Analysis and Findings

CS 451 - Computational Intelligence

Assignment 1 – Evolutionary Algorithm

Shalin Amir Ali (sa06132), Syeda Areesha Najam (sn05985)

For each of the given problem, this pdf contains following,

- Chromosome Representation and Fitness Functions
- Plots of Avg. BSF and Avg. ASF for various combinations of schemes
- Your analysis for various schemes and the scheme that worked best in your case.

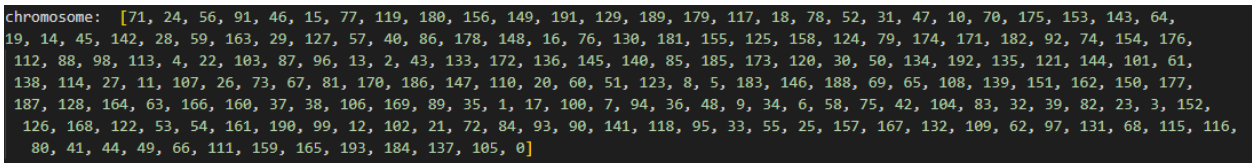
The default parameters are,

- Population size: 30
- Number of offspring to be produced in each generation :10
- No. of generations: 100
- Mutation rate: 0.5
- No of Iterations: 10

<u>1. Travelling Salesman Problem (TSP):</u>

1.1 Chromosome Representation:

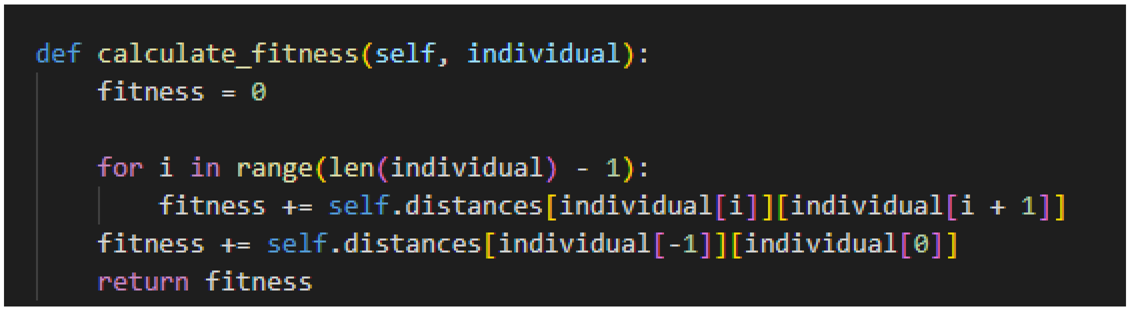
Since there are total 194 cities in the dataset, we initialize our population by random permutation of 194 cities. Each chromosome represents a path passing through all cities.



```
chromosome:  [71, 24, 56, 91, 46, 15, 77, 119, 180, 156, 149, 191, 129, 189, 179, 117, 18, 78, 52, 31, 47, 10, 70, 175, 153, 143, 64,
19, 14, 45, 142, 28, 59, 163, 29, 127, 57, 40, 86, 178, 148, 16, 76, 130, 181, 155, 125, 158, 124, 79, 174, 171, 182, 92, 74, 154, 176,
 112, 88, 98, 113, 4, 22, 103, 87, 96, 13, 2, 43, 133, 172, 136, 145, 140, 85, 185, 173, 120, 30, 50, 134, 192, 135, 121, 144, 101, 61,
 138, 114, 27, 11, 107, 26, 73, 67, 81, 170, 186, 147, 110, 20, 60, 51, 123, 8, 5, 183, 146, 188, 69, 65, 108, 139, 151, 162, 150, 177,
 187, 128, 164, 63, 166, 160, 37, 38, 106, 169, 89, 35, 1, 17, 100, 7, 94, 36, 48, 9, 34, 6, 58, 75, 42, 104, 83, 32, 39, 82, 23, 3, 152,
  126, 168, 122, 53, 54, 161, 190, 99, 12, 102, 21, 72, 84, 93, 90, 141, 118, 95, 33, 55, 25, 157, 167, 132, 109, 62, 97, 131, 68, 115, 116,
   80, 41, 44, 49, 66, 111, 159, 165, 193, 184, 137, 105, 0]
```

1.2 Fitness Function:

Following is the fitness function for knapsack problem,

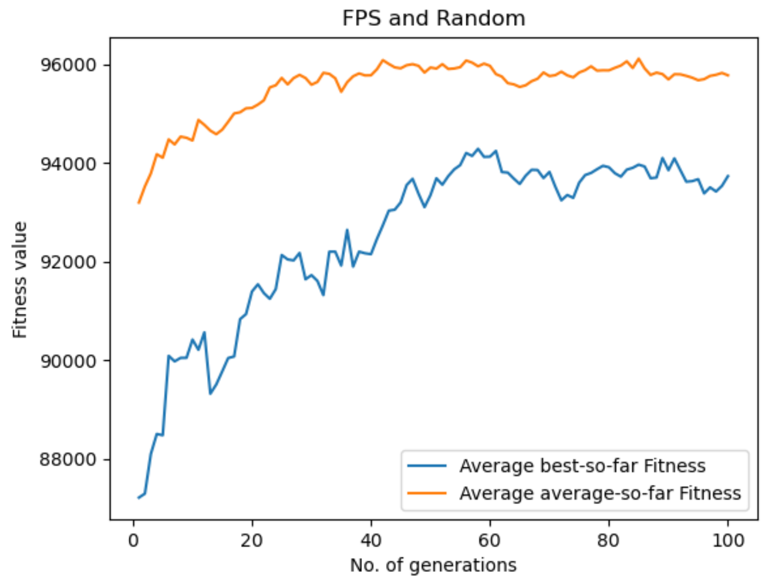


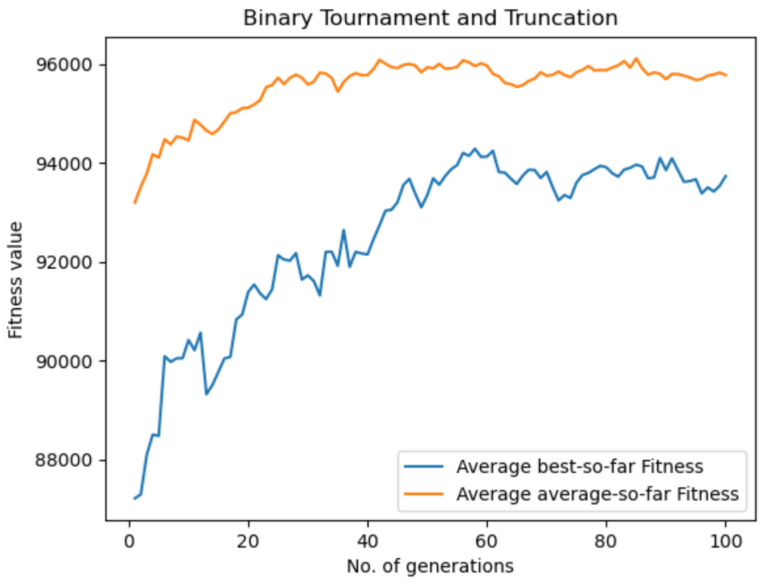
```python
def calculate_fitness(self, individual):
    fitness = 0

    for i in range(len(individual) - 1):
        fitness += self.distances[individual[i]][individual[i + 1]]
    fitness += self.distances[individual[-1]][individual[0]]
    return fitness
```

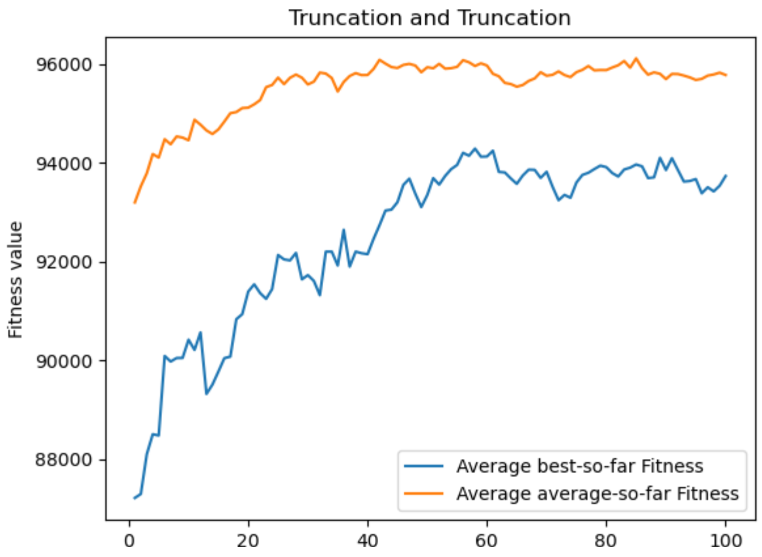
The total cost of travel between the cities is our fitness function in this case. The total cost of travel is calculated as follows: Create a 194x194 matrix storing the euclidean distance between each city. For each pair of consecutive cities in the chromosome, the distance between those cities is retrieved from a distances matrix and added to the fitness. Finally, the distance between the final city and the starting city is added to fitness to complete the loop.

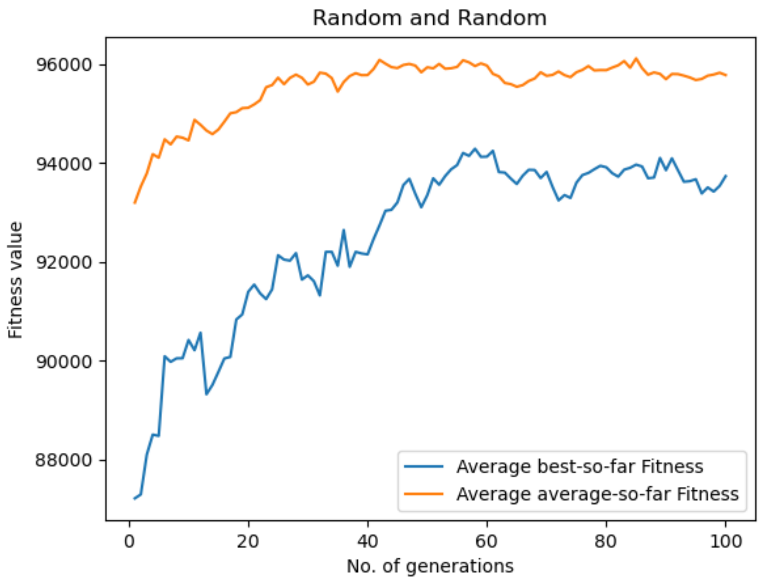
The resulting fitness value is then returned as the output of the method, representing the fitness of the given individual candidate solution.

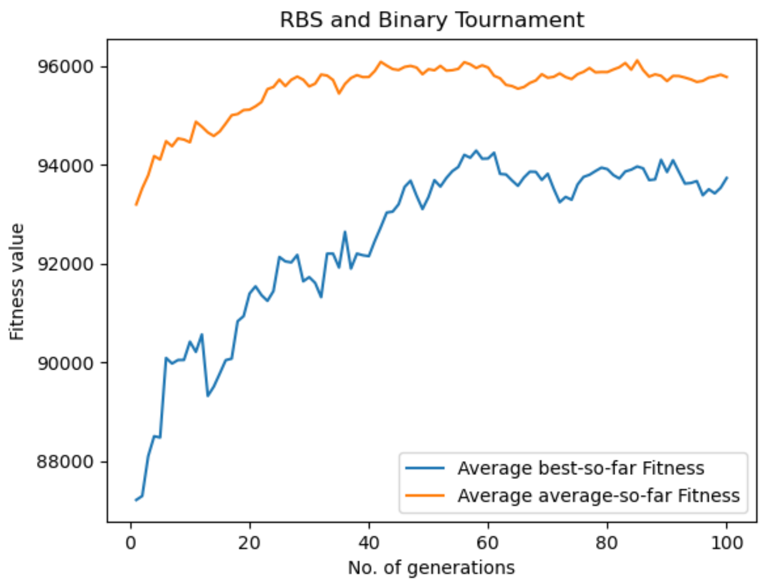
1.3 Plots:

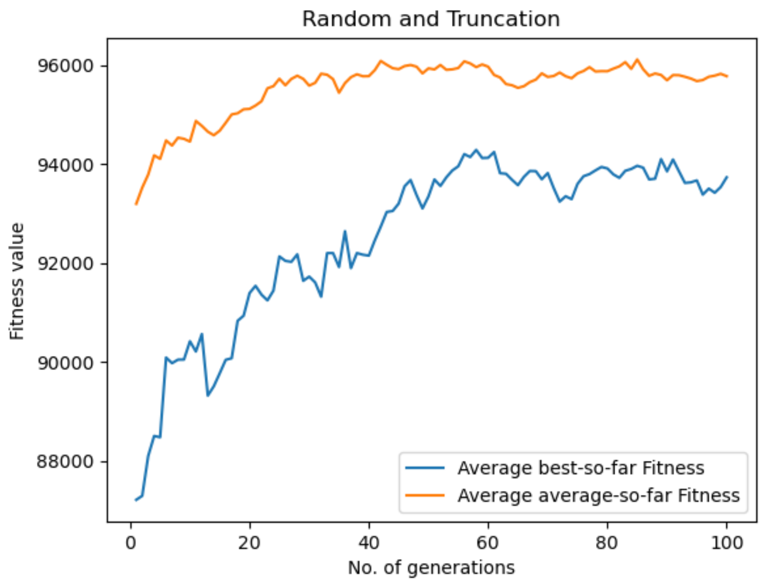



Binary Tournament and Truncation
Fitness value
No. of generations
Average best-so-far Fitness
Average average-so-far Fitness
96000
94000
92000
90000
88000
0
20
40
60
80
100


Truncation and Truncation
Fitness value
Average best-so-far Fitness
Average average-so-far Fitness
96000
94000
92000
90000
88000
0
20
40
60
80
100


Random and Random
Fitness value
No. of generations
Average best-so-far Fitness
Average average-so-far Fitness
96000
94000
92000
90000
88000
0
20
40
60
80
100





1.4 Results:

The results are reported in 'TSP Results.txt'. As per obtained results we see that minimum distance to visit each city exactly once and returns to the origin city is 49918 where the route would be,

[25, 44, 78, 39, 18, 34, 20, 35, 105, 143, 170, 128, 129, 81, 119, 164, 167, 173, 186, 158, 161, 135, 193, 189, 154, 151, 174, 171, 85, 110, 55, 138, 127, 136, 38, 15, 5, 106, 107, 104, 14, 4, 31, 16, 13, 73, 92, 100, 183, 182, 178, 168, 192, 163, 185, 144, 175, 191, 172, 166, 159, 102, 37, 91, 82, 115, 83, 68, 79, 22, 157, 169, 12, 118, 98, 74, 71, 75, 113, 86, 36, 63, 54, 41, 57, 94, 101, 108, 131, 93, 124, 187, 160, 112, 153, 97, 7, 3, 62, 126, 146, 111, 133, 139, 155, 141, 140, 152, 150, 96, 58, 10, 53, 45, 6, 2, 47, 77, 69, 43, 48, 95, 114, 116, 89, 64, 176, 162, 177, 190, 148, 156, 180, 184, 179, 188, 165, 181, 142, 132, 76, 109, 60, 29, 87, 99, 122, 117, 137, 103, 19, 27, 42, 49, 52, 121, 145, 88, 125, 149, 147, 123, 66, 32, 51, 40, 26, 24, 59, 70, 11, 21, 33, 9, 50, 80, 1, 46, 8, 17, 72, 120, 90, 134, 130, 65, 28, 67, 30, 56, 23, 84, 61, 0].

1.5 Analysis:

The ‘TSP Results.txt’ contains solutions to the given TSP problem using different selection schemes, i.e., FPS (Fitness Proportional Selection), Binary Tournament, Truncation, RBS (Rank-Based Selection), and Random. The best possible optimal route reported above shows a chromosome with the best fitness vale of 49918 which is obtained through combination of Truncation selection scheme for both parent selection and survivor selection.

It's difficult to say which selection scheme is "bad" as it depends on the specific problem and its constraints. However, based on the given data, it can be seen that the lowest fitness value is 83238, which is obtained using the Random selection scheme. This suggests that the Random selection scheme does not perform well this particular TSP problem. But again, it depends on the specific problem and its constraints and cannot be generalized.

Additionally, the plots indicate that all the combinations of parent and survival selections schemes have a similar trend in terms of performance.These trends indicate gradual improvement of algorithm, say convergence to best solution. The average average fitness trend shows the overall and stable improvement towards finding best solution, ensuring convergence.

Note**: The tables are submitted under folder, “TSP tables”.

table1_[selectionTech1]_[selectionTech2] = ASF and BSF for generations

table2_[selectionTech1]_[selectionTech2] = average BSF for generations after K iterations

table3_[selectionTech1]_[selectionTech2] = average ASF for generations after K iterations

2. Knapsack Problem:

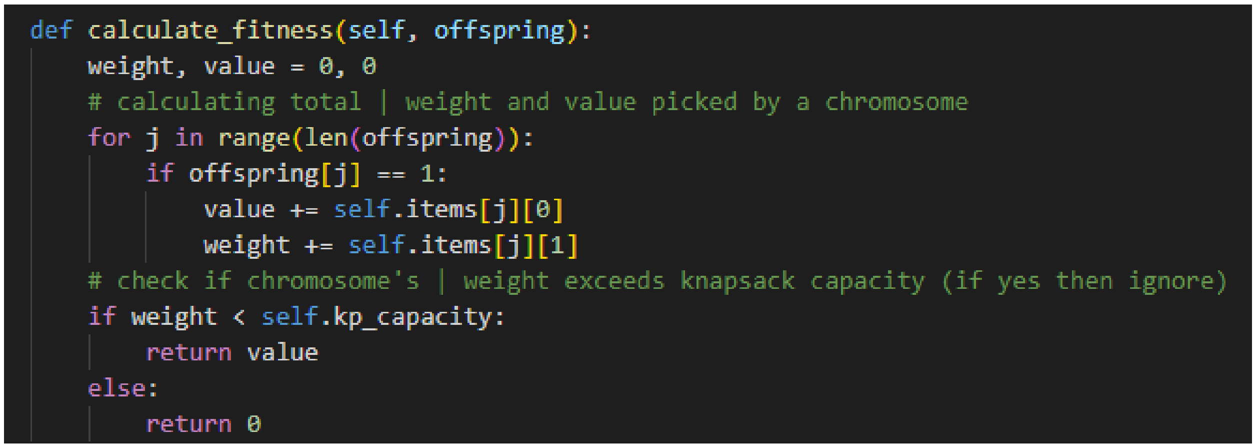
2.1 Chromosome Representation:

For this problem, in a population of 30 chromosomes (as per default parameters), a chromosome was represented as a list of zeros and ones. The length of the said list is equal to total number of items, which are 23 in our case.

[1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0]

Note that 1s represent items to be picked. For instance, 1 at index 0 indicates first item has to be picked and 0 at index 5 indicates item that has not to be picked.

2.2 Fitness Function: Following is the fitness function for knapsack problem,

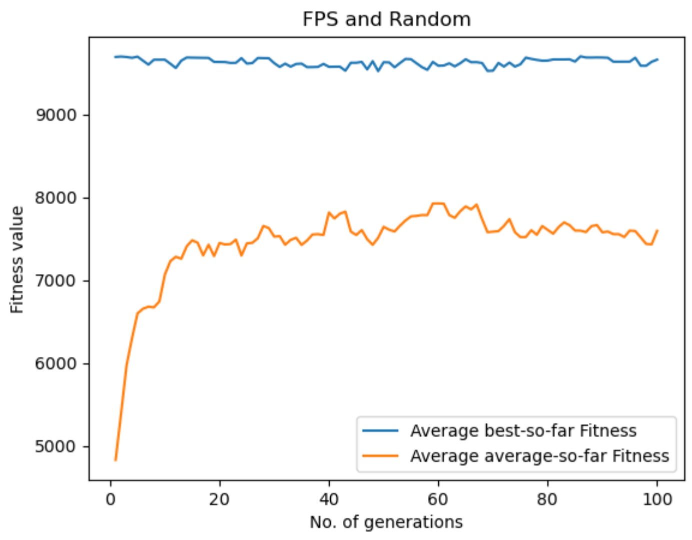


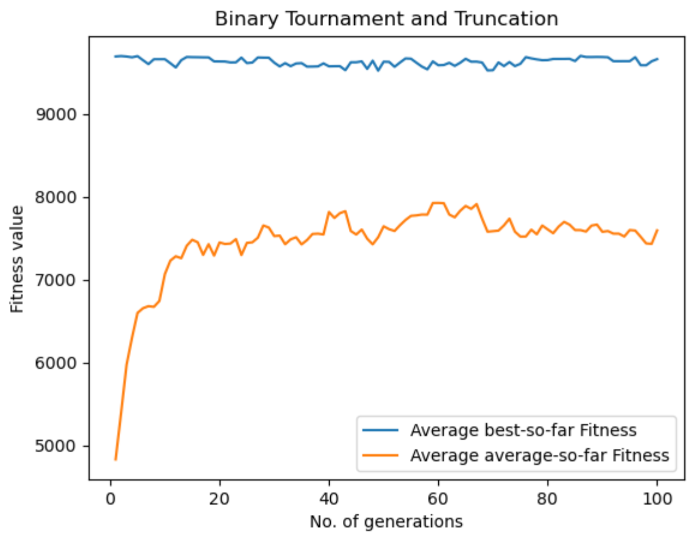
```python
def calculate_fitness(self, offspring):
    weight, value = 0, 0
    # calculating total | weight and value picked by a chromosome
    for j in range(len(offspring)):
        if offspring[j] == 1:
            value += self.items[j][0]
            weight += self.items[j][1]
    # check if chromosome's | weight exceeds knapsack capacity (if yes then ignore)
    if weight < self.kp_capacity:
        return value
    else:
        return 0
```

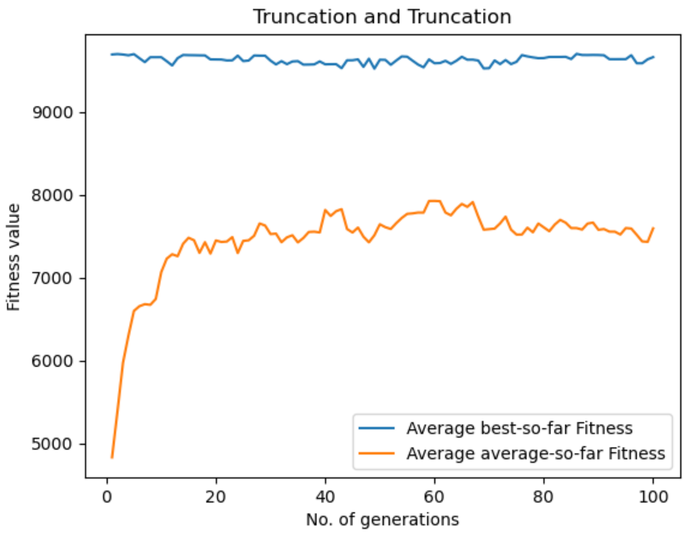
The function takes an offspring chromosome as input and iterates through its genes (j-th item) to determine if it is selected (genes with value 1) or not. If the gene is selected, its value and weight are added to the total value and weight of the chromosome.

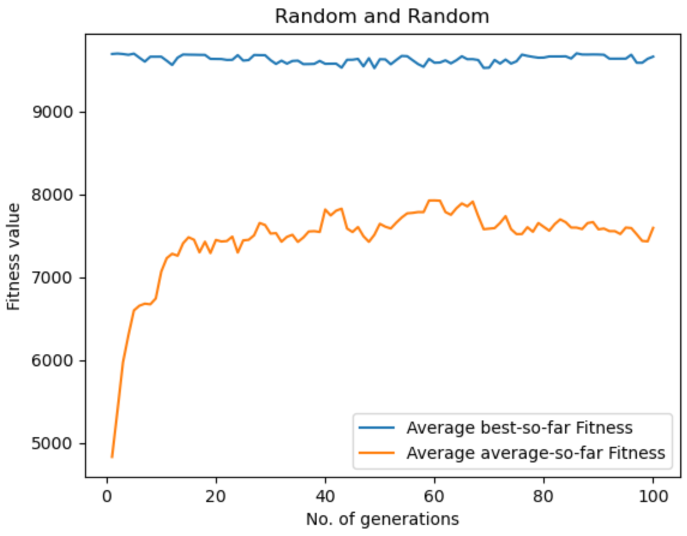
After all the genes are processed, the function checks if the total weight of the chromosome exceeds the knapsack capacity. If it does, the function returns 0 as the chromosome is not a valid solution. If the total weight is within the capacity, the function returns the total value of the items selected by the chromosome as the fitness value.

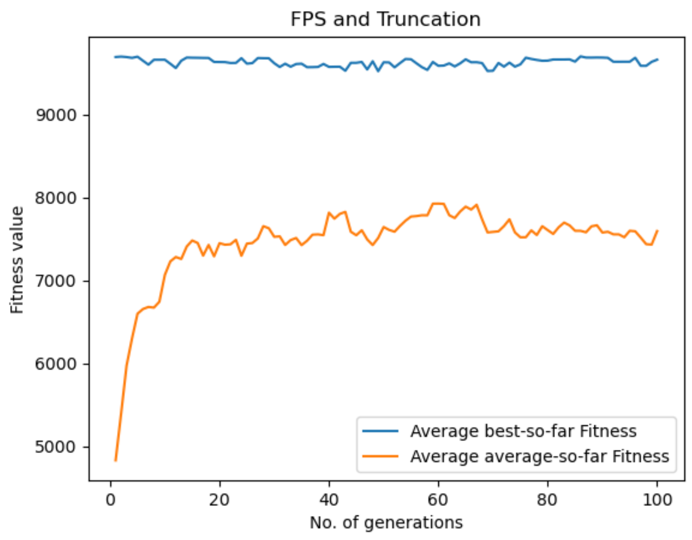
2.3 Plots:

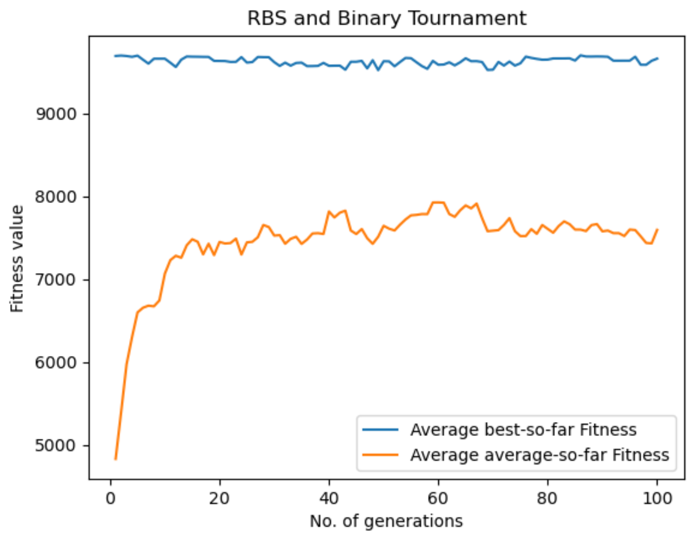

FPS and Random
Fitness value
No. of generations
Average best-so-far Fitness
Average average-so-far Fitness
9000
8000
7000
6000
5000
0
20
40
60
80
100

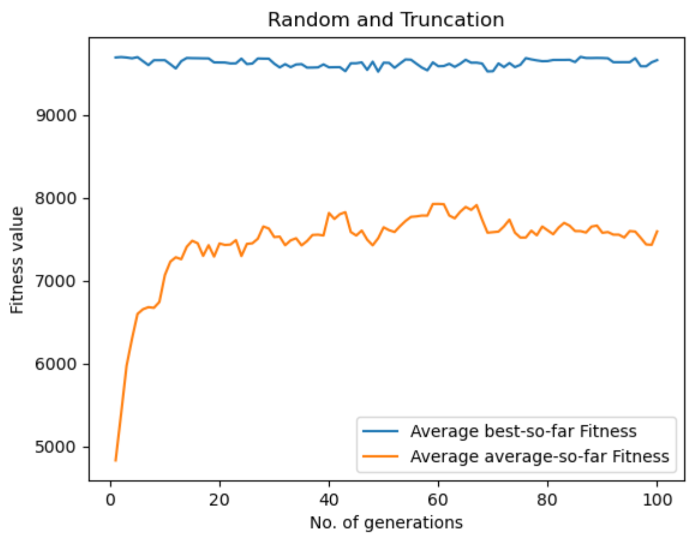

Binary Tournament and Truncation
Fitness value
No. of generations
Average best-so-far Fitness
Average average-so-far Fitness
9000
8000
7000
6000
5000
0
20
40
60
80
100


Truncation and Truncation
Fitness value
No. of generations
Average best-so-far Fitness
Average average-so-far Fitness
9000
8000
7000
6000
5000
0
20
40
60
80
100

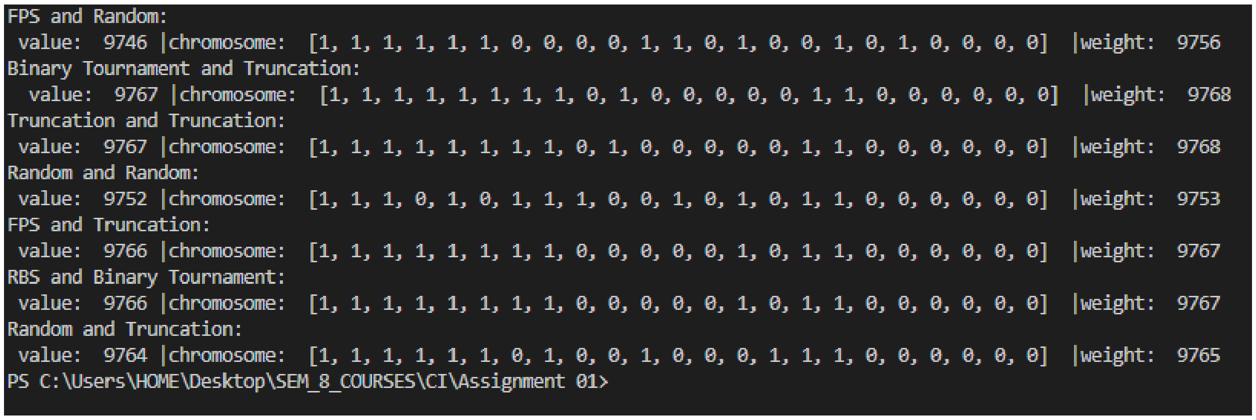

Random and Random
Fitness value
No. of generations
Average best-so-far Fitness
Average average-so-far Fitness
9000
8000
7000
6000
5000
0
20
40
60
80
100


FPS and Truncation
Fitness value
9000
8000
7000
6000
5000
0
20
40
60
80
100
No. of generations
Average best-so-far Fitness
Average average-so-far Fitness


RBS and Binary Tournament
Fitness value
9000
8000
7000
6000
5000
0
20
40
60
80
100
No. of generations
Average best-so-far Fitness
Average average-so-far Fitness



2.4 Results:



```
FPS and Random:
 value:  9746 |chromosome:  [1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0]  |weight:  9756
Binary Tournament and Truncation:
  value:  9767 |chromosome:  [1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0]  |weight:  9768
Truncation and Truncation:
 value:  9767 |chromosome:  [1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0]  |weight:  9768
Random and Random:
 value:  9752 |chromosome:  [1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0]  |weight:  9753
FPS and Truncation:
 value:  9766 |chromosome:  [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0]  |weight:  9767
RBS and Binary Tournament:
 value:  9766 |chromosome:  [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0]  |weight:  9767
Random and Truncation:
 value:  9764 |chromosome:  [1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0]  |weight:  9765
PS C:\Users\HOME\Desktop\SEM_8_COURSES\CI\Assignment 01>
```

For this problem we can say that optimal knapsack would be of value 9767 with weight 9768 (total 11 items out of 23) and the selected items (value, weight, index) would be,

(981, 983, 1)

(980, 982, 2)

(979, 981, 3)

(978, 980, 4)

(977, 979, 5)

(976, 978, 6)

(487, 488, 7)

(974, 976, 8)

(485, 486, 10)

(976, 969, 16)

(974, 966, 17)

Say, item 1, item 2, item 3, item 4, item 5, item 6, item 7, item 8, item 10, item 16 and item 17,

represented by following chromosome,

[1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0]

2.5 Analysis:

The figure attached under Result represents solutions to the given knapsack problem using different selection schemes, i.e., FPS (Fitness Proportional Selection), Binary Tournament, Truncation, RBS (Rank-Based Selection), and Random. Let us see what this figure has to say.

Note that the solutions are presented as chromosomes, where a chromosome represents a combination of items to be included in the knapsack and each element of the chromosome represents whether a corresponding item is included or not. The fitness value represents the total value of the items included in the knapsack. The weight of the knapsack represents the total weight of the items included in the knapsack.

It can be seen that the highest fitness value is 9767, which is obtained using Truncation and Binary Tournament selection schemes (Trruncation and Truncation or Truncation and Binary Tournament). The best chromosome is also the same for both selection schemes. Therefore, we can conclude that Truncation and Binary Tournament selection schemes are good for this knapsack problem.

It can be seen that the lowest fitness value is 9752, which is obtained using the Random selection scheme. This suggests that the Random selection scheme may not perform as well as the other selection schemes for this particular knapsack problem (we saw this for TSP too, as discussed above). But again, it depends on the specific problem and its constraints and cannot be generalized. Thus, *similar to TSP*, it is difficult to say which selection scheme is "bad" as it depends on the specific problem and its constraints.

Moreover, from the plots attached above we see that all the combinations of parent and survival selections used in our program perform roughly same (in terms of trend). These trends indicate gradual improvement of algorithm, say convergence to best solution. The average average fitness trend shows the overall improvement of the population of solutions. Since the average average fitness is increasing, it indicates that the population as a whole is becoming more fit over time. At the same time, the average best fitness curve shows stable trend towards finding better solutions.

Note**: The tables are submitted under folder, “KP tables”.

table1_[selectionTech1]_[selectionTech2] = ASF and BSF for generations

table2_[selectionTech1]_[selectionTech2] = average BSF for generations after K iterations

table3_[selectionTech1]_[selectionTech2] = average ASF for generations after K iterations

<u>3. Graph Coloring:</u>

The graph coloring problem essentially deals with coloring the graph G = (V, E) in such a way that a node adjacent to any given node have unique colors. The parameter that we need to optimize in this problem is to find the minimum number of colors required to color the graph keeping the constraint into consideration. We maintain an adjacency matrix to represent the graph so whenever we need to check for an edge between two nodes, we look up for the value in the adjacency matrix.

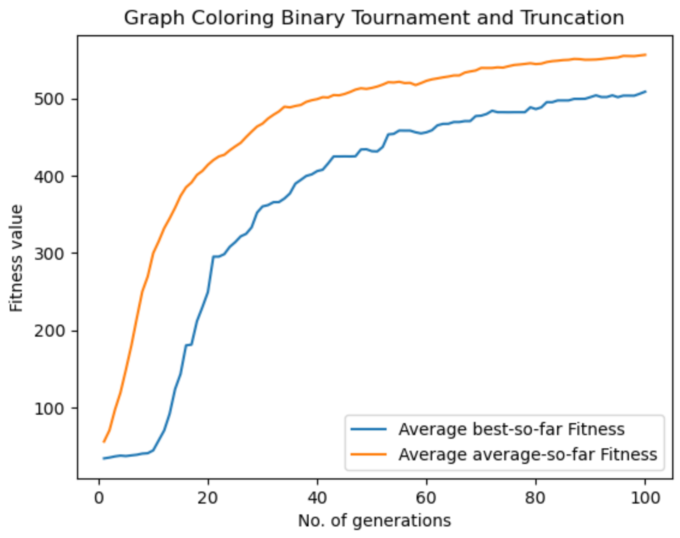
3.1 Chromosome Representation:

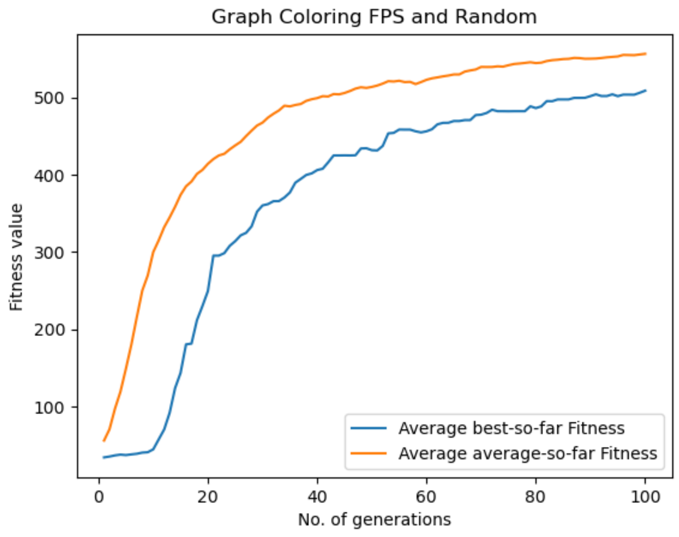
The best chromosome representation would be to have a list of size V = 100 (number of vertices in the dataset) initialized by random colors from 1 to k, where k is equal to degree of the graph. We then initialize the entire population with such chromosomes, each having randomly assigned graph coloring.

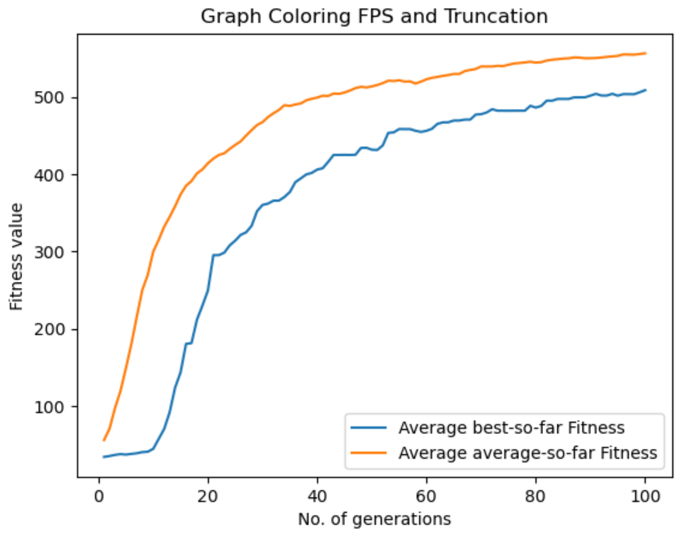
3.2 Fitness Function:

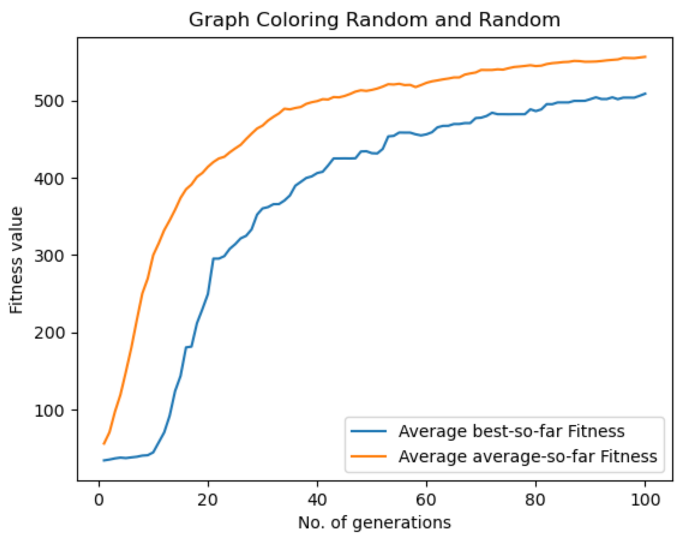
Since we have to minnimize the number of colors to be used, we calculate the fitness of each chromosome by adding 1 as a penalty to the overall fitness whenever two adjacent nodes have the same color value. This would ensure that the chromosomes having higher frequency of adjacent nodes with same colors would be considered less fit. Since this is a minimization problem, the lower the fitness value of a chromosome, the higher the chances of it to be considered as a potential solution.

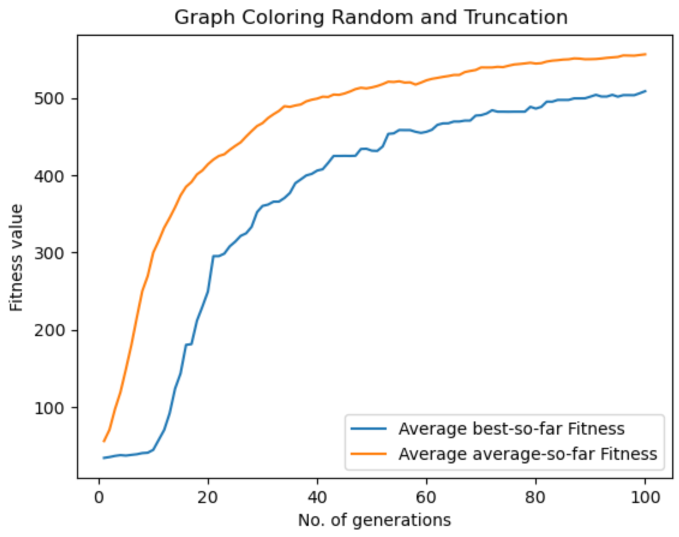
3.3 Plots:

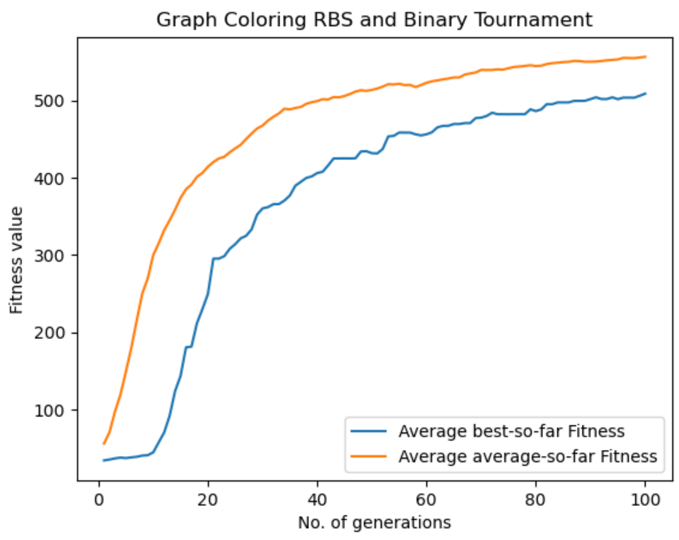


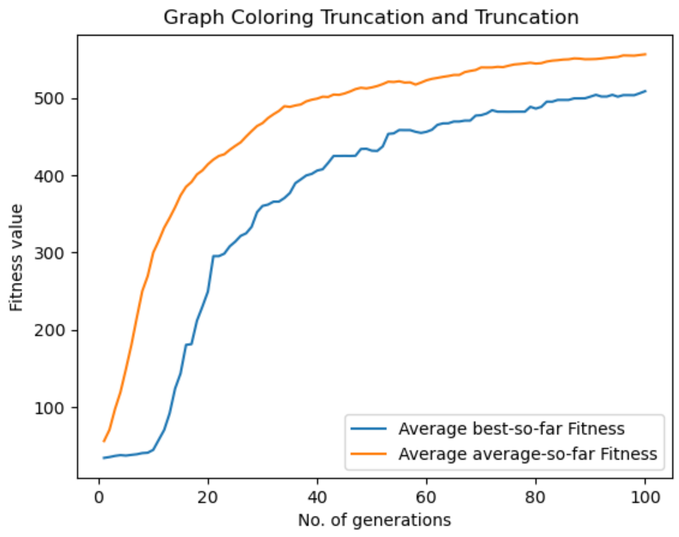

Graph Coloring FPS and Random
Fitness value
No. of generations
Average best-so-far Fitness
Average average-so-far Fitness
0
20
40
60
80
100
100
200
300
400
500


Graph Coloring FPS and Truncation
Fitness value
No. of generations
Average best-so-far Fitness
Average average-so-far Fitness
0
20
40
60
80
100
100
200
300
400
500


Graph Coloring Random and Random
Fitness value
No. of generations
Average best-so-far Fitness
Average average-so-far Fitness

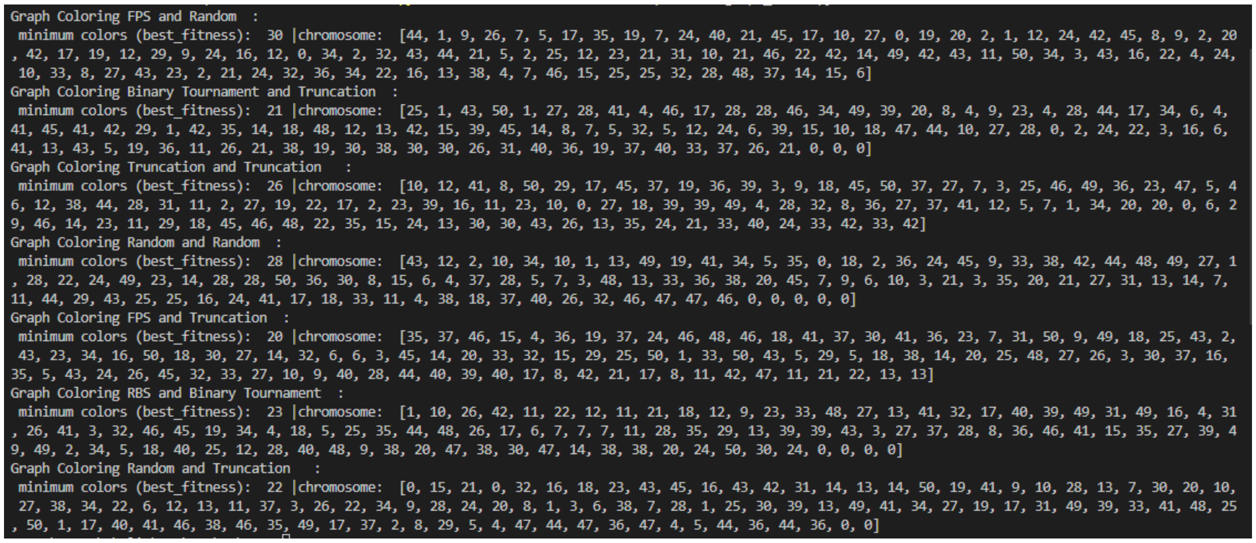

Graph Coloring Random and Truncation
Fitness value
No. of generations
Average best-so-far Fitness
Average average-so-far Fitness





3.4 Results:

From the results below, the best fitness obtained was 20 for the combination of FPS and Truncation as parent and survival selections respectively. The fitness indicates that the minimum number of colors required to color the graph would be 20.



```
Graph Coloring FPS and Random  :
 minimum colors (best_fitness):  30 |chromosome:  [44, 1, 9, 26, 7, 5, 17, 35, 19, 7, 24, 40, 21, 45, 17, 10, 27, 0, 19, 20, 2, 1, 12, 24, 42, 45, 8, 9, 2, 20
, 42, 17, 19, 12, 29, 9, 24, 16, 12, 0, 34, 2, 32, 43, 44, 21, 5, 2, 25, 12, 23, 21, 31, 10, 21, 46, 22, 42, 14, 49, 42, 43, 11, 50, 34, 3, 43, 16, 22, 4, 24,
 10, 33, 8, 27, 43, 23, 2, 21, 24, 32, 36, 34, 22, 16, 13, 38, 4, 7, 46, 15, 25, 25, 32, 28, 48, 37, 14, 15, 6]
Graph Coloring Binary Tournament and Truncation  :
 minimum colors (best_fitness):  21 |chromosome:  [25, 1, 43, 50, 1, 27, 28, 41, 4, 46, 17, 28, 28, 46, 34, 49, 39, 20, 8, 4, 9, 23, 4, 28, 44, 17, 34, 6, 4,
41, 45, 41, 42, 29, 1, 42, 35, 14, 18, 48, 12, 13, 42, 15, 39, 45, 14, 8, 7, 5, 32, 5, 12, 24, 6, 39, 15, 10, 18, 47, 44, 10, 27, 28, 0, 2, 24, 22, 3, 16, 6,
41, 13, 43, 5, 19, 36, 11, 26, 21, 38, 19, 30, 38, 30, 30, 26, 31, 40, 36, 19, 37, 40, 33, 37, 26, 21, 0, 0, 0]
Graph Coloring Truncation and Truncation   :
 minimum colors (best_fitness):  26 |chromosome:  [10, 12, 41, 8, 50, 29, 17, 45, 37, 19, 36, 39, 3, 9, 18, 45, 50, 37, 27, 7, 3, 25, 46, 49, 36, 23, 47, 5, 4
6, 12, 38, 44, 28, 31, 11, 2, 27, 19, 22, 17, 2, 23, 39, 16, 11, 23, 10, 0, 27, 18, 39, 39, 49, 4, 28, 32, 8, 36, 27, 37, 41, 12, 5, 7, 1, 34, 20, 20, 0, 6, 2
9, 46, 14, 23, 11, 29, 18, 45, 46, 48, 22, 35, 15, 24, 13, 30, 30, 43, 26, 13, 35, 24, 21, 33, 40, 24, 33, 42, 33, 42]
Graph Coloring Random and Random  :
 minimum colors (best_fitness):  28 |chromosome:  [43, 12, 2, 10, 34, 10, 1, 13, 49, 19, 41, 34, 5, 35, 0, 18, 2, 36, 24, 45, 9, 33, 38, 42, 44, 48, 49, 27, 1
, 28, 22, 24, 49, 23, 14, 28, 28, 50, 36, 30, 8, 15, 6, 4, 37, 28, 5, 7, 3, 48, 13, 33, 36, 38, 20, 45, 7, 9, 6, 10, 3, 21, 3, 35, 20, 21, 27, 31, 13, 14, 7,
11, 44, 29, 43, 25, 25, 16, 24, 41, 17, 18, 33, 11, 4, 38, 18, 37, 40, 26, 32, 46, 47, 47, 46, 0, 0, 0, 0, 0]
Graph Coloring FPS and Truncation  :
 minimum colors (best_fitness):  20 |chromosome:  [35, 37, 46, 15, 4, 36, 19, 37, 24, 46, 48, 46, 18, 41, 37, 30, 41, 36, 23, 7, 31, 50, 9, 49, 18, 25, 43, 2,
 43, 23, 34, 16, 50, 18, 30, 27, 14, 32, 6, 6, 3, 45, 14, 20, 33, 32, 15, 29, 25, 50, 1, 33, 50, 43, 5, 29, 5, 18, 38, 14, 20, 25, 48, 27, 26, 3, 30, 37, 16,
35, 5, 43, 24, 26, 45, 32, 33, 27, 10, 9, 40, 28, 44, 40, 39, 40, 17, 8, 42, 21, 17, 8, 11, 42, 47, 11, 21, 22, 13, 13]
Graph Coloring RBS and Binary Tournament  :
 minimum colors (best_fitness):  23 |chromosome:  [1, 10, 26, 42, 11, 22, 12, 11, 21, 18, 12, 9, 23, 33, 48, 27, 13, 41, 32, 17, 40, 39, 49, 31, 49, 16, 4, 31
, 26, 41, 3, 32, 46, 45, 19, 34, 4, 18, 5, 25, 35, 44, 48, 26, 17, 6, 7, 7, 7, 11, 28, 35, 29, 13, 39, 39, 43, 3, 27, 37, 28, 8, 36, 46, 41, 15, 35, 27, 39, 4
9, 49, 2, 34, 5, 18, 40, 25, 12, 28, 40, 48, 9, 38, 20, 47, 38, 30, 47, 14, 38, 38, 20, 24, 50, 30, 24, 0, 0, 0, 0]
Graph Coloring Random and Truncation   :
 minimum colors (best_fitness):  22 |chromosome:  [0, 15, 21, 0, 32, 16, 18, 23, 43, 45, 16, 43, 42, 31, 14, 13, 14, 50, 19, 41, 9, 10, 28, 13, 7, 30, 20, 10,
 27, 38, 34, 22, 6, 12, 13, 11, 37, 3, 26, 22, 34, 9, 28, 24, 20, 8, 1, 3, 6, 38, 7, 28, 1, 25, 30, 39, 13, 49, 41, 34, 27, 19, 17, 31, 49, 39, 33, 41, 48, 25
, 50, 1, 17, 40, 41, 46, 38, 46, 35, 49, 17, 37, 2, 8, 29, 5, 4, 47, 44, 47, 36, 47, 4, 5, 44, 36, 44, 36, 0, 0]
```

3.5 Analysis:

The results for each combination of parent and survivor selection technique was visualized in section XYZ.The plots indicate that regardless of the technique, in each iteration the evolutionary algorithm resulted in lower fitness values for around 10 generations, but as the number of generations increased, a sudden increase in fitness values can be observed. This implies that the best-so-far fitness value obtained in the above result was probably from the initial generations. Hence, keeping the number of generations less while tuning other parameters such as population size, mutation rate, and number of offsprings may result in more optimal results.