Artificial Intelligence: Assignment 2

Justin Stickel (6718704)
Computing and Business Co-Op
Kitchener, Canada
js19ge@brocku.ca

***Abstract*—This document serves to provide insight into the associated Java program, thus allowing for a deeper understanding of its components and results.**

I. Introduction

This report will cover the implementation of a genetic algorithm (coded in Java) on the common travelling salesman problem. A genetic algorithm is a meta-heuristic which utilises natural selection and aims to produce better offspring with each generation. Ideally, the longer a genetic algorithm is allowed to run, the closer to an optimal solution you become. This problem entails a salesman that must travel to a series of cities before returning back to the city which he started in. The goal is to optimize the salesman's route by finding the shortest path for the salesman to take, whilst still ensuring that they are travelling to all cities. This solution has real-life uses, as it can be integrated in daily routing problems, such as pizza-delivery, to optimize people's time.

II. Background

The central algorithm that this program is based on is a: Genetic Algorithm. In essence, a genetic algorithm is an 'evolutionary algorithm' which utilises a series of key methods in order to create a variety of solutions and ultimately choose the solution that is viewed as 'best'. The pseudo-code for the genetic algorithm used is represented in **Algorithm 1**

A. Methods

The core methods used in the program are as follows:

1) ***CalcFitness****:* Using Euclidean distance, this method sequentially finds the distance from each city to the next - within the genome. This is accomplished by checking the original data file for the current and next city and storing their x and y coordinates. These coordinates are then substituted into the euclidean distance formula, and the resulting distance is added to the total distance that is stored within the chromosome.

2) ***SelectNewPop****:* This utilises a tournament selection with K=3 to find pairs of parents (equal to the population size) that will later make children. The tournament selection works by randomly choosing K individuals from the population, and from those three, the one with the greatest fitness score is selected to be apart of a pair. This is repeated once to find a second parent for the pair, ensuring that the already selected parent cannot be selected again, as this would cause it to essentially 'mate' with itself and cause crossover to automatically fail.

3) ***Crossover****:* This performs a UOX (Uniform Order Crossover) by randomly creating a mask, and creating two children based off of said mask. The mask is a binary string with length equal to the amount of cities of one of the parents. For each '1' in the mask, the first child inherits the first parent's 'allele' (city) at the corresponding location. The same also happens between the second child and the second parent. For each '0' in the mask, assuming that the city has not already been inherited by the child: Vice Versa happens. If the city was already included, the child simply inherits the next non-included city from the corresponding parent. This method is finished when the children are full and added into the new population to undergo mutation.

4) ***Crossover2****:* A different crossover method that can be toggled instead of the UOX method. This is a unique 'Single-Point Crossover' that works similar to the UOX. Instead of creating a random binary mask, this chooses a single point in the parents' chromosomes to allot for inheritance. A mask is used to make this process more presentable. The binary mask, with length equal to the number of cities, fills with '1's up to the chosen point, and fills the rest with '0's. The only difference after this is that duplicate cities are not initially prevented from being present in children. Instead, children undergo a 'fix' function, which removes any duplicates found and replaces them (in the same allele position) with cities that are not yet present.

5) ***Mutation****:* This introduces a core element of 'evolution' into the genetic algorithm. If mutation succeeds, two random alleles (cities) of the child are swapped with each other. This only happens once to ensure that the child does not completely change from what it was. After this, or if mutation fails, the child is added into the new population.

Algorithm 1 Typical Genetic Algorithm

```
Initialize population
while currentGen ≤ maxGen do
    Evaluate Population
    Pass Elites onto the Next Generation ▷ Best Survive
    Select Parents ▷ Through Tournament Selection
    Perform Crossover ▷ To Create Children
    Mutate Children
    Replace Population
end while
output bestPath
```

III. Experimental Setup

Five tests were run at five different parameters for each crossover method (1 and 2). This was repeated for each provided data file for a total of 100 runs.
The variable parameters are as follows:

1) : Crossover Rate = 100%, Mutation Rate = 0%

2) : Crossover Rate = 100%, Mutation Rate = 10%

3) : Crossover Rate = 90%, Mutation Rate = 0%

4) : Crossover Rate = 90%, Mutation Rate = 10%

5) : Crossover Rate = 100%, Mutation Rate = 50%

*All runs were done using an elite percentage of 10%, a maximum generation of 50 and a population size of 100; to ensure that the genetic algorithm had a broad enough scope for producing more accurate results.

IV. Results

Through completing the experiments outlined in the above setup, we were able to observe the data from each run. When plotted in a graph like **Fig 1**, this provided us with a clear representation of how the best path and average population were improved over each generation. Typically, the older the generation, the lower the average and best fitness (distance) would be. Additionally, this figure in particular shows how the longer the algorithm runs for, the longer it also experiences sideways movement, or 'plateaus', where no progress is being made. This was evident in all experiments, due to the fact that the better the 'best fitness' is, that harder it is for the algorithm to find an even better solution. Aside from this figure, the experiments also uncovered an interesting discovery. Ultimately, from the data gained through our experiments, we were able to find that the Parameter 5 setup yielded better results than the rest. As seen in **Fig. 2**, none of the parameters had any noticeable effect on the "Average Final Pop" fitness. Despite this, what makes Parameter 5 superior is that it consistently produced a better 'Best Fitness', meaning that when using the same seed: at the end of each trial, Parameter 5 typically found a more efficient path than the rest. These results were further confirmed by performing a statistical test known as the "*Mann-Whitney U Test*" as seen in **Fig. 3**.

A. Interpreting the Results - UOX

Referring to **Table II**, for population fitness, parameter 5's results were not significantly different. This is revealed by U at $p<0.05$ being 25, but the calculated U value was 30. Additionally, the p value was found to be greater than 0.05.

For the best fitness, it was shown that parameter 5's results were significantly different. This is verified through the U at $p<0.05$ being 25, and the calculated U value being 11. Additionally, the p value was found to be less than 0.05. This means that with 95% certainty we can say that parameter 5 produces (on average) a shorter 'best path'.

The total metrics from this experiment can be found in **Table I**.

B. Alternate Crossover (Single-Point)

Much like UOX crossover, experiments run with the parameter 5 setup were not significant in reference to population fitness, but were significant in reference to the best fitness. This is shown once again through the Mann-Whitney test results in **Table IV**.

A similar breakdown of the experiment metrics can also be found in **Table III**. It is also notable that when comparing this table with **Table I**, it becomes apparent that the mean fitnesses for both population and best run in the single-point crossover experiment are noticeably lower than those of the UOX experiment. This means that, in general, single-point crossover outperformed UOX in finding a better solution to the travelling salesman problem.

Fig. 1.

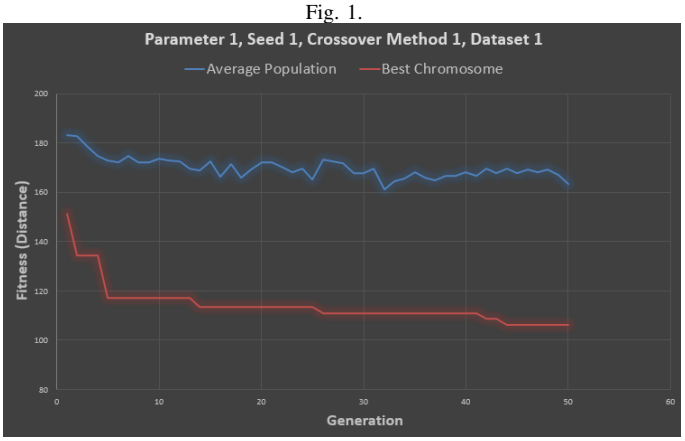


Fig. 2. Data for UOX Crossover

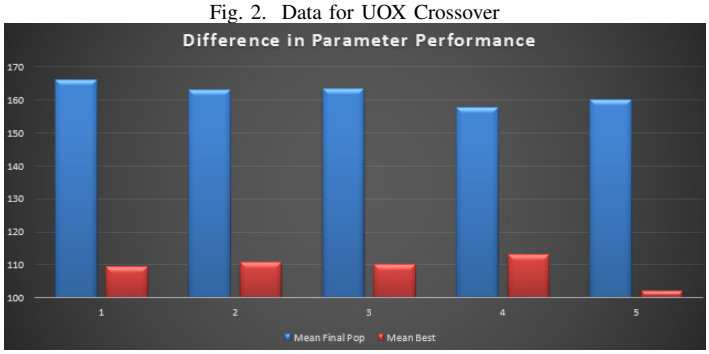


TABLE I
UOX Experiment Results

Fitness Type	Parameter Number				
	1	*2*	*3*	*4*	*5*
Mean Final Population	166.23	162.97	163.34	157.84	160.23
Mean Best	109.43	110.68	110.10	113.04	104.59
Median Population	165.61	165.01	163.31	156.58	160.03
Median Best	107.78	109.48	109.88	112.74	102.94
Min Population	163.41	155.026	160.51	149.15	157.40
Max Population	170.03	169.53	166.23	167.97	164.48
Min Best	106.17	108.94	105.30	106.06	95.09
Max Best	111.28	117.08	118.10	119.77	111.09
Std. Dev (Population)	2.53	5.19	2.48	6.20	2.38
Std. Dev (Best)	3.56	3.51	4.39	4.37	4.08

Fig. 3. Mann-Whitney Test Formula

$$U_1 = n_1 n_2 + \frac{n_1(n_1+1)}{2} - R_1$$

TABLE II
MANN-WHITNEY U TEST RESULTS FOR UOX

Fitness Type	Statistical Values		
	U-Value	*Z-Score*	*P-Value*
Mean Fitness	30	1.32476	0.0934
Best Fitness	11	2.61556	0.0044

V. DISCUSSION & CONCLUSION

One hundred different runs were done to see what parameter/variable combination for a genetic algorithm would yield the greatest results when implemented for a travelling salesman problem. Ultimately, it was seen that higher mutation rates, higher crossover rates, and utilising a single-point crossover were the biggest contributors to lowering the best fitness. The conclusions that can be drawn from this are: for the travelling salesman problem, introducing more variability into new generations allows for a better solution to be found quicker. Naturally, given an infinite amount of time with an infinite amount of generations, all parameter sets would eventually find the best solution; however through this experiment, it is seen that using the aforementioned parameters will, on average, allow you to arrive at the best solution quicker and more efficiently. Additionally, through the calculations shown in **Section IV**, we can verify that this conclusion is accurate with 95% certainty. Additionally, due to single-point crossover yielding better results on average than UOX, we can further conclude that this is due to the single-point crossover's properties. Since the majority of the parents' alleles stay together, it allows for successful combinations of cities to be inherited by the children, rather than in UOX where, on average, children would inherit every other allele from each parent. Because of the way the travelling salesman problem functions, single-point crossover is the better logic choice, since it does not disrupt partial-pathways which are superior (for example, if city 1 -> city 2 -> city 3 was the shortest 3 city combination).

Although the optimal parameters and variables were identified, it is still important to note that no experiments performed actually found the best possible solution for the data sets used. This was most likely a result of the total generations and population size not being large enough; thus for future experiments, a possible extension could be to test which parameter set finds the optimal solution in the least number of generations. Whilst we can infer that the results will not change, this cannot be confirmed until proven.

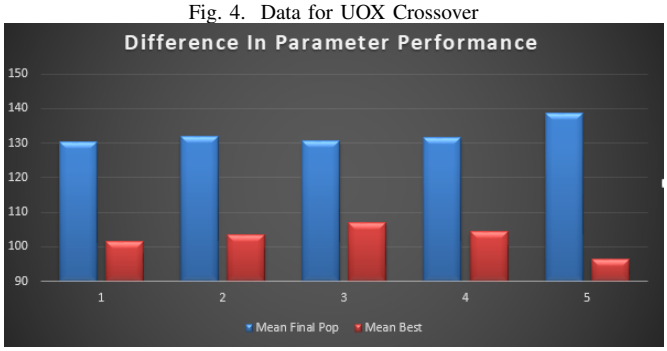
TABLE III
SINGLE-POINT CROSSOVER EXPERIMENT RESULTS

Fitness Type	Parameter Number				
	1	*2*	*3*	*4*	*5*
Mean Final Population	130.46	132.12	130.67	131.49	138.85
Mean Best	101.61	103.47	106.96	104.41	96.43
Median Population	132.36	133.05	133.27	133.64	138.17
Median Best	101.20	103.31	106.83	105.98	95.92
Min Population	119.78	125.78	123.86	125.69	133.64
Max Population	134.20	135.98	136.34	136.10	146.20
Min Best	96.57	99.89	104.48	97.35	93.31
Max Best	110.08	107.03	110.46	107.07	100.25
Std. Dev (Population)	5.52	3.39	4.80	4.43	4.07
Std. Dev (Best)	4.63	2.42	2.18	3.64	2.49

TABLE IV
MANN-WHITNEY U TEST RESULTS FOR SINGLE-POINT CROSSOVER

Fitness Type	Statistical Values		
	U-Value	*Z-Score*	*P-Value*
Mean Fitness	35	1.12476	0.0954
Best Fitness	9	2.82334	0.0236

Fig. 4. Data for UOX Crossover



REFERENCES

[1] Stuart Russel, and Peter Norvig, "Artificial Intelligence A Modern Approach" Fourth Edition, ch. 4.1, April 2020.
[2] Beatrice Ombuki-Berman, In-class material (GA.ppt), 2021.