**Traveling Salesman Problem: Genetic Algorithm**

Jacob Taylor Cassady

Computer Engineering & Computer Science

Speed School of Engineering

University of Louisville, USA

[jtcass01@louisville.edu](mailto:NameID@louisville.edu)

# Introduction

The Traveling Salesman Problem (TSP) is a well-known non-deterministic polynomial-time hard problem that has been studied within mathematics since the 1930s. The "salesman” is given a list of cities with their locations and is asked the shortest route to travel to each city once and then return to the starting point. A program was developed using Python 3.7 and accompanying 3rd party libraries: NumPy, Pandas, and matplotlib to determine the shortest path.

# Approach

The approach taken to solving the TSP was to use a genetic algorithm. Throughout this document cities from TSP will be referred to as “vertices” and the route between a pair of vertices as an “edge”. Development of the algorithm was aided by a graphical user interface (GUI). The GUI displays the previous best chromosome as the algorithm evolves.

The genetic algorithm implemented is inspired by sexual reproduction of gametes in biology. This algorithm retains a constant population of “chromosomes” which are representations of possible solutions/agents for/within the given problem. These chromosomes are a set of alleles that describe its performance. The algorithm makes use of two functions to evolve the population overtime to weed out the poor performers and mate the good performers and continues this cycle until 10 consecutive cycles occur without improvement. For the high-level implementation of the algorithm please refer to **Figure 4** in the appendices.

## Crossover

The mating process is called “crossover”. This process involves creating a new chromosome while inheriting sections of two parent chromosomes. The resultant chromosome is then used to replace poor performers. For reference to the implementation of the crossover method from the perspective of the population, please refer to **Figure 5** in the appendices.

Two crossover methods were defined. Both methods were developed by the authors of this paper; although, they may not be novel in concept. One method alternates the alleles of both parents. This method allows for substantial differentiation between parents and their offspring. In the results section, this method uses the alias CrossOver Every Other (COEO). The second method splits the parent chromosomes in half and then creates a child that has the first half of one parent’s alleles and the second half of the other parents. In the results section, this method uses the alias Halfies. For reference to the implementation of these methods from the perspective of the parent chromosome, please refer to **Figure 6** in the appendices.

## Mutation

An additional method is used after crossover to add more randomization into the process, this method is called “mutation”. The mutation process involves randomly altering the alleles of some percentage of the population.

Only one mutation method was implemented. This method randomly chooses 2 alleles and switches their location in the chromosome. For reference to the implementation of this method, please refer to **Figure 7** in the appendices.

# Results

The genetic algorithm was successfully implemented along with its functions: crossover and mutate. The algorithm produces a route that visits each city; although, the output is not necessarily the shortest route. The algorithm did not have any issues running with only 4 GB of RAM. No mitigation techniques were needed to reduce program memory usage or improve runtime efficiency. Tests were ran using different sample sizes as well as the two different crossover functions for comparison.

## Data

The algorithm was tested on one dataset of 100 randomly generated cities. Within the test file, cities are enumerated, and x and y coordinates are provided. The input data was formatted like the example shown in **Figure 1** below. Additionally, a graphical representation of the graph of cities is shown in **Figure 2**.

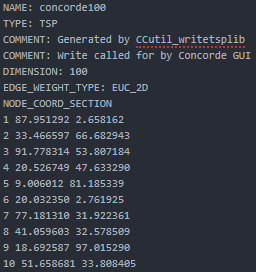


Figure 1 : Random100.tsp Input File Format

A close up of a map

Description automatically generated

Figure 2 : Graphical Representation of Random100.tsp

## Results

As previously mentioned, the algorithm was tested using two different crossover methods as well as two different population sizes. Both algorithms used a crossover probability of 60% and a mutation probability of 2%. Furthermore, costs for the algorithm over time were aggregated to keep track of how the algorithm improved. Below are tables populated with data from test runs. Graphs of the runs’ costs can be found in **Figures 8 - 16** in the appendices. These values were gathered after each cycle of crossover and mutation. Additionally, the graphical output of each run can be found in **Figures 17** - **24** in the appendices.

|  |  |  |  |
| --- | --- | --- | --- |
| COEO | | | |
| **Run #** | **Population Size** | **Runtime** | **Shortest Distance Traveled** |
| 1 | 50 | 663.846877 | 4316.454802 |
| 2 | 50 | 416.008783 | 4294.35774 |
| 1 | 100 | 1510.38222 | 4252.945025 |
| 2 | 100 | 253.966941 | 4218.685616 |

Table 1 : COEO Test Results

**Table 1** above shows the COEO crossover method applied to two different population sizes.

|  |  |  |  |
| --- | --- | --- | --- |
| Halfsies | | | |
| **Run #** | **Population Size** | **Runtime** | **Shortest Distance Traveled** |
| 1 | 50 | 58.3732955 | 4702.190834 |
| 2 | 50 | 53.1031973 | 4680.279964 |
| 1 | 100 | 100.996949 | 4597.500092 |
| 2 | 100 | 184.962564 | 4701.354649 |

Table 2 : Halfsies Test Results

**Table 2** above shows the Halfsies crossover method applied to two different population sizes.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Crossover Method** | **Population Size** | **min** | **max** | **average** | **std\_deviation** |
| COEO | 50 | 4294.358 | 4316.455 | 4305.4063 | 15.62498238 |
| COEO | 100 | 4218.686 | 4252.945 | 4235.8153 | 24.22506042 |
| Halfies | 50 | 4680.28 | 4702.191 | 4691.2354 | 15.49332476 |
| Halfies | 100 | 4597.5 | 4701.355 | 4649.4274 | 73.43626151 |

Table 3 : Crossover Method Performances

**Table 3** above shows a comparison between the two tested crossover methods as well as population sizes. The min, max, average, and standard deviation columns are with respect to the distance traveled throughout the route.

# Discussion

## Performance

Output performance for population sizes of 50 and 100 produced very similar averages. This may be because 50 is already many chromosomes. On the other hand, the two crossover methods performed very differently from each other although the crossover probability, mutation probability, and mutation methods were all the same.

COEO produced cost graphs that monotonically decreased over the series of cycles. On the other hand, Halfies never improved its distance traveled. This may be a consequence of the relative higher level of entropy of COEO when compared to Halfies. It does less to preserve the order of alleles between the two parents than Halfies does. In conclusion, COEO is the preferred crossover method between the two.

## Time

Although some runs produced similar results, the runtime of the genetic algorithm is variable. Some runs might never produce an improvement due to the luck of the random chromosomes generated and thus finish much faster than other. Others might make small improvements after several cycles of no improvements dragging out the runtime.

## Issues

An issue when designing the algorithm was coming up with different crossover methods while attempting to choose the best one for the problem. I ended up choosing the two methods that were the most obvious to me, but a lot of time was spent trying to determine which of those available would be optimal for this problem. If I could change one thing in the implementation, I would add a more descriptive GUI that contained graphs of each member of the population as they evolved. This would help visualize the differences in how the crossover methods behave.

Genetic Algorithms are not deterministic and may produce less than optimal results on the first run. They can be great tools for attempting to solve problems there is not a defined solution for, but they are not the best tools for solving problems that have more deterministic solutions. Genetic Algorithms might be a great choice for implementing a program that works towards a solution when you don’t readily know the next step to take. Especially if it involves a large solution search space, a well-defined fitness function, and a discrete set of alleles.

# References

Wikipedia, Traveling Salesman Problem - <https://en.wikipedia.org/wiki/Travelling_salesman_problem#History>

NumPy Documentation - <https://docs.scipy.org/doc/>

Pandas Documentation - <https://pandas.pydata.org/pandas-docs/stable/>

Matplotlib Documentation - <https://matplotlib.org/3.1.1/contents.html>

# Appendix

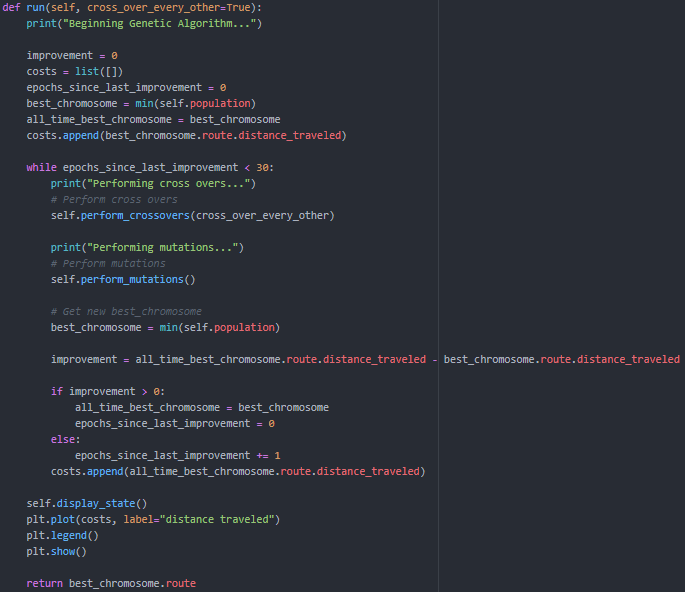


Figure 3 : Genetic Algorithm High Level Function

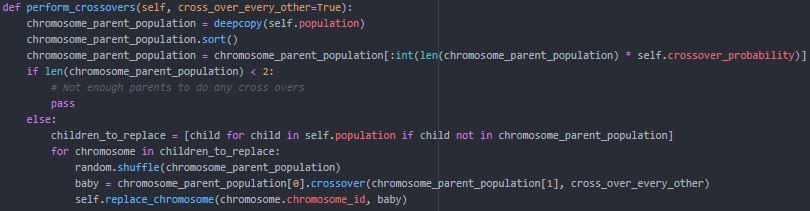


Figure 4 : Perform Crossover Method (From perspective of population)

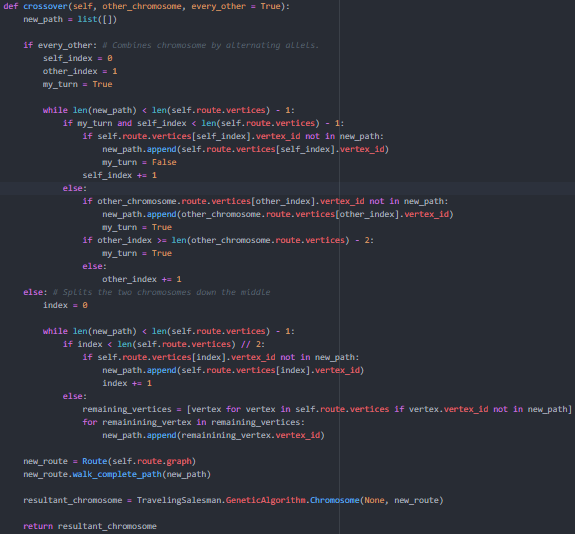


Figure 5 : Crossover Method (from the perspective of a parent chromosome)

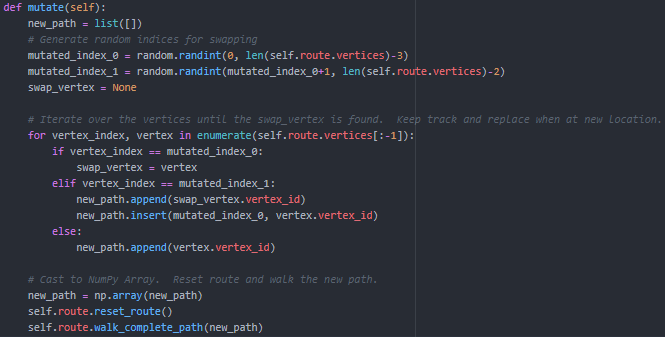


Figure 6 : Mutate Method

A screenshot of a cell phone

Description automatically generated

Figure 7 : COEO 50 Population Costs Run 1

A screenshot of a social media post

Description automatically generated

Figure 8 : COEO 50 Population Costs Run 2

A screenshot of a cell phone

Description automatically generated

Figure 9 : COEO 100 Population Costs Run 1

A screenshot of a social media post

Description automatically generated

Figure 10 : COEO 100 Population Costs Run 2

A screenshot of a cell phone

Description automatically generated

Figure 11 : Halfsies 50 Population Costs Run 1

A screenshot of a cell phone

Description automatically generated

Figure 12 : Halfsies 50 Population Costs Run 1

A screenshot of a cell phone

Description automatically generated

Figure 13 : Halfsies 50 Population Costs Run 2

A screenshot of a cell phone

Description automatically generated

Figure 14 : Halfsies 100 Population Costs Run 1

A screenshot of a cell phone

Description automatically generated

Figure 15 : Halfsies 100 Population Costs Run

A close up of a map

Description automatically generated

Figure 16 : COEO 50 Population Output Run 1

A close up of a map

Description automatically generated

Figure 17 : COEO 50 Population Output Run 2

A close up of a map

Description automatically generated

Figure 18 : COEO 100 Population Output Run 1

A close up of a map

Description automatically generated

Figure 19 : COEO 100 Population Output Run 2

A close up of a map

Description automatically generated

Figure 20 : Halfsies 50 Population Output Run 1

A close up of a map

Description automatically generated

Figure 21 : Halfsies 50 Population Output Run 2

A close up of a map

Description automatically generated

Figure 22 : Halfsies 100 Population Output Run 1

A close up of a map

Description automatically generated

Figure 23 : Halfsies 50 Population Output Run 2