Advanced algorithms
Convex Hull Notes

Søren Dahlgaard

June 15, 2011

1 Disposition

1. Convex hull? Minimum convex set.
2. Graham's Scan + Jarvis march, general idea
3. Lower bound with sorting + intuition it's wrong.
4. Marriage before conquest
   (a) Upper/lower identical.
   (b) General procedure: median, bridge, recurse.
   (c) Finding the bridge. Intuition of slope and removing
   (d) Time complexity
5. Chan's algorithm
   (a) Idea. Merge graham and jarvis.
   (b) Make incremental guesses at h and stop if we're wrong.
   (c) Time complexity.

2 Convex hull

Given a set P of points in the plane E^2 (or space E^3 and more, but we will only look at the 2D example) we want to find the smallest convex set $C \subseteq P$ such that every point in P is in C. We can see it as having a board with nails and wrapping a rubber band around it.

A convex set is a set such that if we pick any point a, b in it, then the line from a to b also lies entirely within the set. Or more exactly. For any t in the interval $[0..1]$ we must have

$$(1-t)x + ty$$

is in the set.

We require that $|P| \geq 3$. For most of the algorithm we will also require that no three points are colinear and that no points have the same x or y coordinates. These are reasonable assumptions for the continuous world. They just add a lot of special cases which is normal with computational geometry.

2.1 Graham's scan

The idea is to start with a point p_0 that has to be in $\mathrm{CH}(P)$. We then iterate through the points of P in order of their polar angle with respect to p_0. We know that p_1 and p_m must also be in C. We keep the current "convex hull" in a stack S and for each p_i we do:

1. While p_i does not make a left turn with the two top poins of S we pop the top element.
2. Add p_i to S.

This clearly runs in $O(n \lg n)$ because we sort the points and then each point is at most pushed and popped once.

Correctness can be shown with the following invariant:

At the start of the ith iteration, S contains $CH(\{p_0, p_1, \ldots, p_{i-1}\})$ in sorted order.

2.2 Jarvis March

Jarvis's March uses "gift wrapping". Again we pick a point that has to be in the hull (the bottom-most, left-most point). We then act like taking some gift paper and stretch it all the way to the right. We then start going up until it touches another point. In other words, we pick the point p_i that maximizes the angle $\angle p_{i-2}p_{i-1}p_i$. This can easily be implemented in $O(nh)$.

To optimize this we might split the run into two. We continue wrapping until we reach p_k with the maximum y-value. Then we have constructed the right-chain (assuming no identical y-values). Do the opposite for the left-chain.

2.3 Lower bound

Some people suggested that $n \lg n$ might be a lower bound because we could otherwise use it to sort n numbers by finding the convex hull of (a_i, a_i^2) for all these n points. While this is true, we can still make algorithms in time $O(nh)$ like Jarvis' March, where h can be less that $\lg n$, surely. Note that in the sorting case we have $h = n$, therefore we might believe that it can be done in $O(n \lg h)$, which we provide to algorithms for below.

2.4 Marriage-before-conquest

Marriage-Before-Conquest is actually a modification of divide-and-conquer. Before solving the subproblems recursively we look at how they will combine in the end and remove a bunch of the problem (in this case points) based on this.

The idea is to compute the top of the hull and the bottom of the hull seperately. We can combine this in constant time adding at most two vertical edges (if the left/rightmost points have different y-values).

The algorithm runs as follows:

1. Find the median x-value of the points and create the line $L = \{(x, y) : x = m\}$, where m is the median x-value.
2. Find the part, b, of the top hull crossing this line. (The bridge).

3. Delete all points under b.

4. Recursively solve P_l and P_r the points left and right of the bridge.

The hard part here is to find the bridge. We use the following idea: A supporting line is a non-vertical straight line that contains at least one point of P such that no other points of P are above the line.

We use a couple of lemmas:

Let $p, q \in P$. If $x(p) = x(q)$ and $y(p) < y(q)$ then obviously p cannot be a bridge point.

Let $x(p) < x(q)$. If the slope $s_{pq} > s_b$ then p cannot be a bridge point. proof: Then q would be above the bridge supporting line, which is a contradiction. The reverse case $s_{pq} < s_b$ implies that q isn't a bridge point.

Let h be a supporting line. $s_h < s_b$ iff h only contains points to the right of L. $s_h = s_b$ iff h has a point on either side.

We now find the bridge like so:

1. Split P into $|P|/2$ pairs of points.

2. Find the median slope of the $|P|/2$ line segments.

3. Translate the slope to make a supporting line (can be done in $O(|P|)$).

4. If this line h only contains points right of L we have $s_h < s_b$ and thus all of the $|P|/2$ lines pq with slope $s_{pq} < s_h < s_b$ can not have q as bridge points. This will eliminat $|P|/4$ points because we chose the median slope.

5. Do this recursively.

The running time of the bridge function is:

$$T(n) = T(3n/4) + O(n) = O(n)$$

This gives a total running time of:

$$T(n, h) = \max_{h_l + h_r = h} \{T(n/2, h_l) + T(n/2, h_r)\} + O(n)$$

Because we have at most $n/$ points to the left of the median and $n/2$ points to the right. The base case here is $T(n, h) = O(n)$ when $h = 2$. This gives us that the worst case of the above recursion is when $h_l = h_r = h/2$. We can show that $T(n, h) = n \log h$ using the substitution method:

$$\begin{aligned}
T(n, h) &\leq \max_{h_l + h_r = h} \left\{ c\frac{n}{2} \log h_l + c\frac{n}{2} \log h_r \right\} + O(n) && (1)\\
&= \max_{h_l + h_r = h} \{\log(h_l h_r)\} + O(n) + c\frac{n}{2} && (2)\\
&\leq O(n) + c\frac{n}{2} \log(h/2)^2 && (3)\\
&= n \log h && (4)
\end{aligned}$$

2.5 Chan's algorithm

The big problem about the marriage-before-conquest algorithm is that it relies on finding the median in linear time. This is very costly in practice thus we strive to find a better algorithm.

The idea of Chan's algorithm is to preprocess the points and splitting them into n/m convex hulls which we compute with graham's scan in $n/m \cdot m \lg m = n \lg m$. If $m = h$ this is obviously $n \lg h$.

With this preprocessing we can do a modified version of jarvis' march. Finding the point p_i that maximizes $\angle p_{i-2}p_{i-1}p_i$ can now be done in $O(n/m \lg m)$ by finding the point for each of the convex hull with binary search and taking the best among these. This gives a total running time of:

$$T(n) = O(n/m \cdot m \lg m + h((n/m) \lg m)) = O(n \lg m + h((n/m) \lg m) = O(n(1+h/m) \lg m)$$

Now, if we pick $m = h$ this will be $O(n \lg h)$ like the mbc algorithm. If we pick m too large it will approach $n(1+0) \lg n$, and if we pick it too low it will approach $O(n(1+h) \lg 1)$. The idea is thus to make a increasing guesses at h (our guess is H) and m (which we want to be h). If the algorithm doesn't calculate a hull in H steps we have guessed too low and we stop it before it runs for too long. Thus we make guesses:

$$(2^{2^1}, 2^{2^2}, 2^{2^3}, \ldots)$$

until $2^{2^t} \geq h$. We set $m = H = 2^{2^t}$. Thus the tth iteration takes

$$T(n) = O(n(1 + H/H) \lg H) = O(n/lgH) = O(n2^t)$$

Thus the total running time is:

$$\begin{aligned} T(n) &= \sum_{t=1}^{\lceil \lg \lg h \rceil} O(n2^t) && (5) \\ &= O(n2^{\lceil \lg \lg h \rceil + 1}) && (6) \\ &= O(n \lg h) && (7) \end{aligned}$$

We might improve this algorithm by picking a better m or H value, removing the points found to be inside C_i when we run graham's scan or more ways.