AI Assignment

Part A

I started by implementing the simple genetic algorithm mentioned in the textbook. I plotted the results for a population of 50 after 50 generations.

Fitness Function

I used a fitness function which checks for coverage and gives a large bonus for full coverage

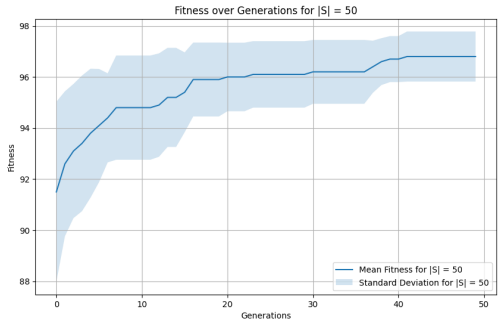
$$f(\text{individual}) = \begin{cases} 100 \times \left(\frac{\text{coverage}}{\text{universe\_size}}\right) & \text{if coverage} < \text{universe\_size} \\ 100 + 10000 \times \left(1 - \frac{\text{num\_selected}}{\text{len(subsets)}}\right) & \text{if coverage} = \text{universe\_size} \end{cases}$$

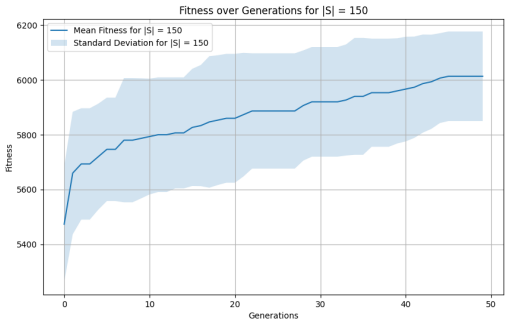
Where:

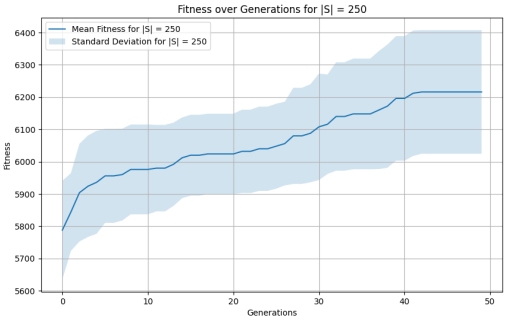
- `coverage` is the number of elements covered by the individual's selected subsets.
- `universe_size` is the total number of elements in the universe.
- `num_selected` is the number of subsets selected by the individual.
- `len(subsets)` is the total number of subsets available.

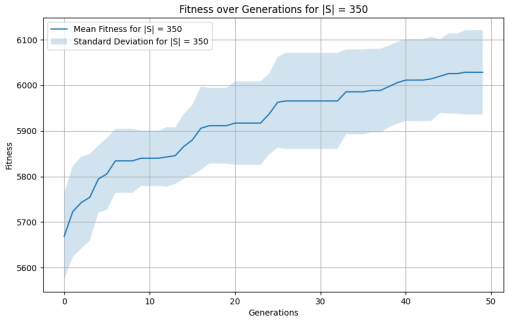
Plots

The plots can be found for the sets of sizes 50, 150, 250 and 350 below. We can observe a large variation since the fitness function changes drastically as an individual moves from partial coverage to full coverage. This large bonus for full coverage makes it likely that we arrive at a solution which covers the entire universe.









Part B

Uniform Crossover

I replaced the crossover method recommended in the textbook with a uniform crossover approach. Unlike the traditional method, which combines genes from parents at a specific cut point, uniform crossover selects each gene independently from either parent with equal probability. This approach enhances gene mixing and promotes greater diversity within the population.

In my test runs, adopting uniform crossover led to an average increase of 200 in the best fitness value compared to the traditional crossover method.

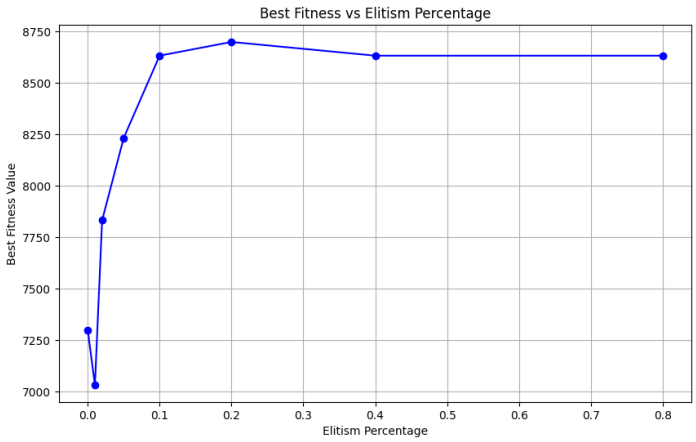
Elitism

As stated in Part A, I started with the naive approach. The first optimization which I used was to implement elitism.

This technique was chosen because:

- It allows us to preserve high-quality solutions
- Improves convergence speed

Below are the graphs for the best fitness achieved with varying percentages for elitism (starting with 0, which indicates no elitism).



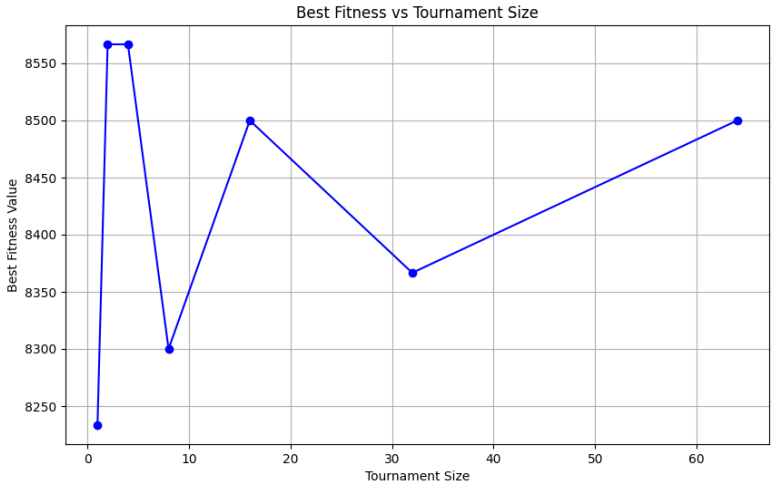
We can observe that elitism leads to an overall improvement in the best fitness we obtain. The results worsen after we reach the elitism value of 0.2; this can be potentially because we do not get enough diversity later on to improve.

Tournament Selection

I started using tournament selection because:

- Increased selection pressure: This increases convergence speed
- Preserving Diversity: Small tournament sizes help us explore the search space since we are not less likely to consistently pick the best individuals

This has a similar effect on fitness as elitism. If we select a large tournament size, we might decrease diversity, and hence, we decrease fitness in the long run. This fact is reflected in our plot. We see a peak at 2 and 4 with a decline later on.

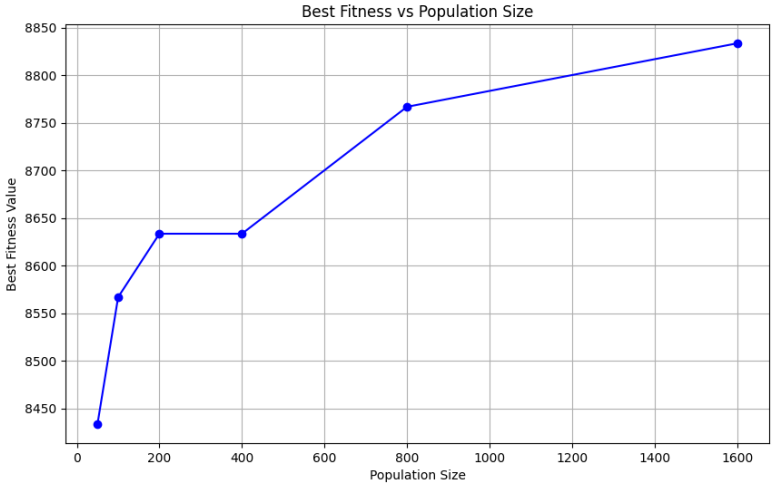


Population Size

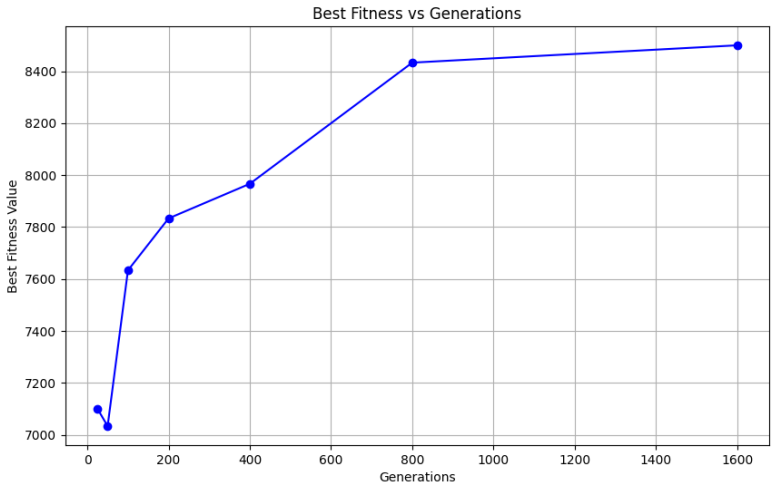
I later increased the population of the genetic algorithm for the following reasons:

- Exploring the search space better
  - This was especially important since I observed my early stopping was getting triggered quite frequently, stopping the exploration.
  - The increased diversity after the increasing population helped combat the stagnation caused by early stopping

We see an increase in our final fitness function with an increase in population size. There is an obvious tradeoff between population size, computational resources required, and time taken for the program to run.



Generations



I next increased the number of generations I was running the algorithm for, this allows us to:

- Explore the search space better and obtain better solutions (especially if the mutation rate is high)

As expected, increasing the number of generations allows us to come to better quality solutions. The tradeoff we have to make when increasing the number of generations is the amount of time taken for the algorithm to converge.

Final Result

Finally, after incorporating all these results, I was able to obtain the set cover by using **19 sets**

```
sidk@SiddhantAsus:/mnt/e/BITS/4-1/AI/Assignment_1$ python3 2021A7PS2606G_SIDDHANT.py
Roll no: 2021A7PS2606G
Number of subsets in scp_test.json file: 150
Solution:
0:0, 1:0, 2:0, 3:0, 4:0, 5:0, 6:0, 7:1, 8:0, 9:0, 10:1, 11:0, 12:0, 13:0, 14:0, 15:0, 16:0, 17:0, 18:0, 19:1, 20:0, 21:0, 22:0, 23:0, 24:
0, 25:1, 26:0, 27:0, 28:1, 29:0, 30:0, 31:0, 32:0, 33:0, 34:0, 35:0, 36:0, 37:0, 38:0, 39:1, 40:0, 41:0, 42:0, 43:0, 44:0, 45:1, 46:0, 47
:0, 48:0, 49:0, 50:0, 51:0, 52:0, 53:0, 54:0, 55:0, 56:0, 57:0, 58:0, 59:0, 60:0, 61:0, 62:1, 63:0, 64:0, 65:0, 66:0, 67:0, 68:0, 69:0, 7
0:0, 71:0, 72:1, 73:0, 74:0, 75:0, 76:0, 77:0, 78:0, 79:0, 80:0, 81:0, 82:0, 83:0, 84:0, 85:1, 86:0, 87:0, 88:1, 89:0, 90:0, 91:0, 92:1,
93:0, 94:0, 95:0, 96:0, 97:1, 98:0, 99:0, 100:0, 101:0, 102:0, 103:0, 104:0, 105:1, 106:0, 107:0, 108:0, 109:0, 110:0, 111:0, 112:0, 113:
0, 114:0, 115:0, 116:0, 117:0, 118:0, 119:0, 120:0, 121:0, 122:0, 123:0, 124:0, 125:0, 126:0, 127:0, 128:1, 129:0, 130:1, 131:0, 132:0, 1
33:0, 134:0, 135:0, 136:0, 137:0, 138:0, 139:0, 140:0, 141:1, 142:1, 143:0, 144:1, 145:0, 146:0, 147:0, 148:0, 149:0
Fitness value of best state: 8833.333333333332
Minimum number of subsets that can cover the Universe-set: 19
Time taken: 26.42 seconds
```