Travelling Salesman Problem: Genetic Algorithm Solution

Ralph Pereira 4554879

Abstract—This is an outline of my findings for the Travelling Salesman Problem using a Genetic Algorithm. The speed was a vast improvement over the brute force method, which comes in at around ”n!” time. The outputs are controlled through four variables: Mutation, cross over,generations, and population size. A level of elitism is involved in picking the best chromosomes for the crossover as well, but there is less control over that in my algorithm. The graphs show the data with the parameters tweaked slightly.

1 Introduction

Genetic Algorithms, when used in computer science, mimics the evolution process by picking the best element out of a population set and passing it along to the next generation. An element of randomness is used in order to make sure the best element, or chromosome, is truly passed on to the next generation. Genetic Algorithms are used to solve many NP-Complete problems in computer science, in this particular research paper it is used to solve the Travelling Salesman Problem. The population in this case is a series of chromosomes that represent travel orders between cities. The Genetic Algorithm is always trying to find the best chromosome in the population, hoping to get to a global best.

2 Objective and problem definition

The objective for this assignment is to use a Genetic Algorithm to solve the well known Travelling Salesman Problem. The Travelling Salesman Problem involves going from point to point, city to city, in a specific order. The Genetic Algorithm is used to find the best path in order to reduce the amount of distance travelled. The greedy solution to this problem would be to calculate all the permutations of cities and just find the shortest path from there. This is only possible for a data set of at most eight cities, not very useful in the real world. The data set needs to be so small because the time complexity for this brute force method is ”N!” time, which means for a data set of around 50 it would be impossible to calculate it within a reasonable time.

3 Summary parameters used

The output for the algorithm is controlled by six different factors that can be controlled to an extent; the generation count, the population size, the crossover rate and method, the mutation rate and method. Fitness value is a term used to describe the total travel distance between cities in a particular chromosome. We are looking for a higher fitness value, in the case of the Travelling salesman that would mean the smallest value.

3.1 Generation Count

The generation count determines the number of generations the algorithm runs for. Each new generation should be better than the last if the algorithm works the way it should. For the given data-set and population size the results indicated that the best solution is around thirty to fifty generations in. After that there are a few improvements, but it’s negligible, maybe an improvement of one decimal point. For most of the tests 150 generations were used.

3.2 Population Size

Population Size, also referred to as the Chromosome count, contains the core data needed

to make the genetic algorithm function. This determines the genetic diversity of the population, the larger the population size the more diverse the population can be. If it's too small the population will be too similar and not have a lot of variety, leading to similar off-springs in the next generation. Conversely if the population is too large it can take the algorithm a very long time to sort through all the chromosomes within the population.

3.3 Crossover Method

Crossover is the creation of a pair of new chromosomes by exchanging the elements of two child chromosomes. The algorithm uses uniform crossover with a bit-mask and a two point crossover. The uniform cross over is done by creating a masking array of 0 and 1 bits created randomly through the array. The algorithm creates two new child elements by retaining the elements with 1 values and populating the rest of the array with eligible cities. The cities cannot repeat as that would not work within the confines of the Travelling Salesman Problem, and realistically would not make sense if you want to reduce the distance travelled. The algorithm then select the child with the shortest travel distance, thus assuring the best chromosomes get passed on. Two point cross over is a lot simpler, the algorithm takes a subsection of one child and fills a new chromosome with it, it then populates the rest of the chromosome with elements from the second child.

3.4 Crossover Rate

The crossover rate is how often the crossover occurs, the higher the rate the more often chromosomes will be crossed over. For most of my tests a 90% crossover is used, as the results show that it is the best. With 100% crossover it can result in slower results.

3.5 Mutation Method

The mutation method used in the algorithm is fairly simple. Two random points on an array are picked and those two elements are swapped.

3.6 Mutation Rate

The mutation rate is how often a single chromosome is mutated. Smaller mutation rates are generally used in order to keep random exploration under a bit of control and to keep the co-mutation minimal. A 0% mutation rate will result in the same chromosomes passed on from generation to generation, which can lead to the algorithm possibly hitting a local min and not the global min.

4 Results

I did six different combinations for the output graphs: 100% Crossover with 0% Mutation, 90% Crossover with 0% Mutation, 100% Crossover with 10% Mutation, 90% Crossover with 10% Mutation, 95% Crossover with 5% Mutation and 100% Crossover with 10% Mutation with a population of 300.

4.1 Fig 1-2.

First set of figures has 100% Crossover with 0% Mutation. With mutation reduced to zero nothing really changes after about 20 generations. This is because the population doesn't change at all from the initial chromosome set, and probably hits a local minimum. A local minimum in this case means the chromosome with the highest fitness just gets passed on from generation to generation, never changing. The results are similar for a 90% Crossover with 0% mutation, with a slight improvement in the best distance. Another observation we can make is that the best and average distances don't differ a lot from generation to generation past 20th generation mark.

4.2 Fig 3-4.

With an increased mutation rate we can see that the best distance separates a lot more. The best distance also reduces quite a bit with more mutation. The chromosomes change quite a bit as the generations progress, due to the mutation rate.

4.3 Fig 5.

With 95% Crossover with 5% Mutation the results are similar to 10% Mutation rate, with a bit less fluctuation in the best distance.

4.4 Fig 6.

With 50% for both cross over and mutation rate we can see that the results don't change much over all, but they do fluctuate quite a bit generation to generation.

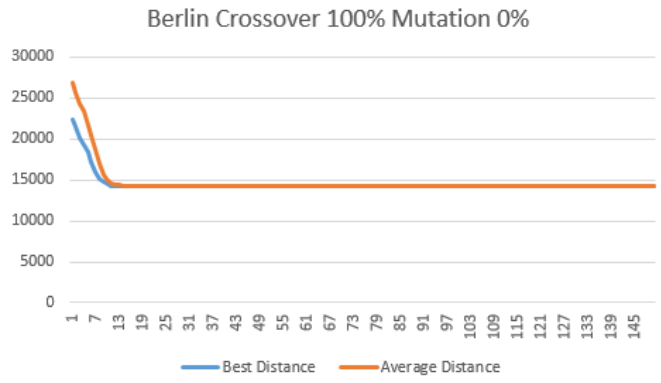
4.5 Fig 7.

A two point cross over is used for this particular test. Two point cross over is described in **3.3**. The results are fairly similar to the uniform cross over, but the generational best changes a bit from generation to generation. The best and average distance never really balance out like with uniform cross over.

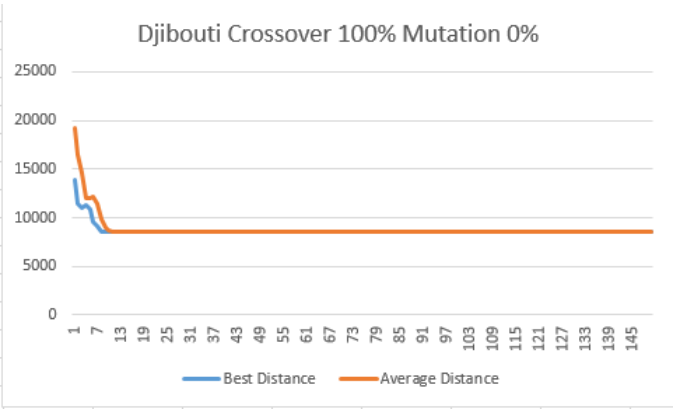
4.6 Fig 8.

The increased population size doesn't really change the overall best distance. The increase in population does show that, while the best does fluctuate between generations, the overall best does not improve. This could mean that the algorithm finds a global max a few generations in, and keeps searching for a better solution but ends up back at the global max.

5 Figures

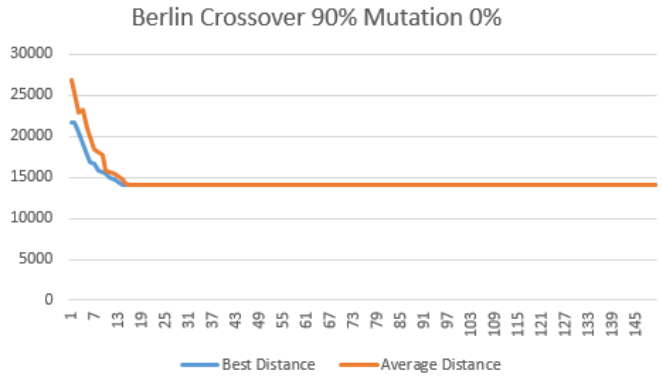


(a) Berlin

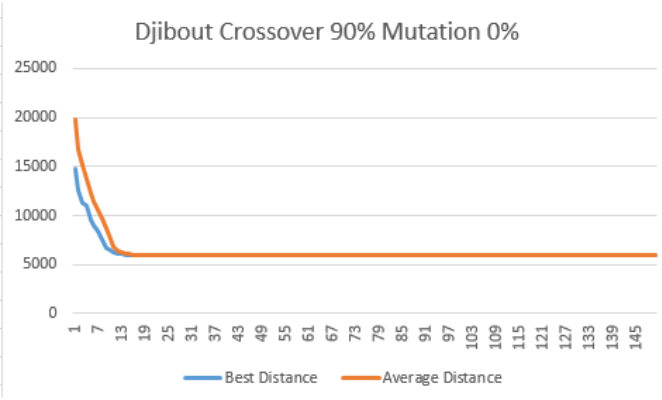


(b) Djibouti

Fig. 1. 100% Crossover with 0% Mutation

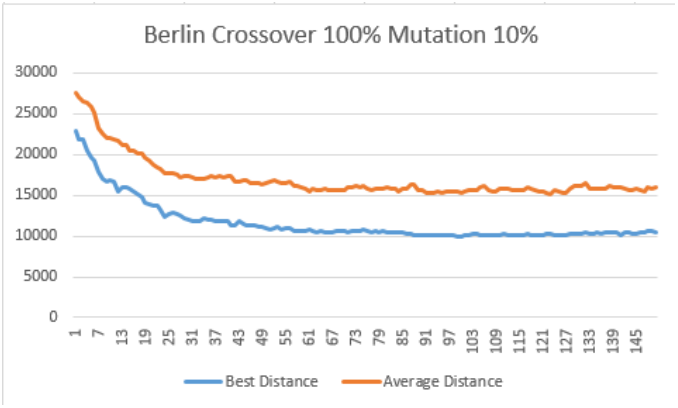


(a) Berlin

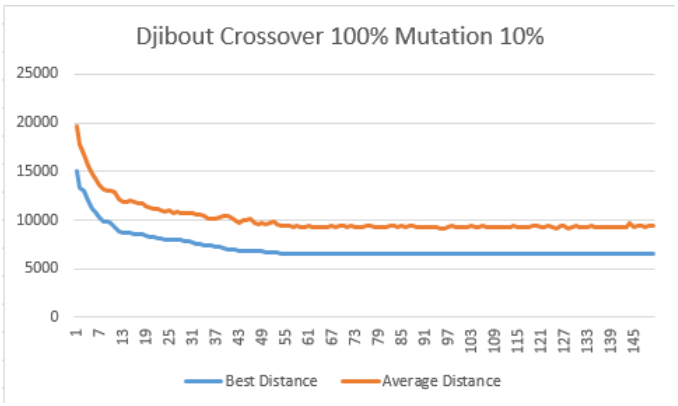


(b) Djibouti

Fig. 2. 90% Crossover with 0% Mutation

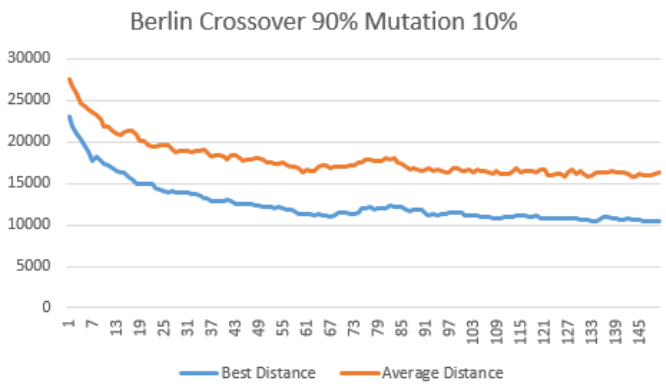


(a) Berlin

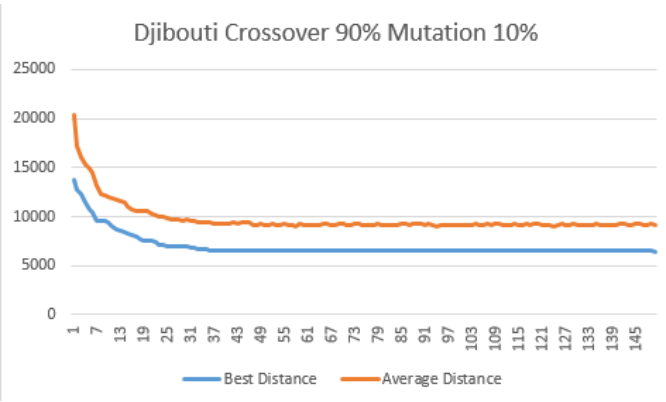


(b) Djibouti

Fig. 3. 100% Crossover with 10% Mutation

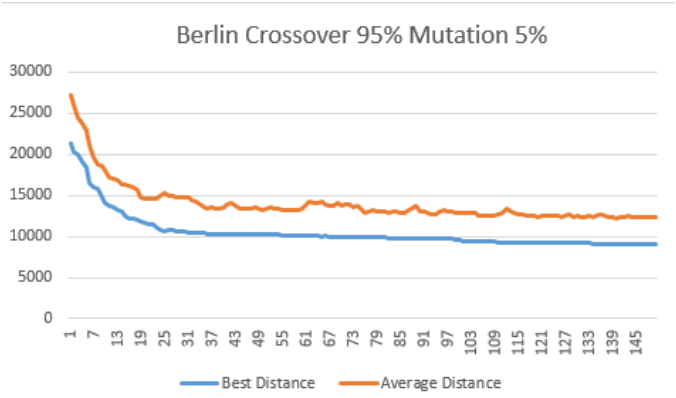


(a) Berlin

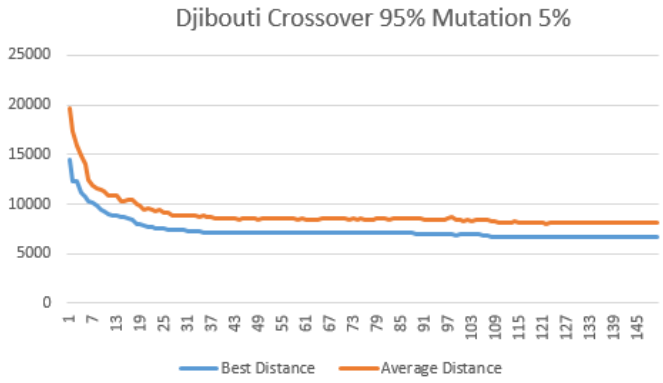


(b) Djibouti

Fig. 4. 90% Crossover with 10% Mutation

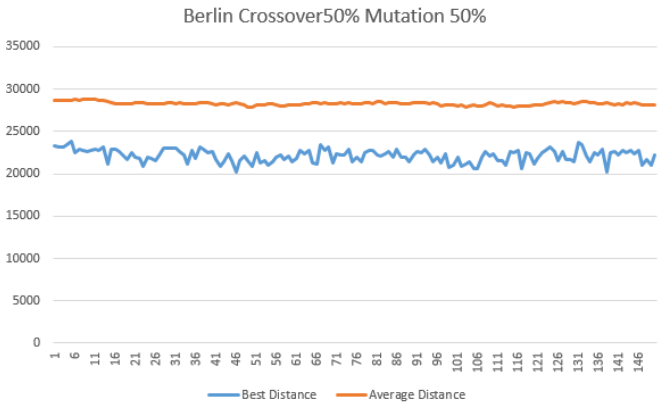


(a) Berlin

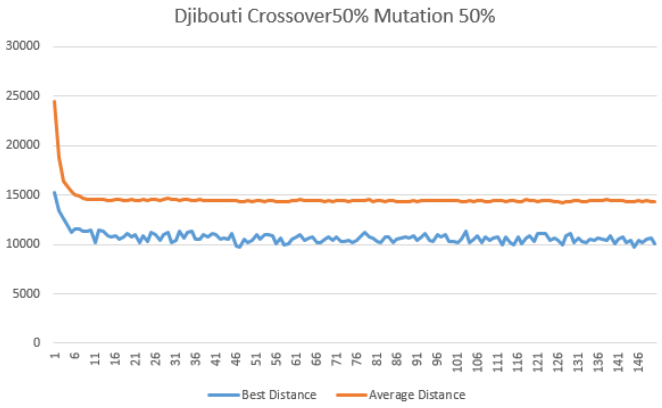


(b) Djibouti

Fig. 5. 95% Crossover with 5% Mutation

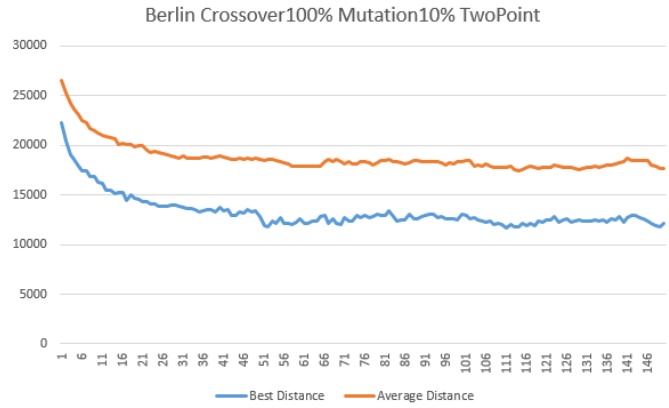


(a) Berlin

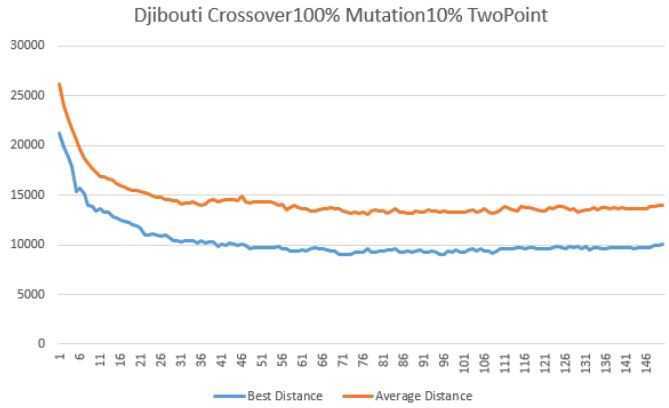


(b) Djibouti

Fig. 6. 50% Crossover with 50% Mutation

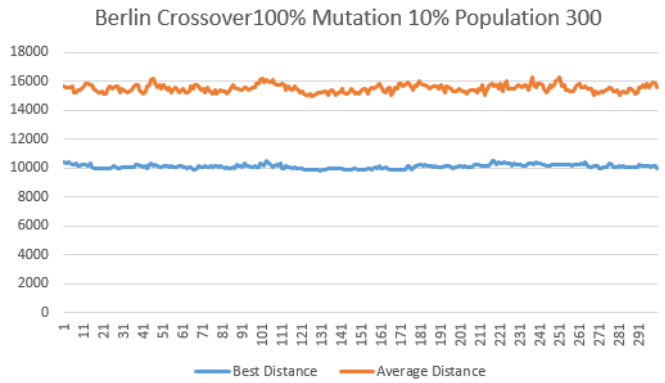


(a) Berlin

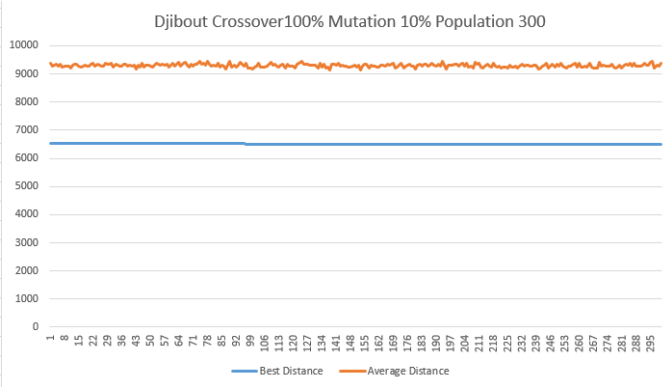


(b) Djibouti

Fig. 7. 100% Crossover with 10% Mutation with a Two Point Crossover



(a) Berlin



(b) Djibouti

Fig. 8. 100% Crossover with 10% Mutation with a Population size of 300

6 Conclusion

This was an interesting look into how a smart program can out-perform a brute force solution. There are a few ways the algorithm can be improved. If the distance values were stored in a 2-d array with the space between cities stored at the corresponding indices instead of re-calculating the distance each time a fair bit of time could be saved. The population size could have been reduced with a slight increase in the number of generations, which would have reduced the search space a bit. A smarter mutation method could have been used, as the algorithm only uses a random swap.

References

[1] S. Russell & P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd Edition. Prentice Hall, 2010.