Investigating Approximate Aglorithms for Traveling Salesman Problem

William Touchstone

April 2020

1 Summary

For this project, I attempted to implement a couple of algorithms. The first of these was the greedy algorithm known as Nearest Neighbor. It starts at a random point and then automatically chooses the nearest neighbor, and does a full path before returning to the start. This was not a suitable algorithm, and it consistently produced a cost of about 32200

The next algorithm I tried to implement was Christofide's. I got pretty far into it, but upon getting stuck and reading more on it, I saw that it required the triangle inequality to be true. Seeing as we were not to assume this, I did not continue from here.

The final and most successful method I did for attempting to solve TSP was through an ant colony optimization algorithm. This algorithm will simulate a large number of ants, pick the best one, and essentially weight the path it took with the ant's "pheromones." Paths that are shorter have more pheromones deposited. Eventually, the algorithm will pick edges that are shorter, more traversed, and therefore more likely to be optimal.

2 Pseudocode of Ant Colony Optimization

```
while there is time remaining:
    Send out many ants -> take best path from ants
    for the edges in this path:
        increment these edges pheremones by (baseline cost - new cost)

psuedocode for ant pathfinding:
start at random location
while not every node visited:
    take path from current node with min(length - pheromones)
complete path
```

3 Analysis of Performance

This is the gist of the basic psuedocode of the algorithm. It's hard to explain what it does in psuedocode, I feel the description in the previous paragraph is more apt. However, this algorithm will complete the TSP for the nodes given with a cost of 28574. This is basically guaranteed, however if the initial ant that forms the baseline ends up with a different path than usual (0.0038% chance of occurring), then the number could end up at about 28900.

The reason for the design of this algorithm comes from observing nature and how ants behave. It's sort of like a genetic algorithm, in that it improves as time goes on, however there are not generations, per se, it's more of a simple iterative process. Wikipedia says that a nearest neighbors algorithm will usually result in a solution within 25% of optimal. The first set of ants are essentially a nearest neighbors search since no pheromones are out yet. Thus, this algorithm will start at on average 25% of optimal and slowly work its way down closer.

My implementation also features random restarts, in the small case that I basically get trapped in a local minima. These random restarts will occur for as long as there is time left to run the program. However, with the number of ants I run and the number of iterations of runs, I'm not sure how well this program will generalize to larger sets. I feel that for it to work as well as it does now, these numbers would need to be increased, but I'm not sure. Also, on the small set my implementation converges. I do not know if this would be the case on a larger set, or if it would potentially just keep changing.