Solving the Set Covering Problem using Genetic Algorithm

In this assignment the goal was to solve the Set Covering Problem(SCP) using a Genetic Algorithm(GA). The SCP is an optimization problem where we are given a Universe U and a collection of subsets S, our goal is to find the minimum number of subsets that would span the entire universe,

In this problem the universe contains all integers from 1 to 100. We can use the subsets from scp_test.json or we can also use the SetCoveringProblemCreator to generate subsets.

State Representation: Each solution (or individual) was represented as a binary vector, where each position in the vector corresponds to a subset in S. A 1 in the vector indicates that the subset is selected, and a 0 indicates it is not.

Fitness Function: The fitness function was designed to reward solutions that cover more elements of the universe and use fewer subsets. The formula for fitness penalized solutions that used many subsets, promoting more efficient solutions.

Population Initialization: The initial population consisted of randomly generated binary vectors representing different combinations of subsets.

Crossover: Single-point crossover was used to combine pairs of parents into offspring, creating new solutions by mixing the characteristics of their parents.

Mutation: A mutation operation with a low mutation rate (1%) was applied to the offspring to introduce random changes in the solutions. A low mutation rate ensures a very low chance of mutation.

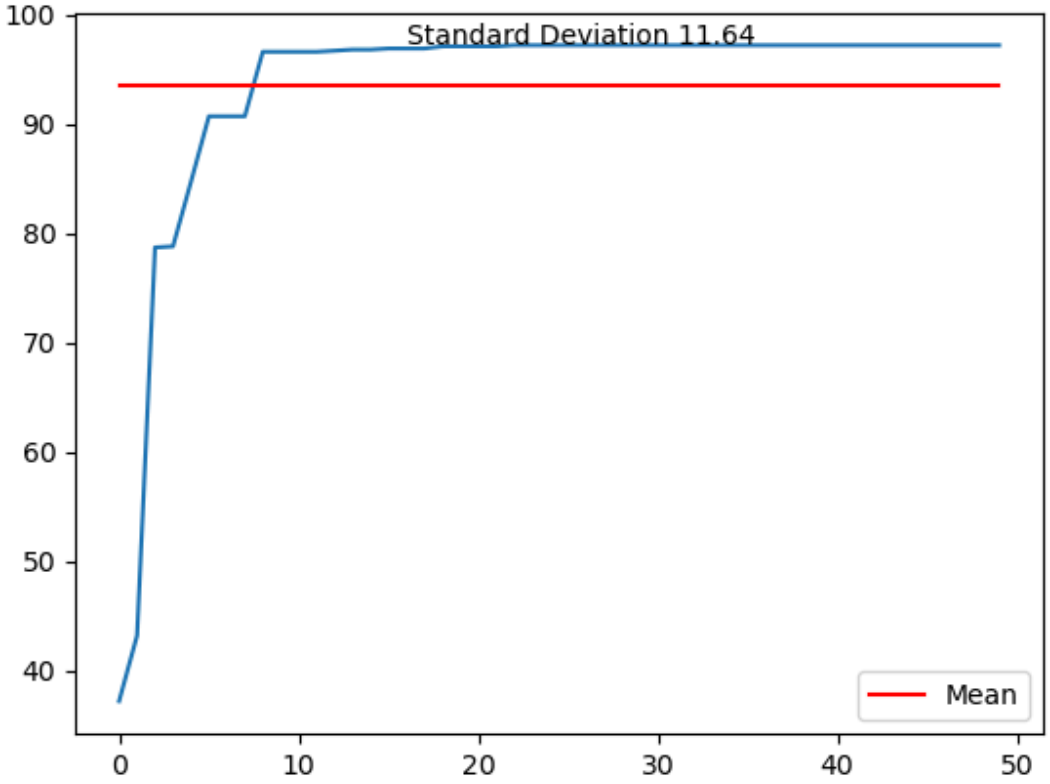
Termination: The algorithm was set to run for 50 generations or terminate after 45 seconds, whichever came first. The best solution found during this time was reported as the final result.

Modifications:
I made various modifications to the genetic algorithm. Firstly I tried varying the population size and number of generations as suggested.

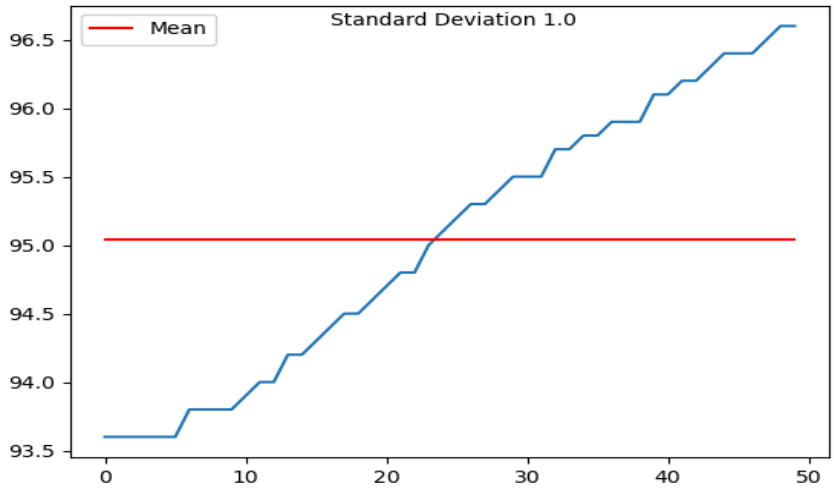
On increasing the population size it was noticed that the genetic algorithm took more time to converge to the final solution but on average had a better best fitness.
On decreasing the population size I noticed that the algorithm took less time to converge to the best solution, but it had a worse best fitness.

On decreasing the number of generations the algorithm converged faster but failed to find the optimal solution. On the other hand increasing the number of generations had a completely different effect on the performance of the code. Most of the improvements happened in the early generations and the fitness increased at a slower rate after a certain number of generations.

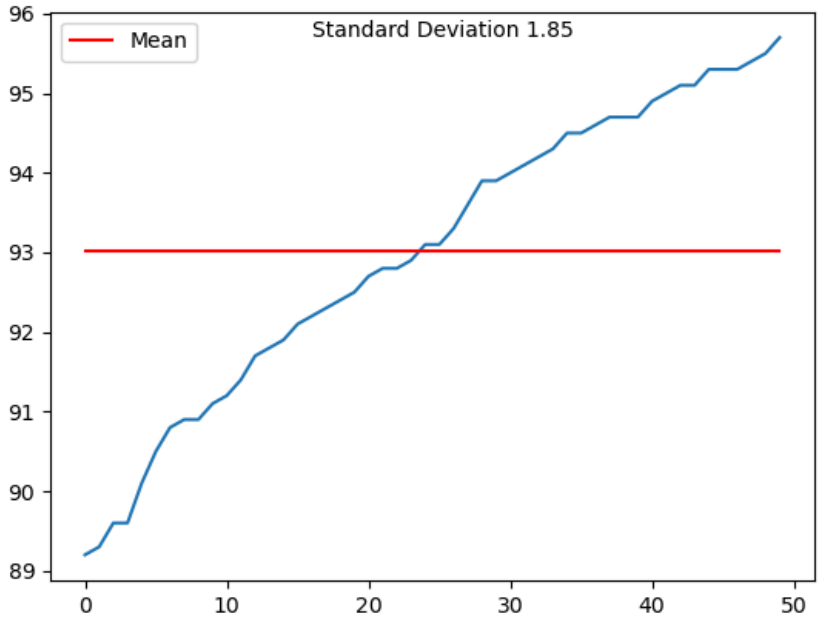
Performance of Genetic Algorithm with subset size of 50



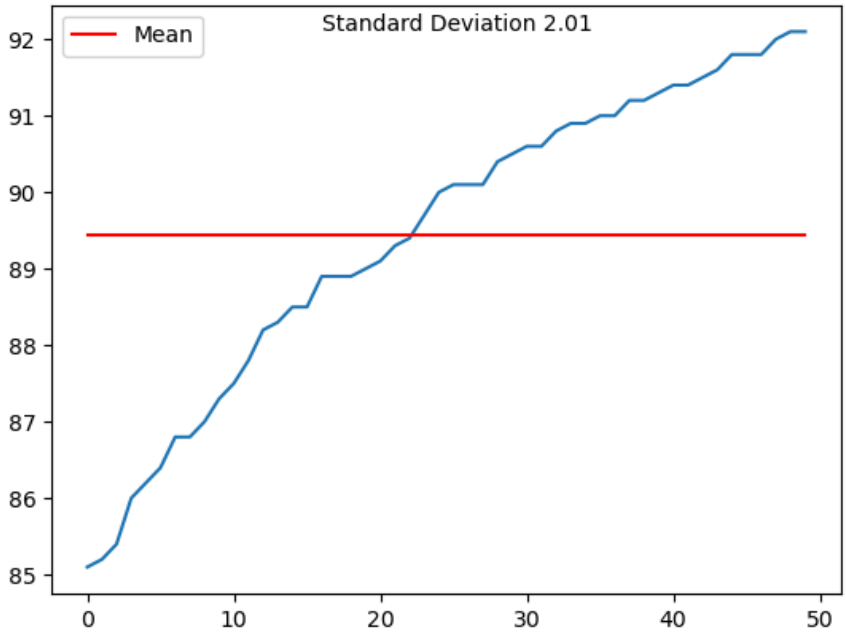
Performance of Genetic Algorithm with subset size of 150



Performance of Genetic Algorithm with subset size of 250



Performance of Genetic Algorithm with subset size of 350



Performance of Genetic Algorithm with different subset sizes