CMSC 754: Spring 1996 Samir Khuller

Kirkpatrick-Seidel's Ultimate Convex Hull Algorithm

This algorithm was published in *SIAM Journal on Computing* (1986).

The basic idea is to develop a divide and conquer algorithm that runs in time $O(n \log h)$ where h is the size of the hull. For simplicity, we will concentrate on only finding the *upper* chain of the convex hull, which connects the leftmost point with the rightmost point.

We will (as before) divide the points in S into two sets S_L and S_R of approximately equal size by using a vertical separating line. A few key observations are in order.

1. We can compute the upper supporting line *before* we compute the convex hulls of S_L and S_R, since these are functions of the point set, and not of the convex hulls of S_L and S_R. (Ofcourse, the computation will be a little trickier since we cannot use the convex hull to find the bridge, as we did in the regular divide and conquer algorithm.)

2. We can delete all points that are immediately "below" the bridge. Specifically, if the bridge joins points p and q, we can delete all points with x co-ordinates between p and q. We recurse on the remaining points to find the upper chain of S'_L and S'_R (remaining sets of points). This will potentially reduce the number of points we recurse on.

There are a few things we need to do to implement the above ideas. The first is that we cannot afford to presort the points before starting the algorithm, so at each divide step we have to invoke a linear time median finding algorithm to split S into equal pieces. (In practice one can use a randomized algorithm to find an approximate median quickly.)

The second hurdle is to find the bridge in $O(n)$ time. This will be done by a method called "prune and search". In this method, we spend cn time, and we either find what we are looking for, or (for example) delete half the points from consideration, and recurse. The running time for such a scheme is

$$cn + c\frac{n}{2} + c\frac{n}{4} \ldots c = O(n).$$

Observe that this actually works, so long as we eliminate a constant fraction of the points.

Before we get into details about how the bridge is actually computed, we will try to analyse the running time as a function of h, the size of the upper chain of the hull. If $h = 1$ in any recursive call, we compute a bridge and realize that the bridge connects the leftmost and rightmost points, and all other points can be eliminated immediately. Hence this takes $O(n)$ time.

If the upper chain of S'_R has size h_R and the upper chain of S'_L has size h_L, then we have that $h = 1 + h_R + h_L$. The running time of the algorithm can be expressed as $T(n, h)$.

$$T(n, h) = cn \text{ if } h = 1.$$

$$T(n, h) = cn + T(\frac{n}{2}, h_L) + T(\frac{n}{2}, h_R).$$

We need to argue that regardless of the split of h between h_L and h_R, the algorithm runs in $O(n \log h)$ time. This is not that hard to prove by induction. Assume $T(m, h') \leq cm \log h'$ for $m < n$ and $1 < h' < h$.

$$T(n, h) \leq cn + c\frac{n}{2} \log h_L + c\frac{n}{2} \log h_R.$$

Now use the fact that $\frac{1}{2}(\log h_L + \log h_R) \leq \log \frac{h_L + h_R}{2}$.

$$T(n, h) \leq cn + c\frac{n}{2}(2 \log(\frac{h_L + h_R}{2})) \leq cn + cn(\log h - 1) \leq cn \log h.$$

Computing the Bridge in $O(n)$ time

We now focus on the problem of computing the bridge. Assume that the bridge has slope K^*. We will "guess" a value for the slope K, and try to figure out how K^* compares to K. Through each point p_i in S, let the line of slope K be ℓ_i. The line ℓ_j which has all the points of S "below" it is the line we are looking for – this line has the highest intercept with the dividing line (or the y axis, since they are all parallel). In $O(n)$ time we can compute which line has this property. (This line may not be unique.) If the line is defined by a point from S_L and a point from S_R this is the bridge we are looking for. If the line is defined *only* by points of S_L then $K^* < K$, and if the line is defined *only* by points of S_R then $K^* > K$.

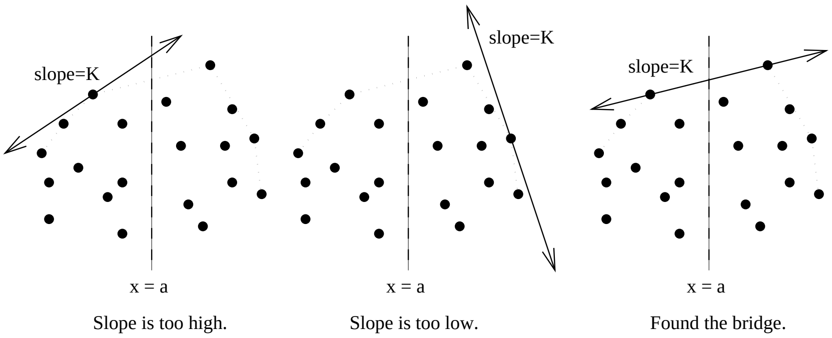

Figure 1: Comparing K and K^*.

In any case, this gives us some information about K^*. To apply the "prune and search" paradigm, we need to somehow discard a constant fraction of the points to continue looking for the bridge. We might "guess" the correct slope, and as soon as we do that the algorithm stops. If our guess is incorrect, we recurse on a constant fraction of the points. The running time is a decreasing geometric series and converges to $O(n)$.

We now outline the method for the pruning step. Consider a pair of points $a, b \in S$. Assume that a is to the left of b. Let the slope defined by this pair of points be K' (see Fig. 2). Suppose $K^* < K$ and $K \leq K'$, then $K^* < K'$. In this case we claim that a cannot be a bridge vertex. Suppose $K^* > K$ and $K \geq K'$, then $K^* > K'$. In this case we claim that b cannot be a bridge vertex. In either case, we are able to eliminate points as being candidate vertices.

How do we guarantee that we can find sufficiently many lines that have slope less than K, and sufficiently many lines with slope more than K? To achieve this, we arbitrarily pair up the points in set S, and compute their slopes. We then select the median slope of this set of $\frac{n}{2}$ lines, and use that as our guess for the slope of the bridge. We either guess the slope correctly and stop, or in either case we eliminate one point from $\frac{n}{4}$ pair's. We are left with at most $\frac{3n}{4}$ points and recurse on this set of points. This completes the prune and search algorithm.

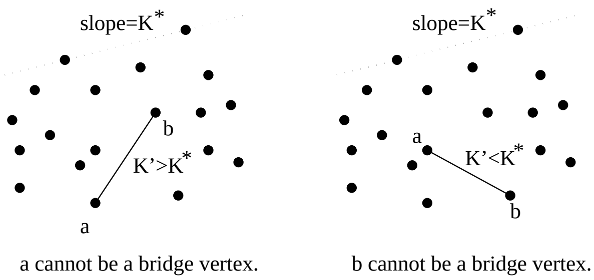

Figure 2: Eliminating points.