Project 4

CS325 — Spring 2015

by Group 2
Vedanth Narayanan
Jonathan Merrill
Tracie Lee

June 7, 2015

Ideas behind the algorithm

There was a lot of research we did before figuring out what we wanted to implement. The single algorithm that was universally praised was Christofides' Algorithm. To succinctly explain, it starts out by creating a minimum spanning tree (MST). The next step is trying to figure out the nodes that have an odd number of edges. Now short cuts need to be created, so there are no vertex has any odd number of edges. A Euler path needs to be found now. Without an Euler path, not all edges can be traversed at least once. This sets us up so we can properly traverse the graph using shortcuts we created, and being able skip over vertices as they seem appropriate. It has been mentioned that Christofides' algorithm is probably one of the best as of now. It is "guaranteed to be at most 50 percent longer than the shortest route," as Erica Klarreich mentions in her Wired article, "Computer Scientists find new shortcuts for infamous Traveling Salesman Problem."

While we were not able to implement a strict interpretation of Christofides' Algorithm, we were inspired by some of its elements. To explain our algorithm, we start with a list of cities that our salesman needs to visit and the corresponding x and y coordinates for each city. We iterate through the list of cities in a for-loop, and use each city in the list as a starting point for a cycle. We then use an inner loop to traverse through the list beginning at that starting city. We iterate through the list, finding the closest city to the starting city, then finding the closest city to that city and so on, working through the map finding the closest node to the current node for the current iteration of the loop. We then use a shortcut to end up back at the starting city once the inner loop has completed the cycle. As the inner loop is traversing the list and finding closest nodes, it keeps track of the overall distance of the current cycle. We also keep track of which cities have been visited by removing each city that we have found from the "unVisitedCities" list and appending each city found to a new list that represents the cycle beginning at the current starting city. Finally, the inner loop returns a tuple that contains a list of the cities in the current cycle as well as the total distance for that cycle. When the outer loop completes, it returns a tuple that contains a list of cycles and each corresponding distance.

"Best" Tours

The "best" tours for the three example instances and the time it took to obtain these tours.

Test Number	Time(seconds)	Final Distance
1	0.28	130921
2	12.81	2975
3	2128.80	1934200

Results

While implementing this algorithm, we drew out little test cases by hand and performed some tests to make sure that it was working it should be. This really helped us out in understanding what we were trying to do. The inputs may just be numbers, but having a visual representation to go along with it helped us with not only understanding what we were trying to implement, but if we were doing a good job of it.
One of the things that sort of threw us off was the fact that Test 1 ran under half a second, but the distance is enormous. Intuitively speaking, when you see that the distance is far, you'd expect there to be a lot of cities. This once again probes the question, if there were so many cities then how was it possible to be finish under half a second? But then we realized that the number of cities we had in Test 1, is not too many, but the distance between them is a lot. This is the reason why the results look a little funky. Similarly, Test 2 takes a reasonable amount of time, but the distance is short. Although there are more than a handful of cities, the distances between them is short.

References

- http://www.wired.com/2013/01/traveling-salesman-problem/
- https://developers.google.com/optimization/routing/tsp
- http://www.thetubechallenge.com/
- https://github.com/branaugj/cs325/blob/master/proj4/proj4.py
- http://www.cs.princeton.edu/ wayne/cs423/lectures/approx-alg-4up.pdf
- http://www.seas.gwu.edu/ simhaweb/champalg/tsp/tsp.html
- http://web.stanford.edu/ saberi/tsp.pdf