**CSE 545 – Artificial Intelligence – Project 3 – TSP Greedy Variant**

Jordan Richardson

BA Computer Science

Speed School of Engineering

University of Louisville, USA

[Jkrich05@louisville.edu](mailto:Jkrich05@louisville.edu)

**Introduction**

This project focused on a variant of TSP using the greedy algorithm. For this problem we used the distance from an edge to a point, for all cities, for each edge. This means we had to consider all edges currently created and consider all cities for each edge. We also had to consider how we inserted the closest city once we found it, instead of just adding it the tour path, we were expected to insert it between the start and endpoints of the edge it is closest too. This eliminates the initial edge and creates two in its places from the closest node to the start and end points of the ex-edge. Due to this, we had to hardcode an edge in to start the problem. We were also required to add a visual display for the output of the program along with distance, path taken, and time to completion.

**Approach**

My approach for this problem was a bit backwards I believe, and it cost me a lot of time in debugging and configuration of the problem. I began by importing what I could from previous projects, getting my display working and as I wanted it, and verified my loops were working with just the shortest distance formula and without a line list whatsoever.

However, what cost me time was then trying to edit the code I already had and implant a line list properly, along with correctly inserting and removing into and from that line list. I also had quite a struggle with the distance calculation at first, and it took me a while to figure out a better solution after emailing the professor, as I was lacking a bit in mathematical skills. It led to me having to create extra definitions, and I even had to entirely redo my graphing as I could not find the problems it was encountering.

My approach for hard coding my first edge was very basic, I wanted to start at city 1, so I simply used the shortest distance calculation from a node to another node, and then hardcoded that in before changing my distance calculation. This then allows my program to build the rest of the tour path based off the first edge.

**Results**

I did not use any libraries for calculations other than sys, math, time, and numpy which means I am unable to compare my efficiency to perfected calculations for these types of problems, I manually code it as I am BACS and I want the mathematical understanding. I am glad I did so because I struggled with the distance calculation for a while before finding something I believe works as intended. I hardcoded my first city as 1 and my second city as the closest to city 1, which was 24. I then had to remove these cities from my unvisited\_cities and append them to my line list so the program could calculate off the first line. I had it loop over the edges in the tour if there were unvisited cities and calculate the distance to every city from each edge, before assigning the minimum distance and adding it to my path, lines, and total\_distance. I also added file selection to my program instead of having it hardcoded and a timer that plots the points one at a time for visualization, however the visualization still doesn’t work as I intended.

We used the greedy approach which simply selects the shortest distance locally without considering the long-term cost until the cycle is completed and arrives back at city 1.

**Results – Data**

For this project, we were given two .TSP files that contained some basic information about how it was generated up to line 7, and a list of cities with city number, longitude, and latitude. Our problems this time had 30 and 40 cities, along with a document explaining on how we should insert the closest node into the path. This meant the fundamental idea of our previous problems gets taken away, as we were now expected to insert the node, in between the edge, eliminate the edge, and connect it to the start and end points of the ex-edge. This added a level of complexity to the problem that I at first did not enjoy, but I learned a ton just about the organization of my code, and how simple mistakes carry throughout the program, and it became fun tracking them down and learning.

**Results – Results**

This is the structure of my code; it helps me to think logically throughout, and I can debug it easier in the long run.

A screenshot of a computer program

Description automatically generated

This is my application for this program using the .TSP file with 40 cities.

A screenshot of a computer

Description automatically generated

Here is the Path and Lines list as they are not visible in the image.

A screenshot of a computer screen

Description automatically generated A black screen with numbers and symbols

Description automatically generated

This is my application for this program using the .TSP file with 30 cities.

A screenshot of a computer

Description automatically generated

Here is the Path and Lines list as they are not visible in the image.

A screenshot of a computer screen

Description automatically generated A screenshot of a computer screen

Description automatically generated

These problems both execute in record time, and I even found a TSP file online with many more cords which I tried to test the efficiency of it.

This is more efficient than brute forcing, and DFS and BFS, considering the number of cities in this problem. However, as you can see from looking at the path, it is not the most optimal solution if the shortest distance overall is required. I did not hardcode any other edges than the first, I just let the program execute, and it seemed optimal so I did not implement any other improvements, though visually I can see a few such improvements such as hardcoding 10 to 16, 16 to 11, 11 to 8, and 8 to 12 would likely eliminate some distance as well as traveling to 12 before traveling to 18.

**Discussion**

During this assignment, I didn’t only learn more about how organization and structure of code affects it. I also gained some mathematical insights, and how to properly use them in my calculations. I honestly have just really realized that the more I struggle with the problem, the more I get out of it, and the more I improve as a programmer. I feel that this is due to just how I learn, along with programming requiring implementation in my cases before the subject becomes cemented. This assignment had me considering my transfer from CSE to BACS as it reignited my passion for math, which the engineering calculus failed to do. I believe many parts of programming, depending on utilization, can absolutely require an in-depth understanding of mathematics, or can even allow a path to optimization. I saw this in this assignment in a few ways, whether that is correctly calculating distance, correctly adding it to my total\_distance, understanding how to delay plotting of my graph correctly, and even the correct addition and removal from my lines list was troublesome. However, I am ecstatic with how it turned out and it works exactly as I wanted it too.

**References**

I did not use any references for my code other than the documentation for PySide6, along with some basic stack overflow examples for my GUI and timer for graphing. Other than those I referenced some previous lectures in attempting to understand the greedy approach better, however it is pretty simple.