CS 451 - Computational Intelligence

Assignment 1

Evolutionary Algorithm

Shamsa Hafeez

Major: Computer Science

Class of 2023

07 February 2023

Contents

1 Travelling Salesman Problem (TSP)

1.1 Chromosome Representation

Each chromosome is in the form of the list. The length of the list is equal to the number of places to visit. Thus, the length is 194. The list consists of tuples whose first and second index represents the x and y coordinates of the city to visit respectively. The index of the city decides the order in which it is visited. For example:

[(x_1, y_1), (x_2, y_2), (x_3, y_3), ..., (x_{194}, y_{194})]

So, this means the person travels from (x_1, y_1) till (x_2, y_2), then from (x_2, y_2) till (x_3, y_3) all the way till from (x_{193}, y_{193}) till (x_{194}, y_{194}). He then travels from last index till first index i.e., from (x_{194}, y_{194}) to (x_1, y_1) directly. Through this way the chromosome represents the path to follow by the travelling salesperson.

1.2 Fitness Function

```
def fitness(self, individual):
    fitness = 0
    for i in range(len(individual)-1):
        fitness += self.distance(individual[i], individual[i+1])
    fitness += self.distance(individual[0], individual[-1])
    return fitness
def distance(self, a, b):
    return math.sqrt((b[0] - a[0])** 2 + (b[1] - a[1]) ** 2)
```

The fitness function calculates and returns the total distance to be travelled when a salesperson passes through a given path. Each individual is a chromosome that shows a path. For example:

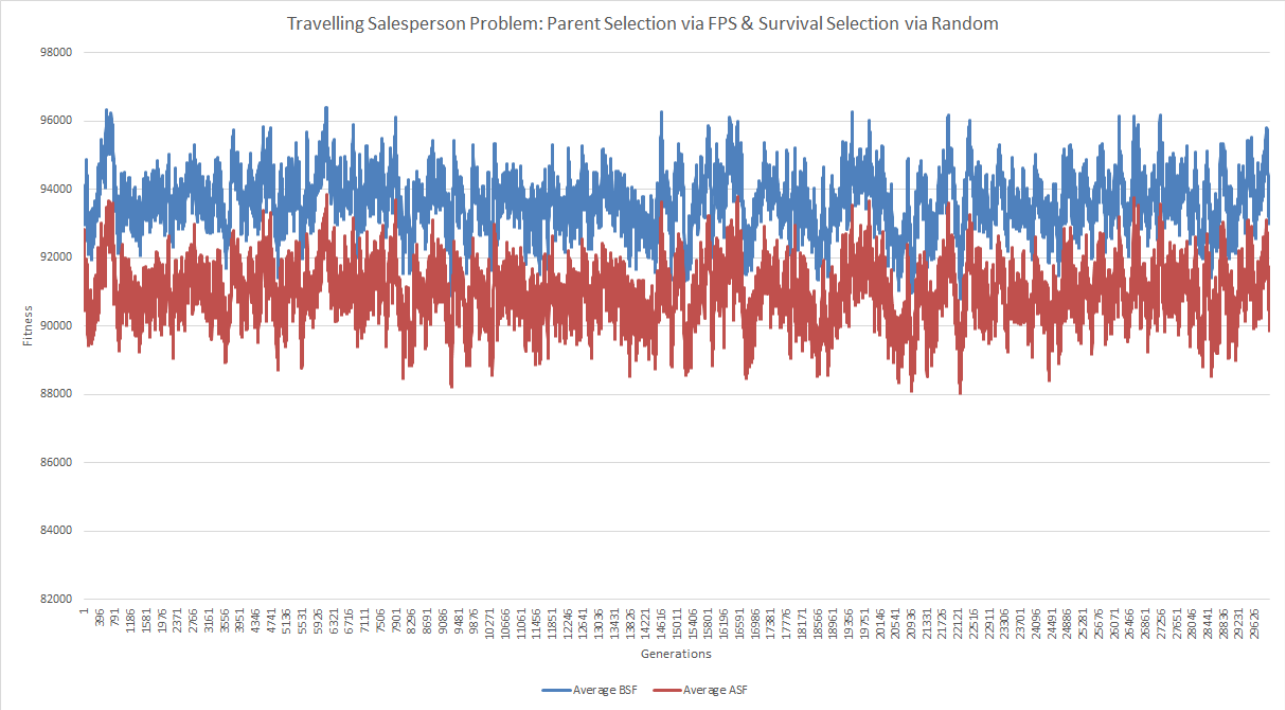
[(x_1, y_1), (x_2, y_2), (x_3, y_3), ..., (x_{194}, y_{194})]

So, this means that the fitness function first adds the distance from (x_1, y_1) till (x_2, y_2), to the distance from (x_2, y_2) till (x_3, y_3) all the way till from (x_{193}, y_{193}) till (x_{194}, y_{194}). It then addes the distance from (x_{194}, y_{194}) to (x_1, y_1) directly. Note that this is a minimization problem and we will minimize the fitness i.e., distance to travel.

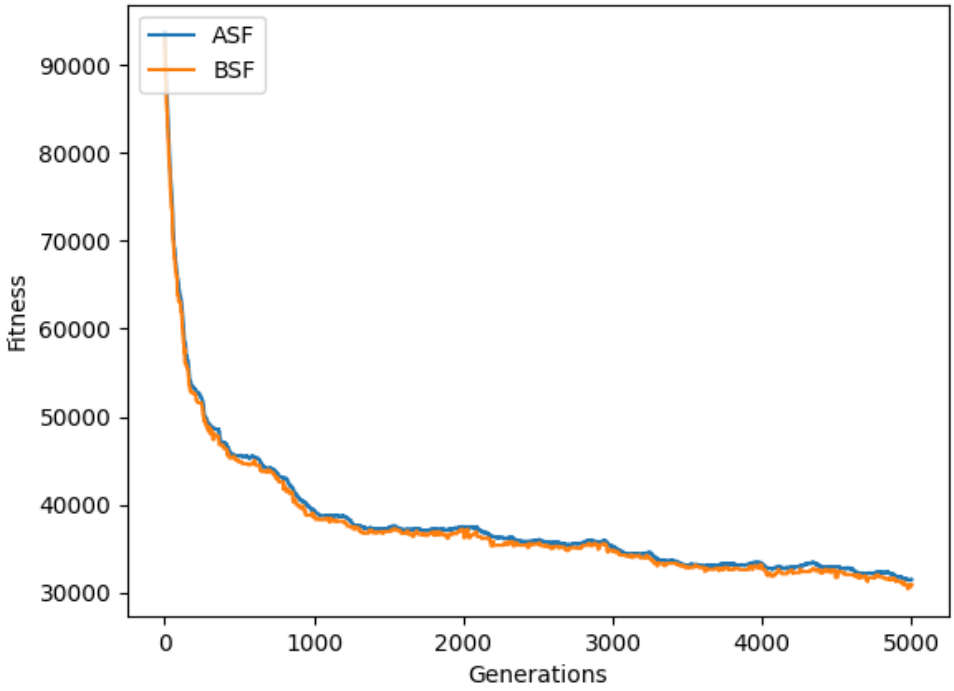
1.3 Plots of Different Selection Schemes

- *The heading of each subsection represents the parent selection scheme and survival selection scheme Respectively*

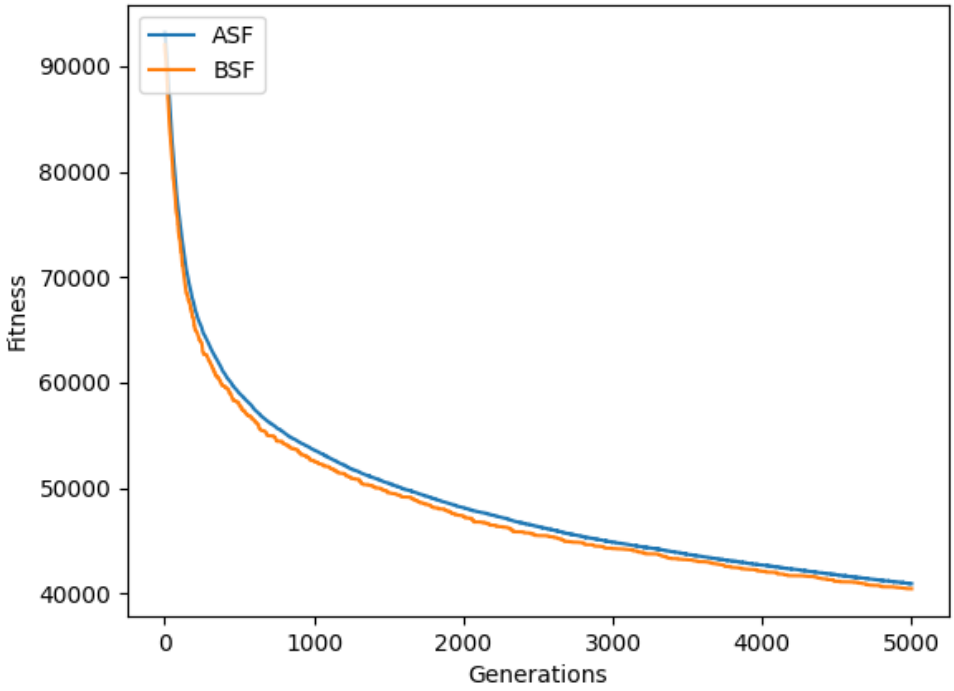
1.3.1 FPS and Random



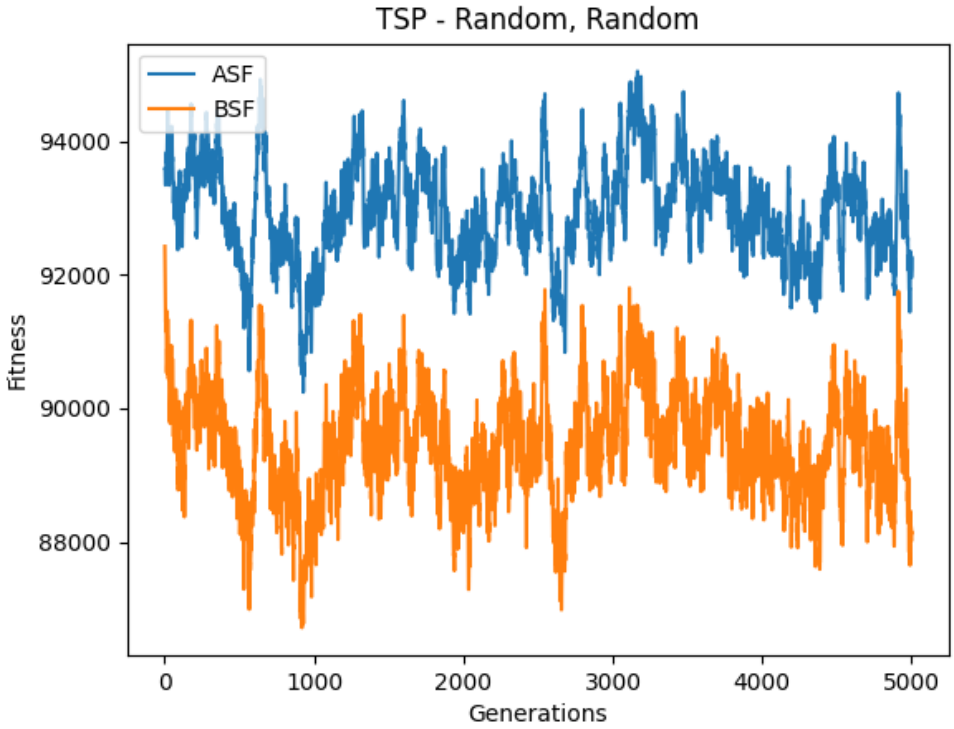
1.3.2 Binary Tournament and Truncation



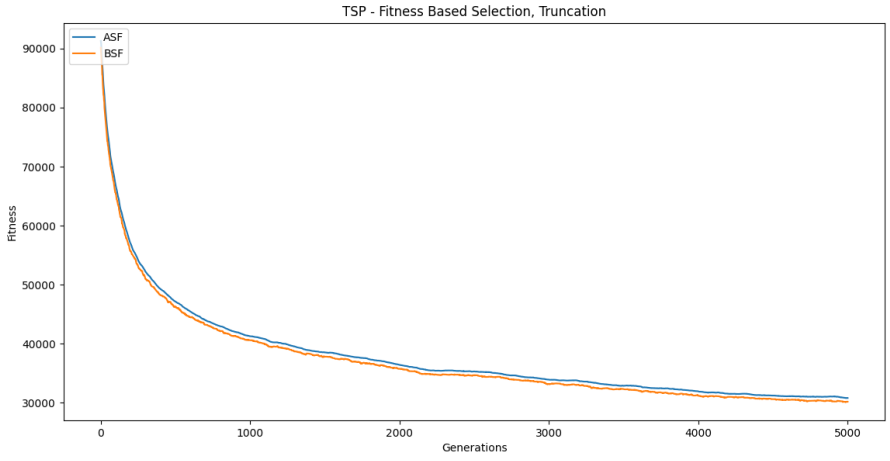
1.3.3 Truncation and Truncation



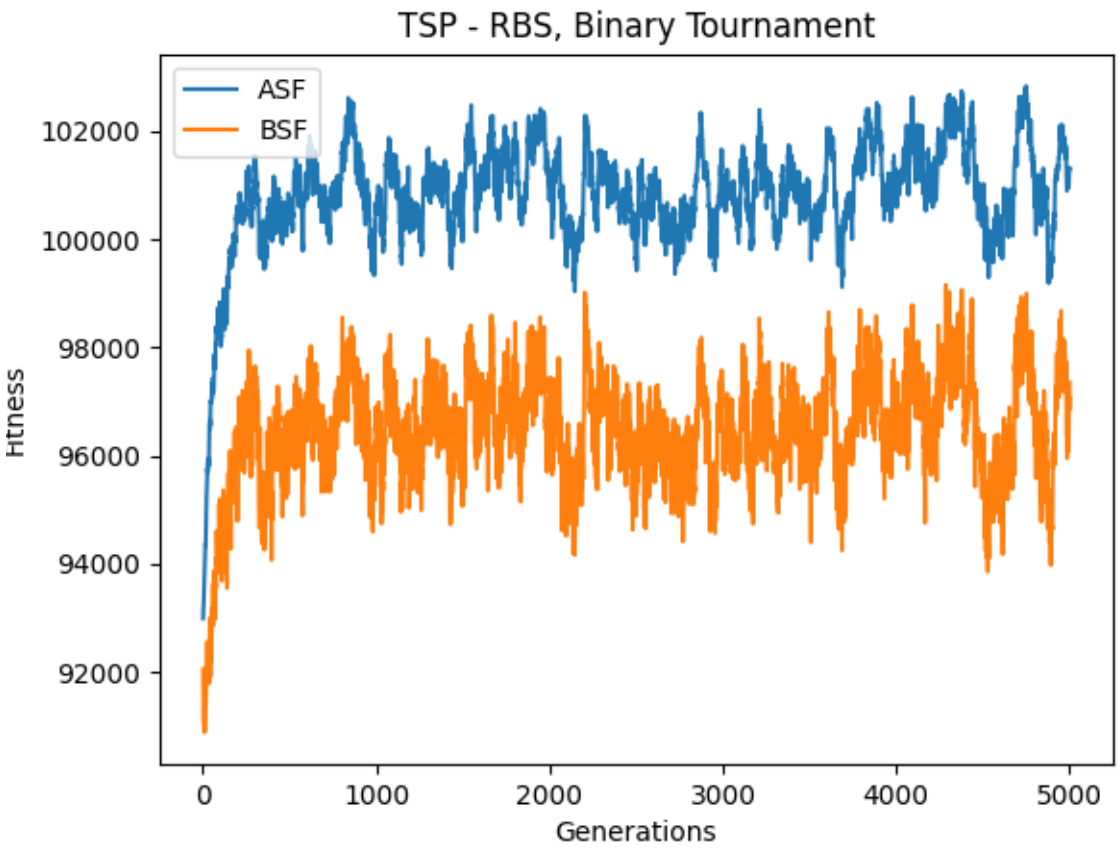
1.3.4 Random and Random



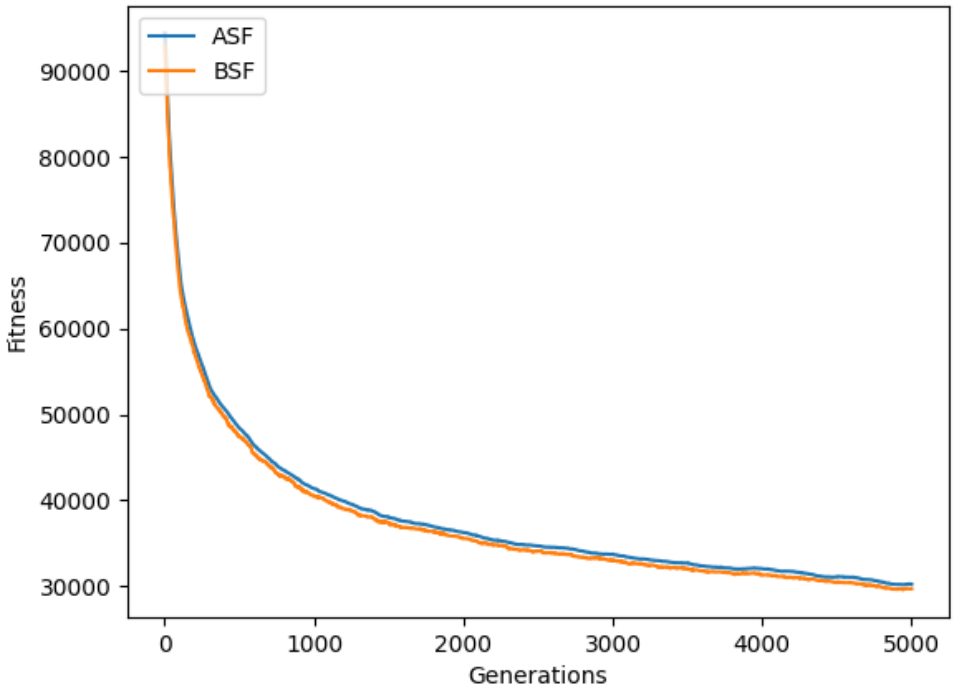
1.3.5 FPS and Truncation



1.3.6 RBS and Binary Tournament



1.3.7 Random and Truncation



1.4 Analysis on schemes

Notice that in the like the knapsack and Graph coloring problem, the TSP problem shows that when the survival selection scheme is more explorative, then no matter whether or not the parent is selected as per the fitness or not, the survivors can be fit or unfit and this randomness is resulting in the graph going zip zag and not converging. Also note that when we are using RBS and Binary Tournament, although RBS is based on relative fitness (and is more exploitative), the explorative nature of Binary Tournament is not allowing the population to converge. Thus, we see there has be the correct balance of exploitation and exploration. Note that graph of 1.3.3 shows that when both the parent and the survivor selection scheme is exploitative then the graph move exactly across its path towards the point where the chromosome traits of population converges. However, we want a pair of selection schemes that performs exploration as well as exploitation so that the global optima is not missed. Therefore the right choice is FPS and Truncation (both exploration and exploitation) and the worst is Random and Random

2 Knapsack Problem

2.1 Chromosome Representation

Each chromosome is a list that consists of tuples.The list represents the bag. Each tuple represents an item. The first element of the tuple represents the profit of the item and the second element of the tuple represents the weight of the item. Note that unlike the chromosome of TSP, the order of the tuples in the list does not matter. For example:

$[(p_1, w_1), (p_2, w_2), (p_3, w_3)]$

This list shows that there are 3 items in the bag: One item has a profit of p_1 and a weight of w_1. Another item has a profit of p_2 and a weight of w_2. Another item has a profit of p_3 and a weight of w_3. Through this way the chromosome exactly represents the configuration of an individual in the population.

2.2 Fitness Functions

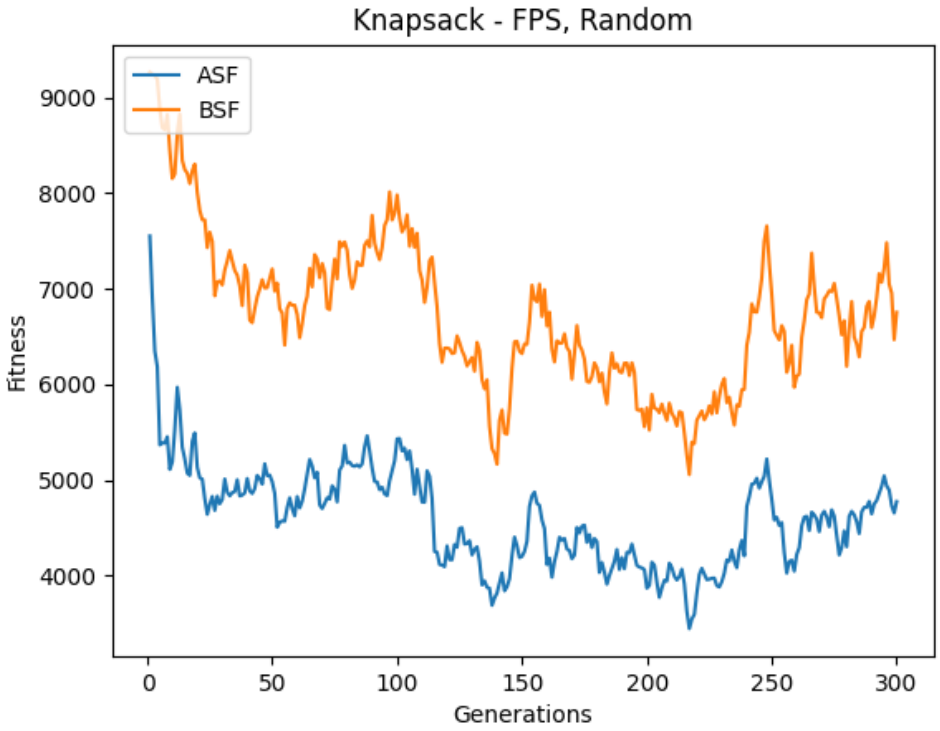
```
def fitness(self, individual):
    fitness = 0
    weight = 0
    for i in individual:
        fitness += i[0] # profit
        weight += i[1]

    if weight > self.wmax:
        fitness = 0
    return fitness # profit to maximize
```

The fitness function returns the total profit of the items in the bag. Note that if the weight of the items in the bag exceeds that maximum weight allowed then the fitness function penalizes it by considering its fitness to be zero. Note that it is a maximization problem (We want to maximize the total profit). In the example mentioned in section 2.1, its fitness would be $p_1 + p_2 + p_3$ if $w_1 + w_2 + w_2$ is less than maximum allowed weight, otherwise the fitness would be zero.

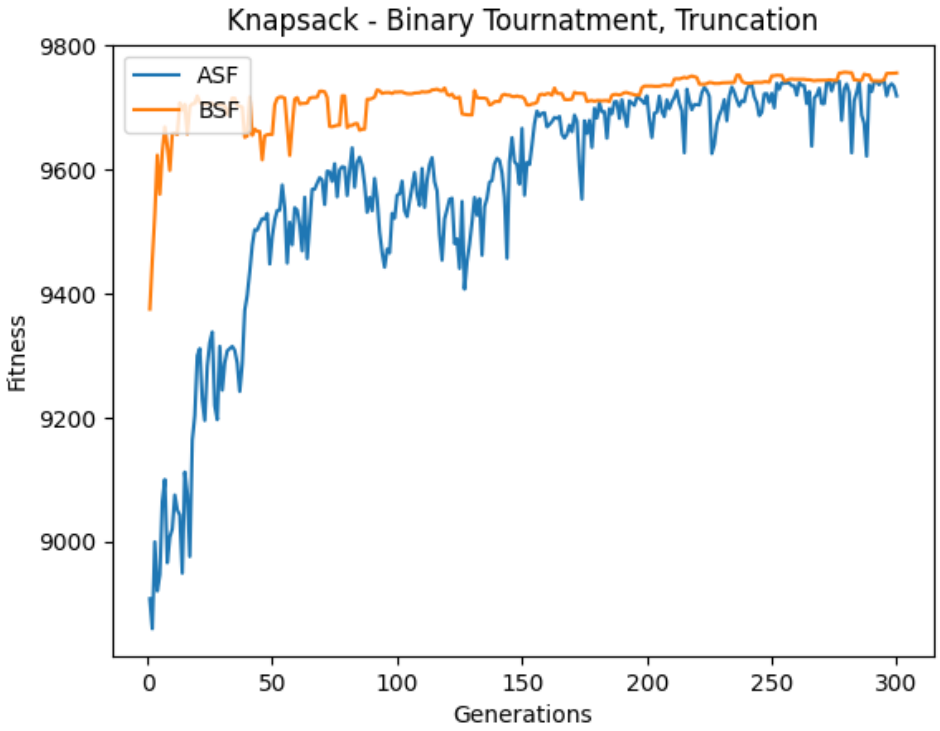
2.3 Plots of Different Selection Schemes

- *The heading of each subsection represents the parent selection scheme and survival selection scheme Respectively*

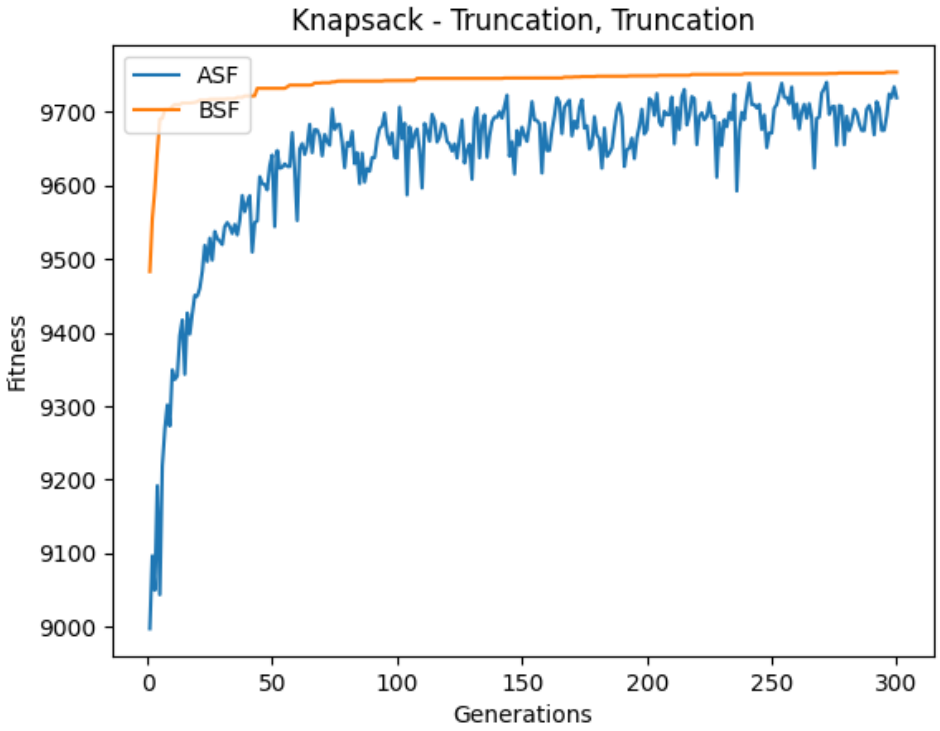
2.3.1 FPS and Random



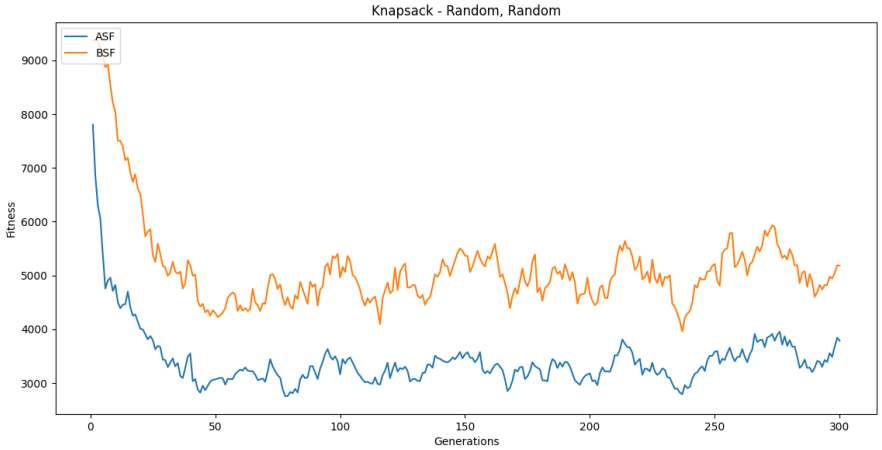
2.3.2 Binary Tournament and Truncation



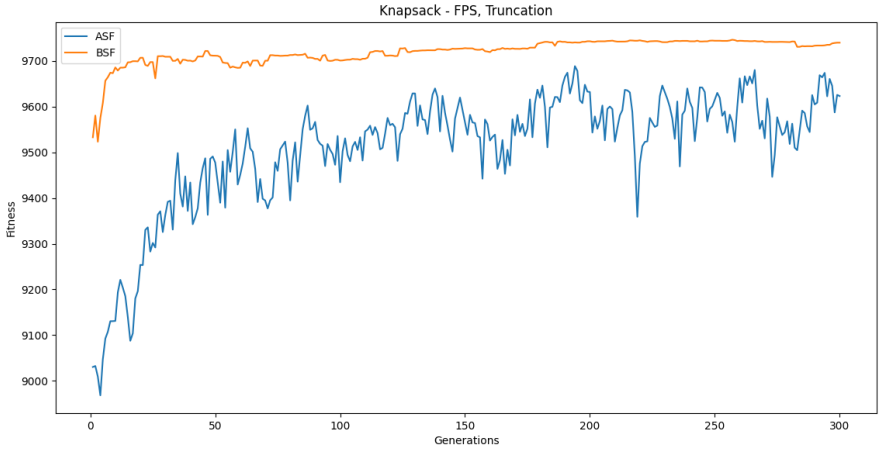
2.3.3 Truncation and Truncation



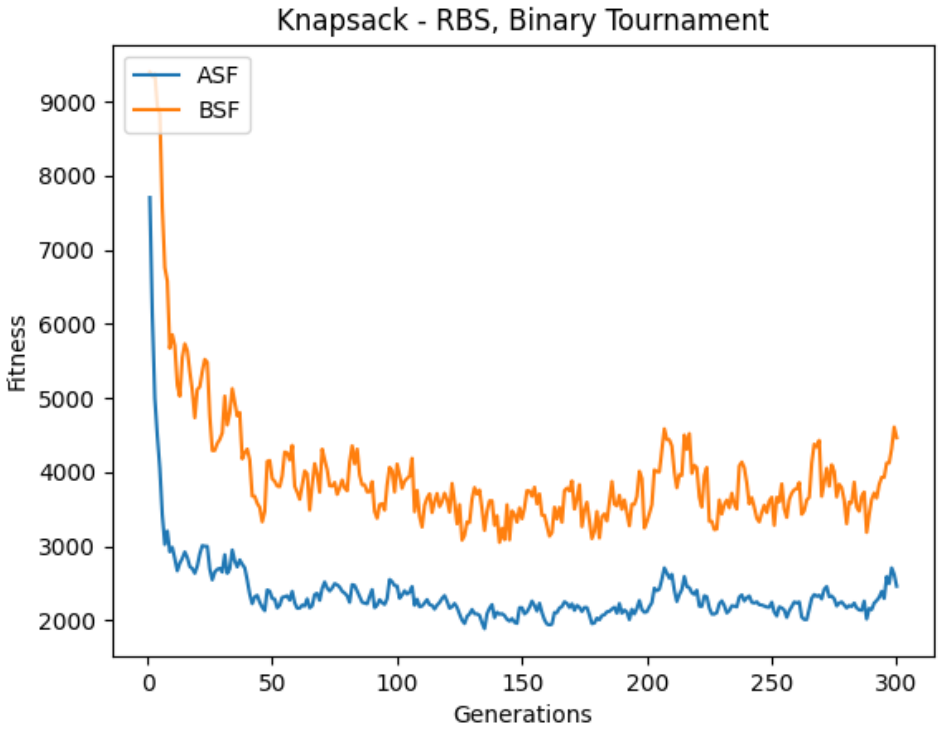
2.3.4 Random and Random



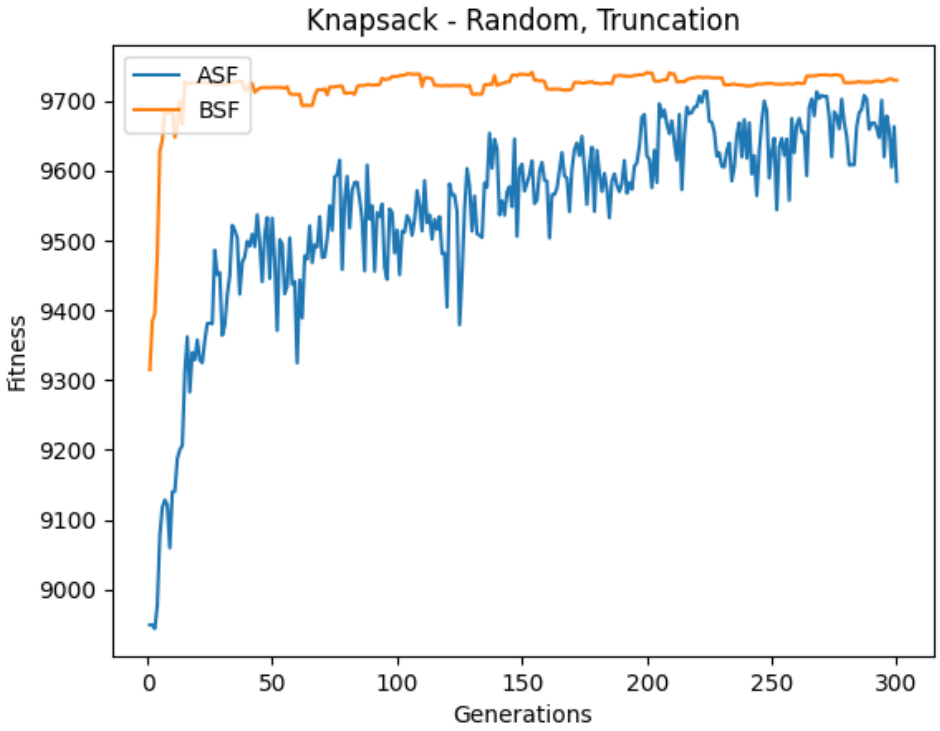
2.3.5 FPS and Truncation



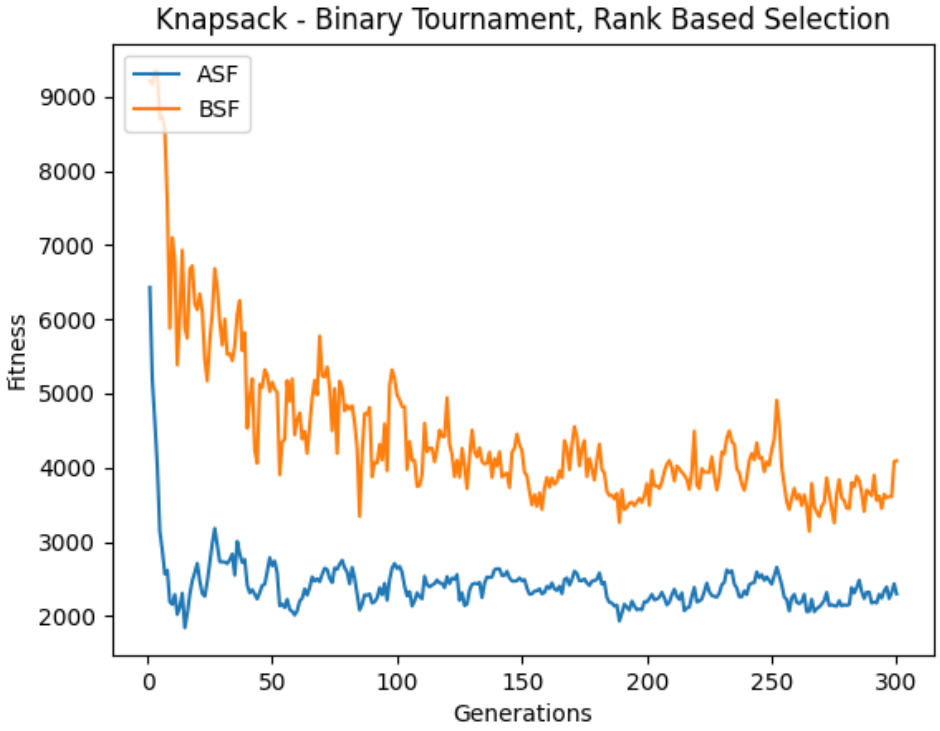
2.3.6 RBS and Binary Tournament

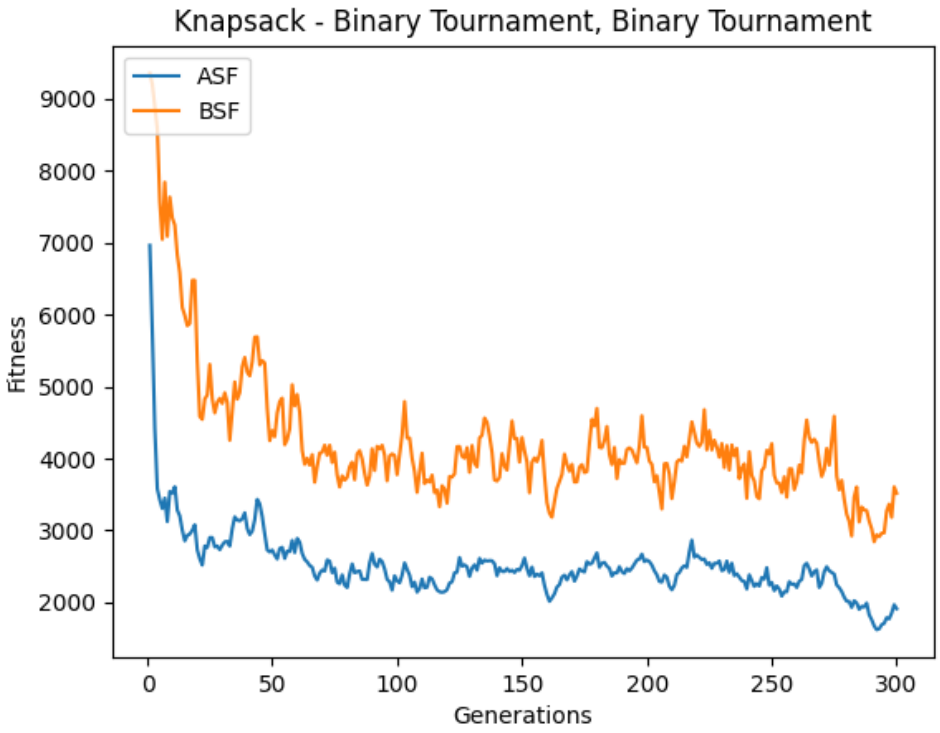


2.3.7 Random and Truncation



2.3.8 Others


Knapsack - Binary Tournament, Rank Based Selection
ASF
BSF
Fitness
Generations
9000
8000
7000
6000
5000
4000
3000
2000
0
50
100
150
200
250
300


Knapsack - Binary Tournament, Binary Tournament
ASF
BSF
Fitness
Generations
9000
8000
7000
6000
5000
4000
3000
2000
0
50
100
150
200
250
300

2.4 Analysis on schemes

Firstly, lets recall that in this problem we aim to "Maximize" our profit. Notice that whenever the survival selection scheme is Random, we do not achieve this goal as depicted in the Graph of 2.3.1 and 2.3.4. This is because we have decreased exploitation by not considering that the fittest should survive (converging us to the appropriate set of solutions). Survivors in each generation are random in nature and can be either fit or unfit. Thus irrespective of the parent selection scheme (FPS, in 2.3.1, imposes greater selective pressure and Random has the least selective pressure) exploitation seems very low.
Notice that in graph of 2.3.6 the fitness is decreasing rather than increasing. The reason can be that even though via RBS, selection of parents is based on relative probabilities, we are using Binary Tournament (BT) to select the chromosomes that would be discarded in the next generation. This technique is usually helpful when either the entire data is unavailable or difficult to fetch. But in our case we had the entire data. Also, only considering 2 sample points for the tournament randomly out of the entire population (10). So even if a bad individual is discarded, we do not know whether it was really the worst of all.
Notice that 2.3.2 , 2.3.3, 2.3.5 and 2.3.7 have Truncation as a survival selection scheme that is allowing the BSF to eventually converge (i.e., sustain in the population despite generations of passing). This is because Truncation encourages exploitation. When the worst fitness individuals are eliminated, it allows the best to survive and reproduce in the next generation (No matter what the parent selection scheme is, since the worst individual dies, whatever member stays in the population has to be the best. Note that we have 5 off springs and 10 population size is 10). 2.3.7 explores more (more ups and downs) since it is Random, Binary Tournament also explores more (ups and downs) but less than Random, Notice that 2.3.3 is steeper than 2.3.5. FPS is exploring first and so taking time to exploit, while, Truncation exploits rapidly. Thus in my case FPS Truncation worked the best as there should be a right balance of exploration and exploitation. Moreover, Random - Random, Binary Tournament - Binary Tournament and Binary Tournament - RBS were worst as they impose least selective pressure. Notice that Orange lines are Above the blue ones since we are maximizing fitness so the Best Fitness of the generation would be greater than the average fitness of the generation.

3 Graph coloring

3.1 Chromosome Representation

Notice that the graph coloring problem needs to store the graph (nodes and edges) as well as the color of each node. Since the graph (nodes and edges) remains the same throughout the process of Evolutionary Algorithm, what we really need to identify an individual chromosome is the color assigned to its nodes. The graph is stored in *graph_map* dictionary where the key is the node and the value is the list of its corresponding neighbourhood nodes. Now, our actual chromosome is a dictionary with key as the node number and the value as the color assigned to it. Note that colors are represented in the form of whole numbers (We can assume it to be arbitrarily any hexadecimal value of color). For example: { 1: 2, 2: 2, 3: 1 }

The above dictionary represents that there are 3 nodes in the graph namely 1, 2 and 3. Node number 1 has a color 2. Node number 2 also has a color 2. Node number 3 has a color 1. Note that the number of colors shall be less than or equal to the number of nodes.

3.2 Fitness Function

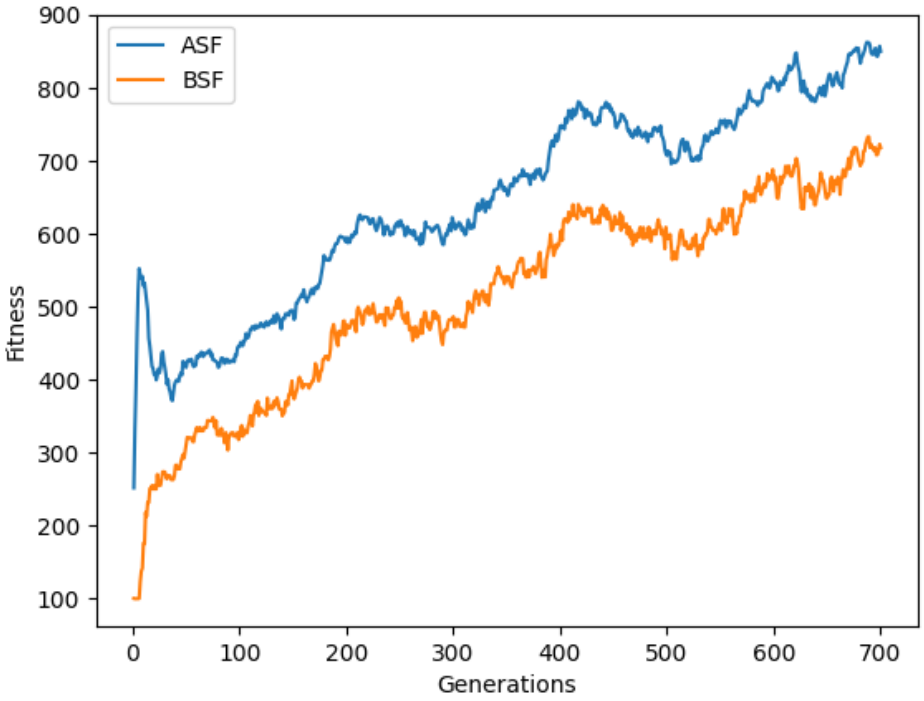
```
def fitness(self, individual):
    total_distinctive_colors = len(set(individual.values()))
    clashes = 0
    for i in self.graph_map:
        for j in self.graph_map[i]:
            if individual[j] == individual[i]:
                clashes += 1
    return (clashes * 50) + total_distinctive_colors # Assuming 50 as our penalty
```

This is a minimization problem and we need to minimize the number of colors used to color the graph. Thus our fitness value would be total distinctive colors used to color the nodes of our map. Since, we have a constraint that no adjacent nodes should be of same color, therefore we cannot tolerate such chromosomes in our population whose adjacent nodes are of same color. Therefore we assign them a high fitness value by adding 50 times the number of colors clashing. Thus, if the least fit chromosomes are discarded first then individuals with clashing colors shall be discarded first.

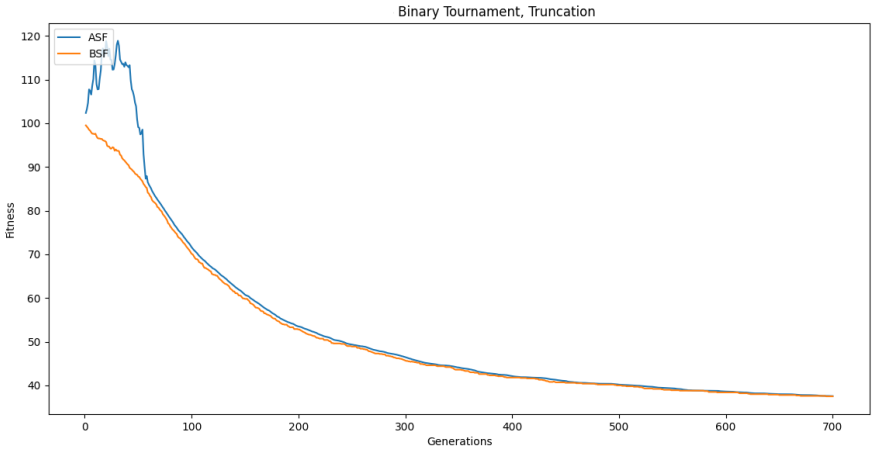
3.3 Plots of Different Selection Schemes

- *The heading of each subsection represents the parent selection scheme and survival selection scheme Respectively*

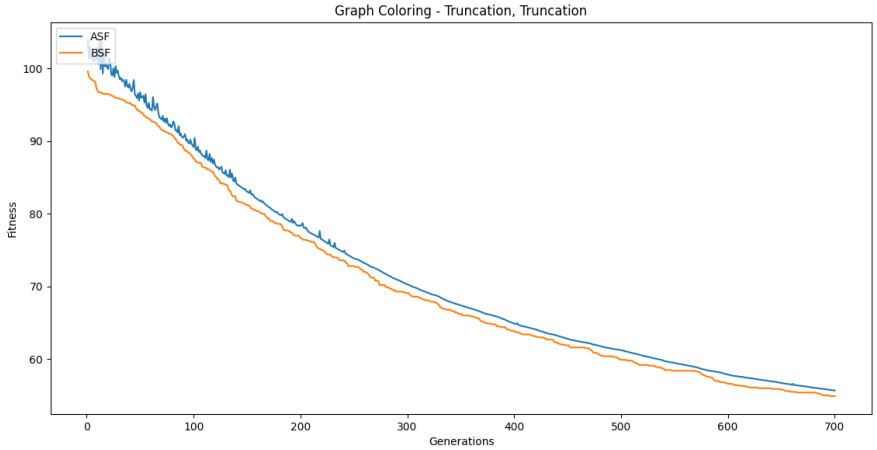
3.3.1 FPS and Random



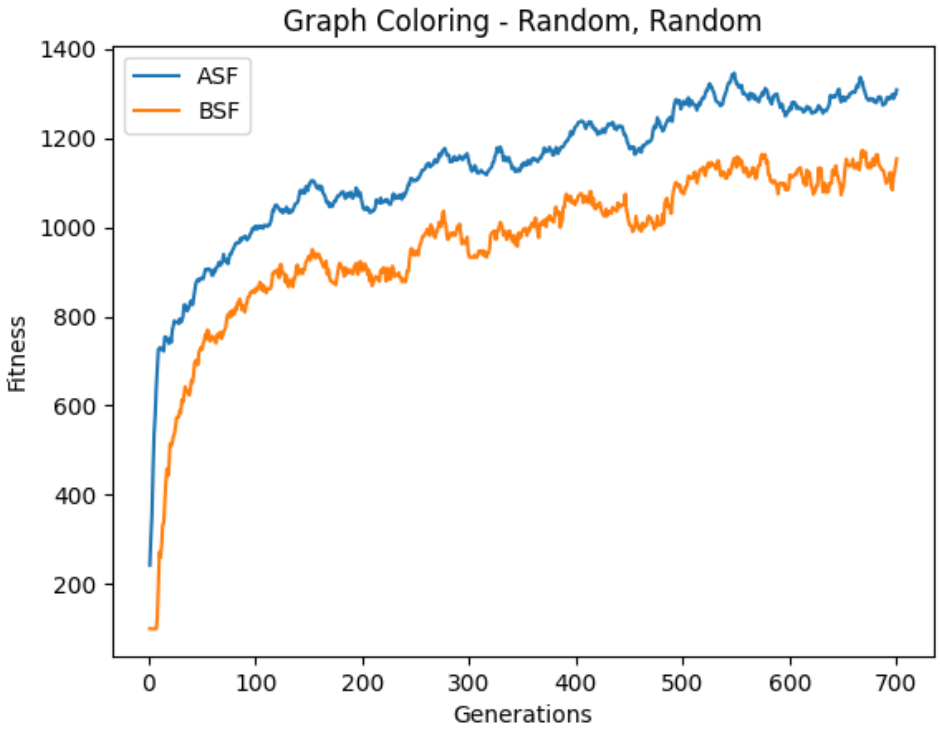
3.3.2 Binary Tournament and Truncation



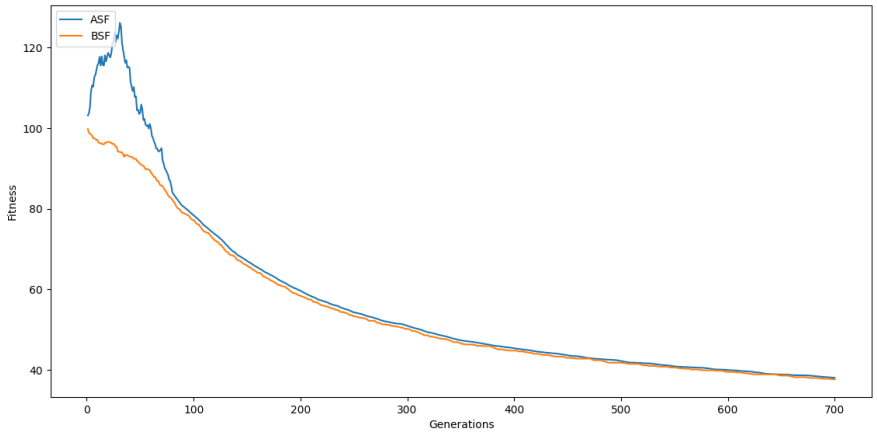
3.3.3 Truncation and Truncation



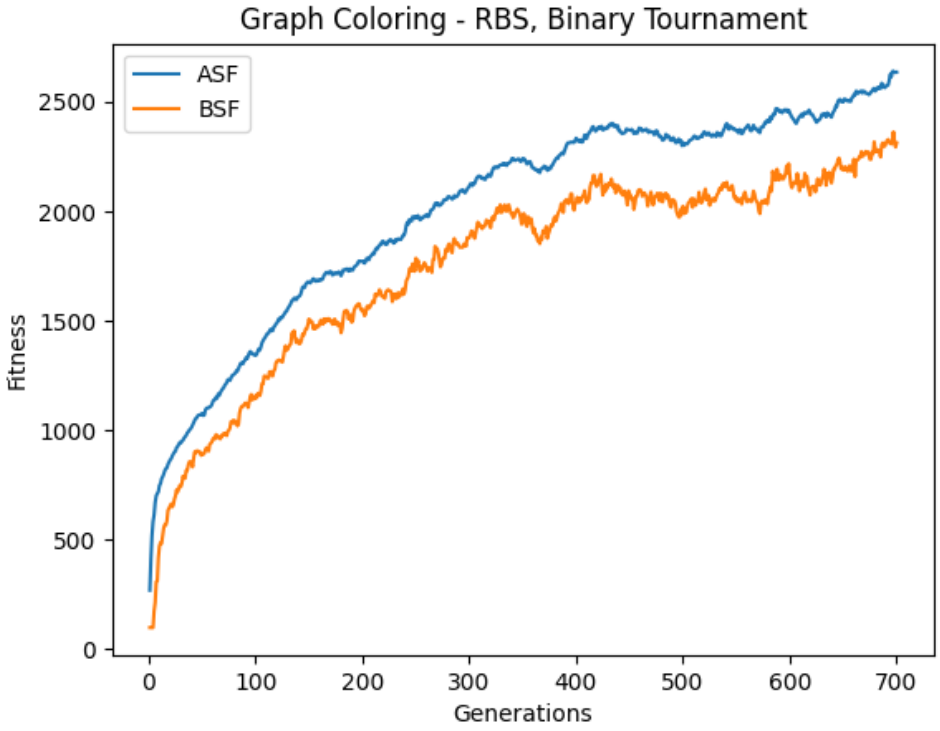
3.3.4 Random and Random



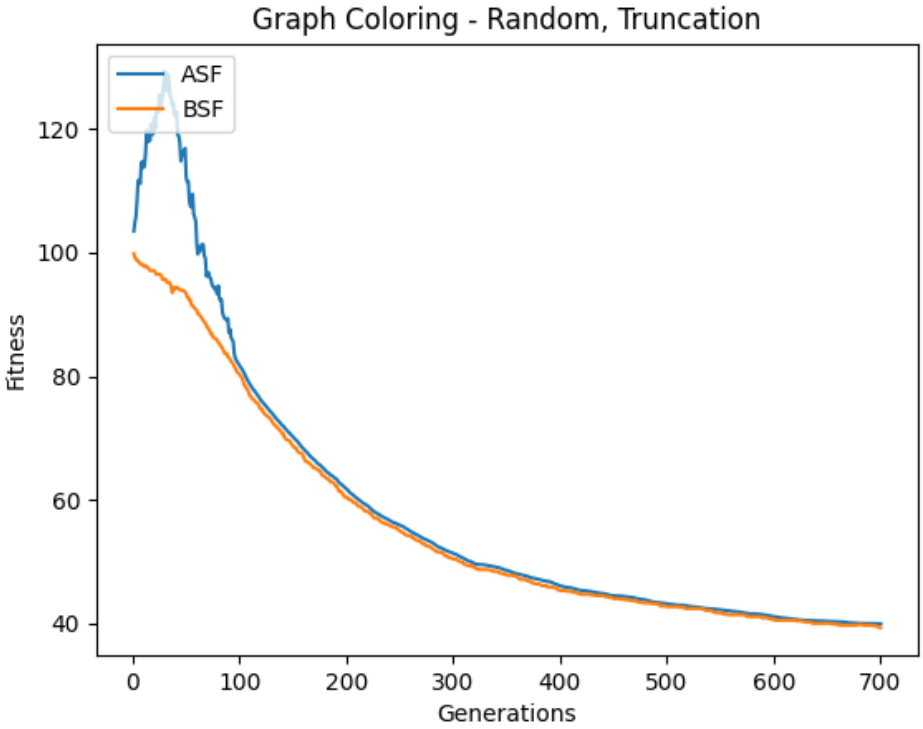
3.3.5 FPS and Truncation



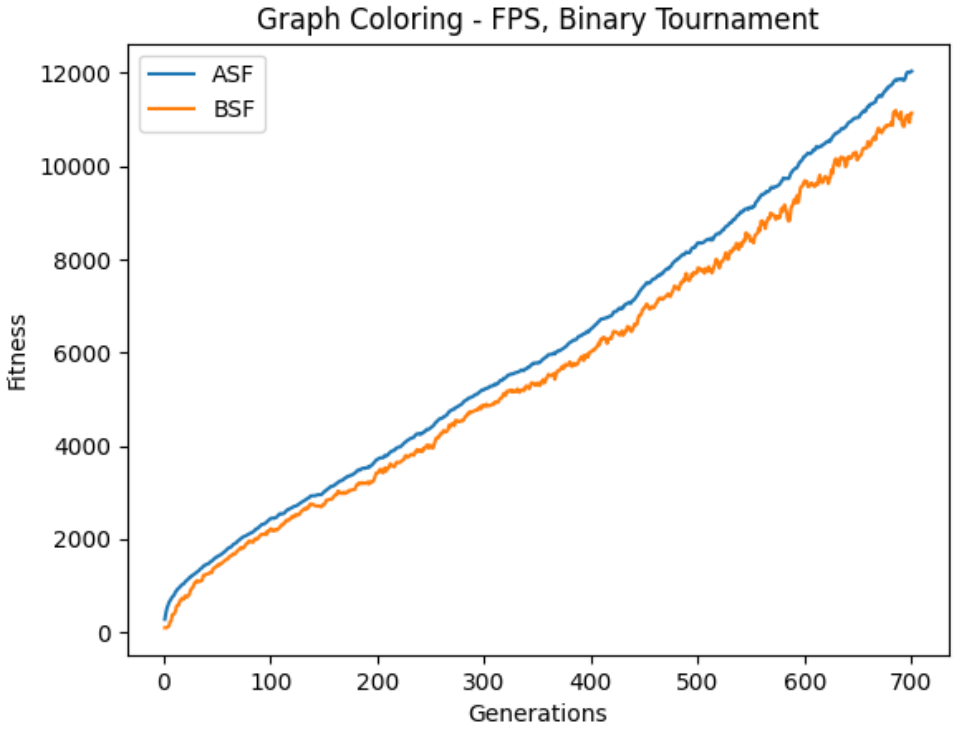
3.3.6 RBS and Binary Tournament

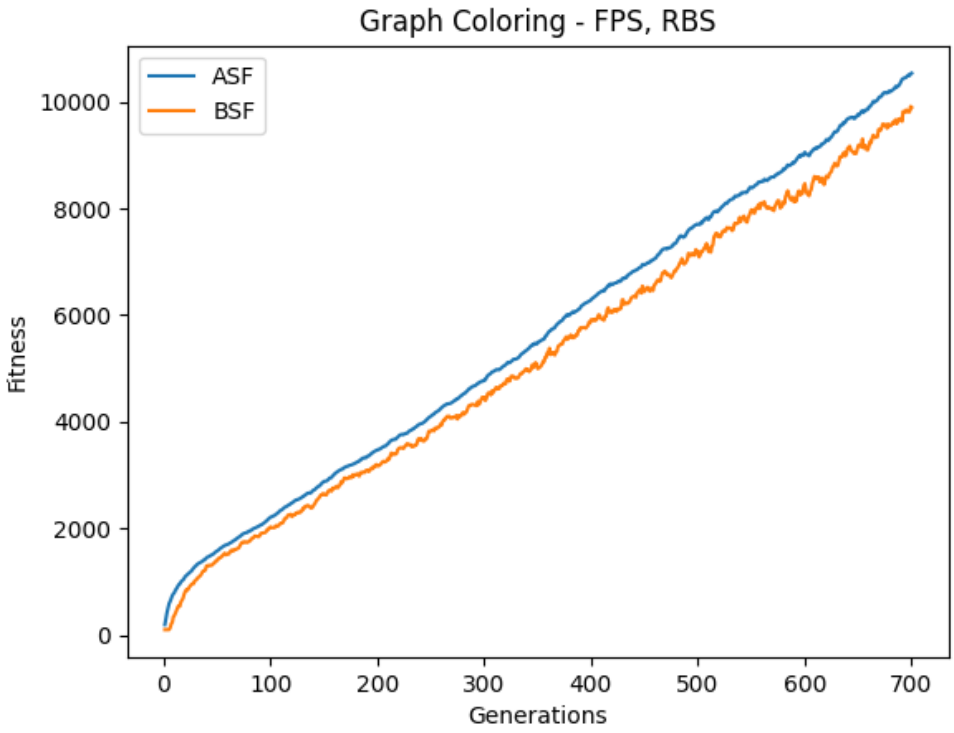


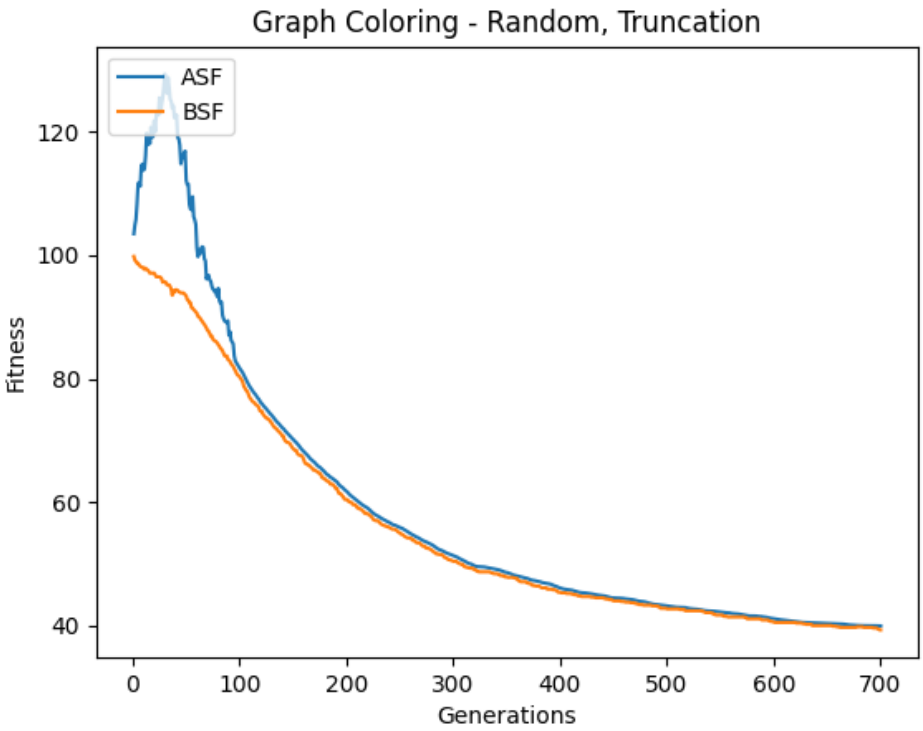
3.3.7 Random and Truncation



3.3.8 Others


Graph Coloring - FPS, Binary Tournament
ASF
BSF
Fitness
Generations
0
2000
4000
6000
8000
10000
12000
0
100
200
300
400
500
600
700


Graph Coloring - FPS, RBS
ASF
BSF
Fitness
Generations
0
2000
4000
6000
8000
10000
0
100
200
300
400
500
600
700


Graph Coloring - Random, Truncation
ASF
BSF
120
100
80
60
40
Fitness
0
100
200
300
400
500
600
700
Generations

3.4 Analysis on schemes

Since this is a minimization problem, we want to minimize the the number of colors. Note that in 3.3.1 Although we are selecting the fittest individuals (based on the probability distribution), notice that since the survival selection is random, the fit individuals might possible eliminate from the population, so the coming generations can have a high fitness value (as selective pressure is low). Thus the graph is showing that as the number of generations proceed, the fitness value has increased(unfit individuals take over the population) with some ups and downs indicating that at some point the randomization might have selected bad individuals to discard. This is the same reason why in 3.3.4 the fitness value is increasing since Survival selection scheme is random.

3.3.2, 3.3.5 and 3.3.7 are showing similar trend. Both ASF and BSF eventually decrease indicating that due to Truncation there has been exploitation and only the fittest survive in the next generation. Also notice that there us a slight peak in ASF in the starting generations. This is due to the degree of exploration that the parent selection scheme allows (All these 3 parent selection schemes use some degree of randomization). However for 3.3.3 the same declining trend is observed but without the initial peak. This is because when least fit individuals produce 5 offspring, the population becomes $10 + 5 = 15$. For survival selection, the two parents would be discarded (if the offsprings are not more unfit) along with their children if they are most unfit. Thus, eventually the fittest would survive allowing less room from exploration as there is no randomization!

Thus we conclude that the best scheme is FPS and Truncation and the worst scheme is Random and Random.
Note that these graphs proof the following order of selective pressure:
Truncation > FPS > RBS > Binary Tournament > Random

Note that the BSF is lower than ASF since it is a minimization problem.