Joshua Vayle and Ben Remer
CS 3510 Programming Project
April 18, 2019

Overview of Our Algorithm

To solve this instance of the Traveling Salesman Problem, we have chosen to utilize a genetic algorithm.

Problem Information:

- Inputs: a graph G = (V,E) with weighted edges (in this case the weight of each edge is equal to the euclidean distance between the (x,y) coordinate pairs of the the respective vertices in a 2D plane.

Problem Objective:

- The objective of the Traveling Salesman Problem is to find the shortest possible cycle of the graph

Algorithm Description

- Load in graph→ put into our city object
- Randomly create the first group of routes (initial population)
- Find the distance of each route
- Rank the routes from smallest to longest
- Mate
    - Keep the top 20 routes as they are
    - For the rest, randomly select an amount of them. Doing this with a higher chance of picking the routes with the least distances
- Breed
    - Keep top 20 routes as they are
    - For each other route in the population choose two routes in the mate selection.
    - Then for each pair basically randomly splice their “genes” together
        - Genes are subsections of the route.
- Mutate
    - Each “child” route undergoes “mutation” wherein random cities in the route are swapped with one another
- Check if best route in current generation is shorter than the stored best route
    - Re-rank the routes in the top 20 then check if the top one if shorter than the current best route
- Repeat until the number of generations specified at the beginning has been reached (or time limit is hit)

Reasons For Choosing a Genetic Algorithm

Genetic algorithms are quite simple, essentially, you are just generating a bunch of random routes, selecting the best ones, and randomly splicing those best routes togethers then

shuffling them a little bit and selecting the best routes again and repeating the same process a bunch of times. This is a very useful way to deal with the fact that since the graph we are given is strongly connected (we were told that every vertex is connected to every other vertex), if we looked at every possible route at the same time the amount of possible solutions would be just huge. The genetic algorithm we created essentially only looks at the random solutions it generates. Over time though since it is always keeping track of the best route is has come up with, the algorithm advances closer and closer to the optimal solution. While it is unlikely that our algorithm would come up with the most optimal solution, there is a good chance that it finds a route that is close to optimal.

WIthin genetic algorithms there are a large number of implementations. The key choices we made with regards to our specific implementation is the choice to use randomized crossover/breeding in our algorithm and random mutation. The main reason why is simplicity. It was relatively simple to implement the random breeding as well as the random mutation. This allowed us to form a solution that we could easily code and while it likely will not produce the most optimal solution, our testing indicates that given a large enough amount of time it is likely that the algorithm will consistently be able to come up with a solution that meets the parameters of the test case, especially for smaller cases like the 30 vertex case in the test file.

So the main advantage of our algorithm is that we are always coming closer to the most optimal solution. Because we are always tracking the best route that we have found so far, the algorithm is always either making progress towards the most optimal solution or staying where it is; it is not going backwards. In addition, because we are only looking at the routes generated by the genetic algorithm, we only consider a fixed number of routes at a time, regardless of the number of vertices in the graph. For example, if we had gone with a dynamic programming approach:
For all possible starting nodes i in the graph G:
If size(G) is 2:
C(G, i) = dist(1, i)
Else size(S)<2:
C(G, i) = min {C(G-{i}, j) + dis(j, i)} where j is a vertex in G, j does not equal i and j does not equal 1.

The advantage of a genetic algorithm over this approach is that this approach considers all of the possible solutions while the genetic algorithm only considers a fixed number of routes at a time (the population size which for us was 20). This means that hte memory footprint of out solution is much lower than the dynamic programming solution. Also, because we are not generating n! possible solutions, we are only making a fixed number of generations, our run-time should be better as the n! gets to be a value that is larger than the population size * number of generations.

Works Cited

https://iccl.inf.tu-dresden.de/w/images/b/b7/GA_for_TSP.pdf
Paper on applying the genetic algorithm to the traveling salesman problem

https://medium.com/@becmjo/genetic-algorithms-and-the-travelling-salesman-problem-d10d1daf96a1
Medium article on applying genetic algorithms to the traveling salesman problem.

https://www.researchgate.net/publication/300038752_TRAVELLING_SALESMAN_PROBLEMS_BY_DYNAMIC_PROGRAMMING_ALGORITHM
We used this paper to determine the quick pseudocode for the basic dynamic programming algorithm for the advantages section of the paper