**Project 4: TSP using**

**GA and Wisdom of Crowds**

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 genetic algorithm and wisdom of crowds 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 last projects, 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)

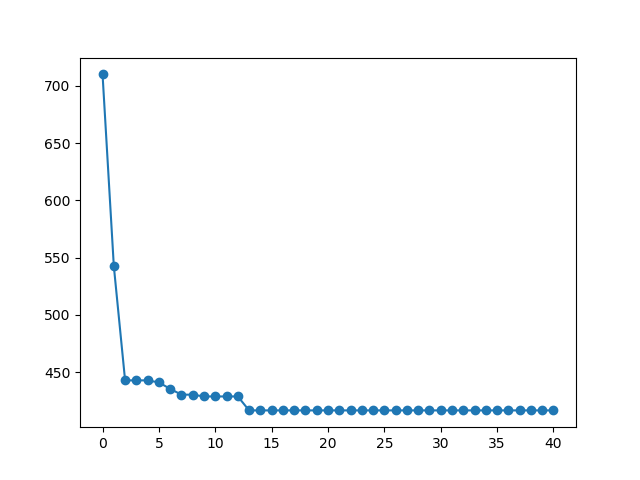
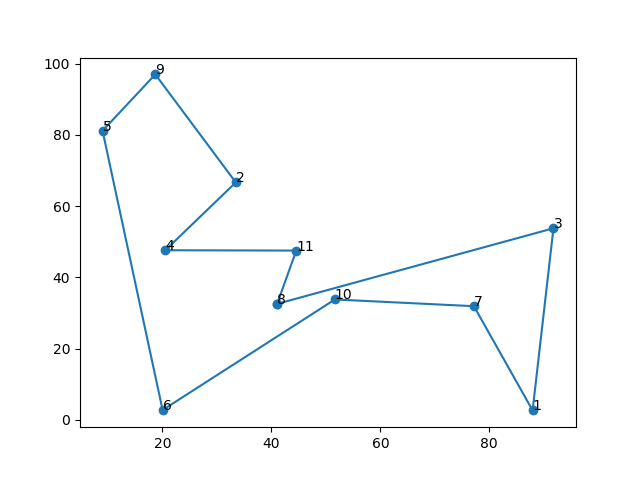
My approach to the problem was so strip out the data from the files provided, make an object based on a class I made that holds the number, the x, and the y coordinates for each of the cities. I reused the parsing method and functions from the previous projects. I first made an initial population of a set amount of paths. I then sorted them by distance and calculated the fitness score of each. I then took a percent of the best in the list. I played around with the amount I took but ended up using the best 25% in the list. After I found the best 25%, I then picked two random cities from it and bred them during this breading process, to cut down on the number of mutations and to avoid excessive fluctuation, I added a while loop that checks if the child that was just bred has a distance greater than the previous generation. If it does, it will bread again until it is less than the previous population. It will, however, pass if it is the distance is the same as the previous generation. If it is for a certain amount of times, it will stop. I used the crossover method which takes a random length from the first parent and puts it into the child then the second parent populates the rest of the child. After repopulating the new population of bred member, I then put it through mutation. I tried many mutation rates, but I found 5% to be the most efficient with my algorithm. Originally, I had it at 10% but it took a lot longer to get to a good result or messed up the results resulting in changing the best path to a path that had a higher distance. After I was done with mutation, I repeated the process of finding the best, breading a new population, and mutating that population. For wisdom of crowds, I ran it 10 times, each time storing the population it returns. I then took a top percentage of each, combined them, and continued. Depending on the number of cities, I changed the amount of generations I ran. For example, with smaller sample sizes, a large amount of generations was not needed. After I found the best path, I used the matplotlib library to return a visual representation of the path. I also took the distance over time array and plotted a simple x y graph also using matplotlib to show the improvement over the generations.

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

I used the city data provided and implemented my GA and wisdom of crowds 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.

**Results**

**Random11.tsp**

****

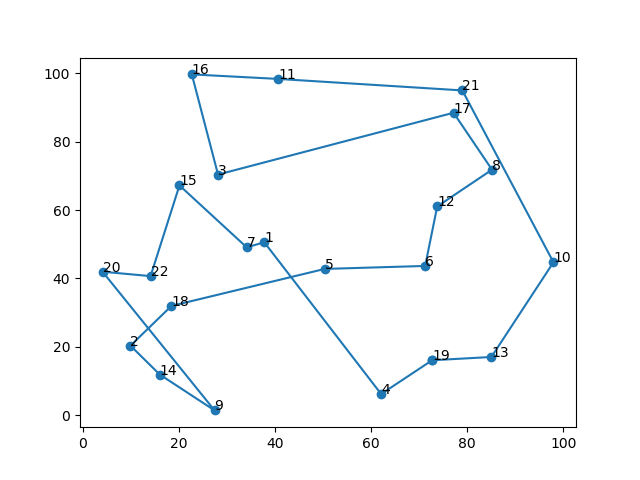
-------------------FINAL PATH---------------------

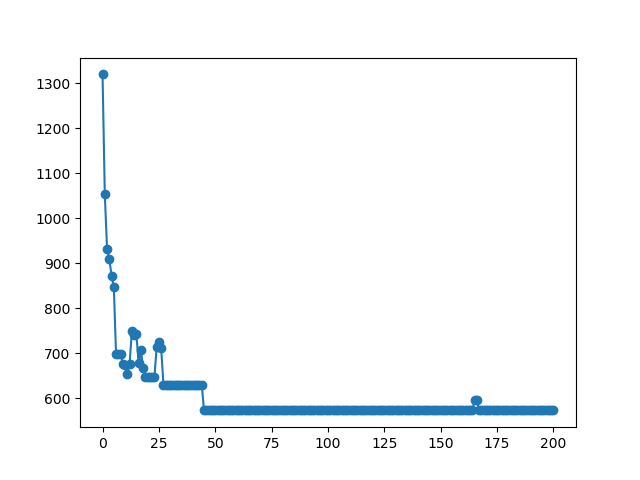
-------Path-------

[8, 3, 1, 7, 10, 6, 5, 9, 2, 4, 11]

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

416.661341258

**Random22.tsp**

****

-------------------FINAL PATH---------------------

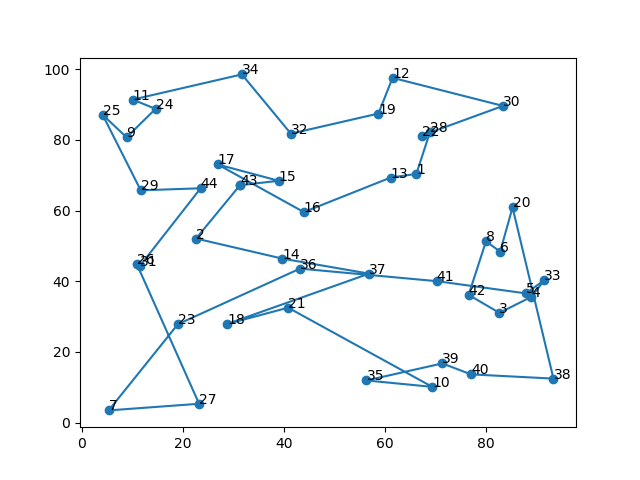
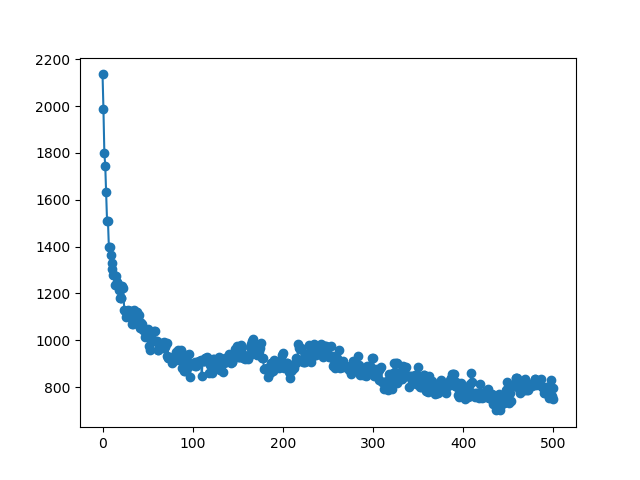
-------Path-------

[7, 12, 21, 8, 17, 11, 16, 15, 3, 5, 6, 13, 10, 19, 4, 2, 14, 9, 18, 22, 20, 1]

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

559.235224581

**Random44.tsp**



-------------------FINAL PATH---------------------

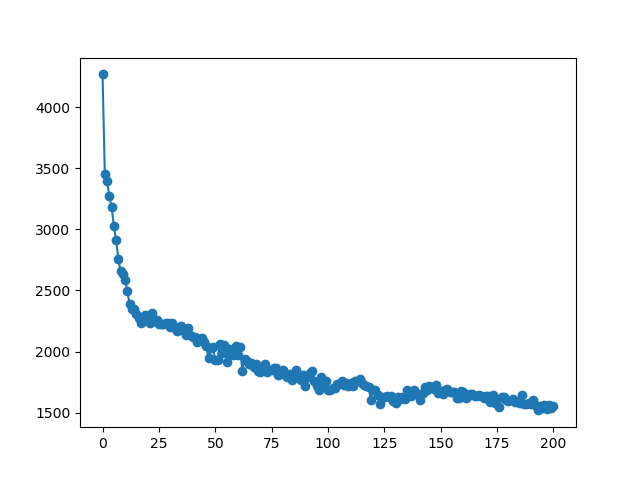
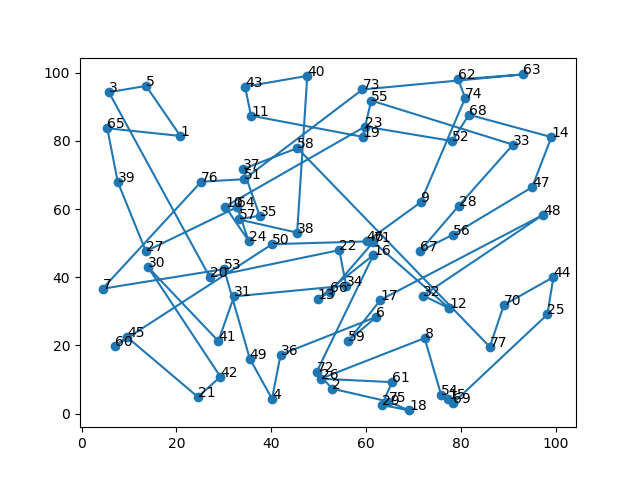
-------Path-------

[43, 15, 17, 16, 13, 1, 28, 22, 30, 12, 19, 32, 34, 11, 24, 9, 25, 29, 44, 31, 26, 27, 7, 23, 36, 41, 5, 33, 4, 3, 42, 8, 6, 20, 38, 40, 39, 35, 10, 21, 18, 37, 14, 2]

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

749.218838385

**Random77.tsp**



-------------------FINAL PATH---------------------

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

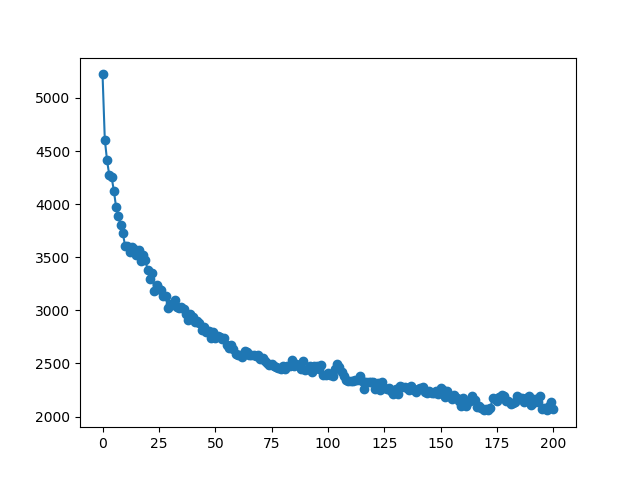
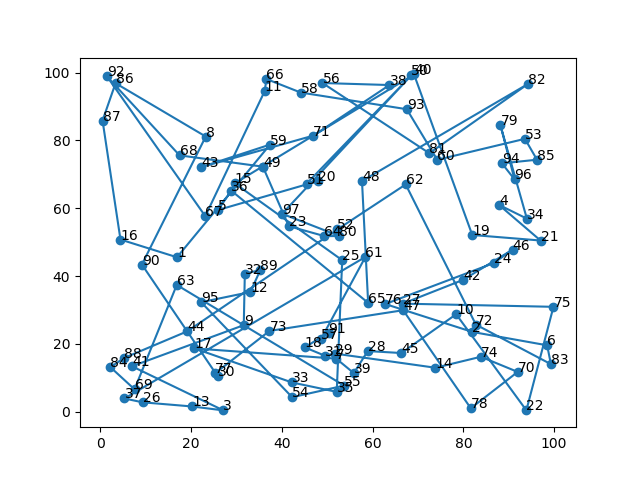
[46, 70, 12, 67, 29, 61, 2, 26, 4, 49, 42, 21, 60, 45, 27, 30, 31, 57, 34, 71, 25, 48, 44, 47, 17, 13, 41, 7, 39, 3, 5, 65, 43, 1, 55, 73, 23, 40, 11, 58, 33, 14, 68, 52, 19,

51, 76, 64, 35, 10, 38, 37, 62, 63, 74, 28, 56, 66, 50, 20, 24, 16, 22, 53, 36, 72, 59, 32, 77, 69, 15, 18, 75, 54, 8, 6, 9]

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

1472.04901789

**Random97.tsp**



-------------------FINAL PATH---------------------

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

[49, 68, 92, 67, 11, 66, 58, 93, 60, 53, 85, 94, 96, 79, 34, 4, 21, 19, 40, 15, 25, 35, 33, 17, 7, 39, 28, 45, 10, 22, 75, 27, 42, 46, 24, 76, 2, 6, 83, 72, 62, 52, 44, 88, 84, 69, 63, 55, 54, 95, 12, 89, 32, 9, 41, 3, 13, 26, 37, 61, 57, 91, 18, 31, 29, 14, 74, 70, 78, 47, 73, 77, 30, 90, 8, 86, 87, 16, 1, 59, 43, 71, 38, 56, 81, 82, 48, 65, 36, 5, 20, 51, 50, 97, 80, 64, 23]

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

2071.63360824

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

Compared to just using the GA, I noticed that my wisdom of crowd seems to run faster but didn’t get quick as close to the result as just my GA did. This was probably an error in my algorithm. For example, it only took around 4 minutes to get to an average of 1600 distance for 100 cities while just my GA took around 9 minutes running time. Although it ran quicker, it was not as effective. The size of the population definitely does matter. It matters just as much as it does when just using GA. The more cities it has to deal with, the more processing it has to do to get, breed, and mutate the population.

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

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