**Traveling Salesman Project 3**

Tyler Hamilton

CECS, 1713904

Speed School of Engineering

University of Louisville

jthami05[@louisville.edu](mailto:NameID@louisville.edu)

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

In this project, I implemented a greedy algorithm to solve the Traveling Salesperson problem (TSP) using Python. The purpose of this project was to determine the effectiveness of the greedy approach to TSP.

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

The greedy algorithm is an algorithm that only looks one step ahead, calculating the best possible route from the current node to the next, and then continuing to do so until every node has been reached.

For the GUI on this project, I used Python’s Tkinter package, creating a canvas on a Tk() instance, scaled the cities up by 5 times to improve readability, plotted them and their routes, and displayed the total distance on the GUI.

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

The greedy algorithm performed very quickly, executing the search for both datasets immediately. The routes traversed were *almost* optimal, with one crossed path for each dataset.

* 1. **Data** (Describe the data you used.)

I used the two .tsp files given in the project. The name of the file to be read was passed in by the user via prompt. I ignored the first sections of the files and read the remaining lines in, as IDs, x-values, and y-values for *city* objects.

* 1. **Results** (Numerical results and any figures or tables.)

The program works as expected, aside from the single crossed path that occurred in both datasets when this algorithm ran.

Random30.tsp:

Running time : 0.00187397003174 seconds.

Total distance: 480.659134879

Path:

city 1 is the 0th element

city 24 is the 1th element

city 30 is the 2th element

city 15 is the 3th element

city 10 is the 4th element

city 16 is the 5th element

city 11 is the 6th element

city 12 is the 7th element

city 18 is the 8th element

city 4 is the 9th element

city 29 is the 10th element

city 13 is the 11th element

city 25 is the 12th element

city 6 is the 13th element

city 20 is the 14th element

city 8 is the 15th element

city 17 is the 16th element

city 7 is the 17th element

city 21 is the 18th element

city 3 is the 19th element

city 19 is the 20th element

city 23 is the 21th element

city 28 is the 22th element

city 22 is the 23th element

city 27 is the 24th element

city 9 is the 25th element

city 5 is the 26th element

city 26 is the 27th element

city 14 is the 28th element

city 2 is the 29th element

Random40.tsp:

Running time : 0.00139689445496 seconds.

Total distance: 612.77864996

Path:

city 1 is the 0th element

city 24 is the 1th element

city 30 is the 2th element

city 15 is the 3th element

city 39 is the 4th element

city 8 is the 5th element

city 10 is the 6th element

city 40 is the 7th element

city 16 is the 8th element

city 11 is the 9th element

city 12 is the 10th element

city 18 is the 11th element

city 35 is the 12th element

city 4 is the 13th element

city 33 is the 14th element

city 29 is the 15th element

city 38 is the 16th element

city 13 is the 17th element

city 25 is the 18th element

city 6 is the 19th element

city 20 is the 20th element

city 31 is the 21th element

city 26 is the 22th element

city 14 is the 23th element

city 2 is the 24th element

city 36 is the 25th element

city 34 is the 26th element

city 23 is the 27th element

city 37 is the 28th element

city 17 is the 29th element

city 7 is the 30th element

city 21 is the 31th element

city 3 is the 32th element

city 19 is the 33th element

city 28 is the 34th element

city 32 is the 35th element

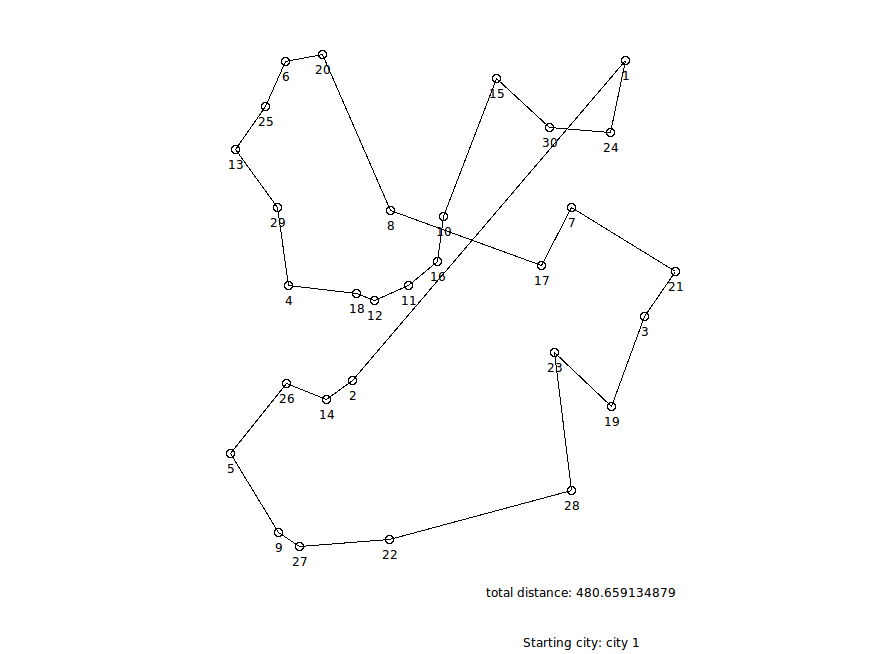
city 22 is the 36th element

city 27 is the 37th element

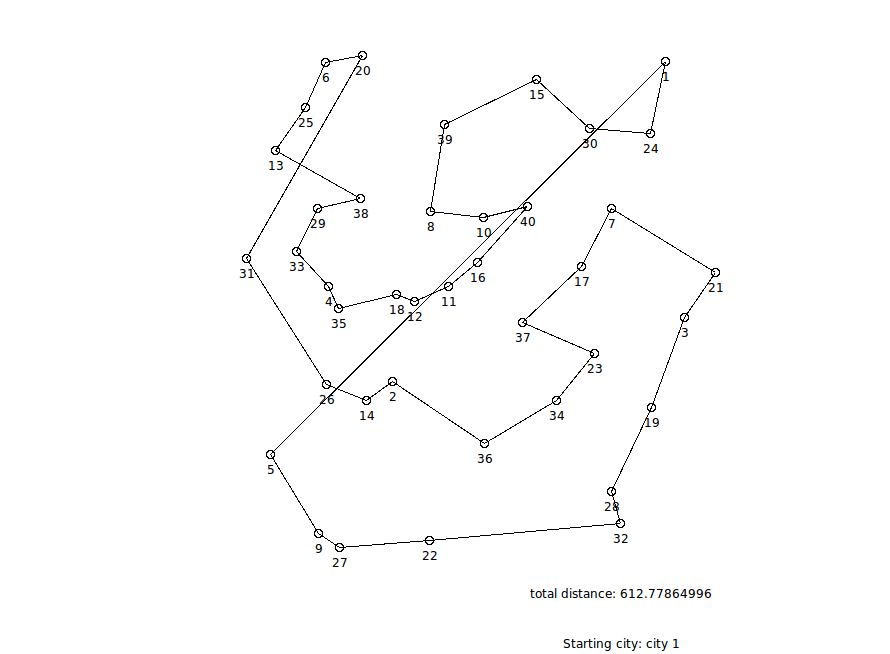
city 9 is the 38th element

city 5 is the 39th element

Traveling Salesman – Greedy Algorithm – Random30.tsp:



Traveling Salesman – Greedy Algorithm – Random40.tsp:



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

The basic process of my program is as such: The dataset is read in as a command line argument, and separated into city objects with x and y coordinates. The greedy algorithm is called on my list, and the first city is visited as the base. Then, the following happens until every city in my original list is visited: all cities in the original city list are looped through, and the distance between each city is sequentially calculated. If this is the first iteration of our loop, the current best distance is set = the resultant distance from the first two cities checked. In all later iterations of the loop, the current best distance is compared to the newly calculated path. If the current best distance is greater than the newly calculated path, the city being analyzed is marked as a possible candidate for addition to the plottedCities list and the new route’s distance becomes the new current best distance. If the current best distance is less than the newly calculated path, the current best route’s end city remains the best possible candidate for addition to the plottedCities list.

After each time the for loop exits, the best possible candidate city is marked as visited and plotted. This city becomes the new root city from which the next calculation will be based off of.

The initial node that I selected for this project was the first node in the dataset, city 1.

The order in which the nodes were visited is displayed above in section 3.2, Results. The order is also displayed in the same section, starting at city 1. All cities are labeled with their ID values.

Because the size of the datasets in this project were different than the size of the datasets in the previous projects (n=30 and n=40 as opposed to n=11 and n=4-12), I am unable to analyze an exact comparison of this algorithm vs. our previous sorting algorithms. However, the greedy algorithm seems to perform one power-of-ten worse than the algorithms from Projects 1 and 2, without taking these dataset size differences into account.

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

[http://effbot.org/tkinterbook/canvas.htm#Tkinter.Canvas.create\_oval-method](http://effbot.org/tkinterbook/canvas.htm" \l "Tkinter.Canvas.create_oval-method)

https://wiki.python.org/moin/TkInter