**Project 4: TSP using**

**Genetic Algorithm**

Trevor Bright

Computer Engineering/Computer Science

Speed School of Engineering

University of Louisville, USA

[t0brig01@louisville.edu](mailto:t0brig01@louisville.edu)

1. **Introduction** (What did you do in this project and why?)

For this project, I used Python through Visual Studio Code and the command line to run and compile the program. This project is using a greedy edge heuristic algorithm to find an optimal path to hit every city and return to the first. I used Python because it is what I am most familiar with for projects like this. This is the same language I used for the first project, so a lot of the code was reused such as the file parsing code and the GUI formatting functions.

1. **Approach** (Describe algorithm you are using for this project)

My approach to the problem was so strip out the data from the files provided, make an object based on a class I made that holds the number, the x, and the y coordinates for each of the cities. I reused the parsing method and functions from the first and second project. Because an instance of TSP with less than 4 nodes is trivially solvable, my program first started by making a triangle, having an edge going from city 1 to 2, 2 to 3, and 3 back to 1. After that, I went through each city, excluding the first 3, and comparing the distance between the city and each of the edge. The edge is chosen based on the shortest distance to the city. The selected edge is then removed and replaced with two edges: one from the from-city of the edge to the city, and one from the city to the to-city of the edge. After I found the path, I used the matplotlib library to return a visual representation of the path.

1. **Results** (How well did the algorithm perform?)
   1. **Data**

I used the city data provided and implemented my greedy edge heuristic algorithm to find an optimal path visiting all cities and return to the starting city. I then used the matplotlib Python library to generate a visual representation of the path.

* 1. **Results**

Random40.tsp

-------Path-------

[1, 30, 15, 39, 20, 6, 25, 13, 31, 33, 29, 38, 40, 10, 8, 4, 35, 26, 5, 9, 27, 22, 36, 14, 2, 18, 12, 11, 16, 37, 17, 23, 34, 28, 32, 19, 3, 21, 7, 24, 1]

-------Distance-------

603.073463345

A close up of a map

Description automatically generated

Random30.tsp

-------Path-------

[1, 30, 15, 20, 6, 25, 13, 29, 10, 8, 4, 26, 5, 9, 27, 22, 14, 2, 18, 12, 11, 16, 17, 23, 28, 19, 3, 21, 7, 24, 1]

-------Distance-------

504.762834639

A close up of a map

Description automatically generated

1. **Discussion** (Talk about the results you got and answer any specific questions mentioned in the assignment.)

While my program had a couple of bugs resulting in not finding the most optimal solution, it is still close to the optimal solution. This algorithm is a lot faster and effective than the previous algorithms used in the last two projects. Even with running 40 cities, the program could be run in less than half a second. This is exponentially faster than brute force and will actually give a valid solution unlike BFS and DFS.

1. **References** (If you used any sources in addition to lectures please include them here.)

Matplotlib library and documentation: <https://matplotlib.org/>