.

Definition 1. *We say that a region $A \in T(R)$ is "good" if the number of intersecting lines in A is small,*

$$|\textit{intersecting lines in } A| = O\Big(\frac{n}{r}\ln r\Big)$$

Definition 2. *We say that a random selection of lines, R, is good if all triangles $A \in T(R)$ are good.*

Lemma 3. *If r lines are selected randomly from n lines in V to form R, then with probability $\frac{3}{4}$, R is good, i.e., each of the $O(r^2)$ triangular regions $A \in T(R)$ encloses $O\Big(\frac{n}{r}\log r\Big)$ intersections.*

Proof. Given V, R with $|V| = n$, $|R| = r = O(1)$, denote S as all intersections of R such that $|S| = O(r^2)$, and denote Δ as a triplet in S, (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 lines intersect in the region enclosed by Δ,

$$\begin{aligned}&\Pr[\Delta \text{ forms a triangle in } T(R)]\\ =&\Pr[\text{end points of } \Delta \text{ lie in } R]\cdot\\ &\Pr[\text{none of the lines intersecting in } \Delta \text{ is in } R \mid \text{end points of } \Delta \text{ lie in } R]\end{aligned}$$

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

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

when we take $r \geq 12$ as required before, and

$$k=\frac{2n}{r}\cdot(7\ln r)=O(\frac{n}{r}\ln r)$$

□

Run time analysis Assuming that all R's selected by the algorithm are "good" (which can be achieved with good probability as Lemma 3 indicates), then a query will take

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

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 $|V'|=n\cdot\frac{\ln r}{r}$.

Meanwhile, building the data structure requires

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

time, where r^2n 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 $|V'|=O\Big(n\frac{\ln r}{r}\Big)$ for each region. To express $T(n)$ explicitly in terms of n, we first expand

the recursion term,

$$\begin{aligned}
T(n) =& r^2 n + r^2\left[r^2 \cdot \frac{n\ln r}{r} + r^2 \cdot T\left(O\left(n\frac{\ln r}{r}\right)\right)\right] \\
=& n \cdot \left[r^2 + r^3 \ln r + \cdots + r^{2d+1}\ln^{2d-1} r\right] \\
& \left(T(n) \text{ is dominated by the leaf term: } d = \log_{O\left(\frac{r}{\ln r}\right)} n\right) \\
=& n \cdot O\left(r^{2d}\right) = n \cdot O\left(r^{2\log_{O\left(\frac{r}{\ln r}\right)} n}\right) \\
=& n \cdot n^{2\log_{O\left(\frac{r}{\ln r}\right)} r} \\
=& n \cdot n^{2\left(1+\frac{\log\log r}{\log\frac{r}{\log r}}\right)} \\
=& n^{3+2\frac{\log\log r}{\log r - \log\log r}} \\
T(n) =& n^{3+O_r(1)}
\end{aligned}$$