Course CS-451: Computational Intelligence

Assignment 01 - Report

Ali Hamza (ah05084)
Haris Karim Ladhani (hl04349)
Synclair Samson (ss05901))

Monday 21st February, 2022

Contents

1 Genetic Algorithm Implementation

1.1 Parameters

We have chosen default parameters as a control to run our tests between what selection schemes give the best results.

- Population Size: 30
- OffSpring per Generation: 10
- Number of Generations: 100
- Mutation rate: 0.4
- Number of Iterations: 40

1.2 Crossover

The generic crossover function works as follows:

1. Select two random individuals from the selected parents
2. Select the first half the first individual and the second half of the second individual to create a child Chromosome
3. Select the first half of the second individual and the second half of the first individual to create a child Chromosome
4. Mutate the child Chromosomes
5. Append the child Chromosomes to the new population

This function slightly changes in TSP as we cannot have repeated genes in a chromosome. A random crossover point is chosen for the first individual and then the remainder of the second individual is added to the child chromosome. The process is repeated with the second individual and the first individuals flipped. The child chromosomes are then mutated and appended to the new population.

1.3 Mutation

The mutation function is quite simplistic where a chromosome is taken permutated on the basis of the mutation rate.

1.4 Chromosome Representations and Fitness Function

1.4.1 Travelling Salesman Problem (TSP)

The Travelling Salesman Problem (TSP) is a problem in which a salesman has to visit a number of cities and return to the starting city. The salesman has to visit each city exactly once and return to the starting city. The distance between two cities is the Euclidean distance.

To make implementation easier we removed the last restriction of the returning to the same city. The cities are represented as a list of coordinates, where each coordinate is a tuple of the form (x,y).

Each chromosome is represented as an ordered array of cities. The population was initialized by simply taking a random permutation of the initial list of cities. The following is a representation of a chromosome:

$$permutation([(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)])$$

The Fitness function is the total distance of the chromosome that was calculated by summing the distances between each city in the chromosome. The distance between two cities is then calculated as the Euclidean distance between two consecutive cities. This computation is slightly expensive as it involves a nested for loop of a time complexity of $O(n^2)$. The fitness of each chromosome can be mathematically represented as follows:

$$\sum_{i=1}^{n-1} \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2}$$

It is obvious that this is a minimization problem therefore the smaller the fitness value the better the solution.

1.4.2 Graph Coloring

The Graph Coloring problem is a problem in which a graph is colored with a number of colors such that no two adjacent vertices are colored with the same color. The graph is represented as a list of edges, where each edge is a tuple of the form (u,v). The maximum coloring of a graph will be the number of nodes in the graph.

Therefore, the chromosome is represented as an array of colors represented as integers ranging from 1 to n, where n is the number of nodes in the graph. The population was initialized by simply taking random integers between 1 and n for each edge. The following is a representation of a chromosome:

$$[randomint(1, n), randomint(1, n), \ldots, randomint(1, n)]$$

The fitness is then the number of color violations present in a solution. The number of color violations is calculated by counting the number of edges that are adjacent to two vertices that have the same color. This computation is slightly expensive as it involves a nested for loop of a time complexity of $O(n^2)$. The fitness of each chromosome can be mathematically represented as follows:

$$\sum_{i=1}^{n-1} \left\{ \begin{cases} 1 & \text{if color}(i) = \text{color}(i+1) \\ 0 & \text{otherwise} \end{cases} \right\}$$

Of course, there is a limitation within this fitness function which is that it does not account for a minimum unique coloring. This adds a level of complexity to the problem which was not implemented for the scope of this assignment. It is obvious that the lower the number of edge violations the better the solution, therefore this is a minimization problem.

1.4.3 Knapsack

The Knapsack problem is a problem in which a knapsack is filled with a number of items such that the total weight of the items is less than or equal to the capacity of the knapsack. The items are represented as a list of tuples of the form (weight, value). The knapsack is represented as a list of tuples of the form (weight, value).

The chromosome is represented as an array of 1's and 0's siginfying whether the item at that particual index was added to the knapsack or not. Therefore, a chromosome looks as follows:

$$[randomint(0,1), randomint(0,1), \ldots, randomint(0,1)]$$

The fitness is then the total value of the items in the knapsack. The total value is calculated by summing the values of the items in the knapsack. The fitness of each chromosome can be mathematically represented as follows:

$$\sum_{i=0}^{n} \left\{ \begin{cases} value(i) & \text{if item}(i) = 1 \\ 0 & \text{otherwise} \end{cases} \right\}$$

The limitation of this fitness function is that it does not account for a high weight knapsack, another parameter that could be considered is the total weight of the knapsack when looking at the fitness of the knapsack - then the fitness function could be the $\frac{total_value}{total_weight}$. This adds a level of complexity to the problem which was not implemented for the scope of this assignment therefore we did not implement it. It is obvious that the higher the total value of the items in the knapsack the better the solution, therefore this is a maximization problem.

2 Statistics and Analysis

2.1 Google Colab Notebook

The link to the Google Colab Notebook for the assignment is here

2.2 Fitness vs Generation Plots

2.2.1 FPS and Random

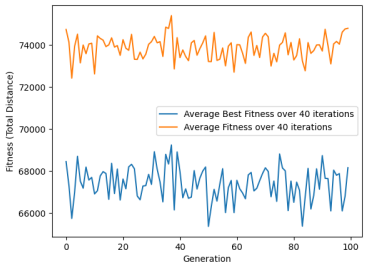

Figure 1: TSP

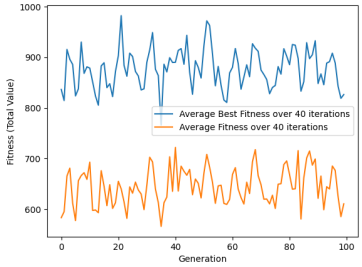

Figure 2: Knapsack

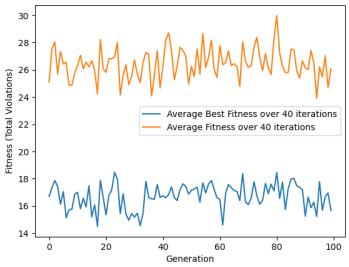

Figure 3: Graph Coloring

2.2.2 RBS and Truncation

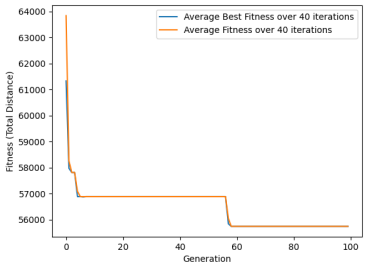

Figure 4: TSP

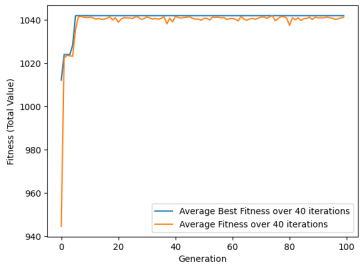

Figure 5: Knapsack

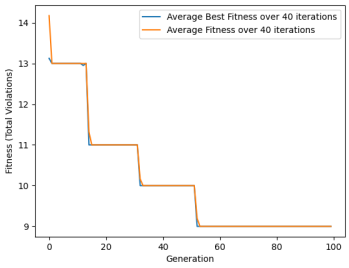

Figure 6: Graph Coloring

2.2.3 Truncation and Truncation

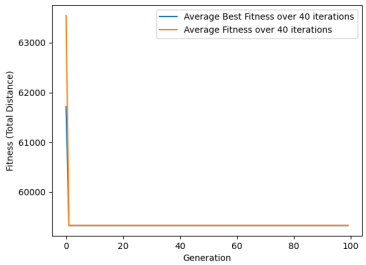

Figure 7: TSP

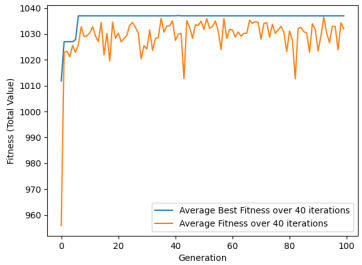

Figure 8: Knapsack

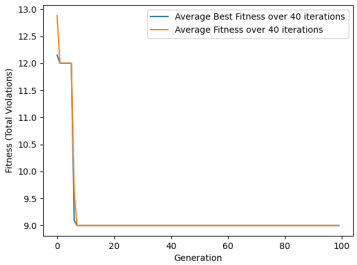

Figure 9: Graph Coloring

2.2.4 Random and Random

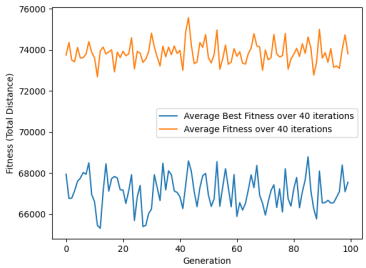

Figure 10: TSP

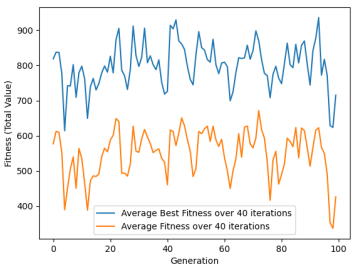

Figure 11: Knapsack

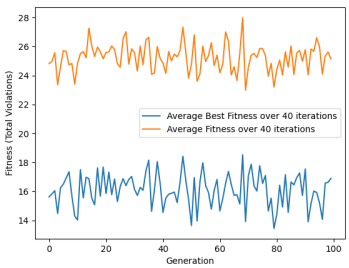

Figure 12: Graph Coloring

2.2.5 FPS and Truncation

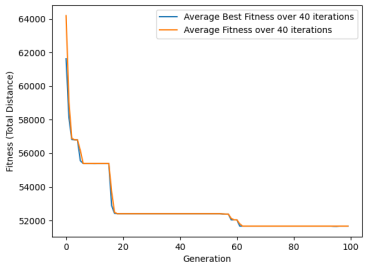

Figure 13: TSP

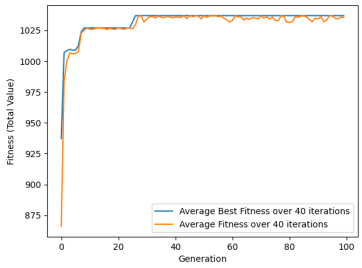

Figure 14: Knapsack

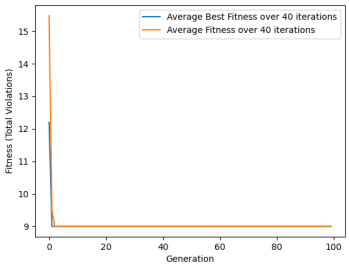

Figure 15: Graph Coloring

2.2.6 Random and FPS

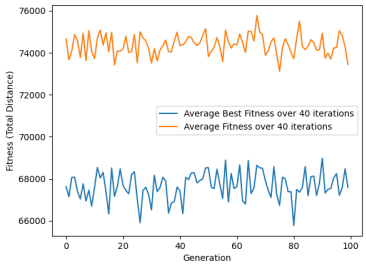

Figure 16: TSP

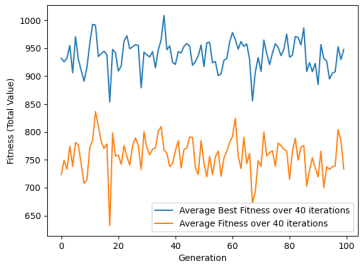

Figure 17: Knapsack

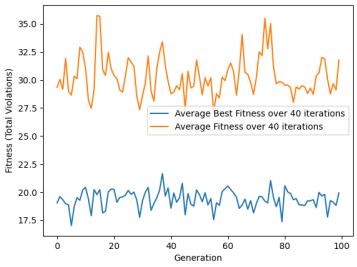

Figure 18: Graph Coloring

2.2.7 Random and Truncation

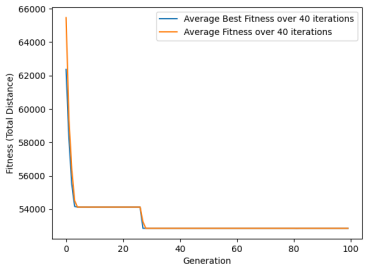

Figure 19: TSP

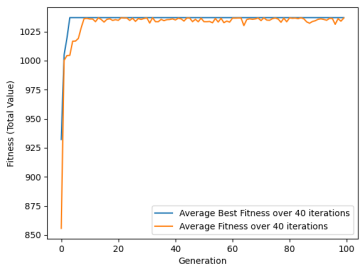

Figure 20: Knapsack

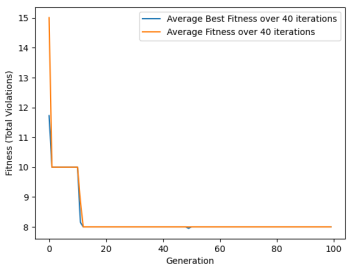

Figure 21: Graph Coloring

2.2.8 Truncation and RBS

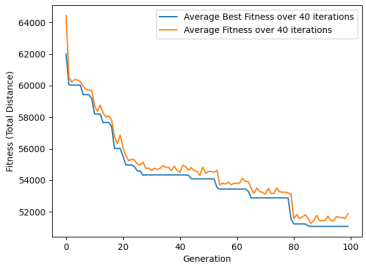

Figure 22: TSP

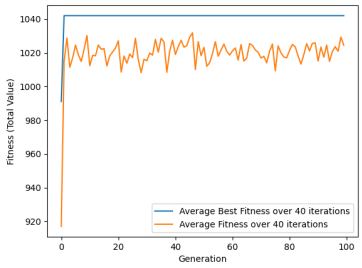

Figure 23: Knapsack

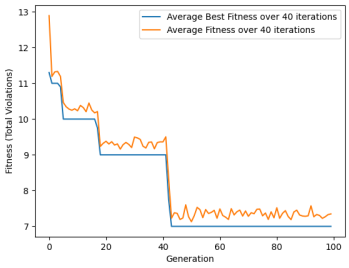

Figure 24: Graph Coloring

2.2.9 RBS and Random

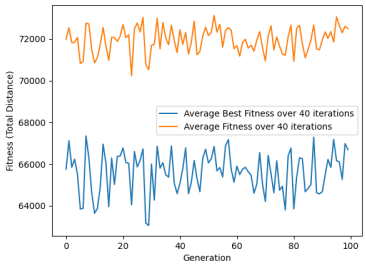

Figure 25: TSP

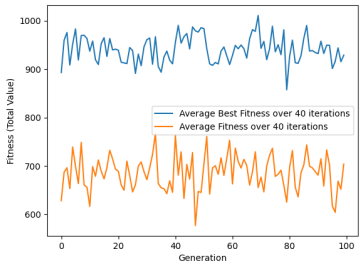

Figure 26: Knapsack

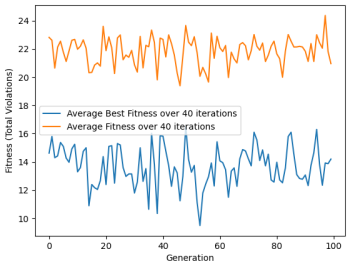

Figure 27: Graph Coloring

Parent Selection	Survivor Selection	Best Fitness	Average Fitness
Knapsack			
Fitness Proportion	Random	982.42	645.23
Rank Based	Truncation	1042	1039.03
Truncation	Truncation	1037.12	1029.05
Random	Random	935.52	550.49
Fitness Proportion	Truncation	1037	1029.77
Random	Truncation	1037	1031.86
Truncation	Rank Based	1042	1019.33
Rank Based	Random	1010.95	685.83
Random	Fitness Proportion	1008.6	757.49
Graph Coloring			
Fitness Proportion	Random	14.5	26.45
Rank Based	Truncation	9	10.14
Truncation	Truncation	9	9.2
Random	Random	13.45	25.24
Fitness Proportion	Truncation	9	9.07
Rank Based	Fitness Proportion	14.72	25.46
Random	Truncation	7.95	8.28
Truncation	Rank Based	7	8.43
Rank Based	Random	9.5	21.83
Random	Fitness Proportion	17.02	30.3
Travelling Salesman Problem			
Fitness Proportion	Random	65361.51	73877.51
Rank Based	Truncation	55745.59	56504.16
Truncation	Truncation	59324.92	59367.14
Random	Random	65303.59	73838.31
Fitness Proportion	Truncation	51655.59	52778.71
Rank Based	Fitness Proportion	63406.15	72516.23
Random	Truncation	52859.87	53402.1
Truncation	Rank Based	51085.47	54714.02
Rank Based	Random	63058.26	71953.08
Random	Fitness Proportion	65778.02	74376.75

2.3 Analysis

In the case of selection where the Parent is Fitness Proportion & Survivor is Random, it can be seen that there are inconsisties across all the optimization problems. This combination of selection has had the same impact on all three and what's worse is that there is no convergence to a single point. The randomness can also be observed through the difference between the average and the best fitness values in all of the combinatorial optimization problems.

In the case of selection where the Parent is Rank Based & Survivor is Truncation, it can be observed that the convergence is almost exponential in the case of TSP and Knapsack. However, with Graph Coloring it still takes some time to converge. This is a combination of selection scheme that seems to have had a similiar positive impact across all cases.

In the case of selection where the Parent is Truncation & Survivor is Truncation, the convergence happens rather quickly. In fact, it's faster than the last selection scheme (RBS and Truncation). However, for Knapsack, at the near point of convergence the pattern seem to be very random for the average average fitness value.

In the case of selection where the Parent is Random & Survivor is Random, it's observed that the pattern here is very random and it seems similiar to the selection scheme combination, FPS and Random. Similiar to the aforementioned selection scheme, this combination has a high margin of difference between the average fitness values and the average best fitness values. Which is another observation that visualizes the randomness in this selection sxheme across all optimization cases.

In the case of selection where the Parent is Fitness Proportion & Survivor is Truncation, where it can be observed that, like some of the previous selection scheme combinations, the convergence here is exponential, more so in the case of Graph Coloring. However, in TSP it did take, relative to other combinatorial optimization problems, more time to converge.

In the case of selection where the Parent is Random & Survivor is FPS, what can be absorved here is that there is no convergence. Just like section 2.2.1 and section 2.2.4, the pattern here is completely random. The difference between the average fitness value and the average best fitness value is also great across all the combinatorial optimization problems. However, what can be noted here is that in all three of the mentioned selection schemes the one thing that has been consistent is that they have at least one selection that is Random.

In the case of selection where the Parent is Random & Survivor is Truncation, it can be observed that there is almost an immediate convergence across all problems. Except for a slight hiccup in TSP and Graph Coloring. What's interesting here is that even though one of the selection is Random but it still manages to converge. This is the best combination so far to use Random selection in.

In the case of selection where the Parent is Truncation & Survivor is RBS, it can be seen that there is convergence. However, the convergence here is not as immediate as expected. It takes sometime to converge.

In the case of selection where the Parent is RBS & Survivor is Random, there is no convergence. As one of the selection scheme is Random, thus the pattern here is random.