Assignment #2: Travelling Salesman

Due: Dec. 4, 2018

The travelling salesman problem is as follows: A salesman has to visit a set of cities. Find the shortest path for the salesman that visits each city exactly once and then returns to the starting point.

If the number of cities is large, then computing the absolute optimal path is extremely difficult; however, there are efficient techniques that compute very good paths. The algorithm below is not actually one of them --- it can get stuck at very poor solutions that are local minima --- but it is a nice exercise in hill climbing.

Hill climbing

For this assignment, you will use simple hill-climbing to compute an approximate solution to travelling salesman, as follows:

- The cities are points in the plane, and the distance between two cities is the geometric distance.
- A *state* is ordering of the cities. The corresponding path goes from the first to the second ... to the last and back to the first.
- The *heuristic function* is just the total length of the path.
- The *operator* is to choose any two cities in the list and swap their position. For instance, suppose that the cities are labelled A ... H and the current state S is the sequence [B,F,C,A,G,H,D,E]. One possible swap would be to swap F with G. That would give you the new sequence [B,G,C,A,F,H,D,E]. If this is shorter, then it is an improvement.
- The start state is just the order in which the cities are read from input.

At each stage of the hill-climbing, you have to consider swaps between all pairs of cities. Therefore, if there are N cities, you have to consider N(N-1)/2 swaps.

Input and output

The input is a file with the coordinates of points. There are two lines: first all the x-coordinates then all the y-coordinates per line. For example:

```
0.0  1.0  2.0  0.0  2.0 1.0
0.0  2.0  0.0  1.0  1.0 0.0
```

is the input for a problem with six cities.

Output: At each iteration of the hill climbing algorithm, show the current sequence of points and the total length of the path.

Things to watch out for

The distance between two cities is the geometric (Euclidean) distance given by the Pythagorean formula. The distance from [Ux,Uy] to [Vx,Vy] is sqrt((Ux-Vx)^2 + (Uy-Vy)^2).

Simple algorithm

The simple version of the algorithm is as follows

The basic algorithm is as follows

```
TravellingSaleman(Path: sequence of points) {
   Compute the initial length of Path
   loop {
      choose the pair of points U,V such that
         swapping U with V in Path has the shortest length;
      if there is no improvement, then return Path;
      swap U and V in the path and decrement the length by the change
    }
}
```

The problem with this is that it requires time $O(n^3)$ on each iterations (all choices of U and V times the time to compute the length). You will get 9 points out of 10 for this.

Improved algorithm

The running time on the inner loop can be reduced to $O(n^2)$ as follows:

There are two different kinds of swaps. If you are swapping two cities that are currently in sequence in the state, then you eliminate two edges and add

two edges. For instance, if you have the sequence [B,F,C,A,G,H,D,E] and you swap F and C, then the new state is [B,C,F,A,G,H,D,E] so you eliminate B-F and C-A and you gain B-C and F-A. (The edge F-C is replaced by C-F, but that doesn't affect the length.) i

The second kind of swap involves two cities that are not in sequence. In that case, then you eliminate four edges and gain four new edges. For instance if you have the sequence [B,F,C,A,G,H,D,E] and you swap F and G, the path loses the edges B-F,F-C,A-G,G-H and gains the edges B-G,G-C,A-F,F-H. If the total length of the edges that have been gained is less than the total length of the edges that have been lost, then the path is now shorter, so this would be an improvement.

For large number of cities, this is an improvement over the simple algorithm because you don't have to recompute the entire length of the path for each swap, you just have to compute the change that the swap makes, which involves only 4 or 8 edges.

The new algorithm is as follows

```
TravellingSaleman(Path: sequence of points) {
   Compute the initial length of Path
   loop {
      for each pair of points U,V, calculate the change to the length
            caused by swapping U and V;
      choose the pair U,V
           that causes the largest decrease in the length of Path
      if no pair of points causes any decrease,
          then return Path;
      swap U and V in Path and decrement the length by the change
    }
```

The only data structure you need is a 2 x N array holding the coordinates of the points in sequence, and a variable that holds the current length. (That's one of the advantages of hill climbing; all you need is the current state).

Sample Input and Output for Assignment #2

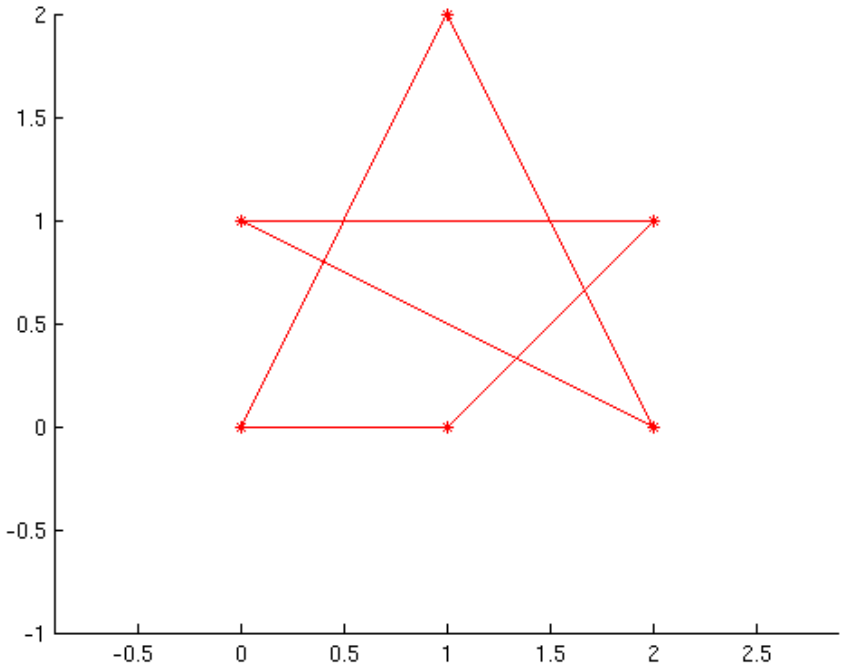
Input

```
0.0  1.0  2.0  0.0  2.0  1.0
0.0  2.0  0.0  1.0  1.0  0.0
```

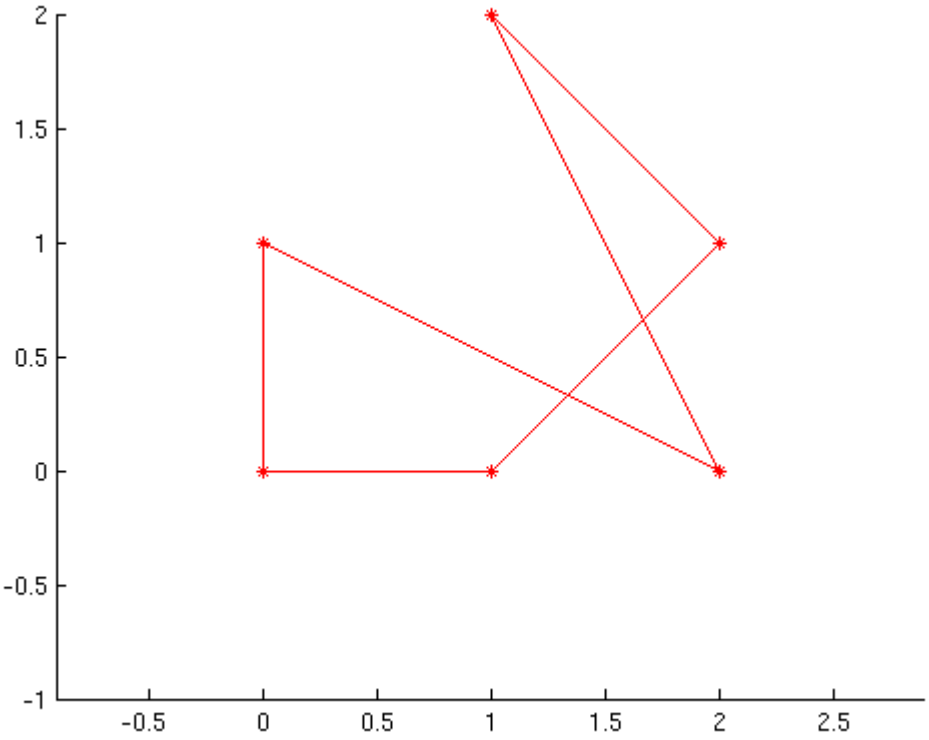
Output

```
Path:
0.0  1.0  2.0  0.0  2.0  1.0
0.0  2.0  0.0  1.0  1.0  0.0

Length = 11.1224

Swap 1 and 5

2.0  1.0  2.0  0.0  0.0  1.0
1.0  2.0  0.0  1.0  0.0  0.0

Length = 9.3006

Swap 2 and 3

2.0  2.0  1.0  0.0  0.0  1.0
1.0  0.0  2.0  1.0  0.0  0.0

Length = 8.0645

Swap 1 and 2

2.0  2.0  1.0  0.0  0.0  1.0
0.0  1.0  2.0  1.0  0.0  0.0

Length = 6.8284

End of hill climbing
```

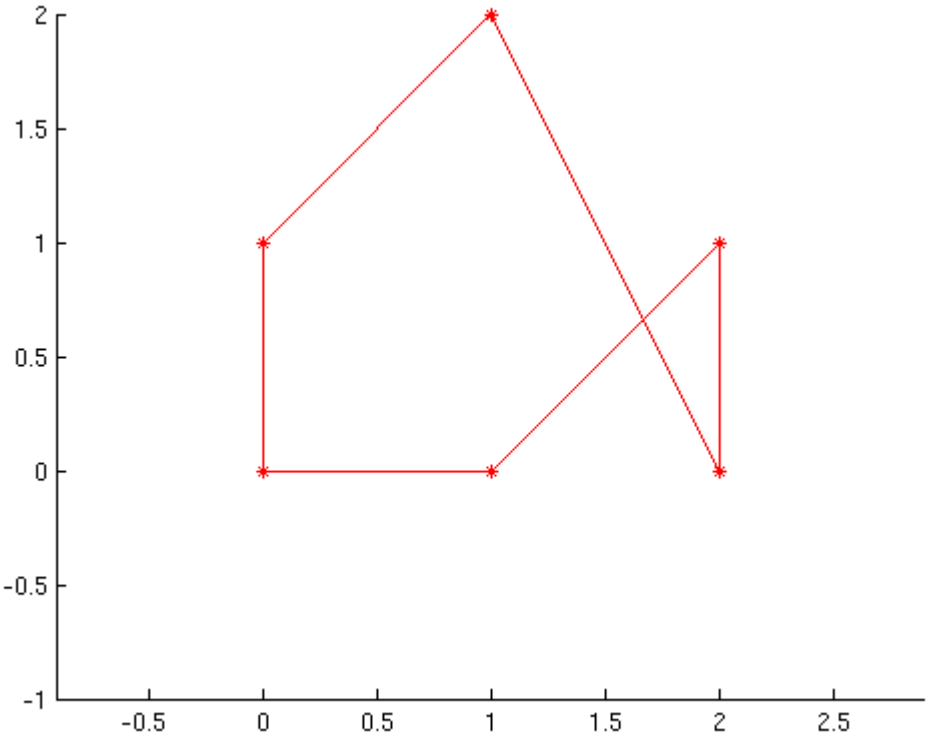
First Path



Second Path



Third Path



Fourth Path