Algorithms and Data Structures

Charles A. Wuethrich

Bauhaus-University Weimar - CogVis/MMC

June 5, 2019

Sorting based algorithms

- Introduction
- Closed path
- Convex Hull
  - Convex hull: Wrapping
  - Convex hull: Graham's Scan
- Inner elimination
- Geometric Cut

Introduction

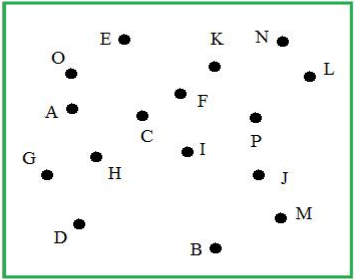
- We have said the other time that we would now start to introduce more complex algorithms
- Clearly, one can combine the basic algorithm types introduced before to solve other problems
- Certainly the most (mis)-used type of algorithms bases on the sorting methods presented at the beginning of the course
- What basically these algorithms do is to transform the problem so that the kernel of the solution relies on sorting.
- The complexity in this case is mostly dictated by the complexity of sorting plus the complexity involved in the transforming step

A first example

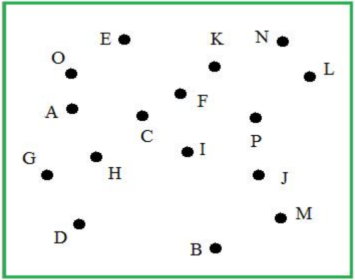
- We already saw one such algorithm: range sorting in 1D
- Remember the simple solution?
  - Sort data
  - Do binary search for the interval border points a and b
  - Pick all elements in between
- As we already said, the complexity here will be
  - Complexity of sorting
  - PLUS the time to extract the elements, which depends from how many points are between the border points
- Next we will present a couple of nice algorithms which solve geometric problems through sorting

Closed path

- Given a set of points, find a non-crossing path through them

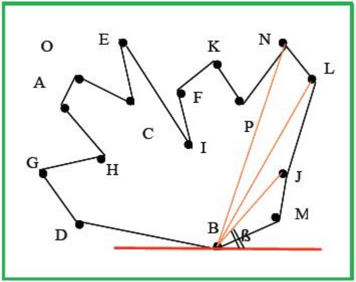


- Let us see how good you are......



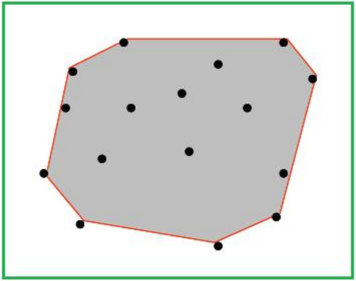
Closed path

- Idea: Choose one point as anchor (Min y: B)
- Use angle between horiz. through B and segment (polar)
- Sort by angle size
- Join consecutive sorted points
- Complexity? Like sorting

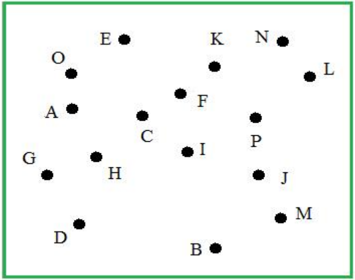


Convex Hull

- A more interesting problem is the problem of the convex hull
- DEF: Convex Polygon π is a polygon ST for any pair of points P_1 ,P_2 in π , the whole segment P_1 P_2 is in π
- Convex Hull problem: Given a set of points, find the smallest convex polygon containing them

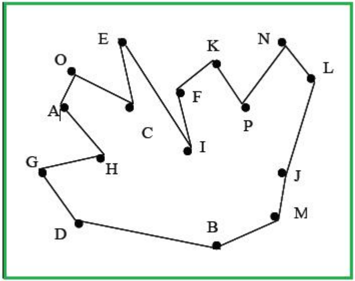


- Let us see how good you are......(part 2)

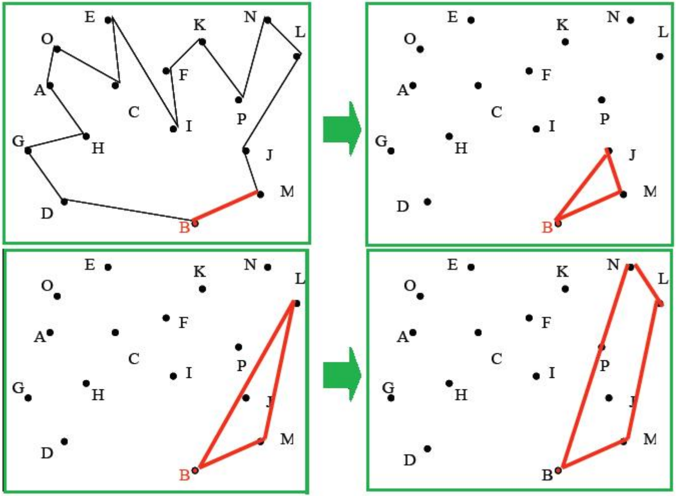


Convex hull: Graham's Scan

- Build the closed path as previously
  - find Min y=B
  - order by growing angle
- Look in Polygon array which points can be eliminated (point in polygon)



Convex hull: Graham's Scan



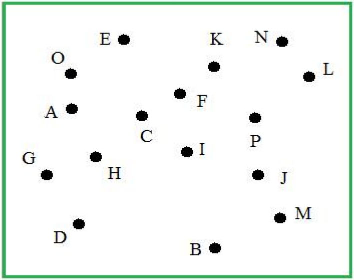
Convex hull: Graham's Scan

- Actually, checking if the old point is in the polygon can be made faster than using the normal point in polygon algorithm, which requires N intersections to be computed per polygon examined.....
- If the points P[1],P[2], ... ,P[N] belong to the CH, and P(N+1) is the next sorted point
  - One has to check only if the angle P(N-1)P(N)P(N+1) is smaller or bigger than 180°
  - If smaller, then P(N) can be eliminated
  - If bigger, it belongs to the CH

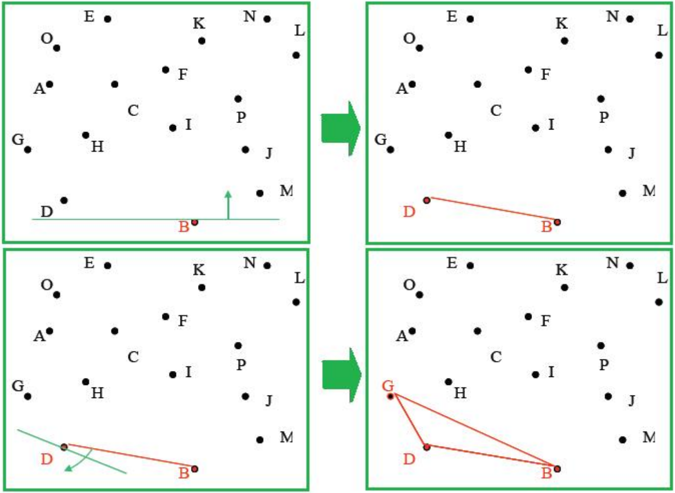
Convex hull: Graham's Scan

- Let us take a look at the complexity of Graham's scan:
  - Sorting first: O(N log N)
  - Run once through each point: linear time O(N)
  - Total: O(N log N+N) $\sim$??

- Wrapping corresponds to what humans do:
    - find first point on hull (e.g. Min y)
    - From this point, move up // line to x axis until second point found
    - Rotate line around this point until next point found.....
    - This corresponds to sorting according to the angle formed with the



Convex hull: Wrapping

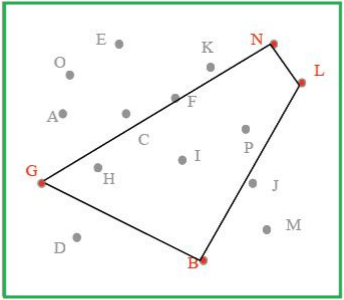


Convex hull: Wrapping

- This algorithm is a slightly more complicated
- Apparently it sorts the points at each step, and this would mean O(M * N log N), where M is the number of points of the CH
- In reality, it does not, because we can rotate the plane so that the previously found CH edge coincides with the x axis
  - Then choose point with minimum y
- This requires M * N steps

Convex hull: Wrapping (Inner elimination)

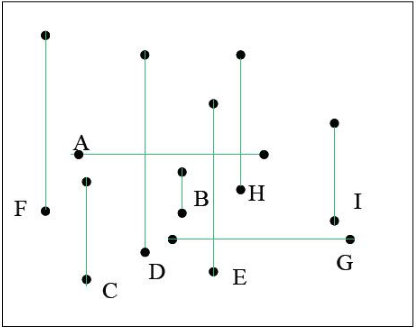
- Additional improvement can be done by clever preprocessing:
  - The four points with Max x, Max y, Min x, Min y belong for sure to the CH
  - Consider the quadrilateral formed by these points
- Throw away ALL inner points to quadrilateral
- This requires N tests (linear)
- Use any other method to continue



Geometric Cut

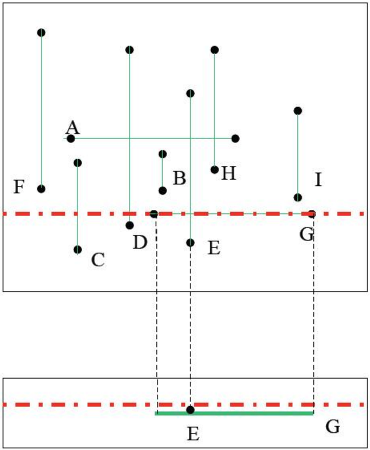
- Another problem which can be solved with a modification of sorting algorithms is the problem of computing the geometric cut
- Basic problem: given a bunch of N objects, do two of them intersect?
- Problem valid for lines, curves, polygons, ...
- Trivial solution: runs in $O(N^2)$ (pairs are $N(N-1)/2$) : insufficient for big datasets

- Let us start from a simple case: crossing of horizontal and vertical lines
- Clearly, both endpoints have either equal x or equal y coordinates
- Problem is more general than expected (VLSI)



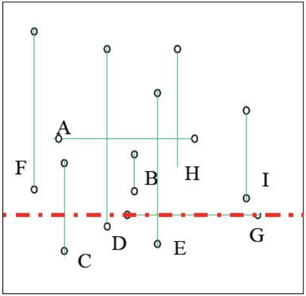
Geometric Cut

- General idea: sweep a horiz. straight line ↑ and look for ∩ :
  - Vertical lines: points
  - Horiz. lines: segments
- Segments intersect if point inside segment
- Problem now 1D



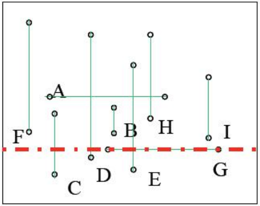
Geometric cut

- Continuous sweeping unnecessary: things change at line endpoints.
- Thus, sort lines by increasing endpoint y: CEDGIBFCHBAIEDHF
- Vertical lines appear twice, horizontal once



Geometric cut

- Continuous sweeping unnecessary: things change at line endpts.
- CEDGIBFCHBAIEDHF
- Use auxilary tree:x-tree (name from coords in tree)
- N lines, I ♯cuts: N log(N+I)



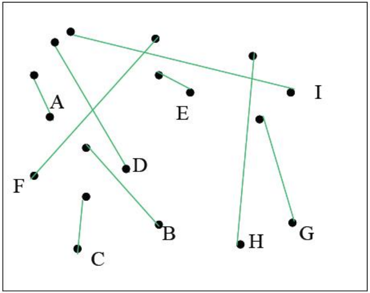
IF start vertical ADD segm. knot
IF end vertical REMOVE segm.
IF start horizontal BEGIN RangeSearch

Geometric cut

- Some optimizations can be used basing on the same principle:
  - Use better data structure for lines
  - Use Active Edge list (adds to clarity and to efficiency)
- What is the complexity of the algorithm?
  Hints:
  - it uses sorting
  - It uses searching....

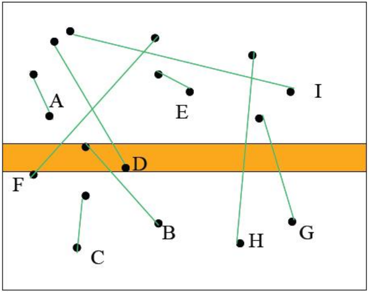
Geometric cut (Bentley-Ottmann)

- What if we choose now generic lines?
  - Range search not sufficient, line ∩ required
  - Line ordering depends also on y coord.
  - 2-D process
- Turns out we can use a similar approach



Geometric cut (Bentley-Ottmann)

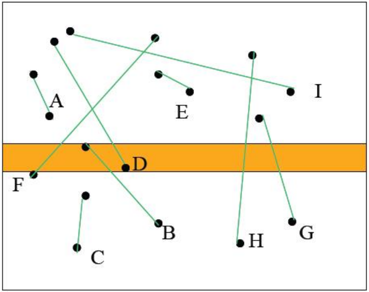
- First sort by increasing y (This creates stripes) and follow ↑ direction.
- Add an entry in binary tree at each line lower point, and remove entry at upper point
- The binary tree reflects the sequence of x coords as before



In the picture example: FBDHG

Geometric cut (Bentley-Ottmann)

- However, bulding the tree needs more care and cannot base on x coord.
- We need a new more general order relation
- Def: A line r is RIGHT of a line s if
  - both endpts. of r lie on the same side of s WRT a point at infinity on the right
  - OR if s is LEFT of r.
  - If r is not left or right of s they must intersect
- Use ccw to check



Geometric cut (Bentley-Ottmann)

- This procedure is an intersection procedure
- Determining if two lines are R or L is NOT transitive: thus careful!!
  - In example, F LEFT of B, B LEFT of D, but F not left of D
- Anytime that something happens in the tree, mutual relations have to be checked
- Summary: Algorithm similar to vertical search, but range search was replaced by new routines with new definitions
- Execution time: ALL intersections can be found in (N+K) log N, where K is the number of intersections.

For further reference, take a look at http://geomalgorithms.com