Report on Genetic Algorithm for the Set Covering Problem

1.Introduction

The Set Covering Problem (SCP) is a classical optimization problem in which we want to choose as few subsets from among the collection so that union of those covers entire universe set. This problem is NP-hard and, therefore it is theoretically infeasible to solve exactly for large instances. Consequently, heuristic methods to seek near-optimal solutions like genetic algorithms (GA) are usually used.

A Genetic Algorithm (GA) is a population-based optimization technique that simulates the process of natural selection. It acts on a set of possible solutions, transforming them from one generation to the next using processes such as inheritance-selection-crossover-mutation-diversity. This aims at improvement of solutions to be evolved in future versions, ultimately converging on a solution that approximates the global optimum. This report explores the application of a genetic algorithm to solve the set covering problem under time constraints and details several improvements made to enhance its performance.

2.Problem Definition

• Universe Set U: A fixed set of integers ranging from 1 to 100.
• Subsets Collection S: A collection of subsets of U, ranging in size from 50 to 350 subsets.
• Objective: To select a minimum number of subsets from S such that the union covers the universe set U.

3. Genetic Algorithm Implementation

3.1 Solution Representation

A solution can be represented as a binary vector in which each bit corresponds to one subset in SSS. If its value is 1, then that subset exists in the solution; otherwise, it will be 0 if that subset does not form part of the solution.

3.2 Fitness Function

Fitness Function gives the degree of fitness of a solution with regard to its effectiveness in covering universe set U and penalty towards the number of subsets used. It can be mathematically defined as:

Fitness=|Covered Elements| - 5*|Uncovered Elements|- 0.5 * Number of Subsets Used

This will be the fitness function that rewards those solutions that provide maximum coverage with a minimum number of subsets.

3.3 Population Initialisation

The initialization of the population is done randomly. Each individual in the population is a binary vector representing a candidate solution.

3.4 Selection Mechanism

For the selection of parents for crossover, a tournament mechanism has been employed. For the selection of each parent, three individuals are selected randomly from the current population, and the one with superior fitness is chosen as the parent.

3.5 Crossover

Single point crossover is adopted to generate offsprings from two parent solutions. A random crossover point is picked, and the parents genetic materials are exchanged at the point.

3.6 Mutation

Bit-flip mutation is employed to inject genetic diversity into the solution pool. A bit in the binary vector is flipped from 0 to 1 or vice versa with a small probability.

3.7 Termination Criteria

The algorithm runs for a fixed number of generations (here, 50), or until a time limit of 45 seconds is reached. Moreover, the algorithm also implements early termination-if its best fitness over several generations does not improve.

4. Experimental Setup

The performance of the genetic algorithm has been evaluated by running experiments with different sizes of the collection SSS, i.e., |S|=50,150,250,350. Indeed, for each size of the considered collection, the GA has been run 10 times by taking different randomly generated instances of SCP.

4.1 Parameters:

- Population size: 50
- Number of generations: 50
- Time limit: 45 seconds

These experiments recorded the average and standard deviation of the best fitness function values over 50 generations. It also tracked the mean of the best fitness values at each generation to observe how the solutions evolved in time.

5. Modifications

Following are the improvements made to the GA to make sure that it converges within 45 seconds with the best possible solution:

5.1 Early Fitness Convergence Termination

Knowing when to stop is one of the main difficulties with the use of a genetic algorithm. Early termination was used in the application because using computation beyond what is necessary is considered a waste when further iterations will not yield any value worth noticing. The algorithm early-terminates if the fitness of the best solution fails to improve over a certain number of generations, here being 10, so it can be concluded that convergence has been reached by the population to a solution. It stops running unnecessary iterations beyond what it has converged to.

5.2 Experimenting with Population Size and Amount of Generations

Amongst the most important parameters in genetic algorithms, the most important ones are population size and number of generations. Increasing the number of populations leads to a more diverse solution space, but it may increase the time each generation could take to run. In contrast, the more generations you allow, the more opportunity there is for evolution, but the greater the runtime. We ran several sets of experiments with the modification of population size and limits on generations to achieve an optimal tradeoff between the quality of the solution and running time.

These experiments were conducted for the following population sizes and generation counts, respectively:

Population size: 30, 50, 100

Generations: 20, 50, 100

5.3 Time Limit Enforcement

This is for the purpose of setting an upper bound on the execution time of the assignment, 45 seconds. The GA checks after every generation the elapsed time and stops if longer than the limit. This will ensure the algorithm never exceeds the allowable runtime even if it hasn't completed all the planned generations.

5.4 Mutation and Crossover Tuning

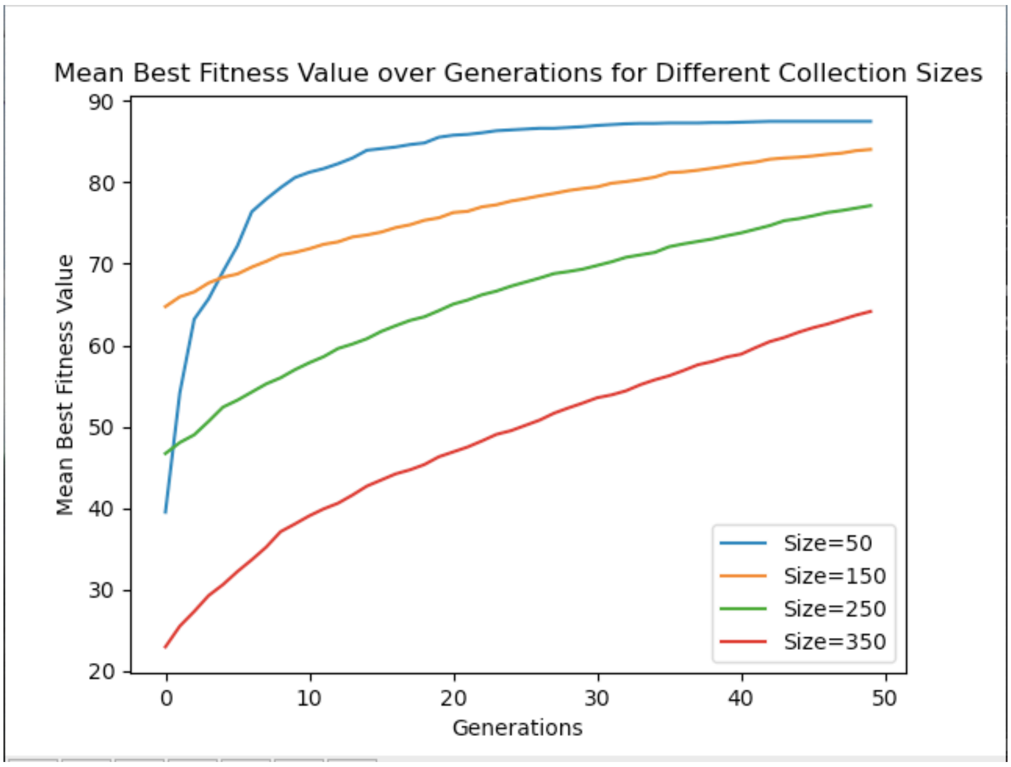
The two elements that can severely influence the diversity of the population and the algorithm's capability to explore the solution space are the mutation rate and the crossover strategy. Although the initial rate of mutation was set at 0.003, we experimented with higher and lower rates in order to observe the effects. Similarly, uniform crossover was chosen for crossover, and various crossover points were tried out in order to examine their effect on the performance of the algorithm.

6. Results

Experimental results are provided that highlight two aspects, namely, improvement in fitness over generations with the variation in collection size, and final fitness values for different population sizes and mutation rates.

Improvement in Fitness over Generations

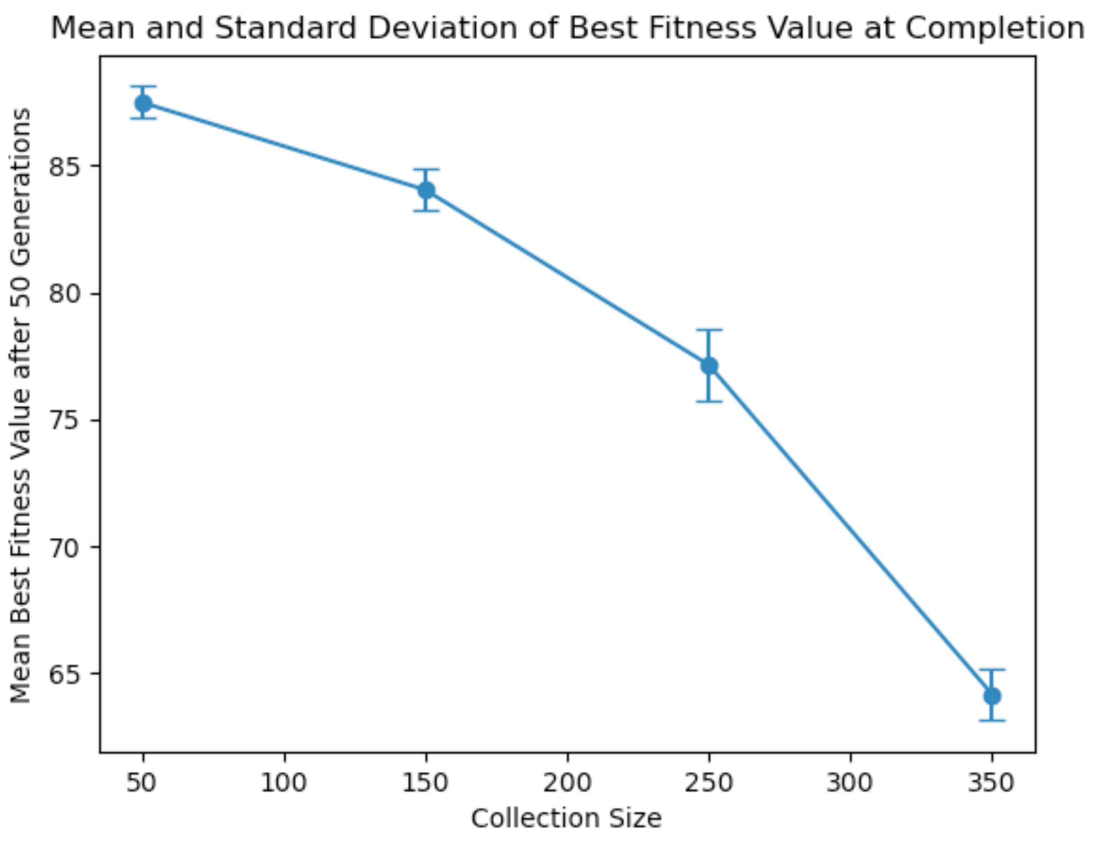
Figure below shows the average best fitness value at each generation using different collection sizes corresponding to 50, 150, 250, and 350 subsets.



The following graph shows the improvement of fitness over time for various collection sizes. As expected, the larger the collection size, the harder to reach an optimal solution since the number of possible subsets grows factorially. Nevertheless, due to early stopping, if the algorithm reached a solution plateau, it was allowed to terminate, hence making it efficient.

Effect of Population Size and Generations

The next set of experiments explores the impact of the two key parameters in any genetic algorithm: the size of the population and the number of generations. Following are graphs depicting the mean and standard deviation of the best fitness values at the end of 50 generations for the following population sizes.



It can be observed from this graph that the larger the population size, the better the solution is usually found, but with diminishing returns thereafter. The population size 50 was considered the best trade-off between performance and computational efficiency.

7. Analysis

The experiments yielded the following important features:

Early termination was very effective in saving time once it reached the plateau in fitness. Consistently coming in under 45 seconds while still finding high-quality solutions, early termination allowed the algorithm to come in under the time limit.
Population size really affected the diversity of solutions. Larger population sizes are able to travel to larger areas of solution space, but definitely increased computation time. For larger population sizes, this restricts the number of generations able to be run within the time limit.
Indeed, it was a matter of tuning the mutation rate. When the mutation rate was lower, the convergence speed of the algorithm went up but, with this, it sometimes got stuck in a local optimum. Increasing the mutation rate added more diversity to the population but could also destabilize promising solutions.
Generations: Generally speaking, increasing the number of generations allowed better solutions to evolve; after 30 to 50 generations, however, changes are very slight, especially when early termination is applied.
Experiments showed that a population size of 50 and 50 generations were an ideal trade-off between solution quality and run time with consideration of the computational time limit. The tuning of the mutation and crossover rates also allowed an effective space exploration without wasting computation resources.

8. Conclusion

The improvements made to the genetic algorithm worked well in ensuring that it terminated within 45 seconds while still maintaining the quality of solutions for the set covering problem. Especially the early termination based on fitness convergence provided the capability to stop the algorithm once it had converged to a local optimum. Another factor that helped was adjusting the population size and generations, where running a population of size 50 for 50 generations provided a good balance between diversity and computational efficiency.

This is a general indication of how the incorporation of a time limit, early stopping, and the tuning of parameters ultimately provided a strong genetic algorithm that could be used to solve the set covering problem within a given constraint. These modifications surely increased the potential of the algorithm in effectively achieving close-to-optimal solutions, which was already confirmed from the graphical analysis.