Assignment 1 – CT421

Liam Caffrey

Student Identification: 21378316

Introduction

As part of assignment one we are to create an algorithm for the traveling salesman problem.

The genetic algorithm was made using a python and implements all the standard aspects of a genetic algorithm.

As is the case with genetic algorithms, it is possible that from the limited testing that results obtained could be due to unlikely / lucky random generations.

Crossover

As apart of the assignment we were asked to use 2 crossover function, both functions the same and end up with the same result, however it is in there implementation where they differ. Both take the first section from parent 1 using it to fill the child, while using parent 2 for the remainder. Function 1 iterates through parent 2 finding elements not in the child and adding them to the end 1 by 1, whereas function 2 creates a list filtered by elements no present in the child and appends the list to the end. Both functions were tested multiple times alongside the mutation functions. While function 1 was easier to implement function 2 saw slightly faster results leading to an overall better performance.

Mutation

2 mutations functions were also tested, function 1 simply swaps the location, randomly, of two cities in the sequence, this is more efficient however provides minimal change. Function 2 however takes a random subsequence within the path and reverses it. This is more expensive however due to the fact it is a larger change it provides a wider range of diversity.

Design choices

The stopping criteria of reaching the max generations or no change within 50 generations was selected with the intention of providing a good chance of reaching the greatest possible solution.

Elitism was also used during the experimentation, at a rate of 10%, this is what saw the greatest result in testing.

The code reads in all files the same way and uses 3 of the same lines to set up files;

```
#filename = "berlin52.tsp"; popSize = 500; maxGenerations = 2000 #small set
filename = "kroA100.tsp"; popSize = 600; maxGenerations = 2000 #medium set
#filename = "pr1002.tsp"; popSize = 150; maxGenerations = 5000 #large set
```

Another aspect that may affect the length of process is a checking method added in to ensure that any generated child contains completely unique cities and does not have any repeats. This was added to guarantee no repeats are present and all paths are of equal length.

For the largest dataset the number of generations and population of generations is quite small however due to the available technology is was what was able to be handled and processed in both a timely and effective manner.

Experimental results and analysis

While each dataset, mutation and crossover combination, where run multiple times, with varying results obtained the data shown is a rough average of the algorithms performance.

Known solutions

Berlin52	**7544**
kroA100	**21282**
pr1002	**259045**

Recorded results

Small Dataset - Berlin52 | population size = 500 | max generations = 2000

Mutation	Crossover	Best Path length	Time (seconds)	Graph
1	1	8994.12	86.52	1
1	2	9822.94	75.29	2
2	1	12698.27	78.93	3
2	2	12573.35	36.43	4

Medium Dataset - kroA100 | population size = 600 | max generations = 2000

Mutation	Crossover	Best Path length	Time (seconds)	Graph
1	1	40871.24	492.08	5
1	2	46574.73	223.66	6
2	1	84651.64	119.60	7
2	2	83787.81	69.13	8

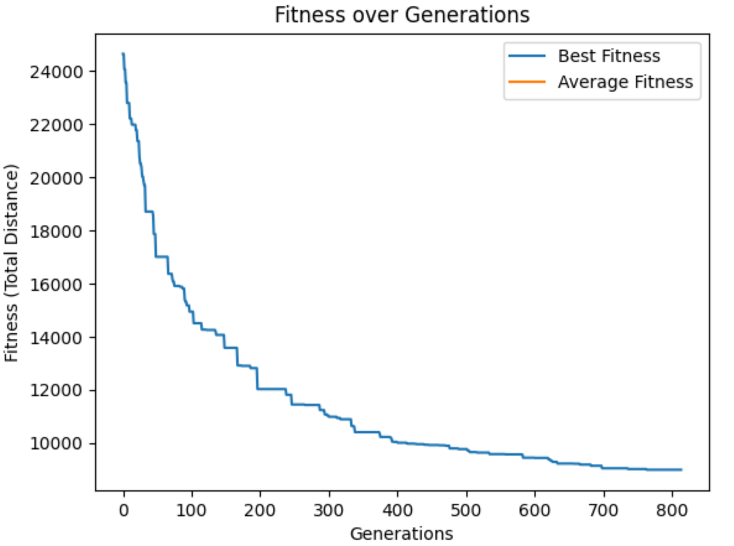
Large dataset – pr1002 | population size = 150 | max generations = 5000

Mutation	Crossover	Best Path length	Time (seconds)	Graph
1	1	3634084.94	2395.32	9
1	2	3154673.15	2789.21	10
2	1	3930626.16	2034.56	11

2	2	4177273.88	1786.42	12

Graphs – Berlin52

Figure 1



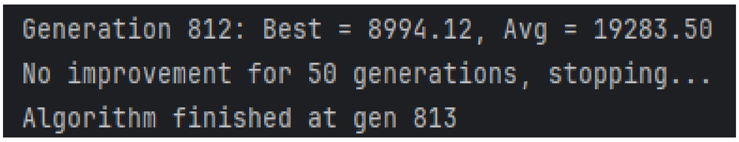
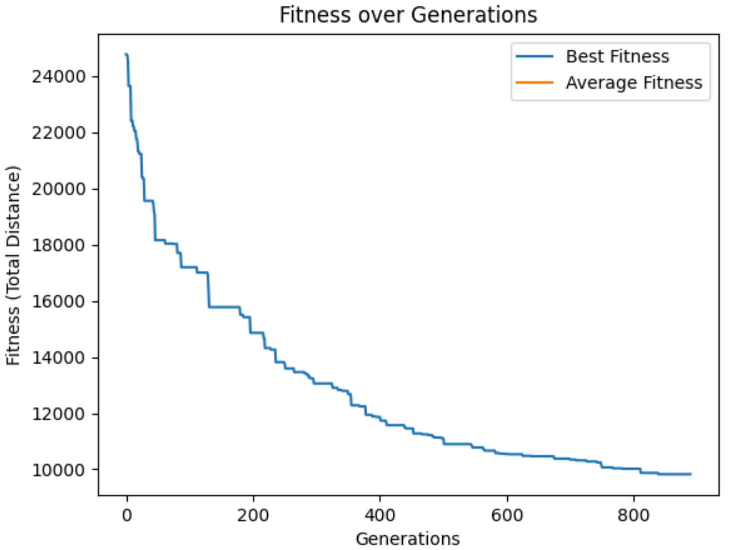

```
Generation 812: Best = 8994.12, Avg = 19283.50
No improvement for 50 generations, stopping...
Algorithm finished at gen 813
```

Figure 2



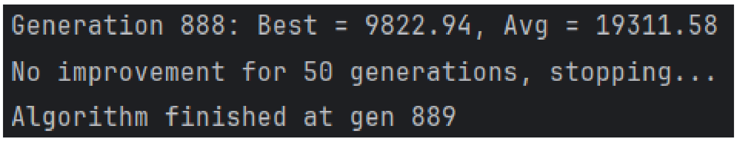
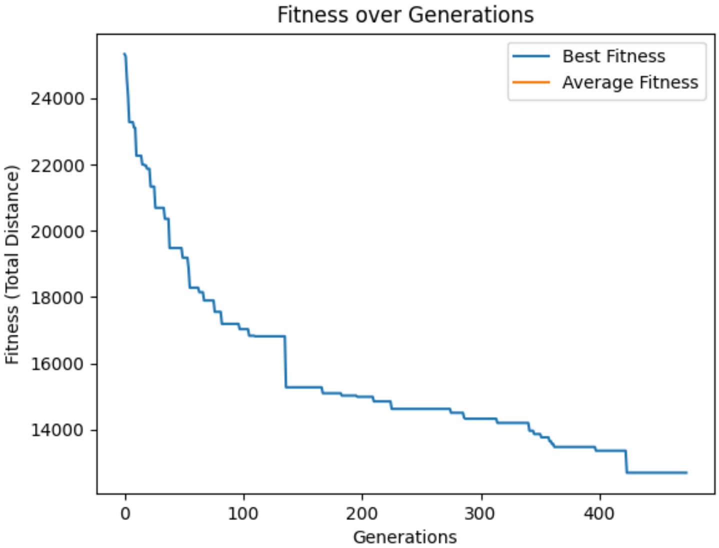

```
Generation 888: Best = 9822.94, Avg = 19311.58
No improvement for 50 generations, stopping...
Algorithm finished at gen 889
```

Figure 3



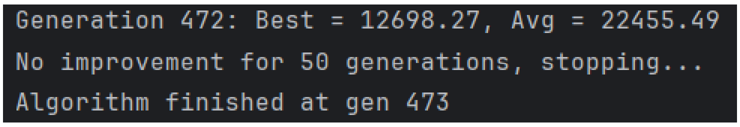
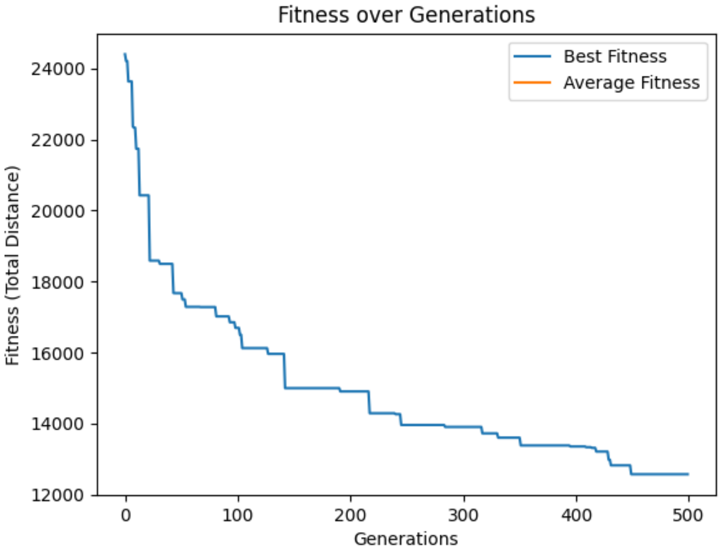
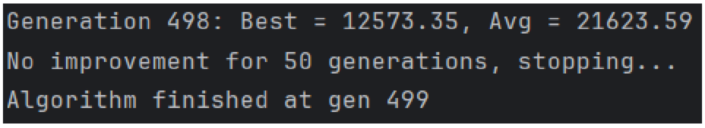

```
Generation 472: Best = 12698.27, Avg = 22455.49
No improvement for 50 generations, stopping...
Algorithm finished at gen 473
```

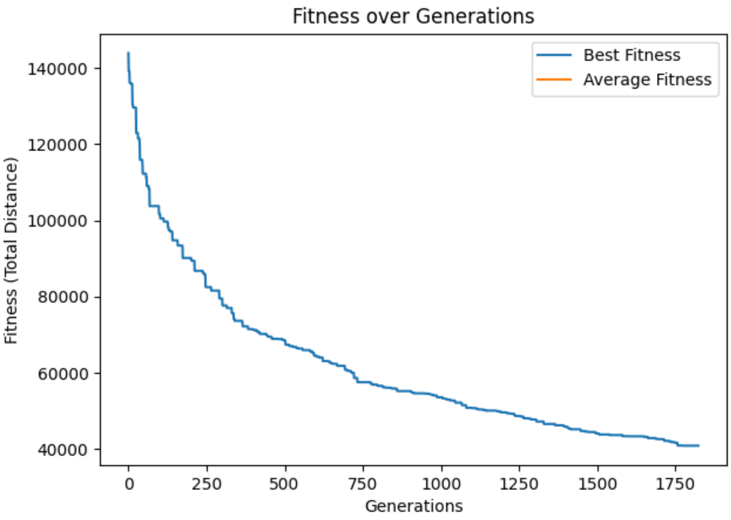
Figure 4

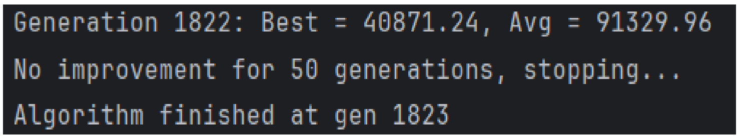



```
Generation 498: Best = 12573.35, Avg = 21623.59
No improvement for 50 generations, stopping...
Algorithm finished at gen 499
```

Graphs - kroA100

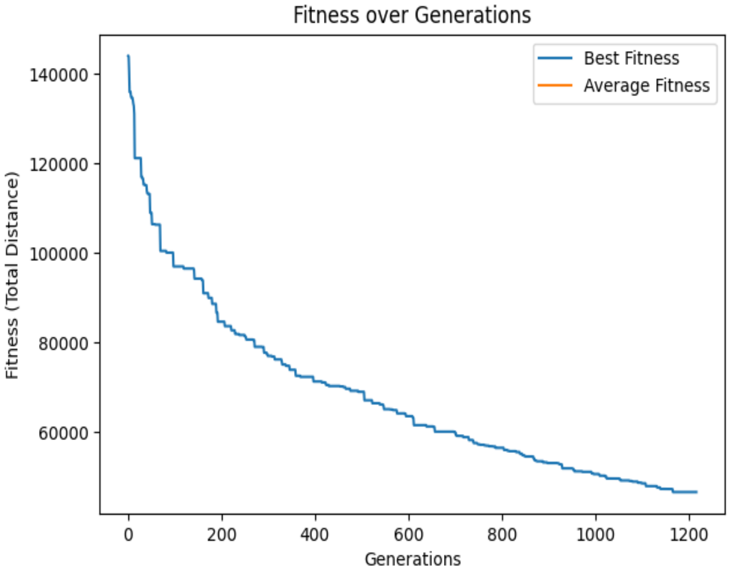
Figure 5

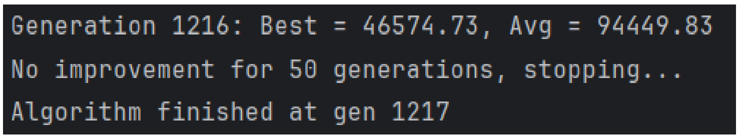




```
Generation 1822: Best = 40871.24, Avg = 91329.96
No improvement for 50 generations, stopping...
Algorithm finished at gen 1823
```

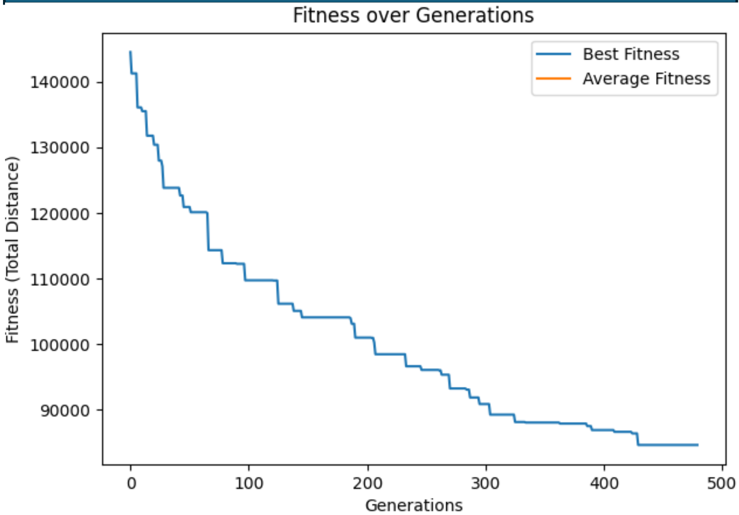
Figure 6

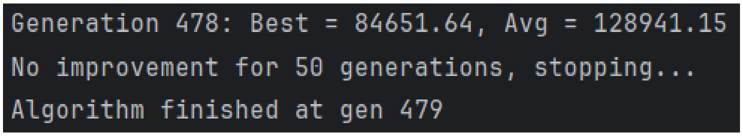




```
Generation 1216: Best = 46574.73, Avg = 94449.83
No improvement for 50 generations, stopping...
Algorithm finished at gen 1217
```

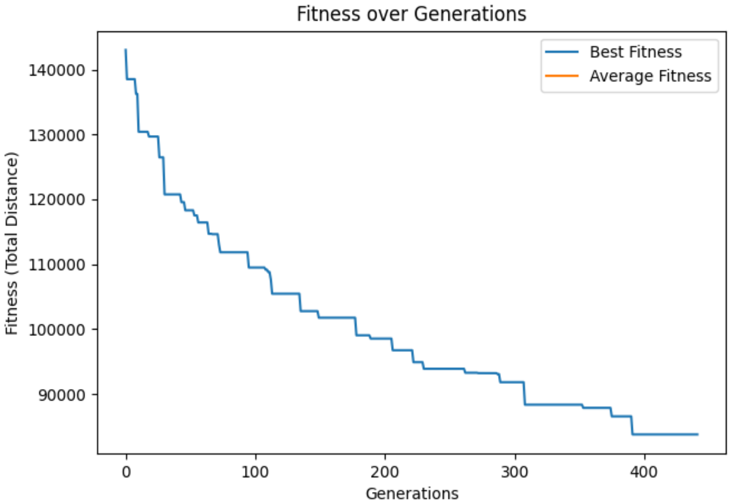
Figure 7

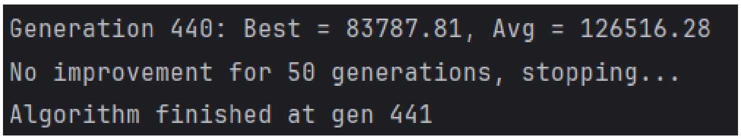




```
Generation 478: Best = 84651.64, Avg = 128941.15
No improvement for 50 generations, stopping...
Algorithm finished at gen 479
```

Figure 8

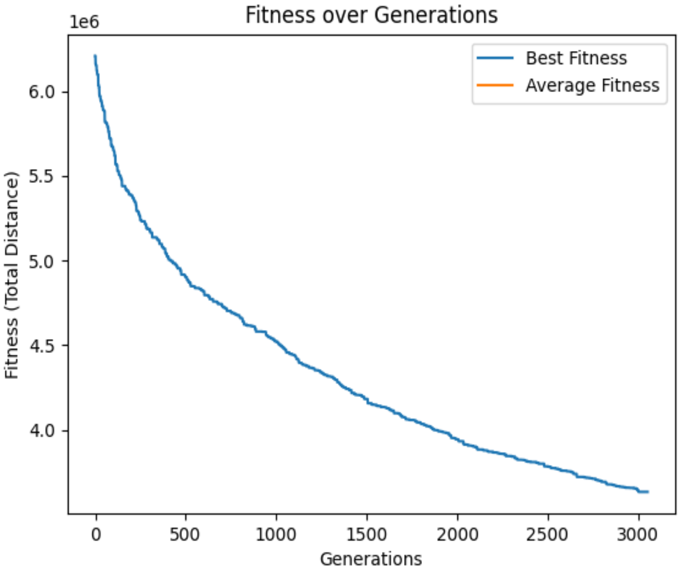


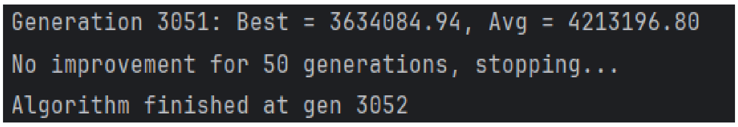


```
Generation 440: Best = 83787.81, Avg = 126516.28
No improvement for 50 generations, stopping...
Algorithm finished at gen 441
```

Graphs – pr1002

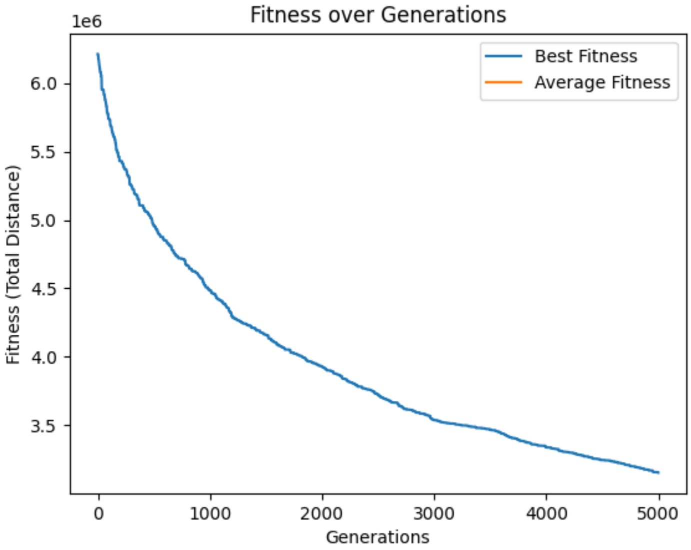
Figure 9

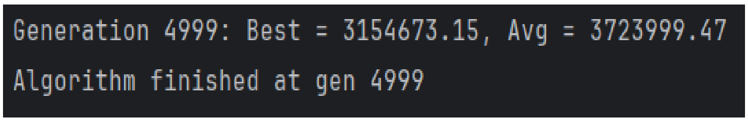




```
Generation 3051: Best = 3634084.94, Avg = 4213196.80
No improvement for 50 generations, stopping...
Algorithm finished at gen 3052
```

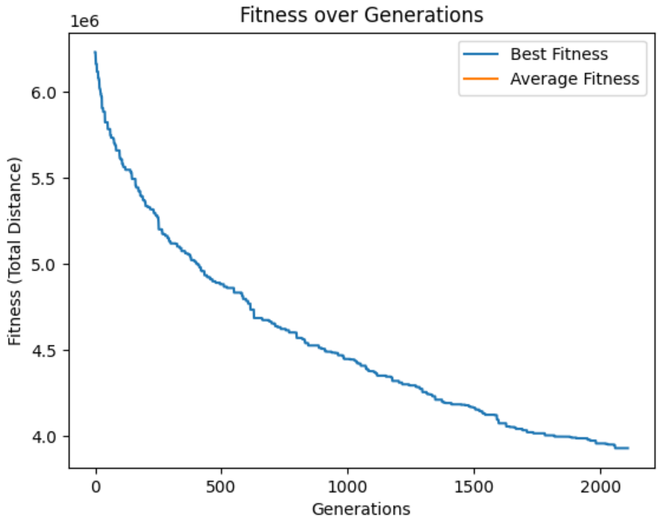
Figure 10

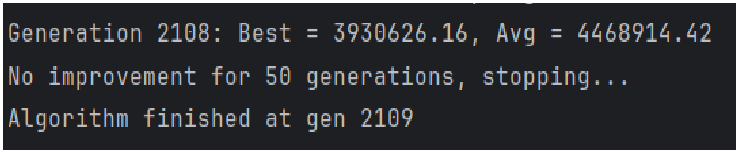




```
Generation 4999: Best = 3154673.15, Avg = 3723999.47
Algorithm finished at gen 4999
```

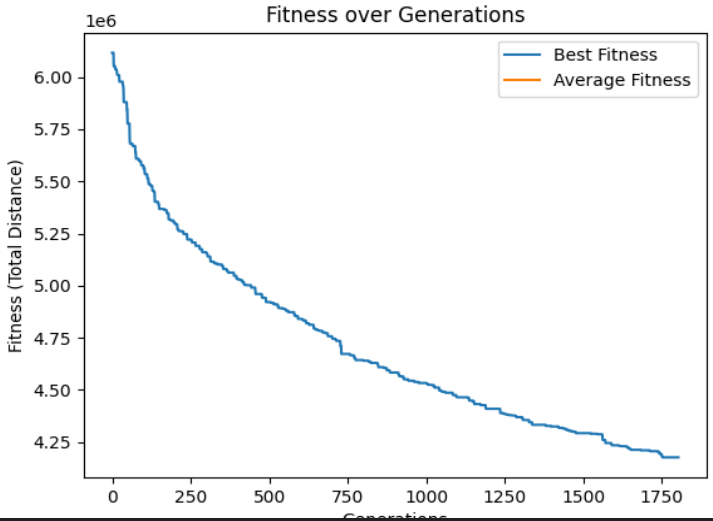
Figure 11

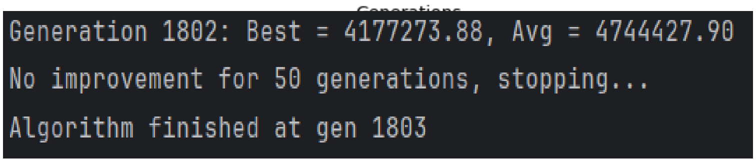




```
Generation 2108: Best = 3930626.16, Avg = 4468914.42
No improvement for 50 generations, stopping...
Algorithm finished at gen 2109
```

Figure 12





```
Generation 1802: Best = 4177273.88, Avg = 4744427.90
No improvement for 50 generations, stopping...
Algorithm finished at gen 1803
```

As shown from the results the optimal solution was never reached. However for the berlin52 dataset the predicted value did come quite close, while for the kroA100 dataset, while still off, compared to the starting values of the first generation, the obtained values where quite close.

As for the largest dataset pr1002, the results were not close. This is be further discussed in the potential improvements section.

Out of the 2 mutation functions used 1 was the clear better option. Mutation 1 while offering less diversity than mutation 2, made it so that the obtained results were better, allowing the genetic algorithm to obtain good paths and focus in on them more and more, improvements where large at first but would slow down nearing the end. This clarity is best seen with the kroA100 dataset where the mutation 1 results produced paths half the length of mutation 2. The mutation 2 function seemed to be too diverse, it would take sections of the gene and inverse them, which seemed to not allow the algorithm to hone in on an particularly good result and continue building upon it.

While the mutations allowed for a results based improvement, from the results obtained it seems crossover lead to a performance improvement. This is a more difficult improvement to track overall as, as previously stated improvements can be due to the randomly generated population. It was also difficult to track as quite a lot of the algorithms would finish before the max generation due to no change being found over 50 generations. However, from testing multiple times with the smaller datasets on average, crossover 2 seemed to be more efficient. This can be best seen again this the kroA100 dataset as with crossover 1 generations took roughly 2.1s on average to form, over the testing preformed with the dataset, while crossover 2’s generations took roughly 2s to form.

Discussion of potential improvements

The main area that needs improvement is the larger datasets, such as pr1002, results were far too far away from the best possible solution, to improve this a larger population size would help out immensely, however with the available technology and large size of the dataset, some issues where encountered which running the algorithm, with a higher population size.

The stopping factor for many of the solutions also came down to no improvements within 50 generations, meaning the max generations had little impact, on the algorithm as a whole. I believe a new mutation may help the algorithm out as a whole, however it would have to be more diverse than mutation 1 and more focused than mutation 2.

A higher elitism or mutation rates may also provide better results, however with a elitism rate of 10% and a mutation rate of 20% they seemed already quite high and where only partially tested with higher values, these tests in the future can be more extensive and hopefully a larger variety of them can lead to more conclusive evidence. A higher mutation rate might can lead to a more diverse population but may lead to less focus from the algorithm, while a higher elitism rate may lead to a convergence of results far earlier than expected.

References

Genetic Algorithms Explained By Example – YouTube Kie Codes

The Knapsack Problem & Genetic Algorithms - Computerphile

Symmetric TSPs – Heidelberg University

The optimal solution of Berlin52 | Download Scientific Diagram - Otman, Abdoun & Tajani, Chakir & Abouchabaka, Jaafar. (2012). Hybridizing PSM and RSM Operator for Solving NP-Complete Problems:

Application to Travelling Salesman Problem. International Journal of Computer Science Issues. 9.

Download citation of PMACO: A pheromone-mutation based ant colony optimization for traveling salesman problem - Shokouhifar, Mohammad & Sabet, Shima. (2012). PMACO: A pheromone-mutation based ant colony optimization for traveling salesman problem. INISTA 2012 - International Symposium on INnovations in Intelligent SysTems and Applications. 1-5. 10.1109/INISTA.2012.6247040.