An Evolutionary Algorithm to Solve the Travelling Salesman Problem

Submitted to Dr. Ting Hu

Submitted by:

Tingrui Hu – 201513025

Weiming Chen – 201504727

Justin Heffernan – 201526514

For Computer Science 3201 – Intro to Nature-Inspired Computing

Due Date: December 10th, 2018

Introduction

The Travelling Salesman Problem (often referred to as TSP) is a classic algorithmic problem in the field of computer science and mathematics. The problem focuses on the optimization of the path travelled by a salesman in his list of cities he must visit. In mathematics the problem and solution are often represented in the form of a graph with the nodes being the cities in which the salesman must visit on his trip.

The problem itself describes a salesman who must travel between N cities. The order in which he does this is something that is irrelevant to the problem, if he only visits each city on his list once in the entirety of the trip and finishes where he began. The cost of moving between cities can be calculated by measuring the distance between the two respective cities. Using this definition, the salesman wants to keep cost and time down as he plans his route.

The Travelling Salesman Problem is classified as a "hard" optimization problem by most computer scientists and mathematicians. This can be blamed on the exponential growth of the computation as the data set, i.e. number of cities, grows with each city that is added. The number of solutions that are present for any given number of cities can be represented by

$$\frac{(N-1)!}{2}$$

due to the need to visit each city once. The number of possibilities is N-1 because the start city is defined and therefore there can only be permutations on the remaining cities. The need to only count half is since each route has an equal route in reverse with the same total length of the trip.

The way the problem will be represented and solved in this report is using an evolutionary algorithm. The goal of the project is to design and implement an evolutionary algorithm that given the list of cities, represented in coordinate format, find the optimal solution for the salesman to travel in order to keep costs down. The project will focus mainly on two separate data sets, the first being the Western Sahara (29 cities) and the second being Uruguay (734 cities). The project will also use advanced methods to increase the effectiveness of the algorithm as well as use runtime optimizations to help increase the speed in which the program will run. For data representation a graphical user interface (GUI) was created to help visual the progression through the algorithm as it tries to find the most optimal path. Within the GUI the user can see the progression of the path being displayed as well as multiple graphs that compare the completeness of that given solution.

Methods

The methods that were used in the construction of the algorithm can broken down into parent selection, crossover, mutation, and run-time optimizations. Each one of these three parts plays a crucial role in the effectiveness of the algorithm in finding the optimal path for the salesman to use.

Parent Selection

The parent selection method that was used is tournament selection. In this method the individuals are selected by running several tournaments among few individuals chosen at random from the population. Once the individuals are selected, their fitness are compared with each other and the individual with the best is selected for crossover. The image shown in figure 1 below shows how the algorithm will select the parents to be used for crossover given a random sample from the population.

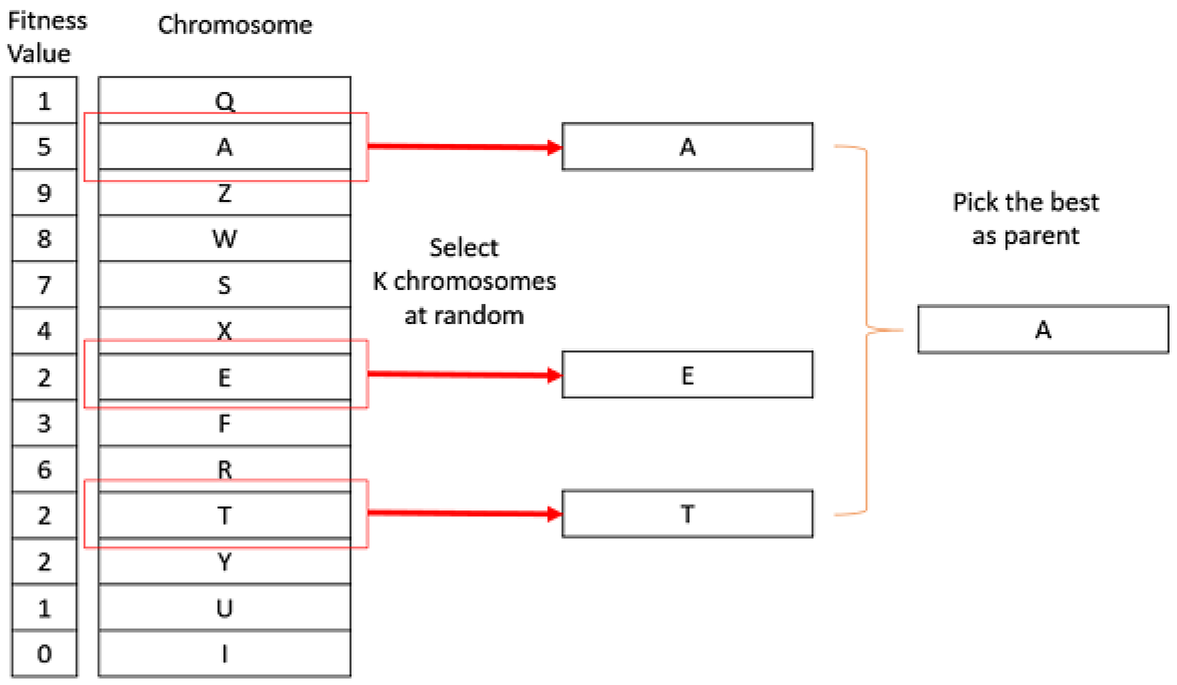


Figure 1

Tournament selection was selected because of the efficient and ease to code, along with the ability to change the selection pressure with ease. Selection pressure is a measure of a chromosome's likelihood of participation in the tournament based on the selection pool size. This method was used with a tournament size of 3 because it resulted in an appropriate selection pressure for the TSP. If it was decided that a higher pressure was needed it would be a simple change of variable to implement. The tournament selection will also ensure that the fittest individuals are selected for crossover, which in turn, increased the efficiency of the algorithm as the potential for less generations to reach the optimal solution is lower by taking the best of each tournament.

Crossover

The crossover method used in the project was order crossover. When first laying out the basis for the algorithm, it was decided that an advanced method would be used here. In the first implementation of the project we used a method called cut on worst gene crossover. This method was implemented by first finding the worst gene in the first and second parents, then using the worst gene's city ID of parent one as the cut point for parent one and then the same for parent two. Once these two were selected a comparison would be made to the distances such that if distance one was greater than distance 2 then apply the modified crossover in both parents at the index that have the larger distance, else apply the modified crossover in both parents at index four. Once this method was tested it was found that the algorithm was getting caught on local maximum and minimums causing it to converge too early before finding the optimal route.

After further testing it was decided that a new crossover method would have to be used to ensure diversity and avoid convergence. After testing some more methods, it was finalized that the order crossover would be used as it preserved diversity and avoid convergence best. The images seen below in figure 2 and 3 help visualized how the crossover was completed in order to ensure each city only appeared once in the children. Figure 4 then shows how the order crossover was implemented in Python.

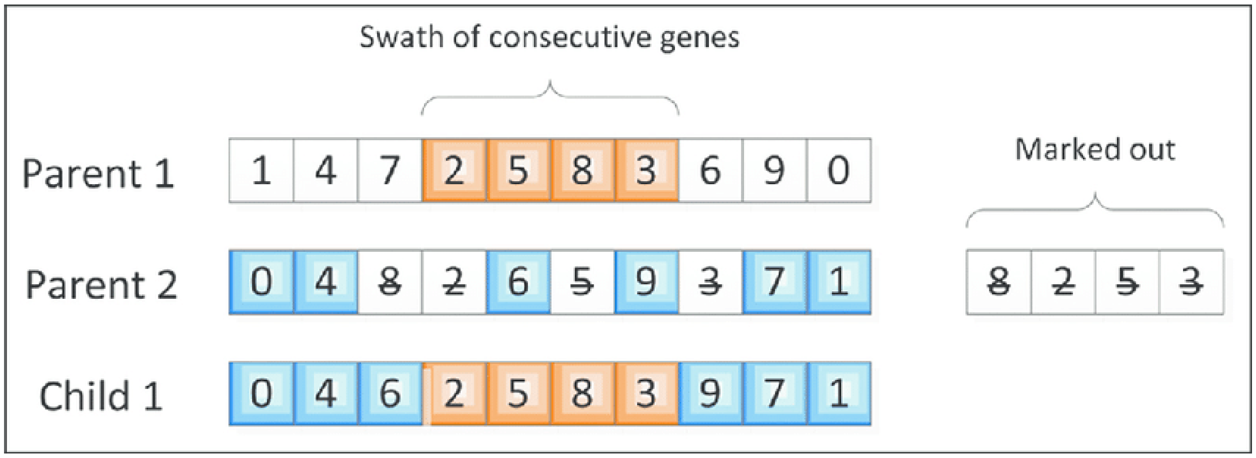


Figure 2

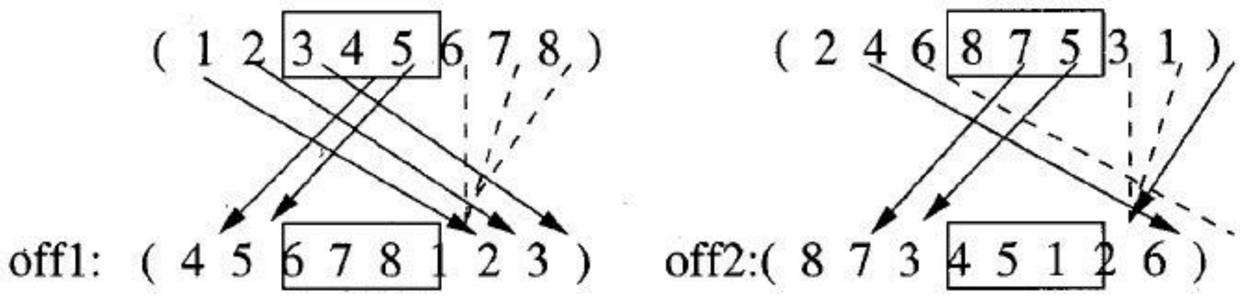
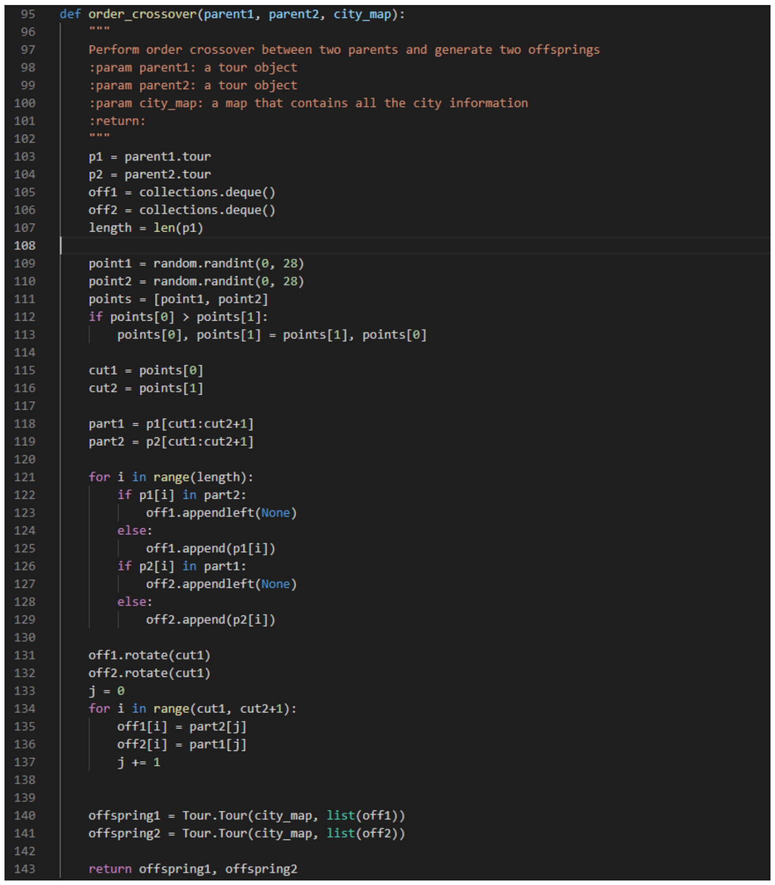


Figure 3



```python
def order_crossover(parent1, parent2, city_map):
    """
    Perform order crossover between two parents and generate two offsprings
    :param parent1: a tour object
    :param parent2: a tour object
    :param city_map: a map that contains all the city information
    :return:
    """
    p1 = parent1.tour
    p2 = parent2.tour
    off1 = collections.deque()
    off2 = collections.deque()
    length = len(p1)

    point1 = random.randint(0, 28)
    point2 = random.randint(0, 28)
    points = [point1, point2]
    if points[0] > points[1]:
        points[0], points[1] = points[1], points[0]

    cut1 = points[0]
    cut2 = points[1]

    part1 = p1[cut1:cut2+1]
    part2 = p2[cut1:cut2+1]

    for i in range(length):
        if p1[i] in part2:
            off1.appendleft(None)
        else:
            off1.append(p1[i])
        if p2[i] in part1:
            off2.appendleft(None)
        else:
            off2.append(p2[i])

    off1.rotate(cut1)
    off2.rotate(cut1)
    j = 0
    for i in range(cut1, cut2+1):
        off1[i] = part2[j]
        off2[i] = part1[j]
        j += 1


    offspring1 = Tour.Tour(city_map, list(off1))
    offspring2 = Tour.Tour(city_map, list(off2))

    return offspring1, offspring2
```

Figure 4

When tested, order crossover then proved to be the more efficient and persevering diversity in the algorithm for the Travelling Salesman Problem. It also in turn resulted in each city only appearing once after crossover as there was a checking done as seen in figure 2 to see if the city selected for the crossover was already in the other parent and if so marked it out.

Mutation

After the failure of an advanced method in the crossover portion of the algorithm, we used an advanced technique in the mutation portion. The advanced method that was used in the algorithm was the use of worst gene with worst gene mutation. In this method the algorithm will take the worst gene from each individual chromosome and once the worst gene is selected from both, they are swapped the from their relative positions with each other. This can be shown in figure 5 below.

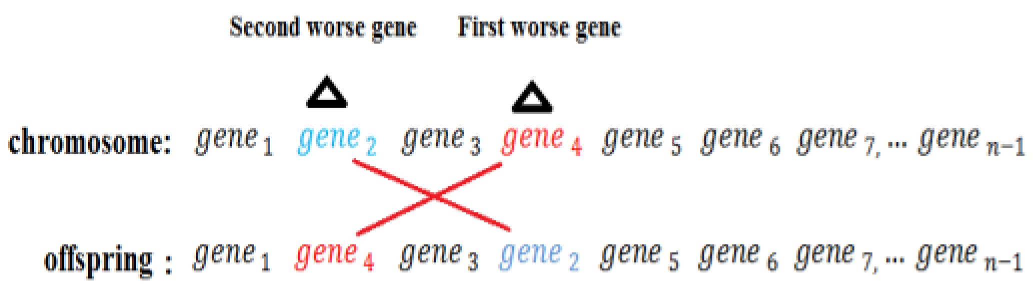

Second worse gene
First worse gene
chromosome: $gene_1$ $gene_2$ $gene_3$ $gene_4$ $gene_5$ $gene_6$ $gene_{7,}$ ... $gene_{n-1}$
offspring : $gene_1$ $gene_4$ $gene_3$ $gene_2$ $gene_5$ $gene_6$ $gene_{7,}$ ... $gene_{n-1}$

Figure 5

Once tested against other mutation methods such as swap and inversion mutation it showed that the method was the fastest and most efficient of the methods in reaching the optimal route for the salesman.

One other mutation method that was tried was worst gene with random gene mutation. In this method the worst gene is selected from the chromosome and then a random gene is selected to be swapped with. This can be illustrated in figure 6 below.

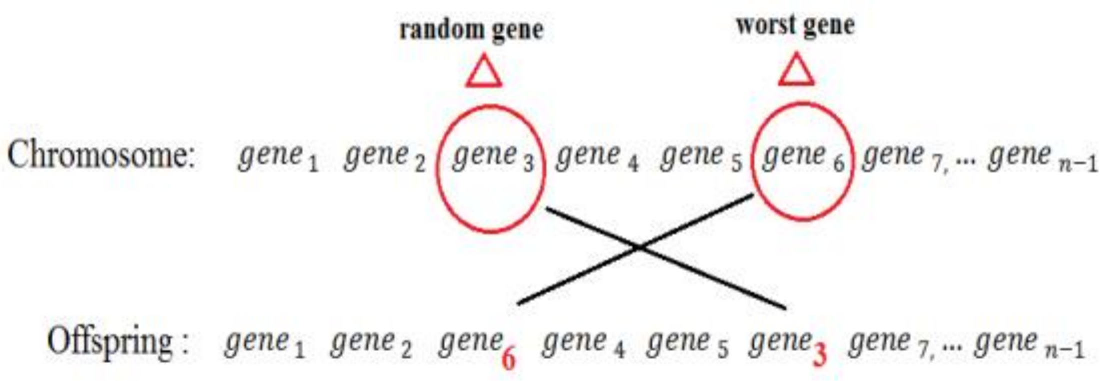

random gene
worst gene
Chromosome: $gene_1$ $gene_2$ $gene_3$ $gene_4$ $gene_5$ $gene_6$ $gene_{7,}$ ... $gene_{n-1}$
Offspring : $gene_1$ $gene_2$ $gene_6$ $gene_4$ $gene_5$ $gene_3$ $gene_{7,}$ ... $gene_{n-1}$

Figure 6

The next mutation method that was compared was inversion mutation. In inversion mutation two random points are selected to use as cut points for the mutation. The data between the two points is then flipped so the last number is the first and the first is the last etc. This is demonstrated below in figure 7.

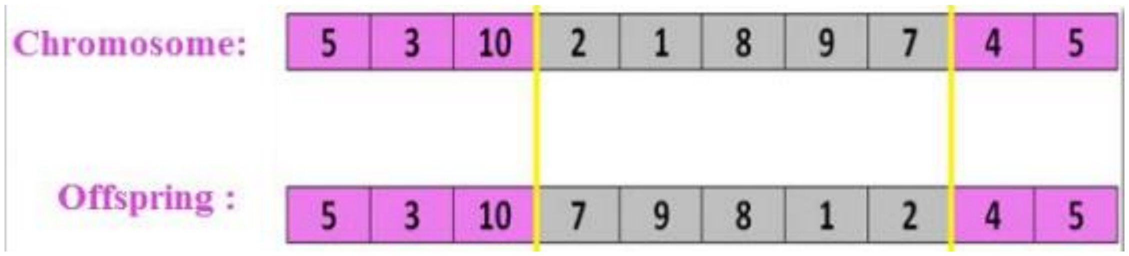


Figure 7

The last mutation method that was discussed was swap mutation. In swap mutation two random points are selected and these are used as indexes for the two points to be swapped within the chromosome. This method can be visualized below in figure 8.

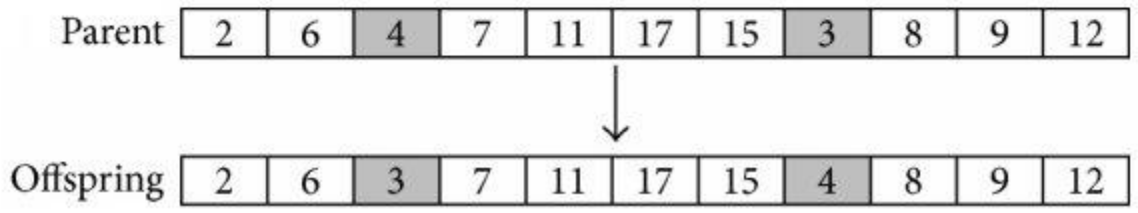
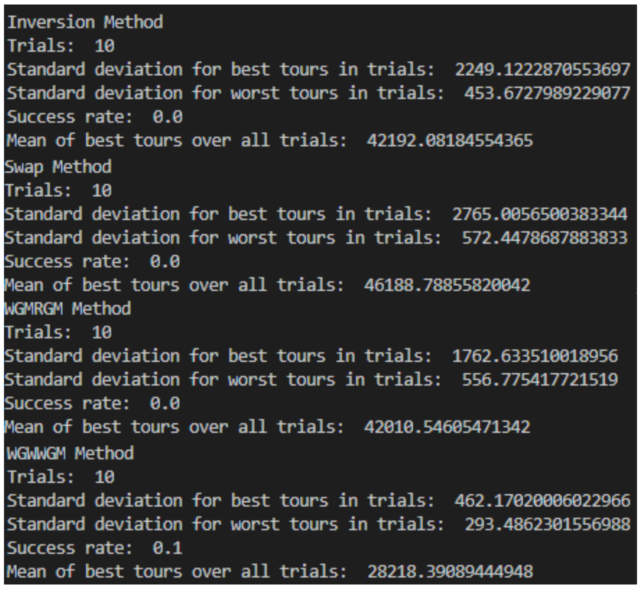


Figure 8

When building the program, all these methods were implemented for testing. This gave the chance to see how to data would be affected by each method and then to settle on the one that gave the best results. The methods were then applied to the Western Sahara data set and 10 trails were run to compute the standard deviation for the best and worst tours, as well as the success rate and mean of the best tours. The results can be seen below in figure 9.



```
Inversion Method
Trials:  10
Standard deviation for best tours in trials:  2249.1222870553697
Standard deviation for worst tours in trials:  453.6727989229077
Success rate:  0.0
Mean of best tours over all trials:  42192.08184554365
Swap Method
Trials:  10
Standard deviation for best tours in trials:  2765.0056500383344
Standard deviation for worst tours in trials:  572.4478687883833
Success rate:  0.0
Mean of best tours over all trials:  46188.78855820042
WGMRGM Method
Trials:  10
Standard deviation for best tours in trials:  1762.633510018956
Standard deviation for worst tours in trials:  556.775417721519
Success rate:  0.0
Mean of best tours over all trials:  42010.54605471342
WGWWGM Method
Trials:  10
Standard deviation for best tours in trials:  462.17020006022966
Standard deviation for worst tours in trials:  293.4862301556988
Success rate:  0.1
Mean of best tours over all trials:  28218.39089444948
```

Figure 9

From figure 9, worst gene with worst gene mutation was the most efficient as it has the best standard deviations, mean of best tours, as well as being the only method to reach a successful optimal tour for the salesman to travel. This was why the WGWWGM method was picked to be implemented into the algorithm.

Runtime Optimization

When working with such a large data set, runtime optimization will be detrimental in ensuring the program will run in an appropriate amount of time. For this project there were a few select runtime optimization that were implemented to help with the computation of the optimal path. The first optimization used is to pre-compute all the data needed to find the optimal path. This includes retrieving the cities from the file and initializing the population. By doing this is negates the need to retrieve a city from the text file every time the algorithm needs the location to compute a distance.

The next runtime optimization that was implemented was the use of a dictionary for storing the city information once retrieved from the text file. By doing this is will make look up for a city faster as the program can simply go straight to the index of the city in dictionary instead of having to traverse a list. Another use of storing data that was used is the use of collections. In using collections, it was possible to create open ended queues which allowed for the removal and addition of items from either end of the queue. This removed the need to push cities and then traverse to place them in their correct locations. This was an important part of the optimization for the crossover portion of the algorithm.

The last runtime optimization used in the project is the use of PyPy. PyPy is a fast, compliant alternative implementation of the Python language. It improved in speed due to the

Just-in-Time complier and improved in memory usage which helped with the larger data sets. Being compatible with Python meant that there is no change in code for the project, but a great deal of performance was gained by using it.

Results

The results for the program was tested on the two datasets, Western Sahara and Uruguay. To compile results the tests were run 10 times for each dataset. Once the data had complied it was then represented in the GUI that was created.

For the Western Sahara dataset, the program was run, and the first 10 trails were generated. The optimal path for the Western Sahara with 29 cities has a length of 27603 approximately. In the trails ran by our algorithm, trail 4 reached the optimal path in 81 generations. The comparison of the optimal path (left) and path generated by our program (right) can be seen below in figure 10.

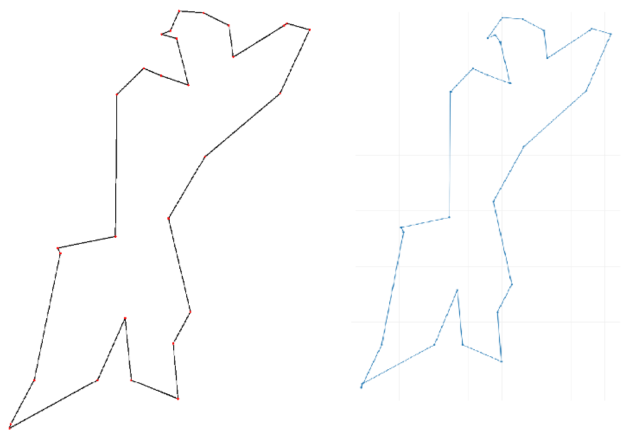


Figure 10

As seen in figure 6, the routes are virtually identical. The algorithm successfully found the optimal route for the salesman to travel. The data returned by the GUI is the distance average, shortest route, and longest route. For trial 4 it shows an average distance of 27631.43, the shortest route being 27601.17, and the longest route being 27634.08. The graph shown below in figure 11 compared the data between the three as the algorithm progressed in trying to find the optimal route, with the route length being on the y-axis and the generation along the x-axis. As visible from the graph, the program converges around the 80th generation mark. This then marks the potential optimal path for the salesman. In correlation with the provided data it is possible to then confirm whether the algorithm found the optimal path.

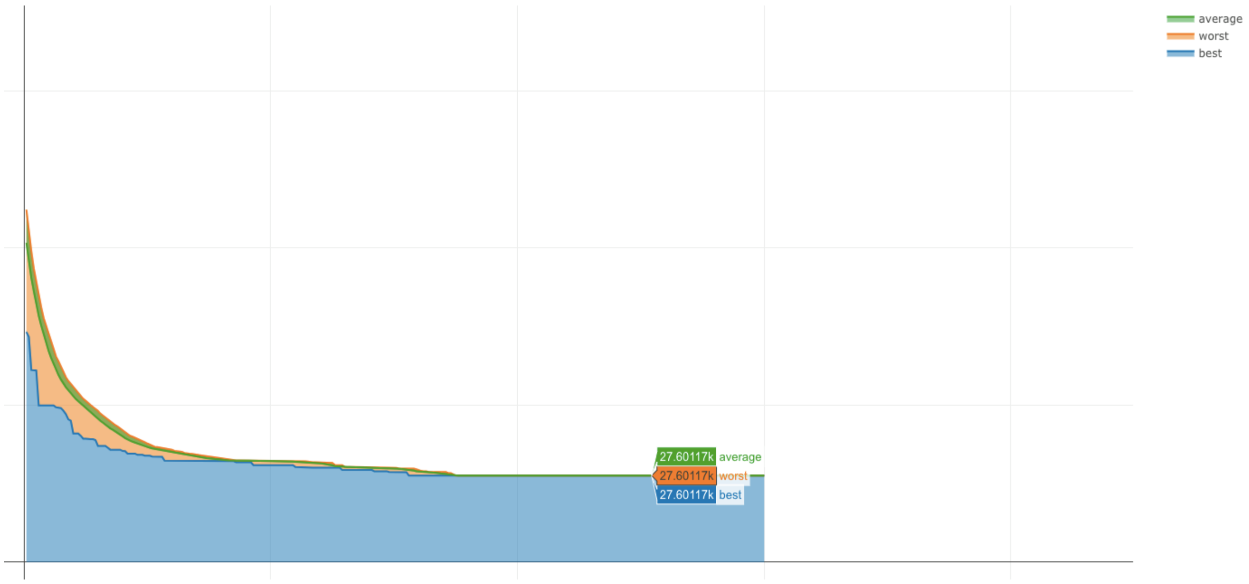
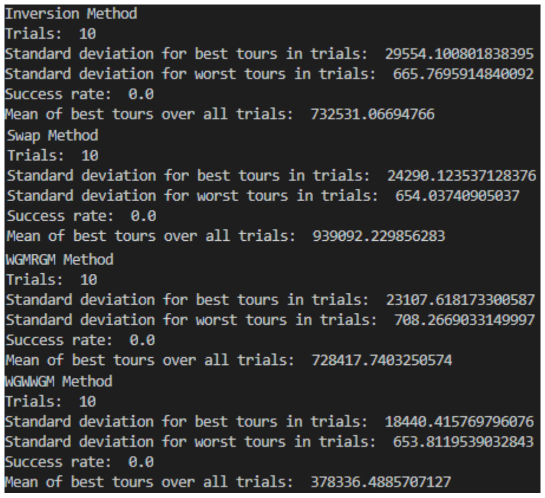


Figure 11

For the Uruguay dataset the algorithm ran into issues with convergence due to the size of the set. Experimenting with trying to increase the diversity yielded no results in getting to the optimal path. With all the different mutation methods discussed the worst gene with worst gene mutation still yielded the best results. The optimal path length for the Uruguay dataset should around approximately 79114 in length. The current algorithm can find a best of 250k in path length. The different mutation methods and the results in which they produced of a 10-trial run are shown below in figure 12.



```
Inversion Method
Trials:  10
Standard deviation for best tours in trials:  29554.100801838395
Standard deviation for worst tours in trials:  665.7695914840092
Success rate:  0.0
Mean of best tours over all trials:  732531.06694766
Swap Method
Trials:  10
Standard deviation for best tours in trials:  24290.123537128376
Standard deviation for worst tours in trials:  654.03740905037
Success rate:  0.0
Mean of best tours over all trials:  939092.229856283
WGMRGM Method
Trials:  10
Standard deviation for best tours in trials:  23107.618173300587
Standard deviation for worst tours in trials:  708.2669033149997
Success rate:  0.0
Mean of best tours over all trials:  728417.7403250574
WGWWGM Method
Trials:  10
Standard deviation for best tours in trials:  18440.415769796076
Standard deviation for worst tours in trials:  653.8119539032843
Success rate:  0.0
Mean of best tours over all trials:  378336.4885707127
```

Figure 12

Below in figure 13 it is visualized the best route that was calculated for the Uruguay dataset.

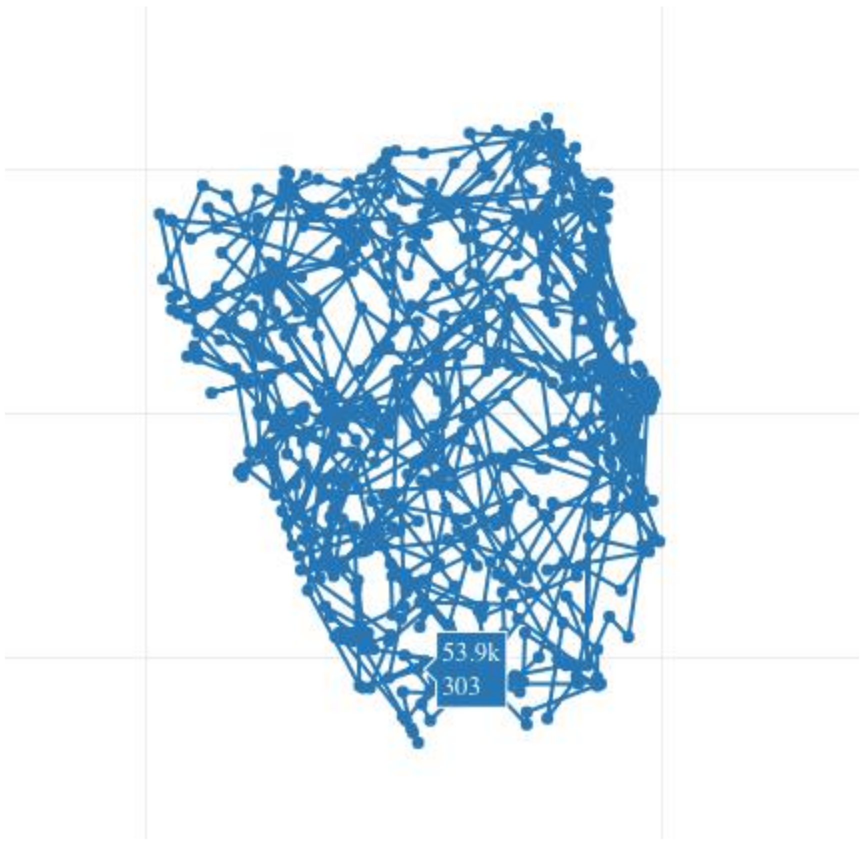


Figure 13

Discussion

After running the program through all the methods and studying the results the worst gene with worst gene mutation was the most effective at producing, in the case of the Western Sahara dataset, the optimal route and in the case of the Uruguay dataset it produced the closest optimal path of any other method. Pairing this method with the most effective parents' selection in tournament selection and the most effective crossover, order crossover, it showed that the algorithm has the possibility of finding the optimal route for the salesman to travel.

If the program was to continue development there would be other methods trialed to increase and preserve diversity to avoid convergence on larger datasets, as well as other run time optimizations. To help during run time multi-threading or multi-processing could be implemented to allow the program to do several computations at once. While there is room for improvement on the algorithm, it did successfully find the optimal path for Western Sahara and came close for Uruguay datasets. This has laid a great basis for the computation of large datasets for the Travelling Salesman Problem.

References

Travelling Salesman Problem:

https://www.geeksforgeeks.org/travelling-salesman-problem-set-1/

Methods:

https://www.researchgate.net/figure/Example-of-WGWWGM-213-Worst-left-and-right-gene-with-random-gene-mutation-WLRGWRGM_fig8_304623320

https://arxiv.org/ftp/arxiv/papers/1801/1801.02827.pdf

https://www.researchgate.net/figure/Swap-mutation-for-TTRPSD_fig4_272094816

https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_parent_selection.htm

https://www.researchgate.net/figure/An-example-of-order-crossover_fig4_282998951

https://www.researchgate.net/figure/Order-crossover-OX1_fig3_226665831

https://arxiv.org/abs/1801.02335