15-492: Parallel Algorithms **Lecturer:** Guy Blelloch
Topic: Computational Geometry **Date:** October 9, 2007
Scribe: Kyle Comer(kcomer@)

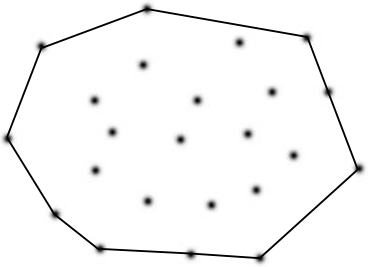
13.1 Review of Graph Algorithms

1. Connectivity and MST
2. Max-Flow
   - P-Complete: No one knows how to solve in poly-logarithmic depth.
   - The best known parallel algorithm is the Pre-flow Push.
     This runs with O(n^2m) Work, O(n) Depth.
3. Reachability
   - Can be done in O($n^3 \log n$) work and O($\log^2 n$).
   - Can be made work efficient with O(M(n)) work and O(n) depth, where M(n) is the time for the best known sequential algorithm.
4. All-Pairs Shortest Path
   - Can be done in O($n^3 \log n$) work and O($\log^2 n$) depth for a dense graph. This method uses matrix multiply but each entry becomes the max of the sums of each pair of elements, instead of the sum of the products of each pair of elements. (For example: $C(i,j) = Max_k(A(i,k) + B(k,j))$)
   - Can be done in O(nm) work and O(n) depth for a sparse graph. This algorithm uses the Bellman Ford algorithm. At each step, every node's distance bcomes the minimum of the neighbors value's plus the edge between the neighbor and itself.
5. Non-Negative Edge Single Source
   - The best sequential algorith is Dijkstra's. This runs in time O($m + nlogn$).
   - There is no known good parallel algorithm.

13.2 Convex Hull

Input: A set of points.
Output: The Elastic Band around them.

Input

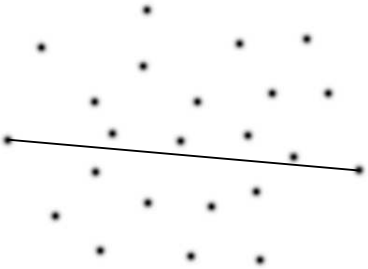


Output

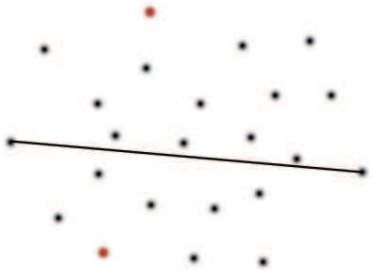
13.2.1 Quick Hull

- Find the minimum and maximum x-coordinates
- Draw a line between the two. (WLOG, we will consider the upper half of the points)
- Find the point furthest above the line
- Generate two new lines
- Throw out all points inside the lines
- Recurse

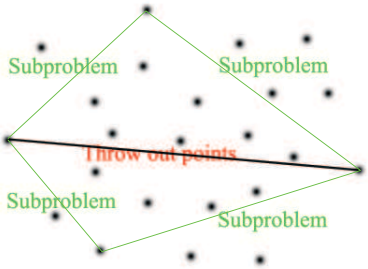
Find min/max x coords



Draw line between them



Find furthest points

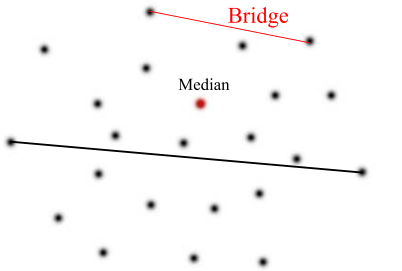


Recurse

Theoretically this algorithm runs in $O(n^2)$ time, but this is only in the worst case. Additionally, the worst case is odd and unlikely to occur. Typically this algorithm will run in $O(nlogn)$ or even $O(n)$. There are other algorithms that will run is $O(nlogn)$.

13.2.2 Randomized Quick Hull

This algorithm requires an algorithm that will tell us the bridge of the convex hull above/below any point on a line. (The bridge is the line segment of the convex hull that is above/below a given point. This algorithm is not too difficult and can be done with two dimensional linear programming in $O(n)$ work and $O(log^2 n)$ depth. Now, instead of finding the point furthest from the line, we find the median point above the line and ask for the bridge above that point. Note that the median point may not be part of the bridge. This will guarentee that the recursive calls are balanced and therefore guarentee the $O(nlogn)$ running time.



Find Bridge of Median Point

13.3 Closest Pair

Input: Set of points in a plane.
Output: Closest pair of points.


Input Output

Technique: Divide and conquer.
We can either split the points in half or the space in half. Since we are dealing with distance between points, it makes more sense to split the space in half. Let us assume we have split the

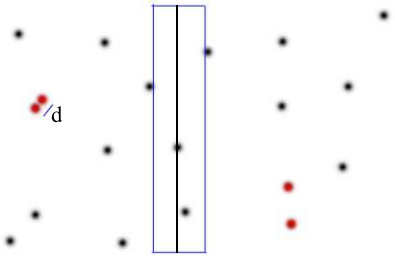
space in half and have found the minimums in both halves, min_L and min_R. What is the minimum of the union of the left and right?
Let $d = min(min_L, min_R)$. Obviously, the minimum will be less than or equal to this value. The only way for 2 points to have a smaller distance is if one is in the left half and the other is in the right half. This will only occur if the points are within a distance d from the dividing line.
If we had a list of all points within d of the dividing line, we could check all pairs to find the minimum. However, this would run in time O(n^2) per step. We can do something smarter. Since points on the same side of the line must be at least distance d apart, each point can only be within d from a small number of points, call this number s. If the points are sorted by their y coordinate, then, for each point, we only must check the points s above and below that point. This will run in time O(n) since s is a constant. If we sort the points by their y coordinate at the beginning and maintain the ordering across the divisions of points, our recurrance for work is W(n) = 2W($n/2$)+O(n). This gives a total work of O($nlogn$) and depth of O($logn$).

The algorithm:

1. Sort by y coordinate
2. CP(Y)
    - Base Case
    - Find x median
    - P_l = Points to the left of the median
    - P_r = Points to the right of the median
    - d_l = CP(P_l)
    - d_r = CP(P_r)
    - d = min(d_l,d_r)
    - P$'$ = Points within d of x median
    - Check each point against a constant number of neighbors
    - d' = Minimum of these comparisons
    - d = min(d,d')
    - return d

Divide points by x coordinate

Recursively find solution



Find points within d of line