CS 3137 Class Notes

1 Finding the Convex Hull of a 2-D Set of Points

- Reference: *Computational Geometry in C* by J. O'Rourke and
  http://www-e.uni-magdeburg.de/mertens/TSP/node1.html (applet)
- In our discussion of the traveling salesperson problem, one method we discussed was the method of cheapest insertion, in which we insert a vertex in an existing cycle to minimally increase the length of the tour. Starting with a simple cycle of k vertices, we keep adding vertices that minimize the change in the tour's new cost. For example, if we have a an edge (u,v) in the path, we add the vertex x such that:

  $$dist(u,x) + dist(x,v) - dist(u,v) \ is \ a \ minimum \tag{1}$$

  We may iterate over all choices of u,v to select this minimum.

  This algorithm begins by computing the $ConvexHull$ of the vertices. Given a set of points S in a plane, we can compute the convex hull of the point set. The convex hull is an enclosing polygon in which every point in S is in the interior or on the boundary of the polygon (see Fig. 1).
- An intuitve definition is to pound nails at every point in the set S and then stretch a rubber band around the outside of these nails - the resulting image of the rubber band forms a polygonal shape called the Convex Hull. In 3-D, we can think of "wrapping" the point set with plastic shrink wrap to form a convex polyhedron.
- A test for convexity: Given a line segment between any pair of points inside the Convex Hull, it will never contain any points exterior to the Convex Hull.
- Another definition is that the convex hull of a point set S is the intersection of all half-spaces that contain S. In 2-D, half spaces are half-planes, or planes on one side of a separating line.

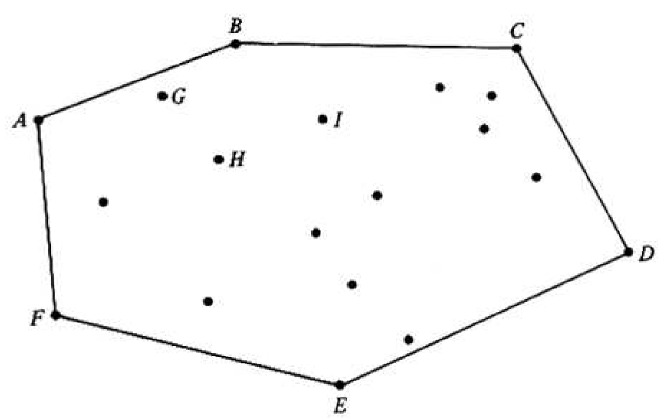


FIGURE 6.27 The convex hull of the points (*A, B, C, D, E, F, G, H, I, . . .*) is the polygon *ABCDEF*.

Figure 1: Convex Hull of a set of points

2 Computing a 2-D Convex Hull: Grahams's Algorithm

There are many algorithms for computing a 2-D convex hull. The algorithm we will use is Graham's Algorithm which is an $O(N\,Log\,N)$ algorithm (see figure 2).

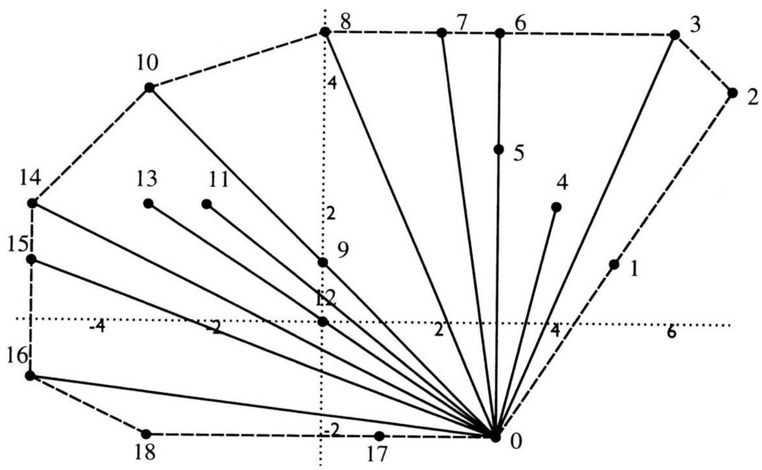
Graham's algorithm is interesting for a number of reasons. First, it is simple to compute and is very intuitive. And for a Data Structures class it is quite compelling, since it uses a number of ideas in we have studied this semester including searching for a minimum value, sorting, and stacks.

1. Given N points, find the righmost, lowest point, label it P_0.
2. Sort all other points angularly about P_0. Break ties in favor of closeness to P_0. Label the sorted points $P_1 \cdots P_{N-1}$.
3. Push the points labeled P_{N-1} and P_0 onto a stack. These points are guaranteed to be on the Convex Hull (why?).
4. Set $i = 1$
5. While $i < N$ do

   If P_i is strictly $left$ of the line formed by top 2 stack entries (P_{top}, P_{top-1}), then Push P_i onto the stack and increment i; else Pop the stack (remove P_{top}).
6. Stack contains Convex Hull vertices.

Notes:

- Strictly left means that the next point under consideration to be added to the hull, P_i, is "left" of the line formed by the two top stack entries - if it is collinear with the two top stack entries, we reject the point.
- One way to determine "left" or "right", we can take a simple cross product of the line formed by the two top stack entries, P_{top-1}, P_{top} and the line formed by the points P_{top-1}, P_i. The sign of this cross product will tell you whether the point P_i is left or right of the line formed by the two top stack entries.



Below is shown the stack (point indices only) and the value of `i` at the top of the `while` loop. The stack is initialized to $(0, 18)$, where the top is shown leftmost (the opposite of our earlier convention). Point p_1 is added to form $(1, 0, 18)$, but then p_2 causes p_1 to be deleted, and so on. Note that p_{18} causes the deletion of p_{17} when $i = 18$, as it should. For this example, the total number of iterations is $29 < 2 \cdot n = 2 \cdot 19 = 38$.

```
i= 1:    0, 18
i= 2:    1, 0, 18
i= 2:    0, 18
i= 3:    2, 0, 18
i= 4:    3, 2, 0, 18
i= 5:    4, 3, 2, 0, 18
i= 5:    3, 2, 0, 18
i= 6:    5, 3, 2, 0, 18
i= 6:    3, 2, 0, 18
i= 7:    6, 3, 2, 0, 18
i= 7:    3, 2, 0, 18
i= 8:    7, 3, 2, 0, 18
i= 8:    3, 2, 0, 18
i= 9:    8, 3, 2, 0, 18
i=10:    9, 8, 3, 2, 0, 18

i=10:    8, 3, 2, 0, 18
i=11:    10, 8, 3, 2, 0, 18
i=12:    11, 10, 8, 3, 2, 0, 18
i=13:    12, 11, 10, 8, 3, 2, 0, 18
i=13:    11, 10, 8, 3, 2, 0, 18
i=13:    10, 8, 3, 2, 0, 18
i=14:    13, 10, 8, 3, 2, 0, 18
i=14:    10, 8, 3, 2, 0, 18
i=15:    14, 10, 8, 3, 2, 0, 18
i=16:    15, 14, 10, 8, 3, 2, 0, 18
i=16:    14, 10, 8, 3, 2, 0, 18
i=17:    16, 14, 10, 8, 3, 2, 0, 18
i=18:    17, 16, 14, 10, 8, 3, 2, 0, 18
i=18:    16, 14, 10, 8, 3, 2, 0, 18,
i=19:    18, 16, 14, 10, 8, 3, 2, 0, 18
```

After popping off the redundant copy of p_{18}, we have the precise hull we seek: $(0, 2, 3, 8, 10, 14, 16, 18)$.

Figure 2: Graham Convex Hull Algorithm example from *J. O'Rourke, Computational Geometry in C*