Solving the Set Cover Problem: A modified Genetic Algorithm approach

Siddharth Chaitra Vivek
2022A7PS0569G
BITS Pilani, Goa Campus

October 5, 2024

Abstract

This report outlines my approach to solving the Set Cover Problem using a Genetic Algorithm. It presents the results from a standard Genetic Algorithm implementation, followed by a detailed explanation of modifications aimed at improving convergence and final accuracy. These modifications include designing a well-defined fitness function, introducing decaying crossover and mutation rates, implementing multiple crossover points for a fixed mixing number, and including local beam search with an increasing rate to address the limitations of the Genetic Algorithm. The modified approach performed as expected, yielding improved results that support the hypothesis.

Problem Statement

Set Covering Problem (SCP): You are provided with a randomly generated instance of the Set Covering Problem (U, S), where the universe set U consists of integers from 1 to 100. The collection S contains m subsets of U. The goal is to find the minimum number of subsets from S whose union covers all elements in the universe set U. In an optimal solution, these selected subsets collectively cover every element in U.

Approaching the Problem

Genetic algorithms (GAs) are optimization techniques inspired by natural selection, where a population of potential solutions evolves over generations to solve complex problems. Each solution (individual) is evaluated using a fitness function, and those with higher fitness are more likely to reproduce through crossover and mutation, creating new solutions. This process repeats, gradually improving the population. This makes GAs useful for solving problem with large search spaces like the SCP.

Since the problem involves selecting subsets, my idea to represent a particular individual/state was though an array the size of the number of subsets containing binary values. This way each bit represents whether or not a certain subset is chosen in the current state, as shown below

$$[\ 1\ \ 0\ \ 0\ \ 1\ \ 0\ \ 0\ \ \cdots\ \ 0\ \ 0\ \ 1\ \ 0\ ]$$

The i-th index being a 1 indicates that the $(i+1)$-th subset is included in the current state.

This representation also allows for easily defining other features of the Genetic Algorithm:

1. **Neighbouring States:** In this case, a neighbouring state would be a state obtained by flipping one value. This represents including an extra subset or excluding one of the current ones.
2. **Crossover:** Crossover is represented easily through simple slicing and exchanging of subarrays of the parents
3. **Mutation:** Similarly, mutation is achieved by flipping each bit in the state with a certain probability.

Using these representations, the following 2 implementations were made

1. **Basic Version:** Follows the explanation in the textbook, implementing everything except culling.
2. **Improved Version:** Additionally introduces decaying crossover and mutation rates, implements multiple crossover points for a fixed mixing number, and includes local beam search as the state reaches closer to convergence

The changes made in the improved version are explained in more detail later in the report

Fitness Function

The aim of the fitness function is to represent how good the current individual/state is. For the SCP, we are aiming to **maximize cover** on the universe with a **minimal number of subsets** as possible. Hence the fitness function I formulated includes 2 terms, one representing each aim.
Let:

- U be the universal set.
- $S = \{S_1, S_2, \ldots, S_m\}$ be the collection of subsets.
- $C(x)$ be the cost of the state x
- x_i be the value of the ith element in the state x

The fitness function $f(x)$ for state x is:

$$f(x) = 100 - (\alpha \times \left(\frac{|U| - |\bigcup_{S_i \in x} S_i|}{|U|} \right) + \beta \times C(x))$$

where

$$C(x) = \frac{1}{m} \sum_{i=1}^{m} \mathbb{I}_{\{x_i = 1\}}$$

and α = 50 and β = 50 are constants. Altering these values would allow to give more importance to one of the aims, but this report doesn't cover experimentation on these hyper parameters as setting equal weight itself yielded a good solution.

The first term that is subtracted aims at maximizing the coverage of the universe. If no part of the universe is covered, this term is equal to 50 and results in a large subtraction from the fitness value. As more of the universe is covered this value decreases towards 0 and fitness function increases.

The second term that is subtracted aims at minimizing the number of subsets used, using all m subsets will make the term equal 50 , but the less used, the term reduced towards 0 as well.

Hence very good states will have a fitness of close to 100, whereas bad states will have low fitness values

Basic Solution

As mentioned earlier, the basic implementation merely follows the textbook, including everything except Culling. It uses the custom made fitness function but doesn't any have any other modifications.
This version performed poorly initially without elitism, averaging between 70-80 subsets needed for full cover. However on implementing elitism, there was improvement but still needed upto 60 subsets. There were still several issues, like slow convergence and not giving a correct (full covered) solution. The results for average over 10 runs (50 generations, 50 population size) is shown in Figure 1.

Improved Version

The improved version includes 4 main changes, this section goes into detail regarding the changes and the motivation/idea behind them. Similar tests as the Basic Solution are shown in Figure 2. The algorithm however performs much better when given more time to explore the search space and hence something like 3000 generations performs much better that just 50, this is show in Figure 3. Further Figure 4 shows one of the solutions that is consistently generated by the code for the SCP given in pre-generated JSON File, achieving full cover in 18 subsets only.

Decaying crossover and Mutation rates

In genetic algorithms (GAs), since getting the balance right between exploration and choosing the greedy solution is key to their success. Early on, when the algorithm is just starting out, having a high mutation rate helps by introducing a lot of variety and prevents the algorithm from getting stuck in sub optimal solutions too soon. High crossover rates at this stage also help by mixing up states. These essentially result in large jumps in state allowing the algorithm to potentially uncover some promising areas of the search space.

As the algorithm progresses and starts to zero in on better solutions, a change in approach is needed. Reducing the mutation rate helps focus the search on greedily choosing solutions. We do this because at this point we have already explored the search space and elitism has allowed for the best states to carry forward. Now we need to improve around this, but if mutation rate stays too high, it can disrupt this fine-tuning process and prevent the algorithm from settling on the best answers. Similarly, lowering the crossover rate helps keep the good solutions intact.

My implementation hence decays both these quantities as follows:

Crossover Rate:

$$0.9 \xrightarrow{\text{decay linearly}} 0.5$$

Mutation Rate:

$$0.05 \xrightarrow{\text{decay linearly}} 0.001$$

Multiple Crossover points

As discussed earlier, the process can be divided into two phases: Exploration and Greedy Search. To enhance the algorithm's exploration capability, I modified the traditional method of using a single crossover point by incorporating multiple crossover points. This adjustment enables larger jumps within the search space by increasing the mixing of states. Implementing this change led to noticeable stagnation at the beginning of the algorithm, which suggested that the approach was effective. Over time, this stagnation diminished as the crossover rate was reduced.

Hybrid Approach using Local Beam Search

Finally the most significant and effective change was implementing a hybrid approach of using Local Beam Search as well.

The genetic algorithm excels at exploring a broad search space and discovering diverse solutions through its crossover and mutation operations. This global exploration helps avoid local optima and ensures that a wide range of potential solutions is considered. However, as the GA progresses, it may face challenges in looking nearby to these solutions due to the inherent randomness and the broad search focus.

Introducing a local beam search towards the end of the GA capitalizes on its ability to perform a more focused, local search around promising solutions. Local beam search, which involves maintaining a fixed number of candidate solutions and iteratively improving them based on local search strategies, yields better results. By focusing on a narrower region of the search space, local beam search helps identify and greedily explore only the most promising areas.

Since there needs to be a balance between exploration and exploitation, this too was done in a dynamic fashion as follows:

$$\text{Beam Search Rate: } 0.01 \xrightarrow{\text{increase linearly}} 0.1$$

Once again, this improved the algorithm visibly as the solutions suddenly improve towards the end of a run as the beam search rate increases.

NOTE: Regarding Hyperparameters

The sections above mention several hyperparameters like α and β in the fitness function and the various initial and final rates and linear decay rate of crossover, mutation and beam search. These values were fine tuned through manual testing and observation. Further improvements could be refining these through methods like grid-search.

Conclusion

To conclude, the report explains the improvements made to the Naive Genetic Algorithm and shows that it results in significant improvement to the results. It leads to a more consistent rate at which the problem is full solved (all 100 elements covered) and is also more optimized (lesser number of subsets needed).

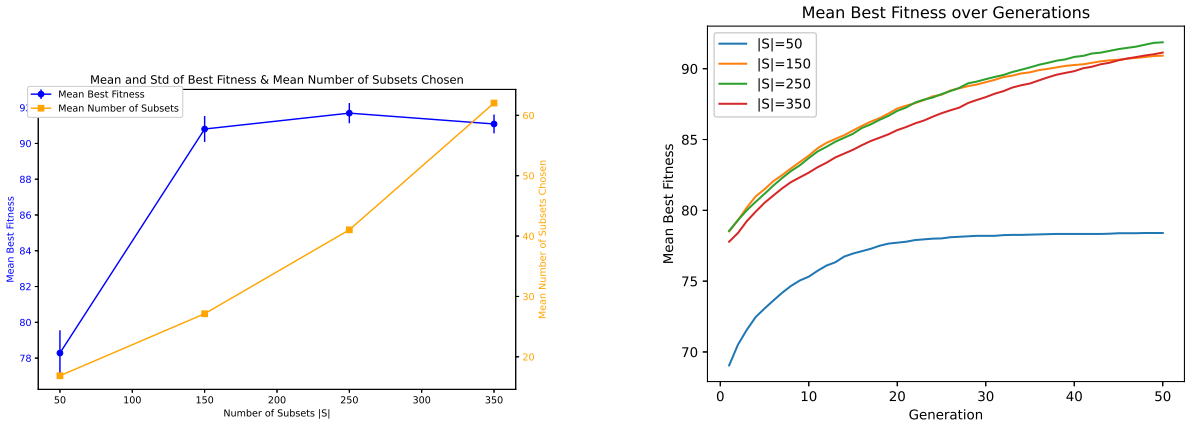

Figure 1: *The graph shows that for low number of subsets (50) the algorithm is only able to cover 77 elements in the universe. Further for higher subset counts, a higher coverage of 90+ is achieved but still doesn't solve the problem with certainty and uses a lot of subsets. The fitness values increase normally without any abnormalities for 100+ subset size.*

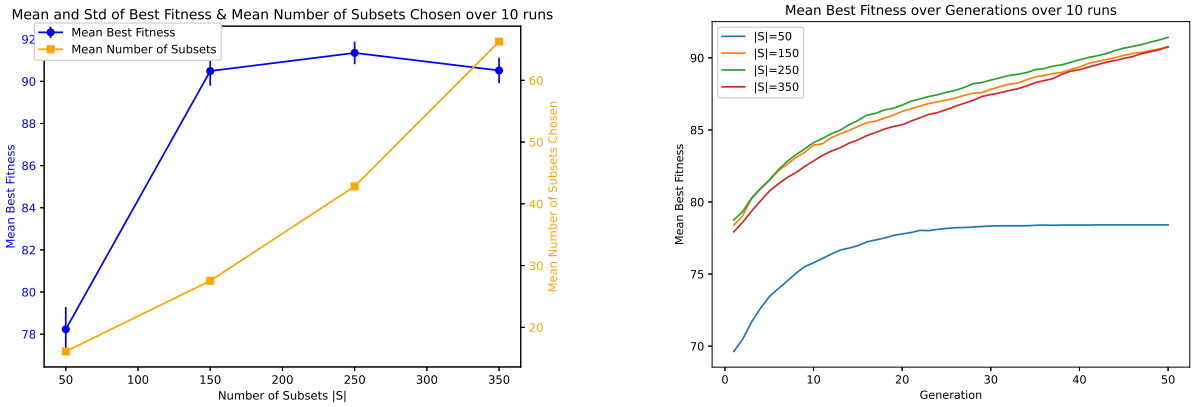

Figure 2: *Same test run for the modified algorithm*

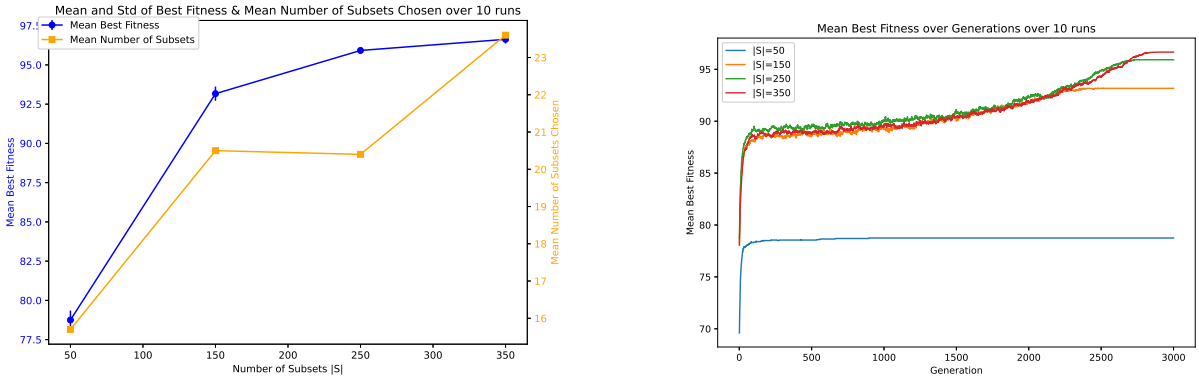

Figure 3: *Over 3000 generations it is visible how the various modifications lead to a much better solution. Several features are also visible like how local beam search pushes the solution more towards the end*