Genetic Algorithm Applied to the Graph Colouring Problem

Shahil Singh Rawat

2020A7PS0138G

Submitted to

Prof. Sujith Thomas

For the course

Artificial Intelligence

CS-F407

Abstract

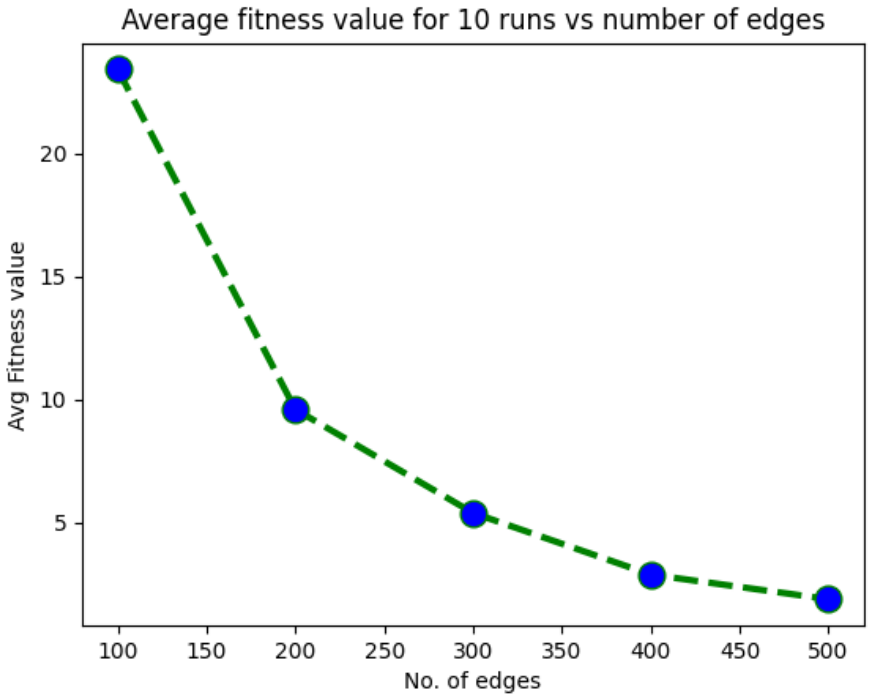
In the assignment, I tried to solve the vertex colouring problem by using the genetic algorithm. The problem was to colour the 50 vertices of a given graph with three colours such that none of the adjacent vertices have the same colour. I tried on different crossover techniques viz. **Single point crossover, two point crossover** and **uniform crossover** and also varied the number of states in the population set. I also varied around the mutation rate and finally arrived at a genetic algorithm that **fully optimises graphs** with 50 and 100 edges and gives around 32 as a fitness score for 200 edges.
The final solution uses two - point crossover method and the mutation probability is 0.2

Approach

a) First, we assign a random colour to each edge from the set of colours {Red, Blue and Green}. This step is repeated 100 times till we have a population set of 100 with 50 vertices each.

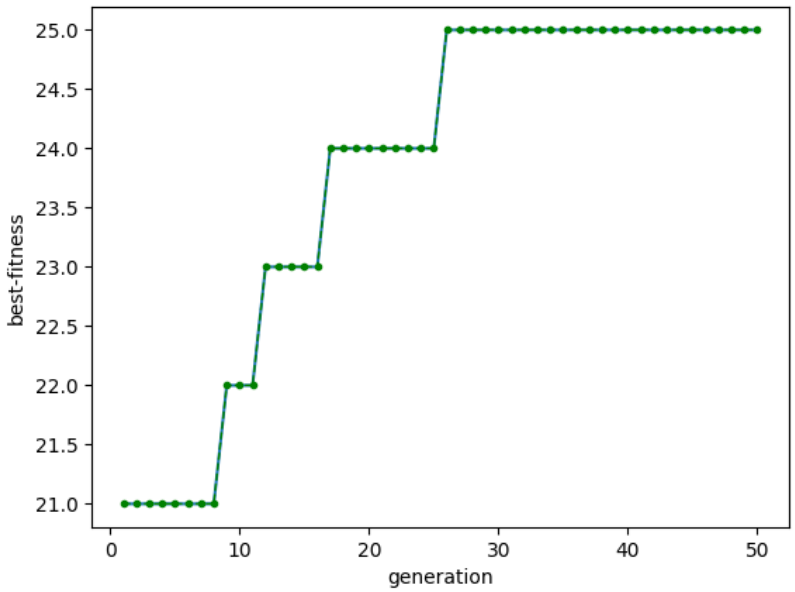
   The genetic algorithm applied here then sorts the population set according to their fitness values and then produces the next generation (children) by crossing over the top 2 fittest states (elitism) and pushes the child set with two parents. The algorithm then mutates the children with a small probability of 0.01. The population size is 100 and the number of generations is 50. The number of vertices is 50 and the edges have been randomly generated with numbers 100, 200, 300, 400, 500.

The plot shown below shows the mean of best fitness states vs number of edges after considering 10 randomly generated graphs.

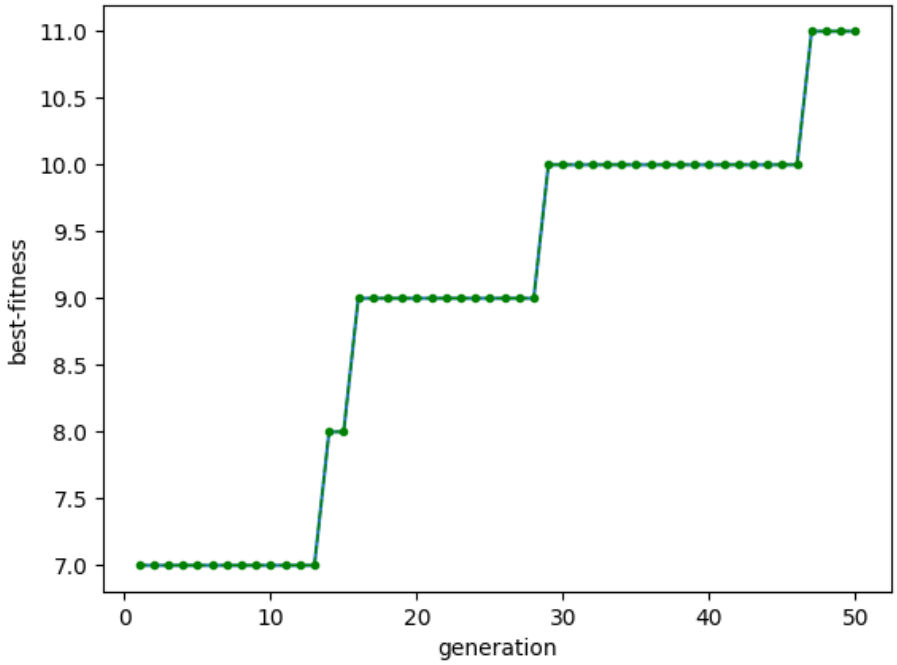


Next, the below plots show the best fitness value variation over the 50 generations for different edge sets

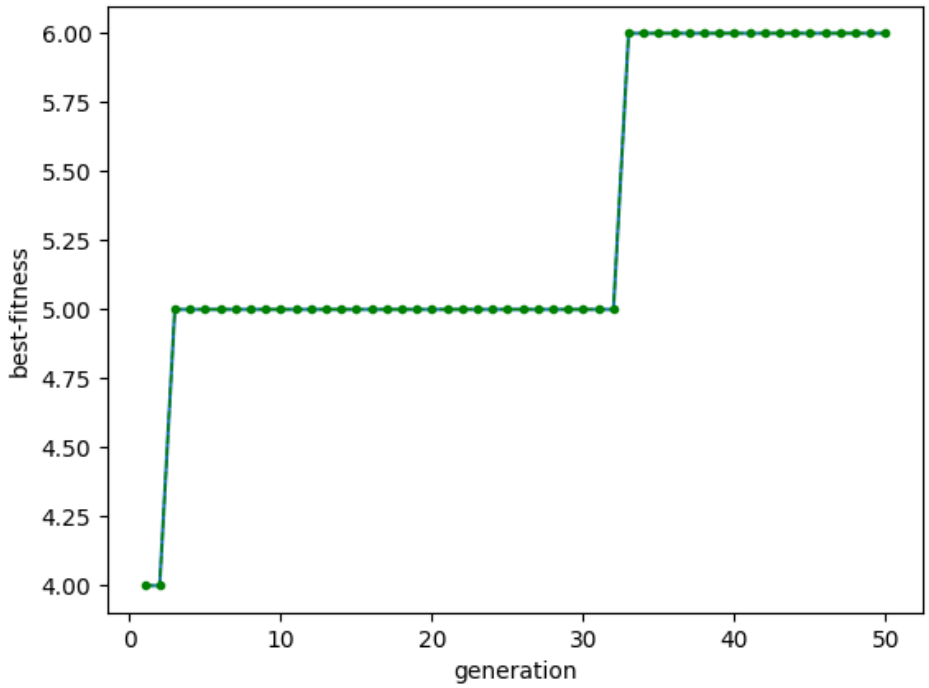
1. 100 edges :



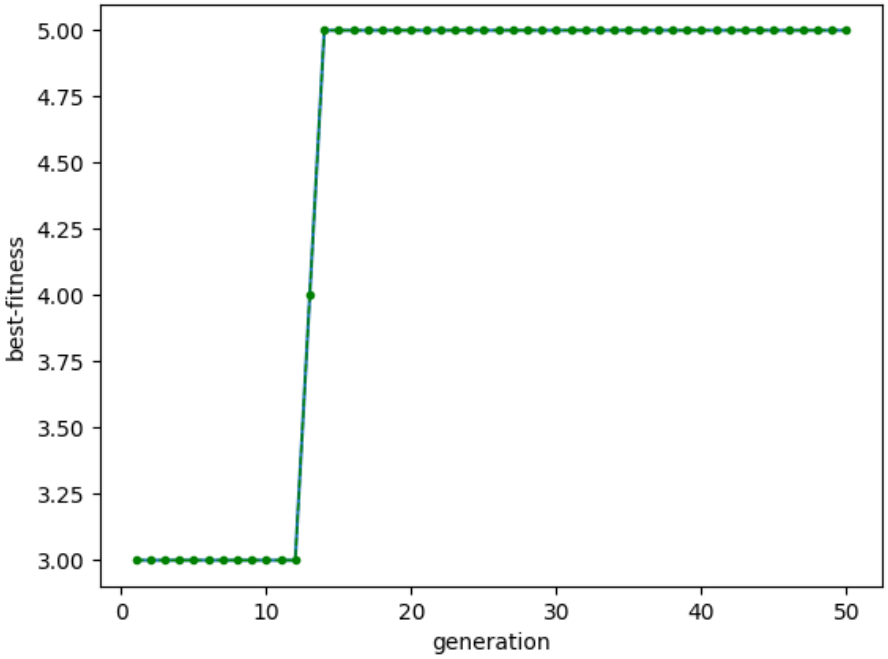
2. 200 edges



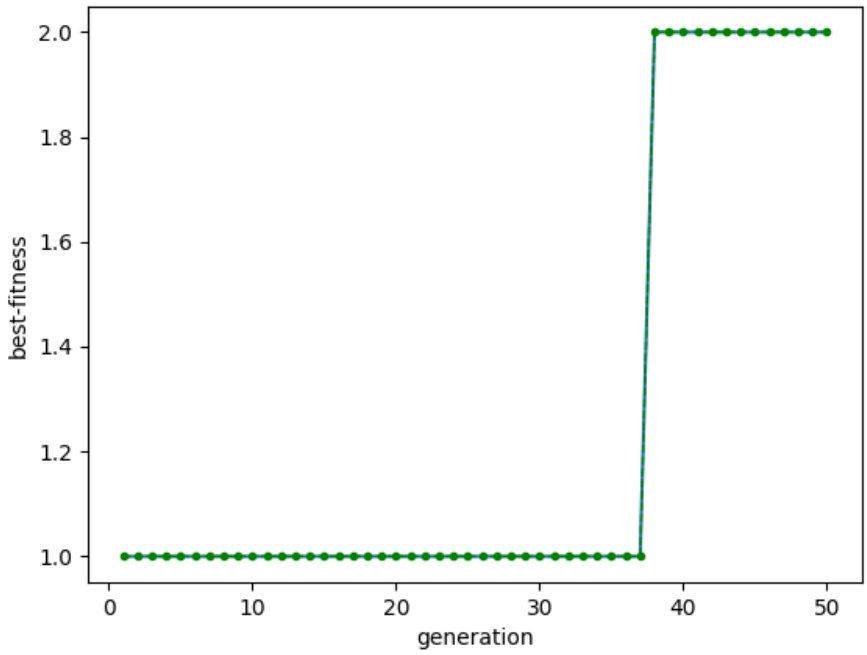
3. 300 edges



4. 400 edges



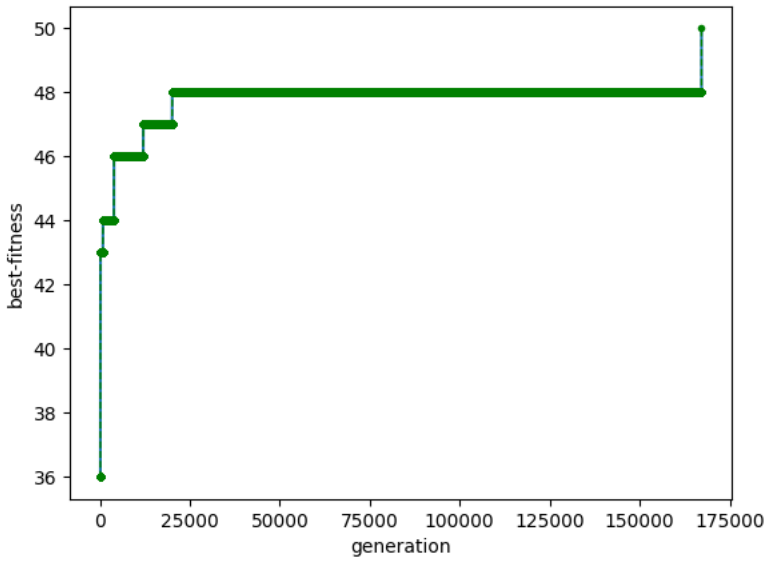
5. 500 edges



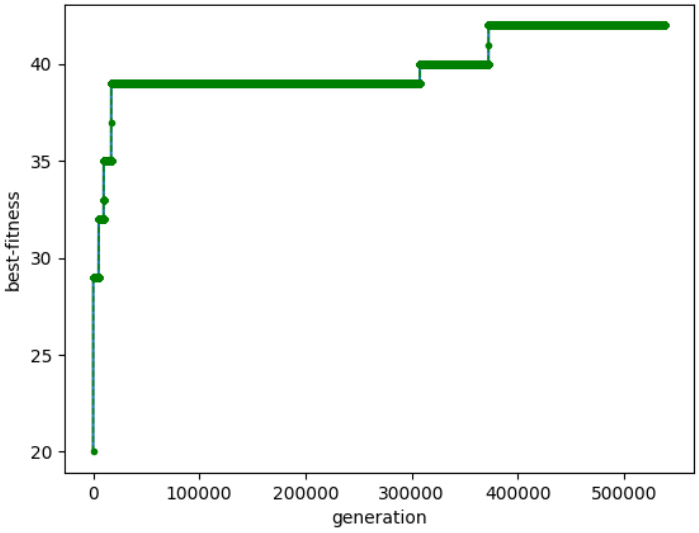
Part b:

1. The first thing I realised was that my algorithm was mutating the child states with the same probability for all generations irrespective of the fitness value. So I changed the probability of mutating from 1 / 100 to 1 / fitnessvalue.

   For the 50.csv file, the best fitness value was 50 for 9 cases out of 10 within 15 seconds and the best time taken was 8.86s.

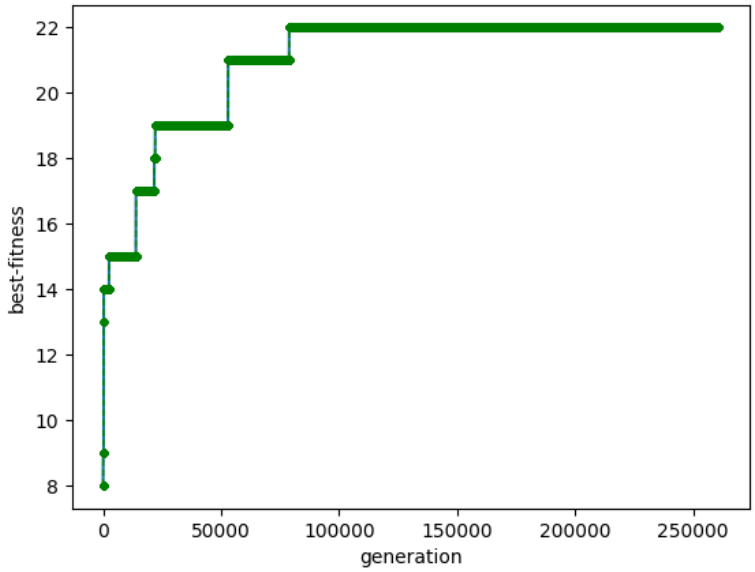


   The average fitness over 10 runs was almost 50.
   For 100.csv the best fitness value from 10 runs was 42 and the generations were 538985.



   The average fitness value for 10 runs was approximately 39.

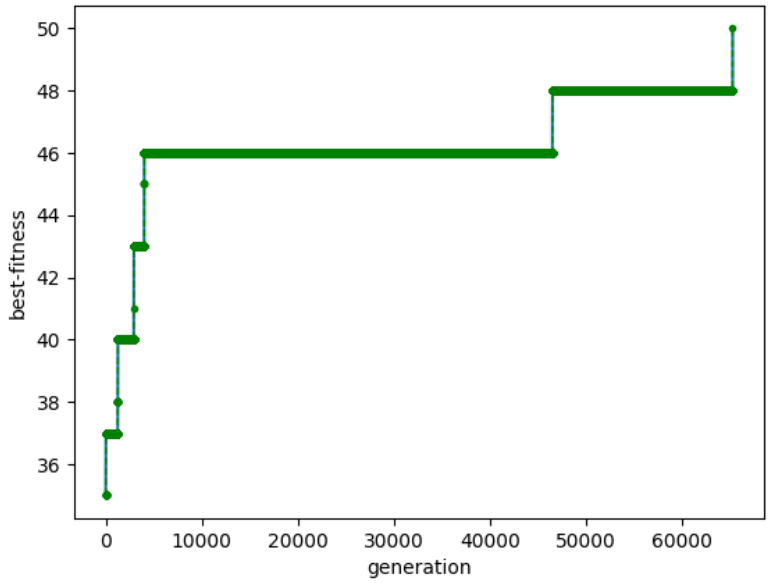
For 200.csv file, the best fitness value obtained after 45 seconds was 22 for 260951 generations

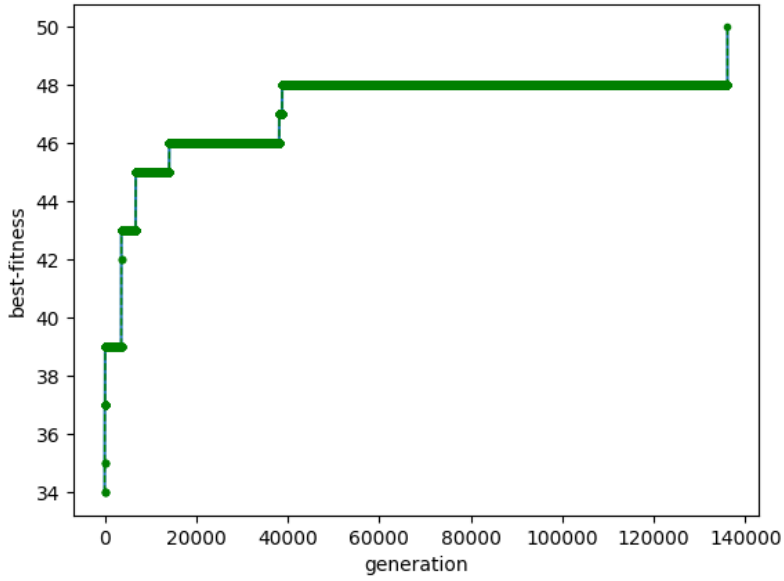


The average fitness for 10 runs was approximately 19.

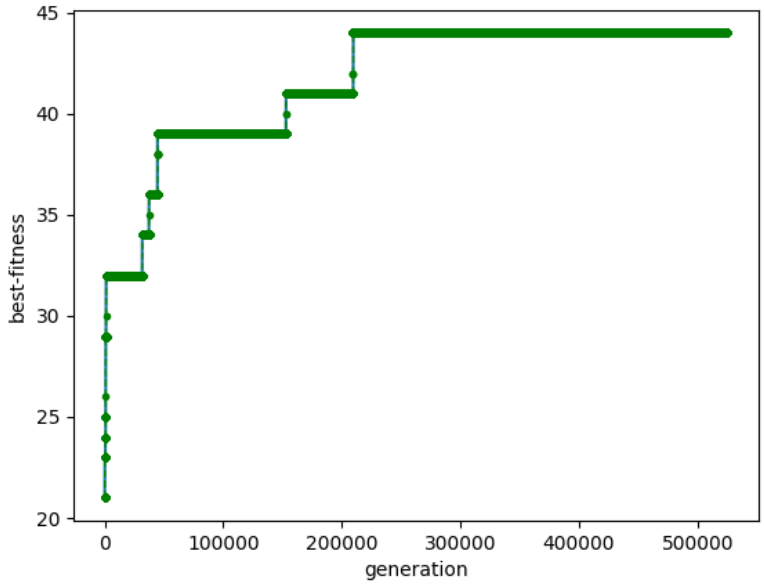
2. Next, while keeping the mutation rate constant, I tried to get the children state by using a two-point crossover instead of one.

   For edges from the 50.csv file, the best fitness value from 10 runs was 50 (full fitness) and the generations were 65296 in 3.51 seconds. The algorithm almost always gave 50 as fitness value within 10 seconds.



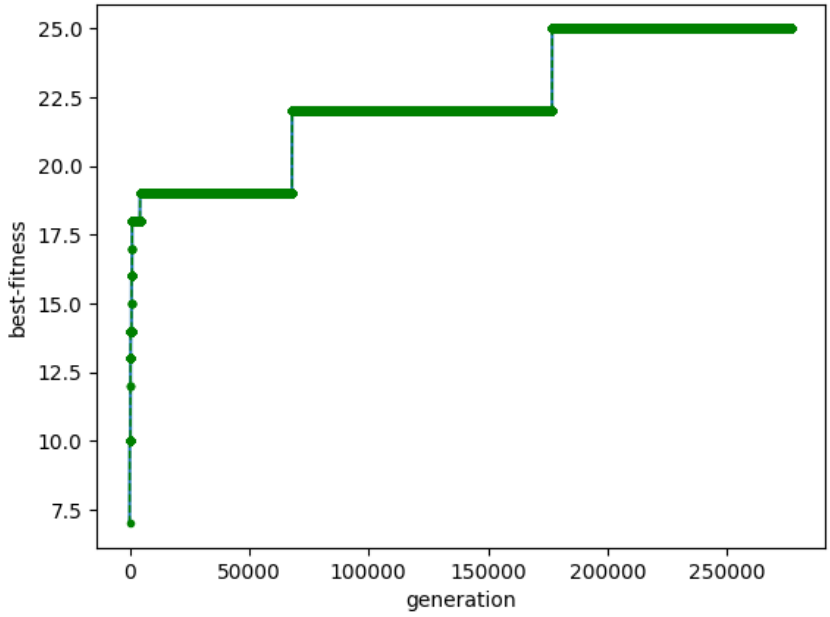


For edges from the 100.csv file, the best fitness value was 44 and 525204 generations were taken.



While the average fitness value score for over 10 runs was 41.5.

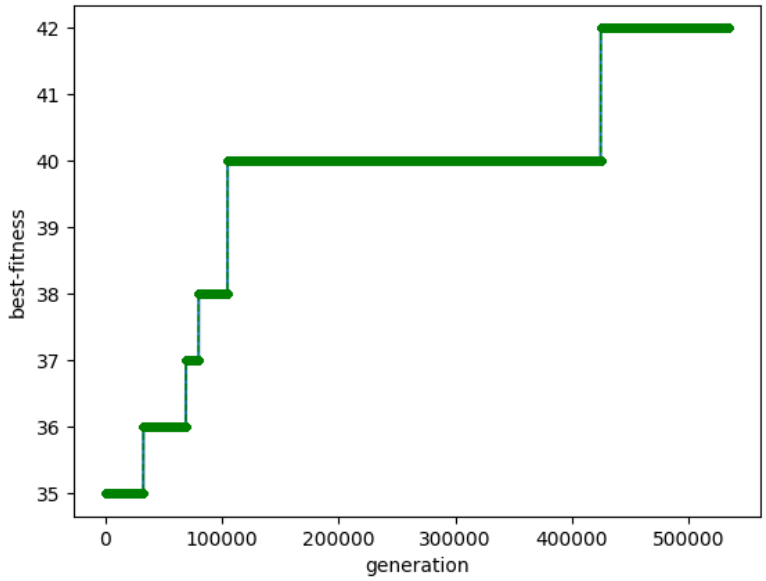
For edges from 200.csv file, the best fitness value graph : (25 fitness value and 277209 generations)



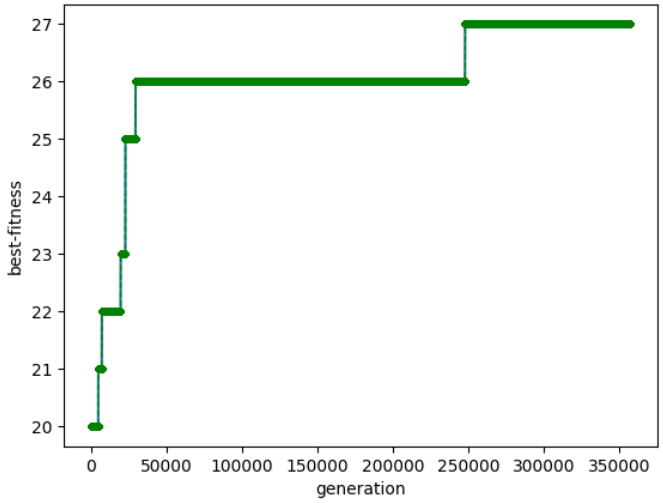
While the average fitness value score for over 10 runs was 21.5.

3. Next, while still keeping the mutation rate constant, I tried to get the children state by using a uniform crossover. Here we simply traverse the arrays, and swap individual indexes if a coin toss comes up heads with probability p

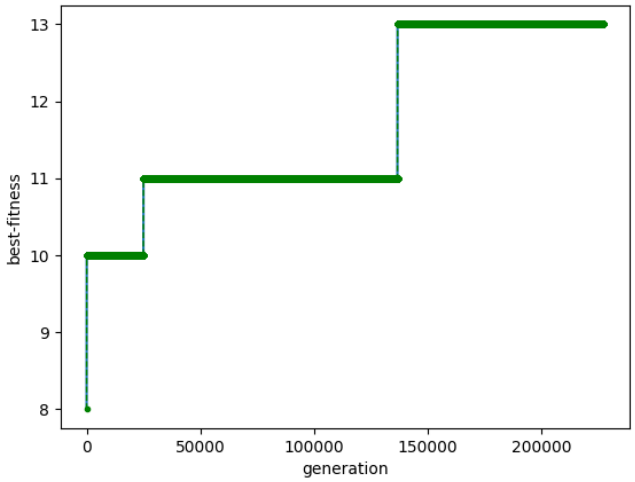
   This crossover method lead to **decrease** in fitness value across all 3 edge sets:
   For 50 edges



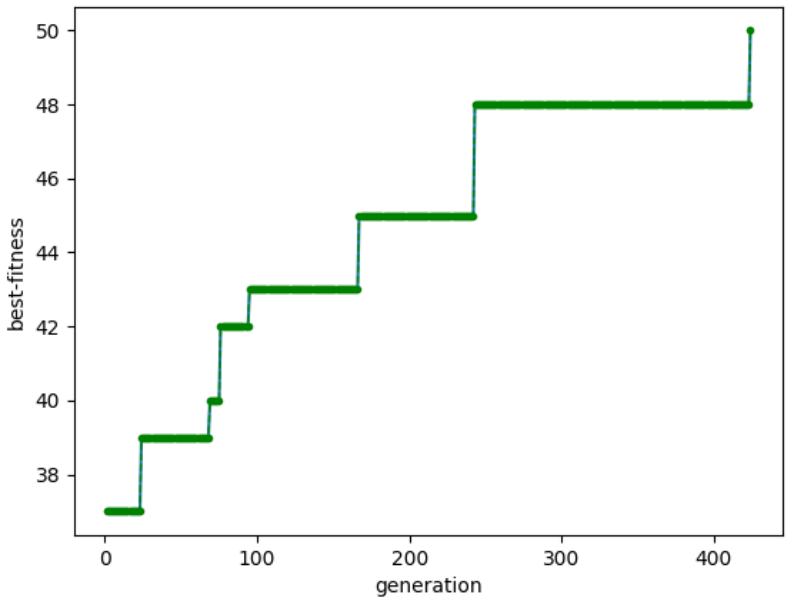
   For 100 edges



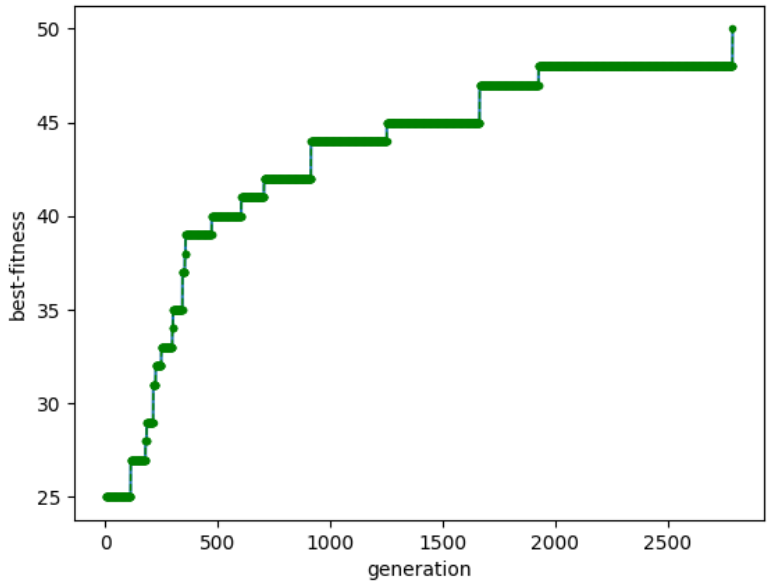
   For 200



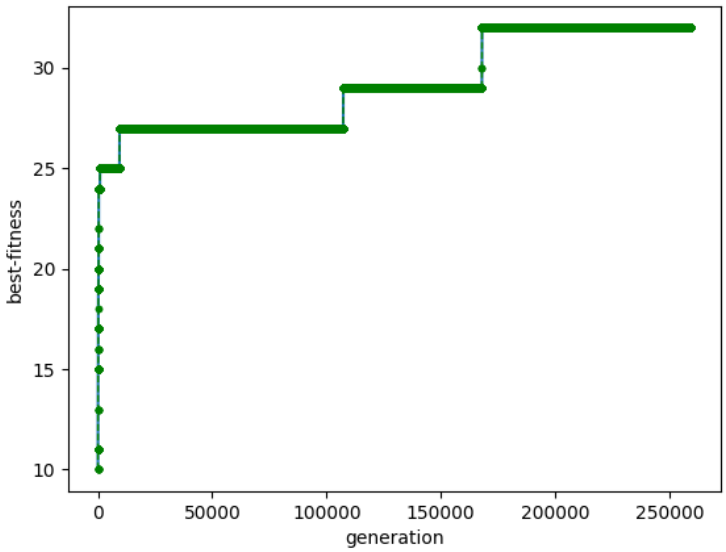
4. Finally, after making various small changes in my code I was able to get to a genetic algorithm that fully optimises 50 and 100 edges under 2 and 5 seconds respectively in most cases. For 200 edges, the highest fitness value obtained was 32.



For 50 edges 424 generations and 0.0909 seconds.



For 100 edges 0.325 seconds and 2790 generations.



For 200 edges 259581 generations and 45 seconds.

Conclusion:

The genetic algorithm finally used a two point crossover method along with a relatively high probability of a state mutating. I have also included my other attempts as comments in the python file for more insights.