**Traveling Salesman Problem: Wisdom of Crowds Using Genetic Algorithms**

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 the ‘wisdom of crowds’ principle along with a series of differing genetic algorithms. A population of 180 chromosomes were split evenly between 6 genetic algorithm variations. The chromosomes were represented as a series of alleles. Throughout this paper, alleles and vertices will be used interchangeably. Edges will be used to refer to adjacent pairs of alleles.

The genetic algorithm variations were generated from cross joining the mutation methods and crossover methods described in sections 2.1.1 and 2.1.2 of this document. The genetic algorithms were run until their population had 25 consecutive generations without improvement. A generation was defined as a series of both performing cross over and then performing mutation. A crowd was then generated by taking a percentage of each algorithm’s population with best performance.

To develop an aggregate answer from the crowd of chromosomes, the crowd was examined to determine common consecutive pairs of alleles by calculating the relative frequency of each edges. A threshold was then used to determine the required number of recurrences of the same edge required throughout the crowd for the edge to be included in the aggregate answer. If two edges were found frequently and included and same starting vertex or the same ending vertex, the edge with the smallest distance was kept and the edge with the longer distance was discarded. If the resultant graph was not complete, a greedy heuristic was used to select the nearest unvisited vertex from an edge already included. The greedy heuristic algorithm is explained in more detail in section 2.3 of this document.

## Genetic Algorithm

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.

### Crossover Methods

The implemented crossover methods have been shown to improve performance in a genetic algorithmic approach to TSP (ABDOUN & ABOUCHABAKA, 2011). Each algorithm was run with an 80% crossover probability, meaning that with each generation the top 20% of the population was used to generate replacements for the bottom 80%.

#### Uniform Crossover

The uniform crossover forms a child by randomly alternating between the two parents. For reference to the implementation of this method please see **Figure 6** in the appendix.

#### Ordered Crossover

The ordered crossover breaks each parent into three sequences, S1, S2, and S3 with matching indices for both parents. The child is then produced by taking S2 from one parent and filling S1 and S3 with alleles from the other parents starting at S1 and leaping genes already included. For reference to the algorithm as formalized in the literature, please refer to **Figure 1** below. For refence to the implementation of this method please see **Figure 8** in the appendix.

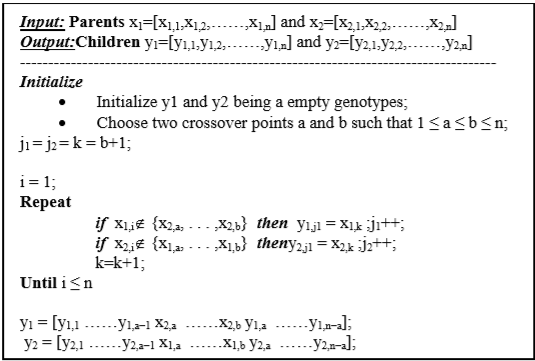


Figure 1 : Ordered Crossover Algorithm (ABDOUN & ABOUCHABAKA, 2011)

#### Partially Mapped (PM)

The partially mapped crossover breaks each parent into three sequences, S1, S2, and S3 with matching indices for both parents. The child is then produced by taking S1 and S3 from one parent and filling in S2 with alleles from the other parent starting at S2 and leaping genes already included. For reference to the algorithm as formalized in the literature, please refer to **Figure 2**. For reference to the implementation of this method please see **Figure 7** in the appendix.

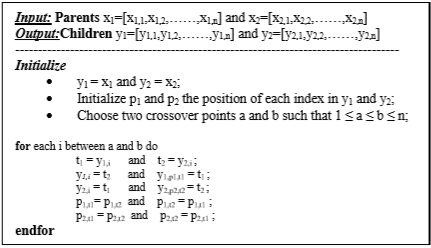


Figure 2 : Partially Mapped Algorithm (ABDOUN & ABOUCHABAKA, 2011)

### Mutation Methods

The implemented mutation methods have been shown to improve performance in a genetic algorithmic approach to solving TSP (ABDOUN & ABOUCHABAKA, 2011). All genetic algorithms were run with a mutation rate of 2%, meaning that each generation had 2% of its chromosomes undergo mutation.

#### TWORS

The TWORS mutation method randomly swaps two alleles’ locations within the chromosome. For reference to the implementation of this method, please see **Figure 9** in the appendix of this document.

#### Reverse Sequence (RSM)

The Reverse Sequence mutation method reverses the sequence of the chromosome. For reference to the implementation of this method, please see **Figure 10** in the appendix of this document.

## GUI

A GUI was developed to visualize different stages of the algorithm. A heat map was developed to understand the crowd’s edge frequency. Additionally, a route solution representation was generated to ensure proper connection of the final path.

### Heat Map

Heat maps were generated to help understand the crowd’s edge frequency. The edges were plotted with their RGB values denoting its frequency within the crowd. The most red colored edges are those that occur least frequent, while the most blue colored edges are those that occur most frequent. Please refer to **Figure 3** below for an example of a heat map generated from the edges with an occurrence rate in the top 80% for a set of 44 cities.

A close up of a map

Description automatically generated

Figure 3 : Heat Map Random44.tsp 20% Superiority Threshold

### Route Solution

A graphical representation of the final solution was generated to ensure it is reasonable. Each edge is colored using its vertices’ IDs to quantize its red and blue color magnitudes while the green magnitude is calculated from the modulus of the starting vertex id with respect to the ending vertex id. For reference to the implementation of this plotting method, please refer to **Figure 4** in the appendix.

A picture containing indoor, object, sky

Description automatically generated

Figure 4 : Route Solution Random44.tsp

## Greedy Heuristic

As noted above, to combine the solutions of the crowd of genetic algorithms the algorithm creates a dictionary of edges across the entire crowd and keeps track of the frequency of each edge. Edges that meet some predetermined “superiority threshold” are kept to develop a fragmented graph. For edges that contain the same vertex, the edge with the highest edge count and lowest distance traveled is retained. The relevant code for creating the fragmented graph is shown in **Figure 12** in the appendix of this document.

Once a recombination route is generated, there could still be some stray vertices. The remaining vertices are iterated over and the nearest vertex to an existing route segment is chosen and “lassoed” into the route segment. For reference to the route’s lasso function please refer to **Figure 13** in the appendix of this document. For reference to the method for choosing the next vertex to insert into the group of route segments, please refer to **Figure 14** in the appendix of this document.

Lastly, once all vertices are part of a route segment, a recombine method is used to connect the route segments into a contiguous route. This is done by iterating over the unvisited starting indices and connecting them in order as they appear in the list of edges generated from the lassoing of vertices. For reference to the implementation of this segment recombination method, please refer to **Figure 15** in the appendix of this document.

# Results

Wisdom of Crowds with Genetic Algorithms was successfully implemented to improve upon the approximation of an optimal solution for the Traveling Salesman Problem. There was an issue with Python’s copy library’s deepcopy method randomly producing errors when dealing with city datasets larger than 44. No mitigation techniques were successful in circumventing this issue to allow for proper testing on larger datasets. This seems to be an issue with the implementation of the described algorithms not the algorithms themselves.

## Data

The algorithms were tested on different datasets ranging from 6 cities to 222. The dataset files were generated randomly. 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 5** below.

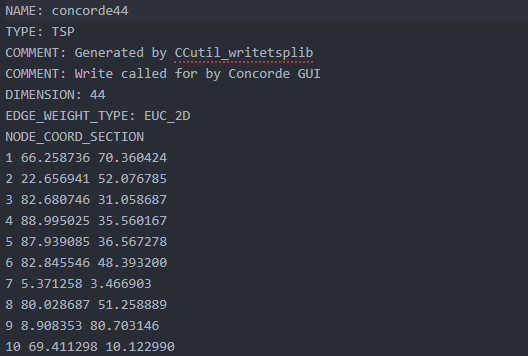


Figure 5: Random44.tsp Input File Format

## Results

Due to the memory issue described above, test results were only gathered for datasets of sizes 6 cities to 44 cities. Successful runs were achieved on each of the datasets just described but tables were only developed for specific tests that highlight the strengths and weaknesses of the wisdom of crowd’s solution purposed in this document.

To use as a benchmark in comparison with the wisdom of crowd’s solution, tests were run using different variations of crossover methods and mutation methods. Each test used the same population size, mutation probability, crossover probability, and epoch threshold (number of epochs without improvement required for completion). For reference to these tests, please refer to **Table 1**.

Table 1 : Genetic Algorithm Individual Results Random22.tsp

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **mutation method** | **mutation prob** | **crossover method** | **crossover prob** | **epoch threshold** | **pop size** | **run time** | **distance traveled** |
| TWORS | 0.02 | Uniform | 0.8 | 25 | 300 | 44.2538182 | 390.858881 |
| RSM | 0.02 | Uniform | 0.8 | 25 | 300 | 24.2687368 | 381.435449 |
| TWORS | 0.02 | Ordered | 0.8 | 25 | 300 | 98.2373263 | 351.045879 |
| RSM | 0.02 | Ordered | 0.8 | 25 | 300 | 85.3636126 | 351.045879 |
| TWORS | 0.02 | PM | 0.8 | 25 | 300 | 51.3056948 | 358.410893 |
| RSM | 0.02 | PM | 0.8 | 25 | 300 | 64.5693509 | 357.623412 |
| TWORS | 0.02 | Uniform | 0.8 | 25 | 300 | 11.7486071 | 389.308905 |
| RSM | 0.02 | Uniform | 0.8 | 25 | 300 | 33.0446600 | 402.009151 |
| TWORS | 0.02 | Ordered | 0.8 | 25 | 300 | 81.4779670 | 351.045879 |
| RSM | 0.02 | Ordered | 0.8 | 25 | 300 | 31.6496074 | 363.384024 |
| TWORS | 0.02 | PM | 0.8 | 25 | 300 | 49.4857442 | 359.547153 |
| RSM | 0.02 | PM | 0.8 | 25 | 300 | 38.0424134 | 351.919141 |
| TWORS | 0.02 | Uniform | 0.8 | 25 | 300 | 18.7359197 | 405.234845 |
| RSM | 0.02 | Uniform | 0.8 | 25 | 300 | 12.7101366 | 397.897589 |
| TWORS | 0.02 | Ordered | 0.8 | 25 | 300 | 58.9454748 | 351.045879 |
| RSM | 0.02 | Ordered | 0.8 | 25 | 300 | 108.881925 | 351.045879 |
| TWORS | 0.02 | PM | 0.8 | 25 | 300 | 40.0469241 | 367.290029 |
| RSM | 0.02 | PM | 0.8 | 25 | 300 | 62.7982237 | 358.410893 |
| TWORS | 0.02 | Uniform | 0.8 | 25 | 300 | 11.1864941 | 396.749793 |
| RSM | 0.02 | Uniform | 0.8 | 25 | 300 | 29.0962438 | 376.928687 |
| TWORS | 0.02 | Ordered | 0.8 | 25 | 300 | 82.7828438 | 351.045879 |
| RSM | 0.02 | Ordered | 0.8 | 25 | 300 | 108.139257 | 351.045879 |
| TWORS | 0.02 | PM | 0.8 | 25 | 300 | 44.8761613 | 357.623415 |
| RSM | 0.02 | PM | 0.8 | 25 | 300 | 53.0247418 | 359.5471535 |

The average and standard deviation of these tests are summarized in **Table 2** below.

Table 2 : Genetic Algorithm Aggregate Results Random22.tsp

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **mutation method** | **crossover method** | **population size** | **average run time** | **run time std dev** | **average distance traveled** | **distance traveled std dev** |
| TWORS | Uniform | 300 | 21.48120981 | 15.5652716 | 395.5381062 | 7.215562601 |
| RSM | Uniform | 300 | 24.77994436 | 8.810553496 | 389.5677196 | 12.24819609 |
| TWORS | Ordered | 300 | 80.36090302 | 16.17921202 | 351.0458799 | 0 |
| RSM | Ordered | 300 | 83.50860077 | 36.25498182 | 354.130416 | 6.169072199 |
| TWORS | Partially Mapped | 300 | 46.42863113 | 5.042090759 | 360.7178729 | 4.452027794 |
| RSM | Partially Mapped | 300 | 54.60868251 | 12.15502145 | 356.8751509 | 3.397058941 |

4 Tests were run on the same file using the Wisdom of Crowds algorithm described in this paper. A table of the results of this test can be found in **Table 3**. The average runtime for these tests was calculated to be 38.8924s with a standard deviation of 6.1338. The distance traveled was on average 372.905 with a relatively large standard deviation of 111.0513. Please note that all tests were ran with a superiority tolerance of 80%.

Table 3 : Wisdom of Crowds Random22.tsp Test

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test Number** | **mutation prob** | **crossover prob** | **population size** | **superiority tolerance** | **run time** | **distance traveled** |
| 0 | 0.02 | 0.8 | 300 | 0.8 | 32.63268375 | 446.9994329 |
| 1 | 0.02 | 0.8 | 300 | 0.8 | 44.9956038 | 243.6437027 |
| 2 | 0.02 | 0.8 | 300 | 0.8 | 43.23842049 | 319.0103765 |
| 3 | 0.02 | 0.8 | 300 | 0.8 | 34.70307946 | 481.9648363 |

Lastly, a test was run to highlight the consequence of different values of superiority tolerance. This test was done on the file Random44.tsp which contained 44 different vertices. An algorithm with the name GA\_TWORS\_UNIFORM refers to a genetic algorithm using a mutation method of TWORS and a uniform crossover method. For reference to the results of this test, please refer to **Table 4**. It is important to note that the genetic algorithms listed here have a chromosome population of 50 while the WOC algorithm has a chromosome population of 300 with 6 different genetic algorithms each having a population of 50.

Table 4 : Wisdom of Crowds Random44.tsp Results

|  |  |  |
| --- | --- | --- |
| **Algorithm** | **run time** | **distance traveled** |
| GA\_TWORS\_UNIFORM | 80.64828849 | 1841.023982 |
| GA\_RSM\_UNIFORM | 68.96369147 | 1884.60208 |
| GA\_TWORS\_ORDERED\_CROSSOVER | 291.1973372 | 1337.673257 |
| GA\_RSM\_ORDERED\_CROSSOVER | 479.8008478 | 1040.436384 |
| GA\_TWORS\_PARTIALLY\_MAPPED | 111.3197362 | 1827.739158 |
| GA\_RSM\_PARTIALLY\_MAPPED | 90.96677256 | 1607.509943 |
| WOC [0.8 Superiority] | 758.8008875 | 987.178483 |
| WOC [0.6 Superiority] | 1136.702333 | 1306.929875 |
| WOC [0.2 Superiority] | 754.098 | 1509.5378 |

# Discussion

There are two main findings I would like to highlight from the tests described in section 3 of this document. The results show the wisdom of crowds solution does produce a more optimal solution than any individual genetic algorithm when the relative number of chromosomes for any given genetic algorithm is sufficiently small with respect to the number of vertices in the graph. Secondly, the results show that a higher superiority threshold leads to a lower resultant distance traveled while not having much of an effect on the run time.

As shown in **Tables 2 and 3**, the wisdom of crowds solution does produce a more optimal solution than any individual genetic algorithm when the relative number of chromosomes for any given genetic algorithm is sufficiently small with respect to the number of vertices in the graph. When both the wisdom of crowd’s solution and the genetic algorithm have access to 300 chromosomes, the wisdom of crowds algorithm does not perform better. This is likely to do with the epoch threshold more easily being met in the WOC’s smaller individual populations. As a consequence, the genetic algorithms have more of a chance to notice a positive change in their population. This is further highlighted in **Table 4** where the genetic algorithms have an equal number of chromosomes to anyone genetic algorithm in the WOC’s crowd.

As shown in **Table 4**, the wisdom of crowds solution does start to perform better when the dataset size is larger and a superiority threshold is utilized. Using a larger superiority threshold does lead to more optimal results even achieving better results than the best genetic algorithm on its own. The WOC solution does take longer to converge; this could be a consequence of the number of cores on my computer as each genetic algorithm in WOC’s crowd is running in parallel. It could also be a consequence of the overhead added from recombination.

# References

ABDOUN, O., & ABOUCHABAKA, J. (2011, October). A Comparative Study of Adaptive Crossover Operators for Genetic Algorithms to Resolve the Traveling Salesman Problem. *International Journal of Computer Applications, 31*(11). Retrieved from https://arxiv.org/ftp/arxiv/papers/1203/1203.3097.pdf

Baraglia, R., Hidalgo, J. I., & Perego, R. (2001, December). A Hybrid Heuristic for the Traveling Salesman Problem. *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, 5*(6), 613-622. doi:10.1109/4235.974843

Yi , S. M., Steyvers , M., Lee, M. D., & Dry , M. J. (2011). Wisdom of the Crowds in Traveling Salesman Problems.

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

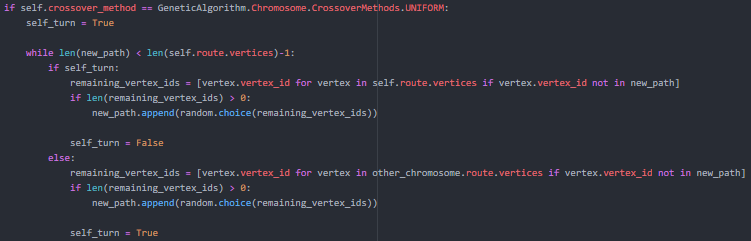


Figure 6 : Uniform Crossover Method

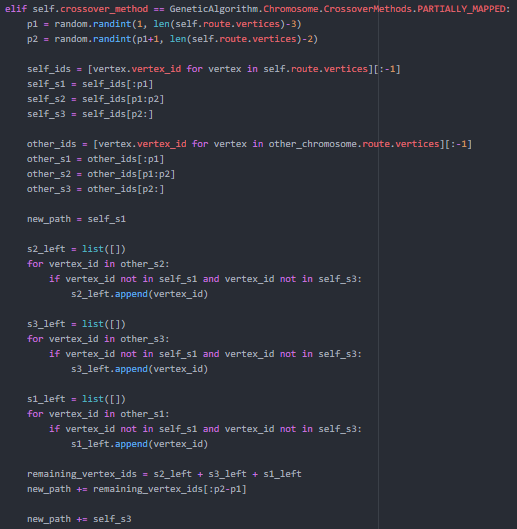


Figure 7 : Partially Mapped Crossover Method

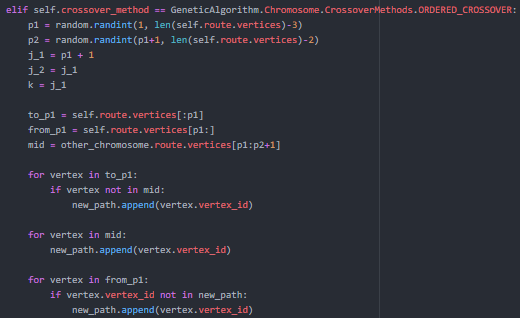


Figure 8 : Ordered Crossover Method

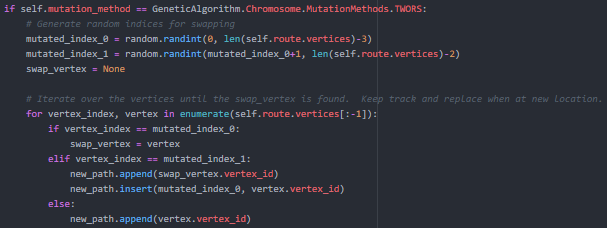


Figure 9 : TWORS Mutation Method



Figure 10 : Reverse Sequence Mutation Method

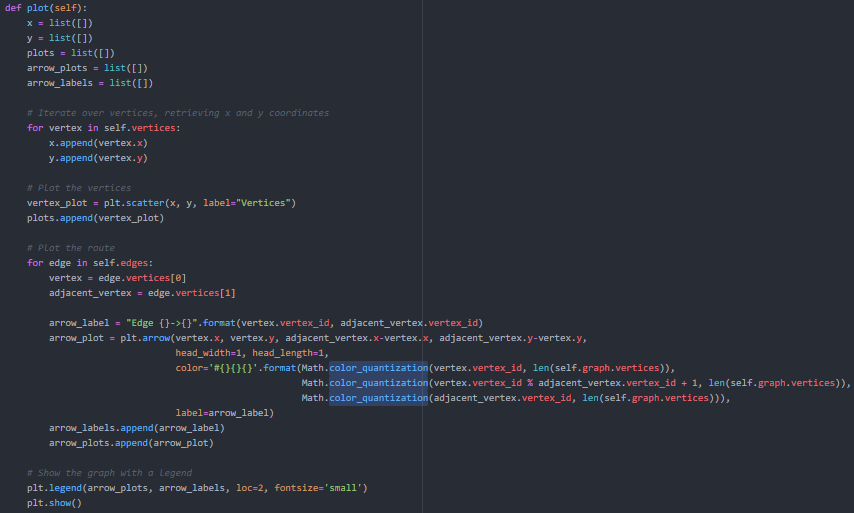


Figure 11 : Route Solution Plotting Method

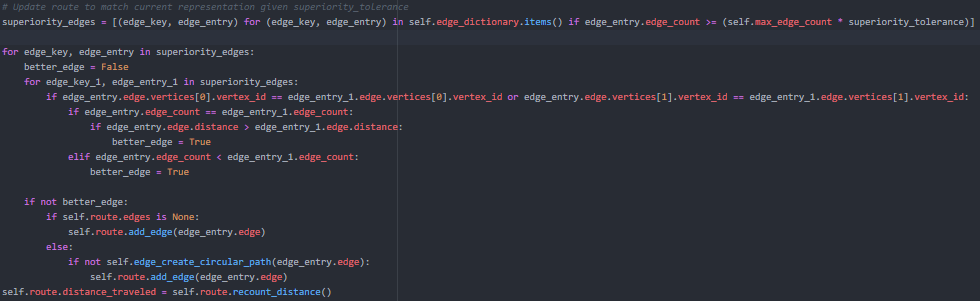


Figure 12 : Relevant Code Route Recombination

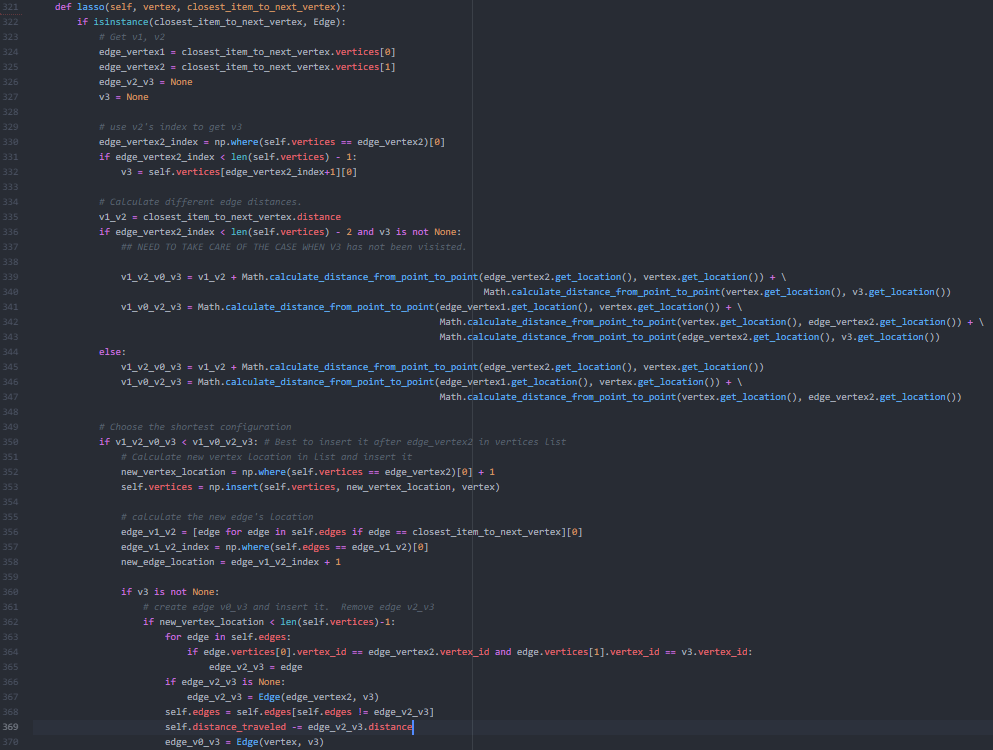




Figure 13 : Route Lasso Method

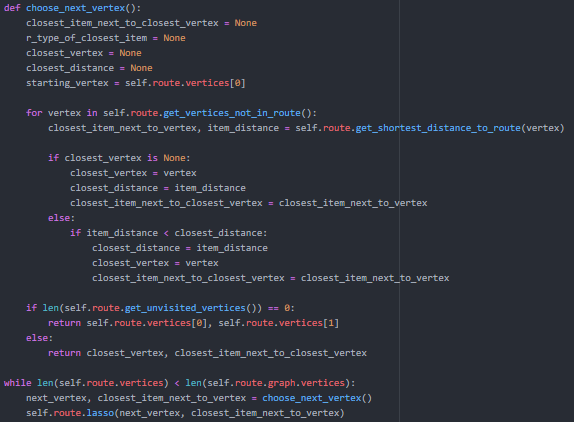


Figure 14 : Relevant Code Greedy Choose Next Vertex

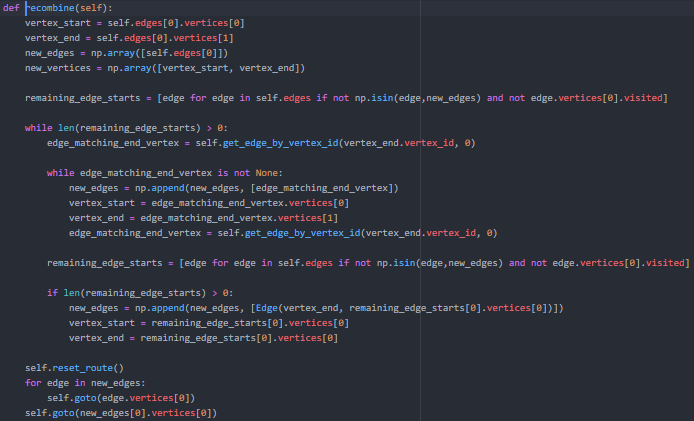


Figure 15 : Segment Recombine Method