**CSE 545 – Artificial Intelligence – Project 4 – TSP Genetic Algorithm**

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 a Genetic Algorithm. The basis of this specific problem is like TSP in that you have a starting city, you must navigate through every city node, and return to the starting city. We were given a data set of 100 unique cities and their coordinates. To create a genetic algorithm for the TSP problem involves a little bit of creativity. It needs to have a chance of mutation, parents, and a cross over method which is basically mating between parents. You must also have a random population to start, and that population must breed, and then selectively select the most “Fit” children to bring into the next generation so that your distance improves over time and through generations. This enables you to find an optimal path for the TSP problem with the lowest cost.

**Approach**

My approach for this problem was simple as it seems to me. I began my project by bring over my TSP Reading and Calculate Distance functions from the first 2 projects, then I began work on creating a function that will create random paths through the cities while ensuring they set up a basic route to the starting city, and then storing these paths. I had it calculate the fitness of these paths by taking the inverse of the distance. Then I implemented the ordered crossover method to induce mating of parents with different offspring, along with a swap mutation method to further induce the chance of more optimal paths. However, I still wasn’t done, I had to implement other base necessities for my code such as a termination condition revolving around how many generations, I want it to progress through before a final output. I also had to ensure that the population is replaced with the new population consisting of changes and better fitness. I also needed to create a best path function that collects the best paths of each generation so I can view the change over time. Once I was able to have my code run in the terminal without issues, I then implemented tournaments and elitism to further increase the benefits to distance over time. However, I realized that as it gets closer and closer to the optimal path, it slows down drastically and has trouble finding the best path, so while I feel the genetic algorithm may be great at large sets of data and narrowing in on the best path, it seemed almost impossible to calculate the best path. I also later introduced a dynamic mutation rate that increases as distance does not improve with time.

**Results**

I did not use any libraries to do the calculations to complete the TSP GA. I instead decided to thoroughly code my whole project by hand and iterate through it systematically.

While this has drastically sped up my code in relation to other students, I also had trouble with many issues in the GA that took time to solve. I kept getting None in my children, while the parents had all 100 cities included in their path so I could not figure out where it was acquiring None from.

I also had problems with my starting and ending cities swapping even when I felt I had explicitly coded it not to do so, but somehow it was still occurring. I later solved this, and it was simply a list subtracting the length incorrectly.

I wanted the GUI to be interactable in the sense of variables and graphing. So, I used Spin boxes for many of my values so they can be freely manipulated by the user with an initial basic setting. I was unable to get my graph to update as it progressed in the terminal, and instead it views at the end, and you can view the linear change over generations as well as the changes in route (increments of 100) through generations in a second graph.

**Results – Data**

For this project, we were given a single .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 100 cities, along with a document elaborating on how we should our problem using 2 different mutation rates and crossover methods, so we end up with 4 datasets. However, I will say I had a ton of trouble in trying to get another crossover method to work, I tried doing the partially mapped crossover method, but I could not get -1(place holder) to stop appearing in my cities many generations later in some cases.

For this project, I have both two different mutation methods, and two different crossover methods, So I will do my 4 datasets all with the same mutation method, and differing crossovers as I feel that is where I was getting the most varied results.

My stopping criteria changed a few times throughout the project, I began with when distance doesn’t improve, and then changed to when distance doesn’t improve over x generations, to then simply setting the stopping criteria as a generation value so I could let me code continue to search over a long period of time.

For crossover method, we not only have a crossover rate, which I will choose to leave at 1 for purposes of this report, but we also have two methods. One being the basic ordered crossover, and the other being edge recombination. However, for my Mutation, I have both swap mutation, and inversion mutation and for this report we will only use inversion as it provided better results.

My runs began at about 10-20 seconds for 500 population and 1000 generations before I began to involve graphing and printing progress throughout generations. However, now it takes a few minutes to run through all the generations.

For my datasets I choose to set up the base with 1000 population, 1000 generations, 50 elites, and a tournament size of 25 as default. My variations are below.

|  |  |
| --- | --- |
| 1A Edge Recombination, Mutation Inversion with a rate of 0.01 | 2A Ordered Crossover, Mutation Inversion with a rate of 0.01 |
| 1B Edge Recombination, Mutation Inversion with a rate of 0.1 | 2B Ordered Crossover, Mutation Inversion with a rate of 0.1 |

My results graphs and final paths are below.

**Results – Results**

1A Edge Recombination, Mutation Inversion with a rate of 0.01

A screenshot of a computer

Description automatically generated

Generation: 999

Best Total Distance: 1129.6768693947354

Best Fitness: 0.0008852088832586132

Average: 1414.0253963292946

Median: 1329.5276978976306

Standard Deviation: 330.0925252626406

Mutation Rate: 0.01

Best Path: [29, 40, 6, 25, 60, 20, 98, 13, 56, 84, 31, 33, 64, 59, 62, 58, 42, 5, 80, 44, 82, 9, 27, 67, 88, 78, 57, 77, 26, 50, 22, 45, 52, 100, 55, 61, 96, 65, 48, 2, 76, 14, 12, 18, 47, 69, 97, 51, 81, 79, 68, 15, 43, 30, 72, 73, 95, 92, 1, 37, 7, 36, 38, 66, 21, 3, 53, 19, 91, 32, 63, 28, 39, 89, 71, 24, 85, 83, 54, 8, 10, 90, 94, 74, 17, 75, 41, 34, 46, 86, 23, 93, 70, 99, 16, 11, 49, 87, 35, 4, 29]

This is the run I expected to do the best overall as it has both a more optimal mutation and crossover method, while the other runs have a higher mutation rate which can ruin children or worse crossover method.

1B Edge Recombination, Mutation Inversion with a rate of 0.1

A screenshot of a computer

Description automatically generated

Generation: 999

Best Total Distance: 829.8132841560184

Best Fitness: 0.0012050903728506516

Average: 1006.3595379973772

Median: 856.6073971003733

Standard Deviation: 393.1665417278291

Mutation Rate: 0.1

Best Path: [87, 48, 2, 76, 14, 77, 26, 50, 78, 65, 88, 22, 67, 27, 9, 82, 57, 44, 80, 58, 5, 42, 62, 59, 64, 33, 31, 84, 29, 56, 13, 25, 40, 6, 60, 98, 20, 81, 79, 68, 15, 95, 92, 1, 71, 24, 73, 43, 30, 72, 85, 89, 94, 83, 10, 90, 16, 99, 70, 93, 86, 23, 41, 39, 17, 74, 75, 7, 36, 37, 38, 66, 21, 3, 53, 19, 91, 63, 28, 32, 45, 52, 100, 96, 55, 61, 34, 46, 49, 12, 18, 47, 11, 69, 8, 54, 51, 97, 4, 35, 87]

This totally blew me out of the water, as I did not expect the higher mutation rate to help so much. I understand the difference mutation makes, but generally with my other crossover method, lower rates were working better with swap mutation, so maybe that has changed while using inversion.

2A Ordered Crossover, Mutation Inversion with a rate of 0.01 A screenshot of a computer

Description automatically generated

Generation: 999

Best Total Distance: 1374.5326138728724

Best Fitness: 0.0007275200238300696

Average: 1642.8532442750195

Median: 1562.6127036911741

Standard Deviation: 353.877199716499

Mutation Rate: 0.01

Best Path: [38, 66, 21, 3, 53, 19, 34, 86, 23, 46, 61, 55, 32, 91, 41, 39, 17, 74, 94, 90, 49, 78, 26, 77, 2, 65, 48, 76, 14, 50, 5, 87, 59, 64, 84, 56, 33, 31, 80, 57, 9, 67, 44, 27, 22, 45, 52, 100, 28, 63, 96, 88, 82, 42, 58, 62, 29, 25, 40, 6, 60, 13, 98, 79, 68, 83, 10, 51, 54, 8, 16, 93, 70, 99, 18, 69, 11, 47, 12, 4, 35, 97, 20, 81, 15, 1, 92, 71, 24, 73, 95, 43, 72, 30, 85, 7, 89, 75, 36, 37, 38]

This one went as expected from coding these, I know that ordered crossover simply doesn’t really compare to the edge recombination.

2B Ordered Crossover, Mutation Inversion with a rate of 0.1.

A screenshot of a computer

Description automatically generated

Generation: 999

Best Total Distance: 1095.0363134754873

Best Fitness: 0.0009132117242999406

Average: 1421.397960242255

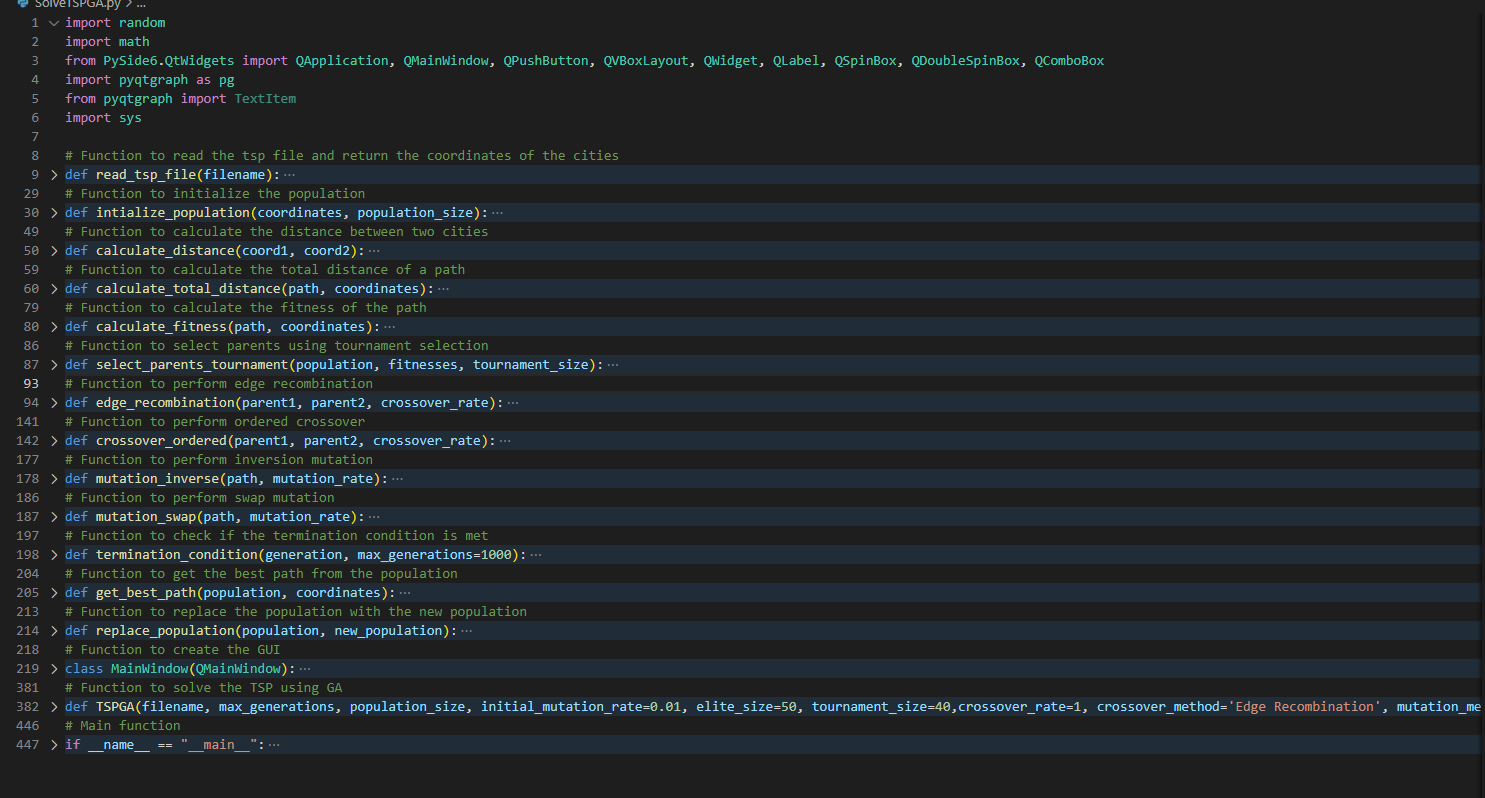
Median: 1315.6889573331243

Standard Deviation: 424.25234772766856

Mutation Rate: 0.1

Best Path: [46, 34, 19, 91, 32, 28, 63, 61, 55, 96, 100, 52, 45, 22, 57, 27, 80, 42, 5, 58, 44, 82, 9, 67, 88, 65, 48, 12, 47, 49, 2, 87, 76, 77, 26, 14, 50, 78, 62, 59, 64, 31, 35, 4, 97, 18, 11, 69, 54, 8, 84, 33, 13, 25, 6, 98, 60, 20, 40, 29, 56, 51, 81, 83, 68, 15, 79, 73, 92, 72, 30, 43, 95, 1, 71, 24, 36, 89, 85, 7, 75, 17, 74, 94, 90, 10, 70, 16, 99, 93, 41, 39, 37, 38, 21, 66, 3, 53, 23, 86, 46]

This went as expected, while it did perform better than the lower mutation rate with edge recombination, in comparison to edge recombination at the same mutation rate, it got absolutely crushed as edge recombination almost achieved the optimal path.



This is the structure of my code according to my desired functions and classes.

**Discussion**

The biggest problem I had in the implementation of this project would be the crossover method. Mutation was very straightforward both in understanding and implementation, however crossover seemed easy to understand but was hard to implement for me. I spent more time on this project than any of the others due to both the extended time and the difficulty. There were quite a few days where I did not have much other schoolwork and I spent sun up to sundown working on this project, and some of those I still walked away without the program functioning how I needed it to. It was hard for me to sit still and keep working on my crossover when method after method would not work to fix my problems. I was getting None in my children, with no None in the parent and the correct array length for children, and still available cities to take from a parent. I put break points all throughout my code trying to trace it down, and even multiple times stepped into, over and over throughout my code but still could not understand how it was missing it. I was able to fix it after rewriting much of my code around my crossover method, however it was very frustrating to have to do so.

One thing I would thing If I could do this project again would be to do it with even more time, a different GUI, and perfecting my code before working on the graphical interface as it adds a level of complexity when you try to add to the application, while it is very possible, I find it very annoying. So, if I could do it over, and I honestly most likely will as I have been placing these projects on my GitHub as I am proud of myself for doing them so well. I would take my time and plan for my end goal from the start and develop it accordingly to my difficulties which would be some of the array knowledge especially concerning large data sets and how things carry between them as this is what hung me up most in this project.

I learned a lot from the GA assignment, both my short comings and strengths, and how I can improve on them to be a better programmer. I realized mistakes in my GUI upon adding so many variables and I basically had to squeeze things together because of what I chose to use instead of rewriting my whole GUI. I feel like what I gained most from this is a deeper understanding of how important scalability can be for an application, as it has been preached to me in classes, but it is hard to understand until I hit the wall I did with this project. This is because the data set size is increasing, we are manipulating the data in new ways and multiple times, as well as making it all function together and be efficient in working towards a solution, while also being able to interpret the results. It has honestly helped me to realize how much better I can be, and why it is important TO BE better than I am now, mainly because of how much time can be wasted due to not realizing how important it was before.

My opinion of GA is that it is probably my favorite we have done all semester, I wish we could do another variation of GA before the class ends. It honestly has just enough data and manipulation included within it, that I feel that any programmer will benefit and learn from implementing it if they apply themselves. I also think it is a unique method of solving TSP, I have not used any type of GA in any other class nor heard of it before this class, and it put an interesting new spin on problem solving for me. I am very glad I took this class with you as a professor, as I think these assignments will benefit my learning and portfolio.

**References**

I have no references other than searching for solutions to my array problems and manipulation regarding my crossover method.