CS381 Lecture Notes on the Pointwise Max of Lines

Computing the pointwise max of n lines

Problem definition

The input is an n-element list L whose ith element describes an affine function $f_i(x) = a_i x + b_i$. The ith element of L contains the constants a_i and b_i, since these two constants completely describe the function f_i. The elements of L are not given in any particular order. Recall that the graph of an affine function is a straight line. The *pointwise MAX* of the functions $f_1, \ldots, f_n$ is defined to be the function h such that

$$h(x) = \max_{1 \le i \le n} f_i(x)$$

In other words the graph of h is the "upper envelope" of the graphs of the n input functions. Note that the function h is made up of a number of "pieces" each of which is a straight line segment. Observe that h consists of no more than n such straight line pieces. This implies that h can be described by a list $\hat{L}$ that has no more than n elements. The ith element of $\hat{L}$ contains a description of the ith line segment of which h is made up.

Sketch of the algorithm

Given L as its input, the following algorithm produces $\hat{L}$ as its output in $O(n \log n)$ time.

1. Sort the lines by increasing slopes (i.e., increasing a_i values). This takes $O(n \log n)$ time.
2. Go through the sorted list of lines, and for each line l encountered update the current description of h (i.e., the current $\hat{L}$) in the following way:
   (a) If the segment at the end of $\hat{L}$ is below l then delete it from $\hat{L}$. Keep doing this until the segment at the end of $\hat{L}$ (call it segment s) intersects line l.
   (b) Replace, in $\hat{L}$, the segment s by the portion of s that is above l, and append to the end of $\hat{L}$ the portion of l that is above the line that contains s.

Observe that processing line l could take time proportional to n in the above updating of $\hat{L}$, because it could lead to the deletion from $\hat{L}$ of that many segments that are below l. However, the *total* time (summed over all such lines l) of updating $\hat{L}$ is $O(n)$ because there can be no more than $n-1$ such segment deletions from $\hat{L}$ (that is, once a segment is deleted from $\hat{L}$ the line containing it has effectively disappeared and will not be considered in future updating of $\hat{L}$).

Computing the minimum of the pointwise max

Problem definition

The f_i and h are defined as in the above, and we assume that no two input lines are parallel to each other. Below is an $O(n)$ time algorithm that, when given as input the functions $f_1, \ldots, f_n$, computes the minimum of h, that is, it computes an x value (call it $\hat{x}$) that minimizes $h(x)$, together with the corresponding $h(\hat{x})$.

Sketch of the algorithm

The idea is to view this as a search problem for the pair of lines f_i and f_j whose intersection is the desired minimum. The following observation is needed for performing this search in linear time: For any particular x value, even when we do not know $\hat{x}$ we can test in $O(n)$ time whether that x is smaller than the desired $\hat{x}$ or not: First we find which line (call it l) is above all the others for this value of x, and then we look at the slope of l: If the slope of l is negative (respectively, positive) then x is smaller (respectively, larger) than $\hat{x}$. (Of course if the slope of l is zero then $x = \hat{x}$.) This observation is used in the algorithm that is sketched next.

1. If n is smaller than 20 then solve the problem directly by using the algorithm of the previous section to compute h and then determining its minimum; this is OK because $20 \log 20$ is just a constant. Otherwise continue with the next steps.

   Note. Constants other than 20 can also be used to define this "bottom of the recursion".

2. If n is odd then we spend $O(n)$ time determining whether f_n is one of the two lines we seek by computing its intersection with all the other $n-1$ lines, then using these to determine whether (and which) segment s on f_n appears on h: If such an s exists and one of its endpoints is the $\hat{x}$ we seek, then we are done (simply return that endpoint of s as the answer). Otherwise we discard f_n and continue to the next step with the remaining lines (whose number is now even, so the following steps can assume that n is even).

3. Use the selection algorithm to find, in $O(n)$ time, an x-coordinate (call it x^*) that equi-partitions the x coordinates of the $n/2$ points $p_1, \ldots, p_{n/2}$ (i.e., half of them are less than x^* and half are larger than x^*).

4. Compare, in $O(n)$ time, x^* to $\hat{x}$ (recall that we have observed that this can be done in $O(n)$ time even though we do not know $\hat{x}$). This allows us to "throw away" $n/4$ of the n lines (in the sense that we know that none of the lines we are throwing away can be one of the two lines we are searching for). Which lines we throw away depends on whether $x^* < \hat{x}$ or $\hat{x} < x^*$:

   (a) If $x^* < \hat{x}$ then for every point p_i to the left of x^* (and there are $n/4$ such points), we can throw away the lower-sloped one of the two lines f_{2i-1} and f_{2i} whose intersection is p_i. This is because in the region where the minimum lies (which is to the right of x^*) the lower-sloped of $\{f_{2i-1}, f_{2i}\}$ is surely below the higher-sloped one and therefore it cannot possibly determine $\hat{x}$. After throwing away these $n/4$ lines, we recurse on the remaining $3n/4$ ones.

   (b) If $\hat{x} < x^*$ then for every point p_i to the right of x^* (and there are $n/4$ such points), we can throw away the higher-sloped one of the two lines f_{2i-1} and f_{2i} whose intersection is p_i. This is because in the region where the minimum lies (which is to the left of x^*) the higher-sloped of $\{f_{2i-1}, f_{2i}\}$ is surely below the lower-sloped one and therefore it cannot possibly determine $\hat{x}$. After throwing away these $n/4$ lines, we recurse on the remaining $3n/4$ ones.

   (c) If $x^* = \hat{x}$ then we are done and can return the point $(x^*, h(x^*))$ as the answer.

The recurrence for the above "prune and search" algorithm is of the form $T(n) = T(3n/4) + cn$ and its solution is $O(n)$.