2021A7PS2627G

Kislay Ranjan Nee Tandon

Brief:

This implementation of a genetic algorithm tackles the Set Covering Problem (SCP) by representing potential solutions as binary strings. Each bit in the string corresponds to a subset in the collection, with a one indicating inclusion of the subgroup and a 0 indicating exclusion.

The algorithm starts by initializing a population of random individuals. Each individual is evaluated using a fitness function that balances two goals: maximizing the coverage of the universe (i.e., ensuring that the selected subsets cover all elements from the universe) and minimizing the number of subsets used.

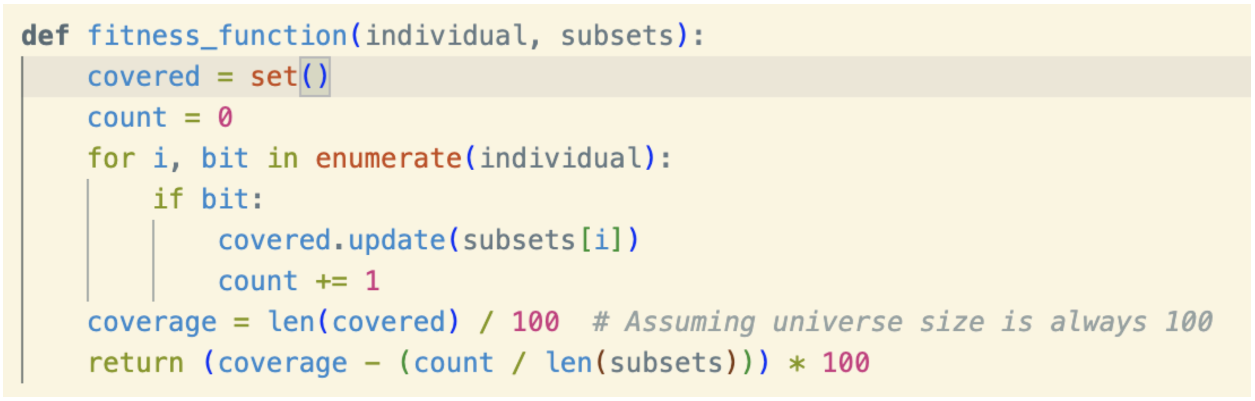
Selection is based on fitness, where fitter individuals are likely to be chosen as parents for the next generation. The crossover function combines two parent solutions to create offspring, while mutation introduces small changes, maintaining genetic diversity and helping the algorithm escape local optima.

The algorithm runs for several generations or until a time limit is reached. Throughout the process, the best solution and its fitness are tracked. The approach is repeated for different collection sizes, demonstrating the algorithm's performance under various conditions. This implementation effectively demonstrates how genetic algorithms can be adapted to solve complex combinatorial problems like SCP.

The Default mutation rate is 0.01.

Question 1 :

Implementing Genetic Algorithm for SCP



```python
def fitness_function(individual, subsets):
    covered = set()
    count = 0
    for i, bit in enumerate(individual):
        if bit:
            covered.update(subsets[i])
            count += 1
    coverage = len(covered) / 100  # Assuming universe size is always 100
    return (coverage - (count / len(subsets))) * 100
```

Fitness Function Summary:

The fitness function evaluates how effectively a subset selection covers the universe while minimizing the number of subsets used. It balances coverage (the proportion of the universe covered by the selected subsets) with efficiency (using the fewest subsets possible). The goal is to maximize this balance, resulting in higher fitness values for better solutions.

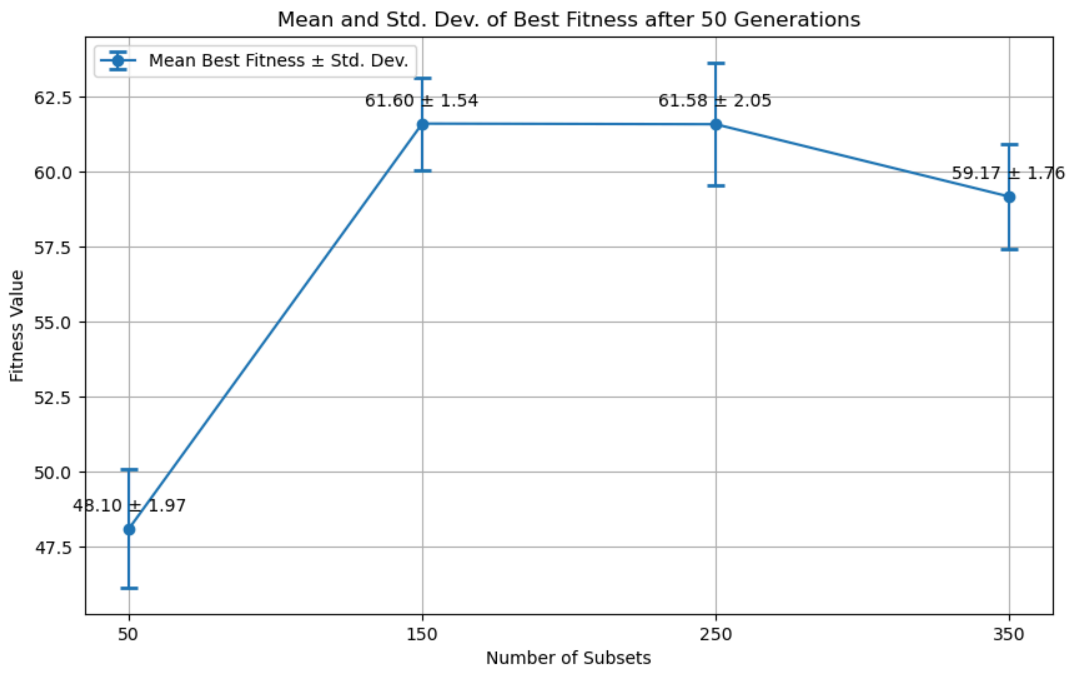


Figure 1: Mean Best Fitness Value over 50 generations

This graph illustrates the performance of the genetic algorithm for different subset sizes (50, 150, 250, 350) over 50 generations.

Key observations:

- Mean fitness improves significantly as the number of subsets increases from 50 to 150
- Peak performance is observed around 250 subsets
- A slight decrease in performance occurs when increasing to 350 subsets
- Standard deviation decreases up to 150 subsets, indicating more consistent results
- Beyond 250 subsets, the standard deviation rises slightly, suggesting reduced consistency

.

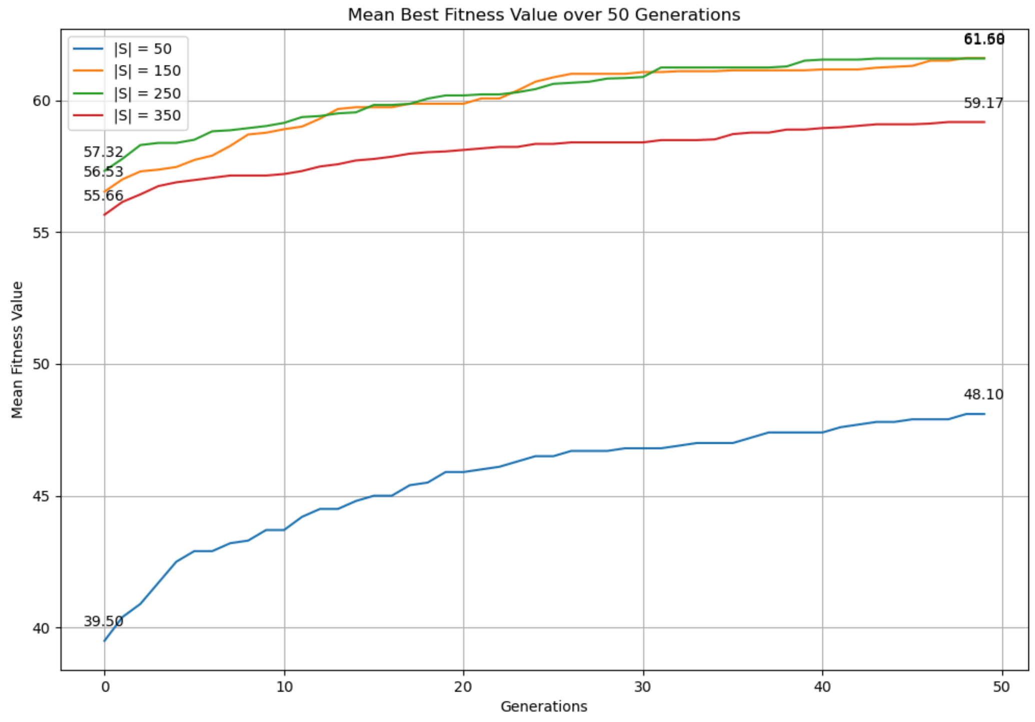


Figure 2: Mean and Standard Deviation of best Fitness values after 50 generations.

This figure provides a snapshot of the algorithm's performance after 50 generations for various collection sizes.

Key findings:

- Optimal performance is achieved with |S| = 150 subsets (mean fitness: 61.60 ± 1.54)
- Performance plateaus and slightly decreases for larger collections
- Consistent performance across multiple runs, as indicated by small standard deviations
- Significantly lower fitness for |S| = 50 (48.10 ± 1.97), likely due to insufficient subsets

In conclusion, the genetic algorithm shows effective optimization behavior for the SCP, with performance peaking at a collection size of around 150 subsets. The algorithm consistently improves solutions over generations, with most improvement occurring in early generations. Future work could explore tuning algorithm parameters or alternative genetic operators to enhance performance, especially for more significant problem instances.

Figure 2

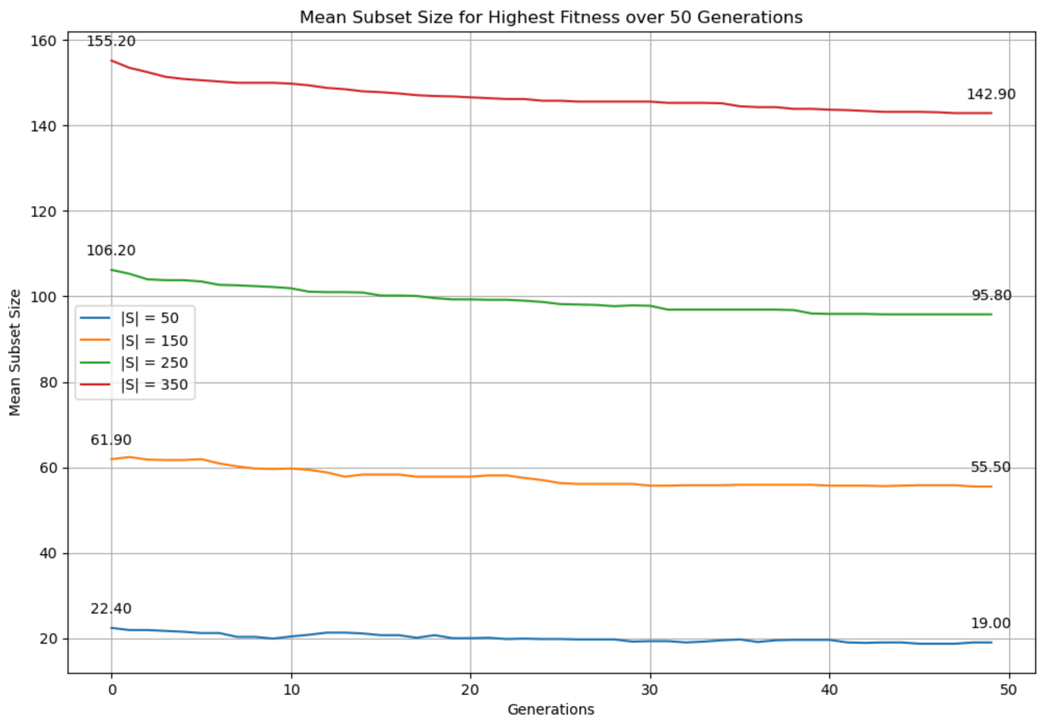


Figure 3: Mean Subset size over a generation

This graph shows how the mean subset size for the highest fitness solution changes over 50 generations for different collection sizes.

Observations:

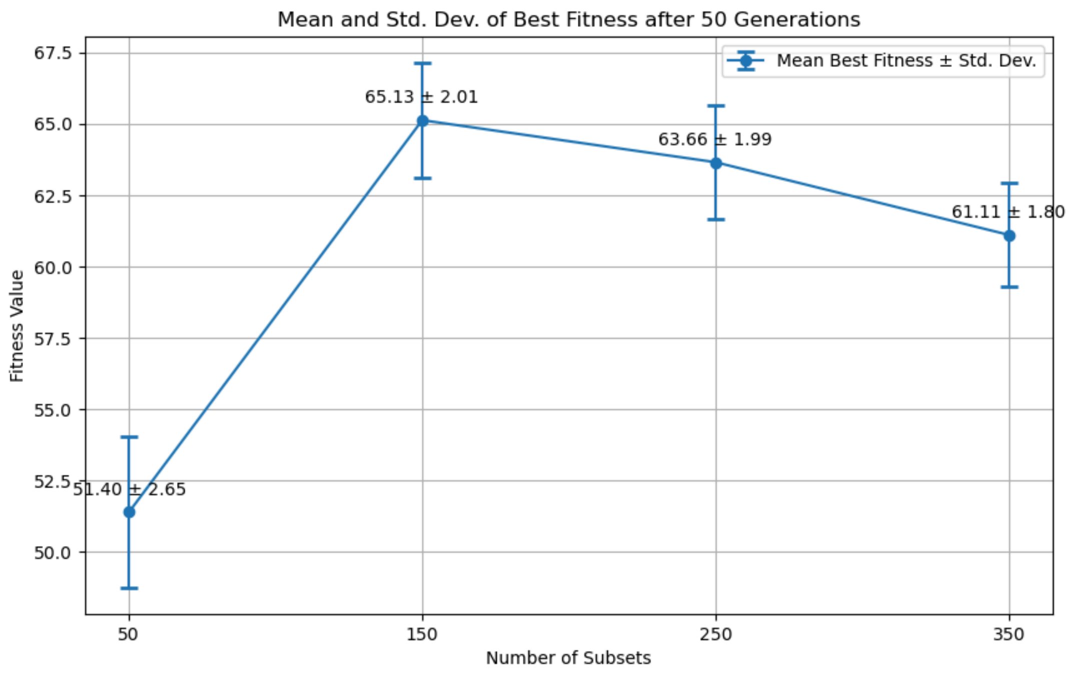
1. Larger collection sizes result in solutions with larger subset sizes
2. Mean subset size decreases over generations for all collection sizes
3. The improvement rate is fastest in early generations, slowing down later
4. Smaller collection sizes show more stability in subset size across generations

Question 2 :

Improving the Algorithm

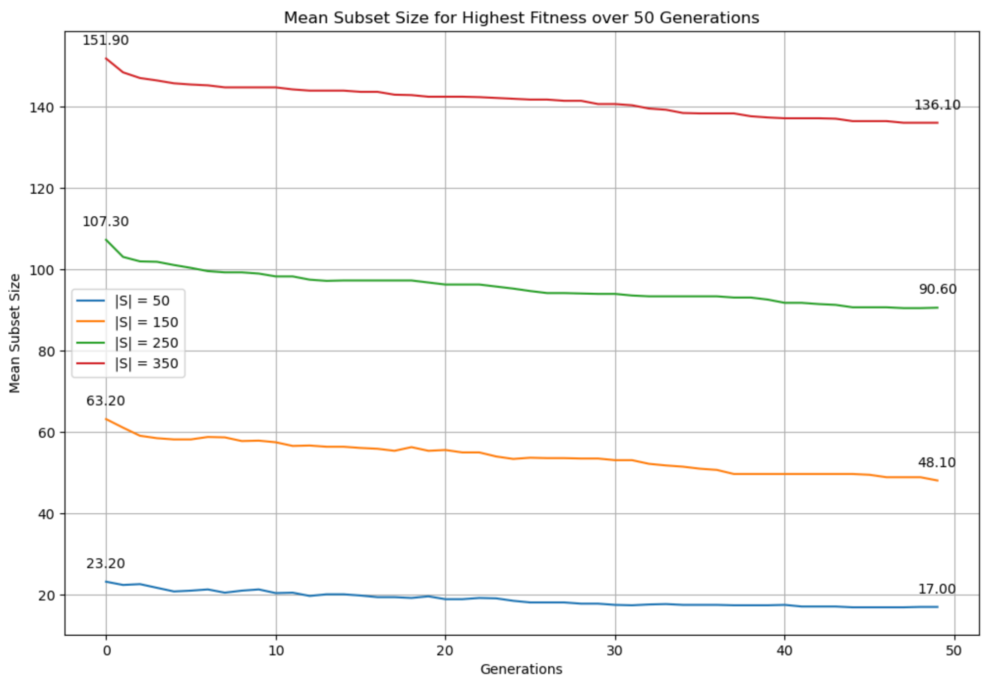
Inserting both Child

In genetic algorithms, the crossover operation is crucial for combining the genetic information of two parent solutions to produce new offspring. The modification to return two children instead of one enhances the diversity of the population, which can lead to more effective exploration of the solution space. By selecting a random crossover point, the algorithm creates two new individuals: one by combining the first segment of the first parent with the second segment of the second parent and the other by doing the reverse. This approach ensures that both offspring inherit different traits from each parent, increasing the likelihood of discovering optimal solutions. Such modifications can significantly improve the performance of genetic algorithms in solving complex optimization problems by maintaining a healthy balance between exploration and exploitation.



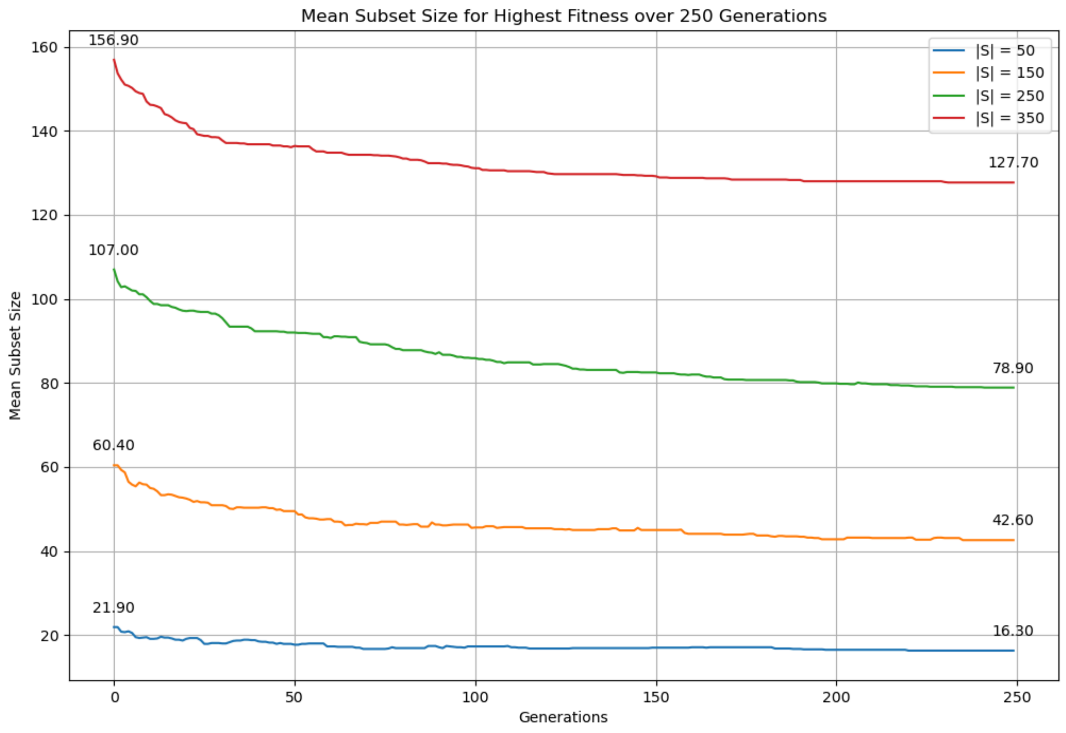
The modification to the genetic algorithm, which now returns two children instead of one during the crossover operation, has a 2.5% increase in the mean fitness values. The image shows that the mean best fitness value increased by three units after implementing the change. This enhancement demonstrates the effectiveness of the new crossover strategy in producing better solutions and improving the algorithm's overall performance. The graph clearly illustrates the positive impact of the modification, with higher mean fitness values indicating more optimal solutions.

Which reduced the number of subsets used by approx 5%.

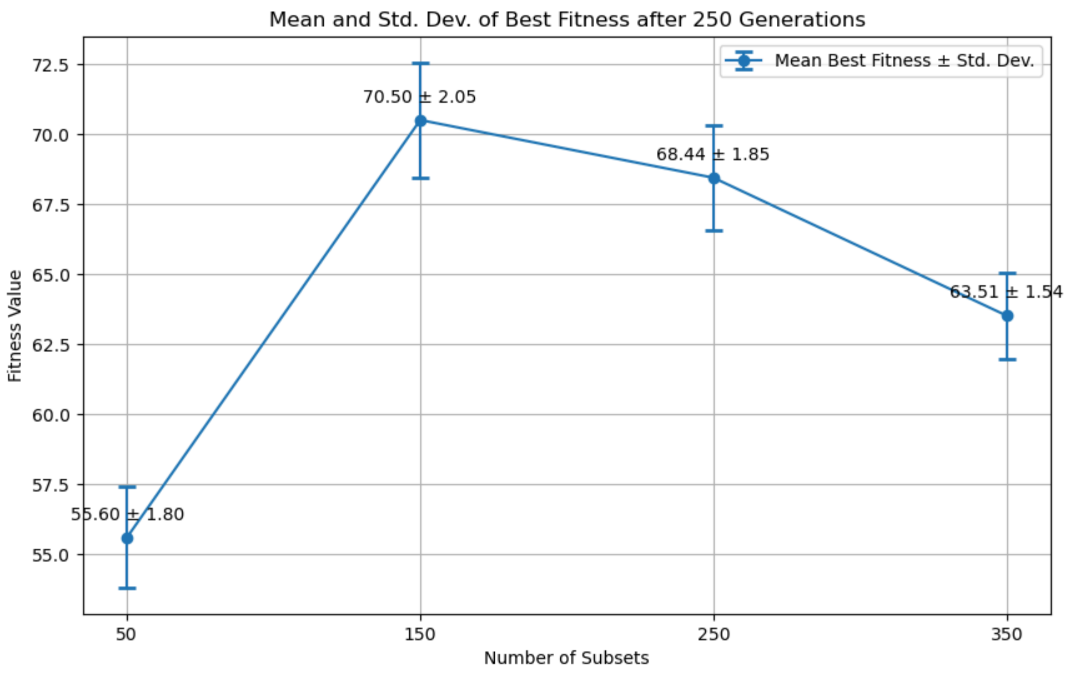


Changing Mutation Rate and Number of Generation

In genetic algorithms, the mutation rate is crucial for maintaining genetic diversity and exploring the solution space. Initially, a higher mutation rate, such as 0.2, encourages exploration by introducing more variability. This helps the algorithm discover new solutions and avoid local optima. As the algorithm progresses, gradually decreasing the mutation rate to around 0.01 shifts the focus from exploration to exploitation; also, to see this over significant observable distance, we are increasing generations from 50 to 250.



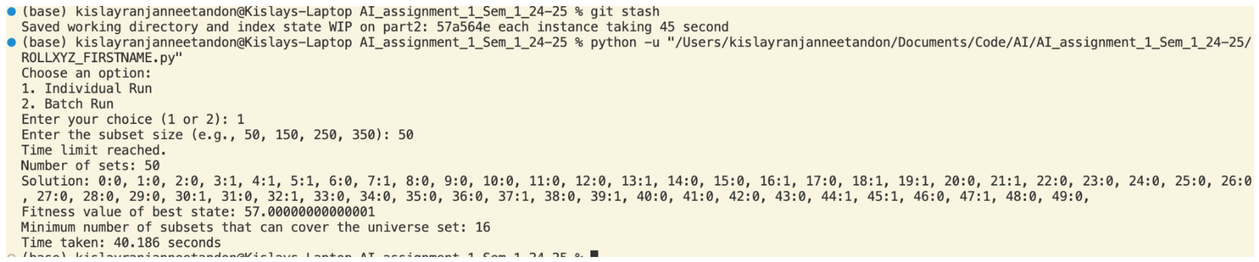
The new graph with 250 generations and a decreasing mutation rate from 0.3 to 0.01 shows significant changes compared to the earlier values observed over 50 generations. Initially, the mean subset sizes were 136 for 350 generations, 90 for 250 generations, 48 for 150 generations, and 18 for 50 generations. These values reflect the initial exploration phase with higher mutation rates, which introduced more variability and helped discover new solutions. With the increase to 250 generations, the mean subset sizes decreased to 127.7 for 350, 78.9 for 250, 42.6 for 150, and 16.3 for 50 generations. This represents percentage decreases of approximately 6.1%, 12.3%, 11.3%, and 9.4%, respectively. These changes indicate that the algorithm effectively shifted from exploration to exploitation, fine-tuning the solutions and preserving well-adapted individuals. The gradual decrease in the mutation rate allowed the algorithm to balance exploration and exploitation, improving optimization results. The extended generations provided a more significant observable distance, making the impact of different mutation rates more evident and resulting in better overall fitness optimization.



Also, there was a significant increase in mean fitness values by approx 2.5%.

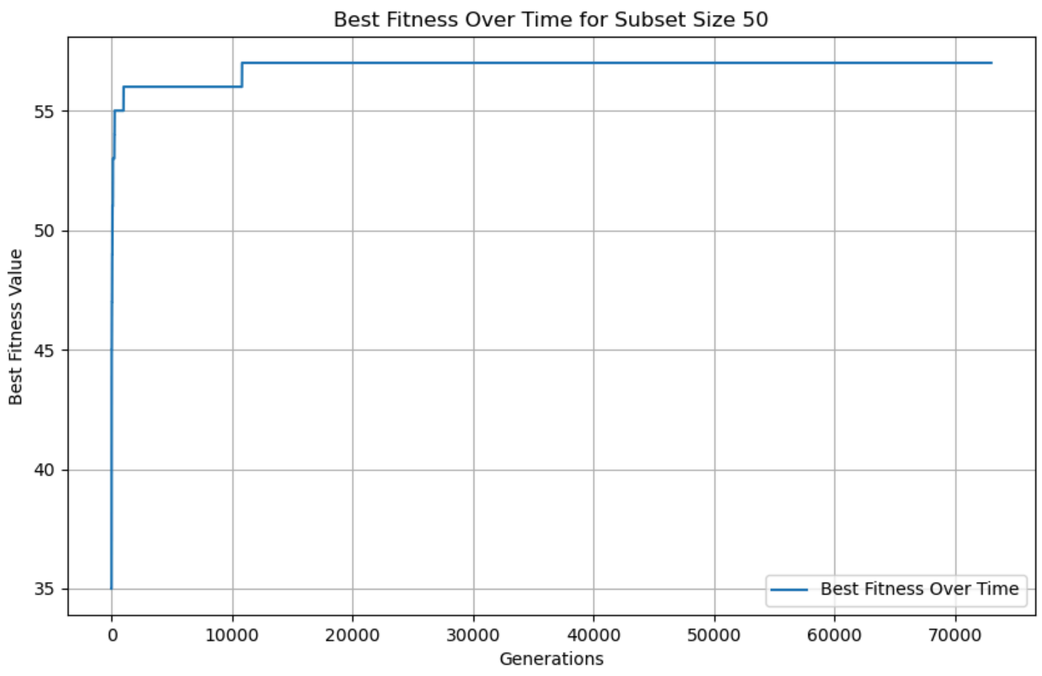
Utilizing a time limit of 40 seconds

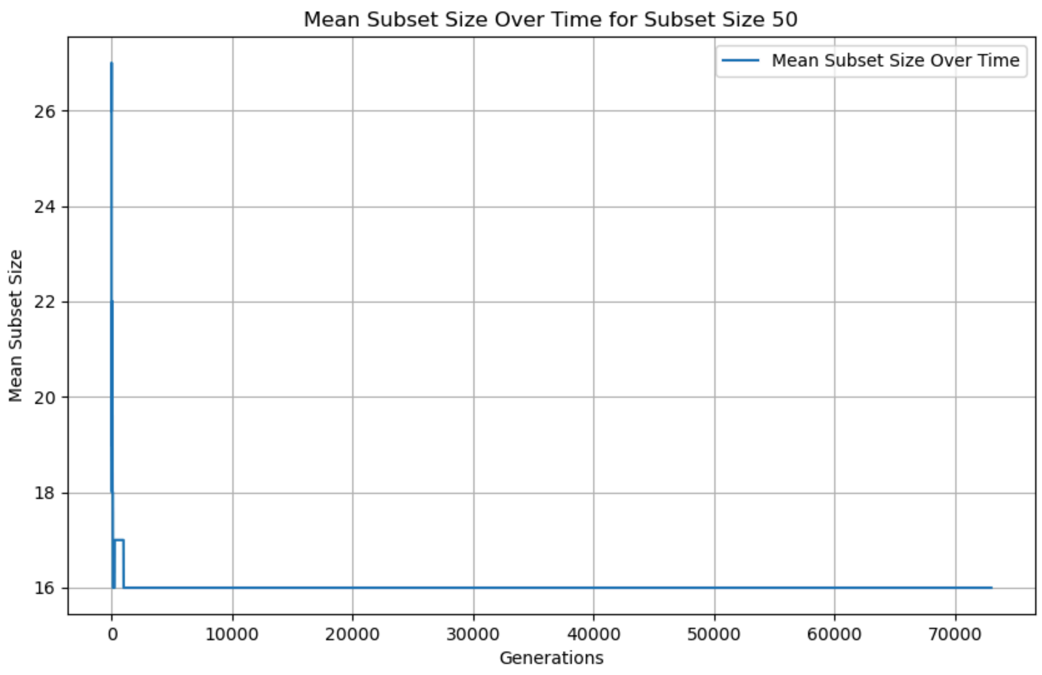
In the experiment, the genetic algorithm was modified to run each instance for a fixed duration of 40 seconds. This adjustment aimed to evaluate the impact of a time constraint on the algorithm's performance. It was observed that the 40-second runtime influenced the quality of the solutions. The following results were given for each of these subsets.

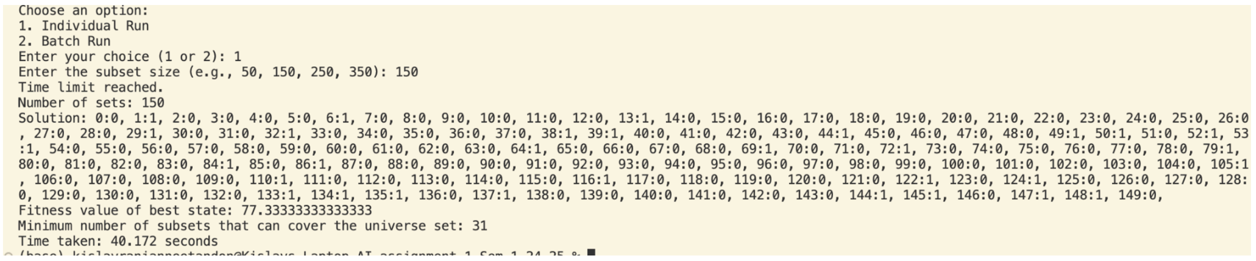


```
● (base) kislayranjanneetandon@Kislays-Laptop AI_assignment_1_Sem_1_24-25 % git stash
  Saved working directory and index state WIP on part2: 57a564e each instance taking 45 second
● (base) kislayranjanneetandon@Kislays-Laptop AI_assignment_1_Sem_1_24-25 % python -u "/Users/kislayranjanneetandon/Documents/Code/AI/AI_assignment_1_Sem_1_24-25/
  ROLLXYZ_FIRSTNAME.py"
  Choose an option:
  1. Individual Run
  2. Batch Run
  Enter your choice (1 or 2): 1
  Enter the subset size (e.g., 50, 150, 250, 350): 50
  Time limit reached.
  Number of sets: 50
  Solution: 0:0, 1:0, 2:0, 3:1, 4:1, 5:1, 6:0, 7:1, 8:0, 9:0, 10:0, 11:0, 12:0, 13:1, 14:0, 15:0, 16:1, 17:0, 18:1, 19:1, 20:0, 21:1, 22:0, 23:0, 24:0, 25:0, 26:0
  , 27:0, 28:0, 29:0, 30:1, 31:0, 32:1, 33:0, 34:0, 35:0, 36:0, 37:1, 38:0, 39:1, 40:0, 41:0, 42:0, 43:0, 44:1, 45:1, 46:0, 47:1, 48:0, 49:0,
  Fitness value of best state: 57.00000000000001
  Minimum number of subsets that can cover the universe set: 16
  Time taken: 40.186 seconds
```

Or a subset size of 50, introducing elitism did not significantly improve fitness values. The algorithm achieved satisfactory fitness levels within approximately 20,000 iterations. The minimal impact from elitism suggests that, in this case, the change from a fixed number of generations to allowing the algorithm to run for 40 seconds did not lead to noticeable enhancements. The results were consistent with what was previously observed, indicating that the benefits of elitism might not be as pronounced for smaller subset sizes.

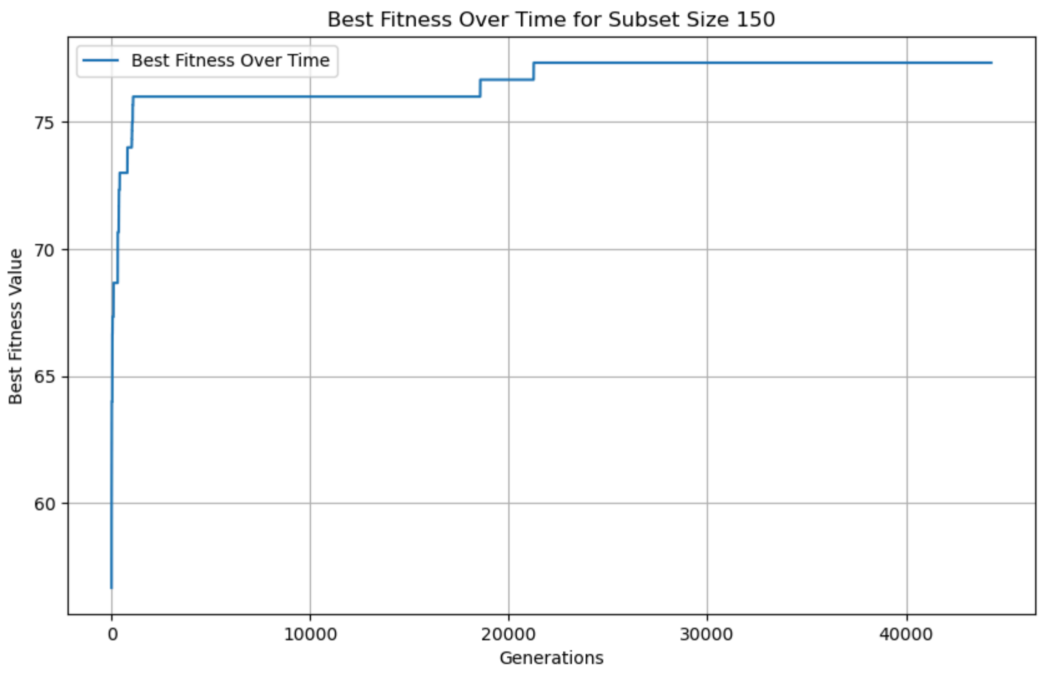


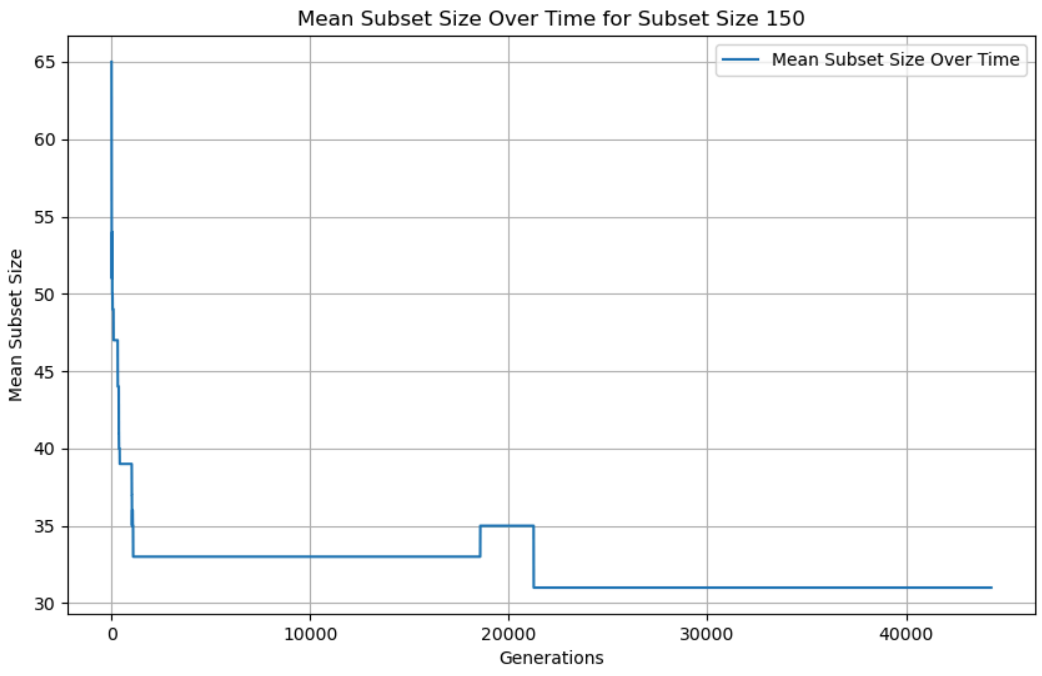


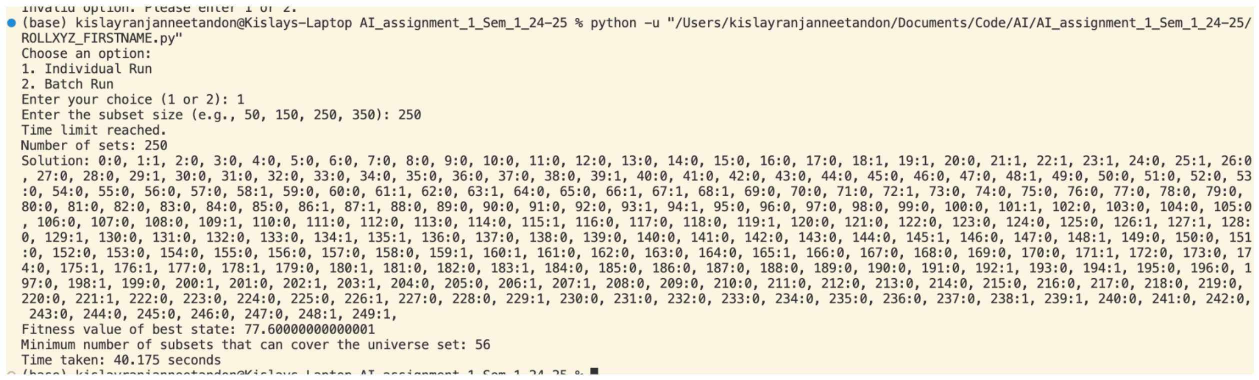


```
Choose an option:
1. Individual Run
2. Batch Run
Enter your choice (1 or 2): 1
Enter the subset size (e.g., 50, 150, 250, 350): 150
Time limit reached.
Number of sets: 150
Solution: 0:0, 1:1, 2:0, 3:0, 4:0, 5:0, 6:1, 7:0, 8:0, 9:0, 10:0, 11:0, 12:0, 13:1, 14:0, 15:0, 16:0, 17:0, 18:0, 19:0, 20:0, 21:0, 22:0, 23:0, 24:0, 25:0, 26:0
, 27:0, 28:0, 29:1, 30:0, 31:0, 32:1, 33:0, 34:0, 35:0, 36:0, 37:0, 38:1, 39:1, 40:0, 41:0, 42:0, 43:0, 44:1, 45:0, 46:0, 47:0, 48:0, 49:1, 50:1, 51:0, 52:1, 53
:1, 54:0, 55:0, 56:0, 57:0, 58:0, 59:0, 60:0, 61:0, 62:0, 63:0, 64:1, 65:0, 66:0, 67:0, 68:0, 69:1, 70:0, 71:0, 72:1, 73:0, 74:0, 75:0, 76:0, 77:0, 78:0, 79:1,
80:0, 81:0, 82:0, 83:0, 84:1, 85:0, 86:1, 87:0, 88:0, 89:0, 90:0, 91:0, 92:0, 93:0, 94:0, 95:0, 96:0, 97:0, 98:0, 99:0, 100:0, 101:0, 102:0, 103:0, 104:0, 105:1
, 106:0, 107:0, 108:0, 109:0, 110:1, 111:0, 112:0, 113:0, 114:0, 115:0, 116:1, 117:0, 118:0, 119:0, 120:0, 121:0, 122:1, 123:0, 124:1, 125:0, 126:0, 127:0, 128:
0, 129:0, 130:0, 131:0, 132:0, 133:1, 134:1, 135:1, 136:0, 137:1, 138:0, 139:0, 140:0, 141:0, 142:0, 143:0, 144:1, 145:1, 146:0, 147:1, 148:1, 149:0,
Fitness value of best state: 77.33333333333333
Minimum number of subsets that can cover the universe set: 31
Time taken: 40.172 seconds
```

In the case of a subset size of 150, the introduction of elitism had a substantial positive impact. The fitness value increased from 70 to 77, and the minimum subset size used improved from 42 to 31. These improvements were observed after extending the algorithm's runtime to 40 seconds rather than limiting it to a fixed number of generations. This extended runtime allowed the algorithm to explore more solutions and refine its performance, resulting in higher fitness values and more efficient subset usage.

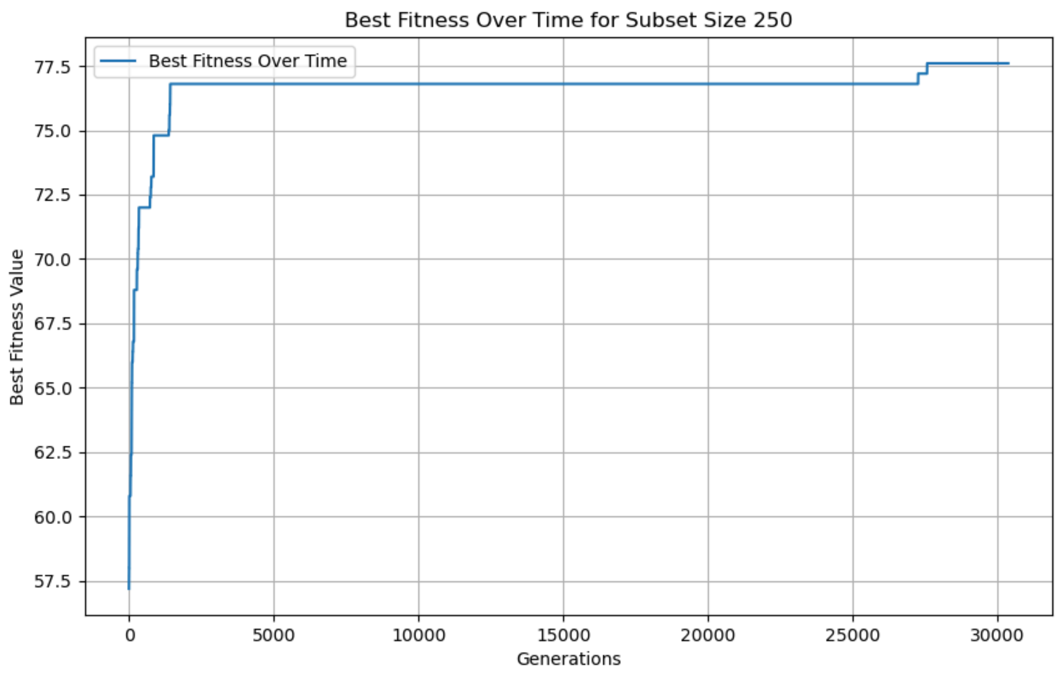

Best Fitness Over Time for Subset Size 150
Best Fitness Over Time
Best Fitness Value
Generations
60
65
70
75
0
10000
20000
30000
40000

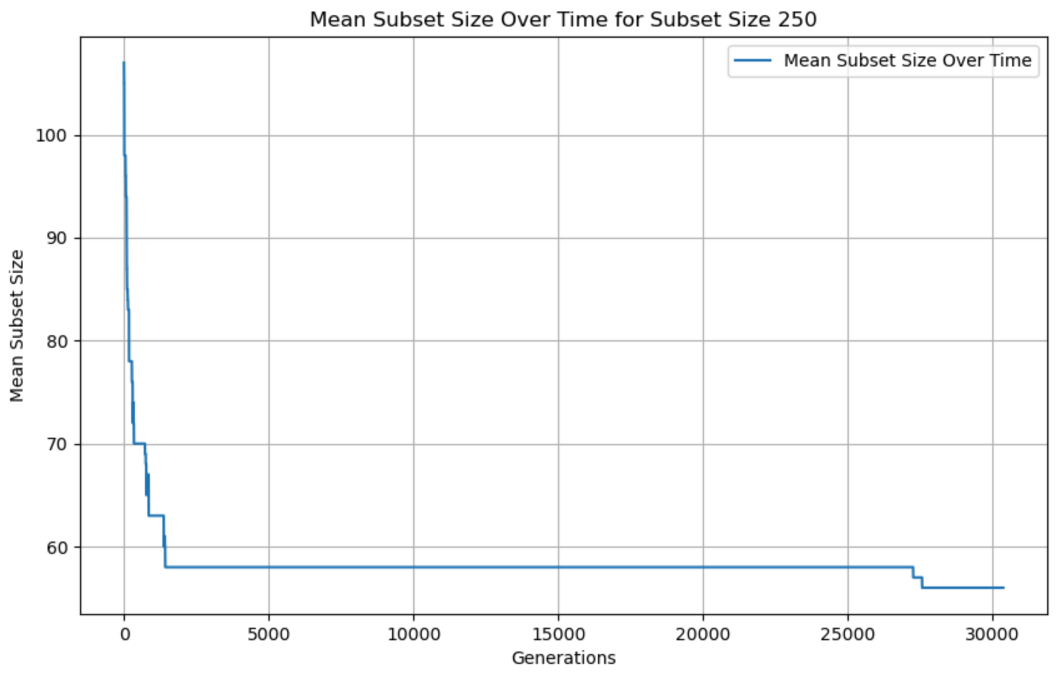

Mean Subset Size Over Time for Subset Size 150
Mean Subset Size Over Time
Mean Subset Size
Generations
30
35
40
45
50
55
60
65
0
10000
20000
30000
40000

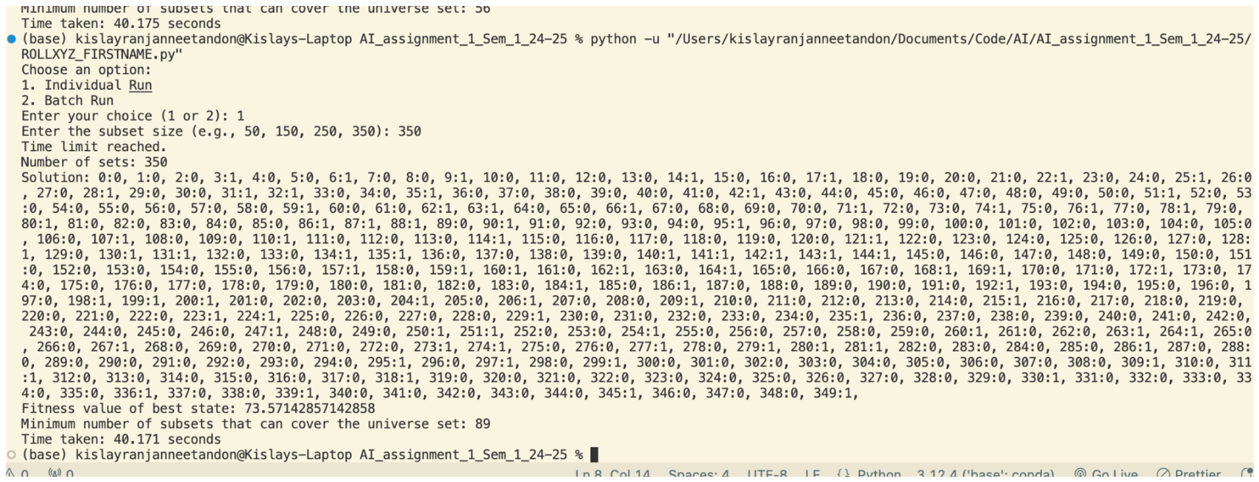


```
Invalid option. Please enter 1 or 2.
(base) kislayranjanneetandon@Kislays-Laptop AI_assignment_1_Sem_1_24-25 % python -u "/Users/kislayranjanneetandon/Documents/Code/AI/AI_assignment_1_Sem_1_24-25/
ROLLXYZ_FIRSTNAME.py"
Choose an option:
1. Individual Run
2. Batch Run
Enter your choice (1 or 2): 1
Enter the subset size (e.g., 50, 150, 250, 350): 250
Time limit reached.
Number of sets: 250
Solution: 0:0, 1:1, 2:0, 3:0, 4:0, 5:0, 6:0, 7:0, 8:0, 9:0, 10:0, 11:0, 12:0, 13:0, 14:0, 15:0, 16:0, 17:0, 18:1, 19:1, 20:0, 21:1, 22:1, 23:1, 24:0, 25:1, 26:0
, 27:0, 28:0, 29:1, 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, 45:0, 46:0, 47:0, 48:1, 49:0, 50:0, 51:0, 52:0, 53
:0, 54:0, 55:0, 56:0, 57:0, 58:1, 59:0, 60:0, 61:1, 62:0, 63:1, 64:0, 65:0, 66:1, 67:1, 68:1, 69:0, 70:0, 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, 85:0, 86:1, 87:1, 88:0, 89:0, 90:0, 91:0, 92:0, 93:1, 94:1, 95:0, 96:0, 97:0, 98:0, 99:0, 100:0, 101:1, 102:0, 103:0, 104:0, 105:0
, 106:0, 107:0, 108:0, 109:1, 110:0, 111:0, 112:0, 113:0, 114:0, 115:1, 116:0, 117:0, 118:0, 119:1, 120:0, 121:0, 122:0, 123:0, 124:0, 125:0, 126:1, 127:1, 128:
0, 129:1, 130:0, 131:0, 132:0, 133:0, 134:1, 135:1, 136:0, 137:0, 138:0, 139:0, 140:0, 141:0, 142:0, 143:0, 144:0, 145:1, 146:0, 147:0, 148:1, 149:0, 150:0, 151
:0, 152:0, 153:0, 154:0, 155:0, 156:0, 157:0, 158:0, 159:1, 160:1, 161:0, 162:0, 163:0, 164:0, 165:1, 166:0, 167:0, 168:0, 169:0, 170:0, 171:1, 172:0, 173:0, 17
4:0, 175:1, 176:1, 177:0, 178:1, 179:0, 180:1, 181:0, 182:0, 183:1, 184:0, 185:0, 186:0, 187:0, 188:0, 189:0, 190:0, 191:0, 192:1, 193:0, 194:1, 195:0, 196:0, 1
97:0, 198:1, 199:0, 200:1, 201:0, 202:1, 203:1, 204:0, 205:0, 206:1, 207:1, 208:0, 209:0, 210:0, 211:0, 212:0, 213:0, 214:0, 215:0, 216:0, 217:0, 218:0, 219:0,
220:0, 221:1, 222:0, 223:0, 224:0, 225:0, 226:1, 227:0, 228:0, 229:1, 230:0, 231:0, 232:0, 233:0, 234:0, 235:0, 236:0, 237:0, 238:1, 239:1, 240:0, 241:0, 242:0,
 243:0, 244:0, 245:0, 246:0, 247:0, 248:1, 249:1,
Fitness value of best state: 77.60000000000001
Minimum number of subsets that can cover the universe set: 56
Time taken: 40.175 seconds
(base) kislayranjanneetandon@Kislays-Laptop AI_assignment_1_Sem_1_24-25 %
```

For a subset size of 250, elitism demonstrated notable benefits. The fitness value rose from 68 to 77, and the mean subset size decreased from 80 to 56. The improvements became evident with the algorithm running for 40 seconds instead of a limited number of generations. This additional runtime enabled the algorithm to achieve better fitness values and optimize the subset size used more effectively, highlighting the advantage of allowing the algorithm to run longer.

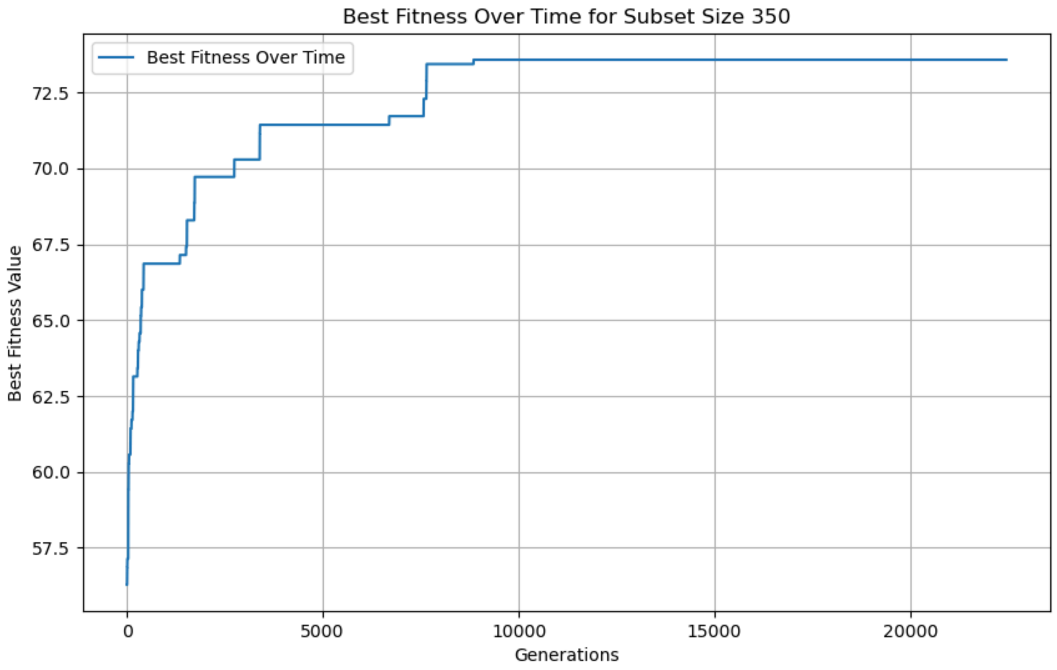


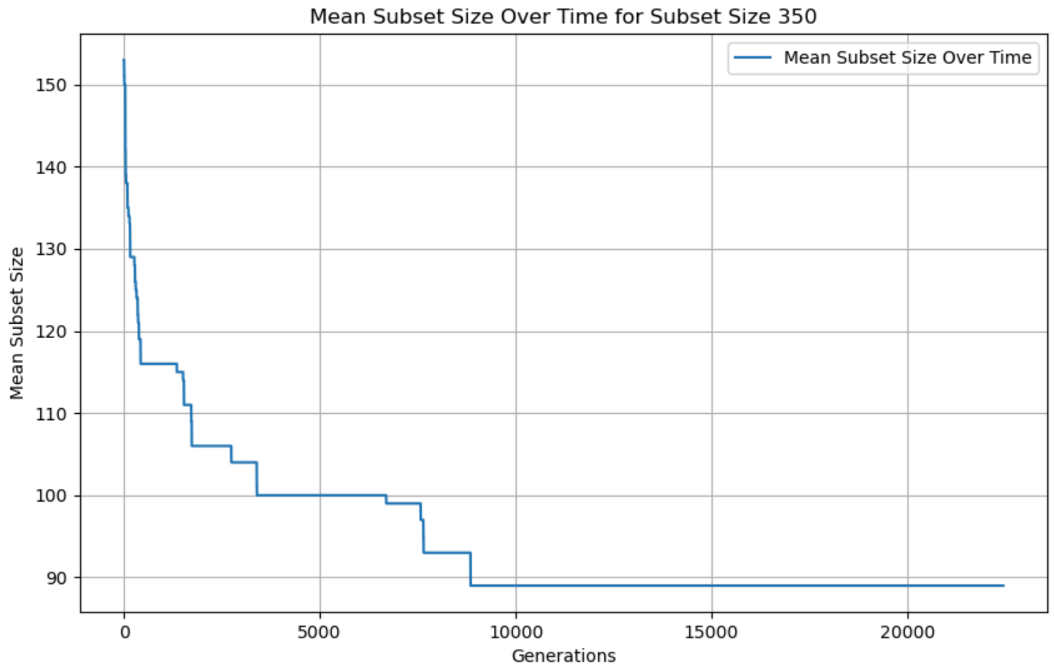




```
Minimum number of subsets that can cover the universe set: 56
Time taken: 40.175 seconds
● (base) kislayranjanneetandon@Kislays-Laptop AI_assignment_1_Sem_1_24-25 % python -u "/Users/kislayranjanneetandon/Documents/Code/AI/AI_assignment_1_Sem_1_24-25/
ROLLXYZ_FIRSTNAME.py"
Choose an option:
1. Individual Run
2. Batch Run
Enter your choice (1 or 2): 1
Enter the subset size (e.g., 50, 150, 250, 350): 350
Time limit reached.
Number of sets: 350
Solution: 0:0, 1:0, 2:0, 3:1, 4:0, 5:0, 6:1, 7:0, 8:0, 9:1, 10:0, 11:0, 12:0, 13:0, 14:1, 15:0, 16:0, 17:1, 18:0, 19:0, 20:0, 21:0, 22:1, 23:0, 24:0, 25:1, 26:0
, 27:0, 28:1, 29:0, 30:0, 31:1, 32:1, 33:0, 34:0, 35:1, 36:0, 37:0, 38:0, 39:0, 40:0, 41:0, 42:1, 43:0, 44:0, 45:0, 46:0, 47:0, 48:0, 49:0, 50:0, 51:1, 52:0, 53
:0, 54:0, 55:0, 56:0, 57:0, 58:0, 59:1, 60:0, 61:0, 62:1, 63:1, 64:0, 65:0, 66:1, 67:0, 68:0, 69:0, 70:0, 71:1, 72:0, 73:0, 74:1, 75:0, 76:1, 77:0, 78:1, 79:0,
80:1, 81:0, 82:0, 83:0, 84:0, 85:0, 86:1, 87:1, 88:1, 89:0, 90:1, 91:0, 92:0, 93:0, 94:0, 95:1, 96:0, 97:0, 98:0, 99:0, 100:0, 101:0, 102:0, 103:0, 104:0, 105:0
, 106:0, 107:1, 108:0, 109:0, 110:1, 111:0, 112:0, 113:0, 114:1, 115:0, 116:0, 117:0, 118:0, 119:0, 120:0, 121:1, 122:0, 123:0, 124:0, 125:0, 126:0, 127:0, 128:
1, 129:0, 130:1, 131:1, 132:0, 133:0, 134:1, 135:1, 136:0, 137:0, 138:0, 139:0, 140:1, 141:1, 142:1, 143:1, 144:1, 145:0, 146:0, 147:0, 148:0, 149:0, 150:0, 151
:0, 152:0, 153:0, 154:0, 155:0, 156:0, 157:1, 158:0, 159:1, 160:1, 161:0, 162:1, 163:0, 164:1, 165:0, 166:0, 167:0, 168:1, 169:1, 170:0, 171:0, 172:1, 173:0, 17
4:0, 175:0, 176:0, 177:0, 178:0, 179:0, 180:0, 181:0, 182:0, 183:0, 184:1, 185:0, 186:1, 187:0, 188:0, 189:0, 190:0, 191:0, 192:1, 193:0, 194:0, 195:0, 196:0, 1
97:0, 198:1, 199:1, 200:1, 201:0, 202:0, 203:0, 204:1, 205:0, 206:1, 207:0, 208:0, 209:1, 210:0, 211:0, 212:0, 213:0, 214:0, 215:1, 216:0, 217:0, 218:0, 219:0,
220:0, 221:0, 222:0, 223:1, 224:1, 225:0, 226:0, 227:0, 228:0, 229:1, 230:0, 231:0, 232:0, 233:0, 234:0, 235:1, 236:0, 237:0, 238:0, 239:0, 240:0, 241:0, 242:0,
 243:0, 244:0, 245:0, 246:0, 247:1, 248:0, 249:0, 250:1, 251:1, 252:0, 253:0, 254:1, 255:0, 256:0, 257:0, 258:0, 259:0, 260:1, 261:0, 262:0, 263:1, 264:1, 265:0
, 266:0, 267:1, 268:0, 269:0, 270:0, 271:0, 272:0, 273:1, 274:1, 275:0, 276:0, 277:1, 278:0, 279:1, 280:1, 281:1, 282:0, 283:0, 284:0, 285:0, 286:1, 287:0, 288:
0, 289:0, 290:0, 291:0, 292:0, 293:0, 294:0, 295:1, 296:0, 297:1, 298:0, 299:1, 300:0, 301:0, 302:0, 303:0, 304:0, 305:0, 306:0, 307:0, 308:0, 309:1, 310:0, 311
:1, 312:0, 313:0, 314:0, 315:0, 316:0, 317:0, 318:1, 319:0, 320:0, 321:0, 322:0, 323:0, 324:0, 325:0, 326:0, 327:0, 328:0, 329:0, 330:1, 331:0, 332:0, 333:0, 33
4:0, 335:0, 336:1, 337:0, 338:0, 339:1, 340:0, 341:0, 342:0, 343:0, 344:0, 345:1, 346:0, 347:0, 348:0, 349:1,
Fitness value of best state: 73.57142857142858
Minimum number of subsets that can cover the universe set: 89
Time taken: 40.171 seconds
○ (base) kislayranjanneetandon@Kislays-Laptop AI_assignment_1_Sem_1_24-25 % █
```

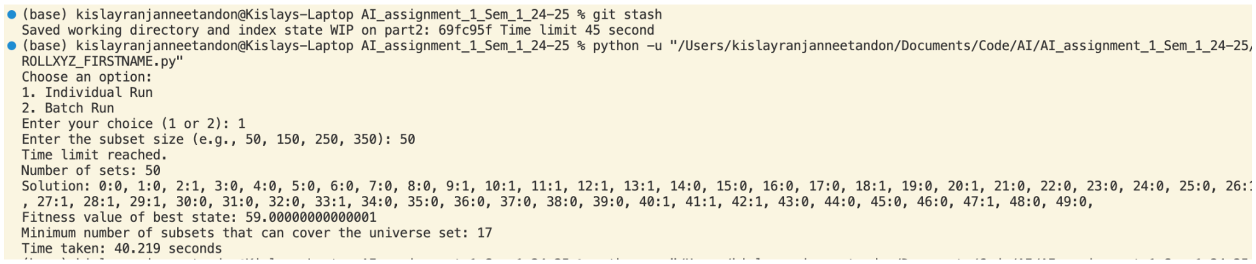
With a subset size of 350, the introduction of elitism led to significant improvements. The fitness value increased from 63 to 73, and the mean subset size used reduced from 130 to 89. These enhancements were realized by extending the algorithm’s runtime to 40 seconds, and this longer execution time allowed the algorithm to explore better and exploit potential solutions, resulting in higher fitness values and reduced subset sizes. The results underscore the effectiveness of allowing the algorithm more time to find optimal solutions.


Best Fitness Over Time for Subset Size 350
Best Fitness Over Time
Best Fitness Value
72.5
70.0
67.5
65.0
62.5
60.0
57.5
0
5000
10000
15000
20000
Generations


Mean Subset Size Over Time for Subset Size 350
Mean Subset Size Over Time
Mean Subset Size
150
140
130
120
110
100
90
0
5000
10000
15000
20000
Generations

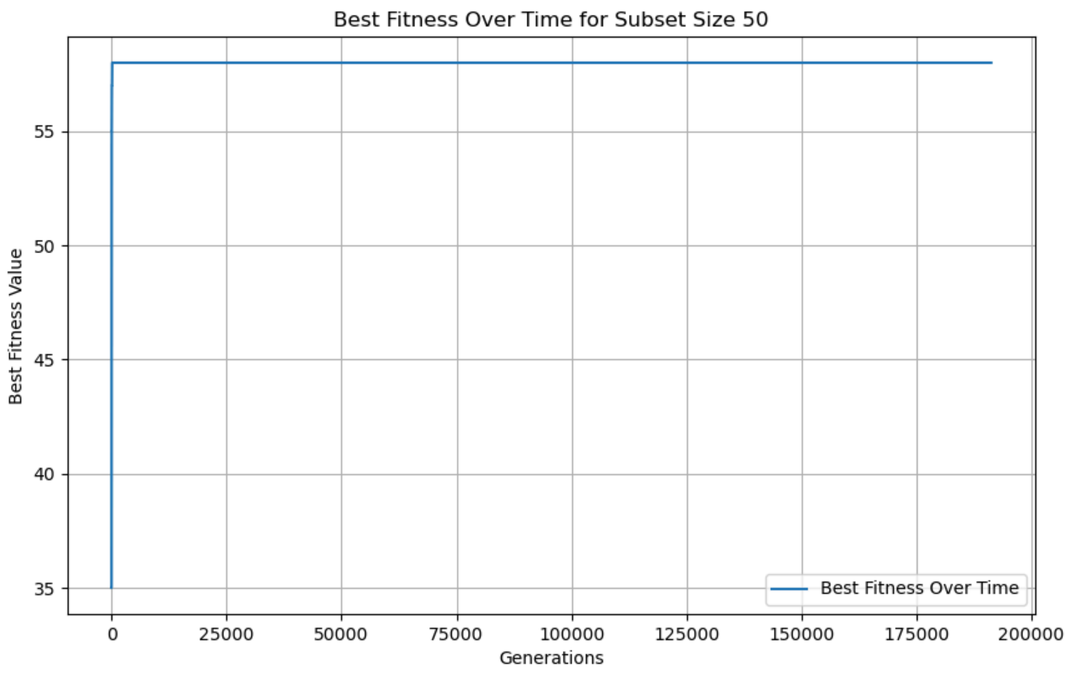
Introducing Elitism

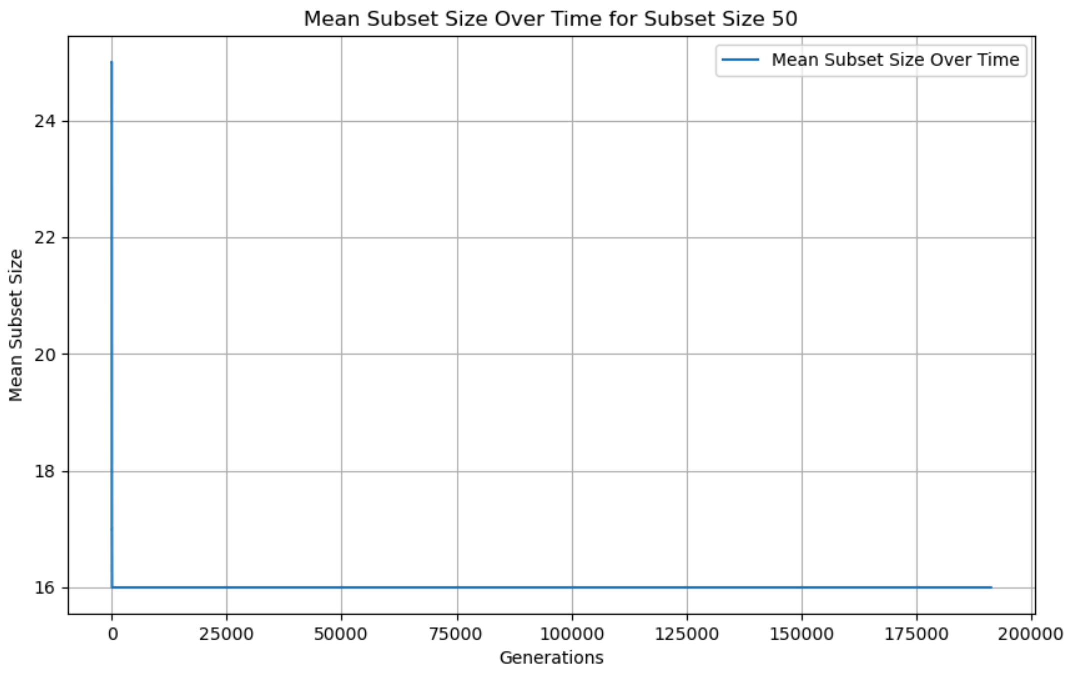
Elitism is a technique used in genetic algorithms to preserve the best-performing solutions from one generation to the next. By ensuring that the best individuals (or elites) are carried over unchanged, elitism helps maintain high-quality solutions and improves convergence rates. This approach can prevent the loss of optimal or near-optimal solutions due to the randomness of crossover and mutation processes. Introducing elitism involves selecting a portion of the population with the highest fitness and directly passing it to the next generation. In contrast, the rest undergoes genetic operations like crossover and mutation.

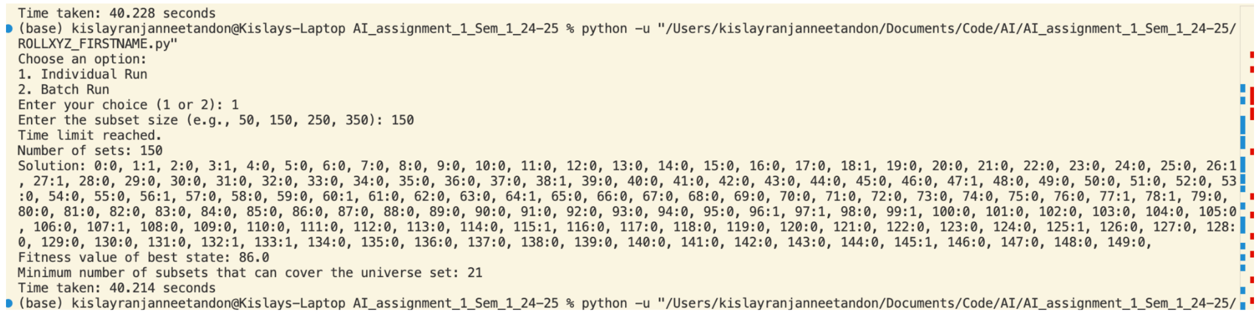


```
● (base) kislayranjanneetandon@Kislays-Laptop AI_assignment_1_Sem_1_24-25 % git stash
  Saved working directory and index state WIP on part2: 69fc95f Time limit 45 second
● (base) kislayranjanneetandon@Kislays-Laptop AI_assignment_1_Sem_1_24-25 % python -u "/Users/kislayranjanneetandon/Documents/Code/AI/AI_assignment_1_Sem_1_24-25/
  ROLLXYZ_FIRSTNAME.py"
  Choose an option:
  1. Individual Run
  2. Batch Run
  Enter your choice (1 or 2): 1
  Enter the subset size (e.g., 50, 150, 250, 350): 50
  Time limit reached.
  Number of sets: 50
  Solution: 0:0, 1:0, 2:1, 3:0, 4:0, 5:0, 6:0, 7:0, 8:0, 9:1, 10:1, 11:1, 12:1, 13:1, 14:0, 15:0, 16:0, 17:0, 18:1, 19:0, 20:1, 21:0, 22:0, 23:0, 24:0, 25:0, 26:
  , 27:1, 28:1, 29:1, 30:0, 31:0, 32:0, 33:1, 34:0, 35:0, 36:0, 37:0, 38:0, 39:0, 40:1, 41:1, 42:1, 43:0, 44:0, 45:0, 46:0, 47:1, 48:0, 49:0,
  Fitness value of best state: 59.00000000000001
  Minimum number of subsets that can cover the universe set: 17
  Time taken: 40.219 seconds
```

After implementing elitism, the fitness value for a subset size 50 improved to 58, and the subset size used decreased to 16. This represents a significant enhancement compared to previous results. The introduction of elitism appears to have contributed to a more effective selection of high-quality solutions, resulting in better fitness and a more efficient subset size.

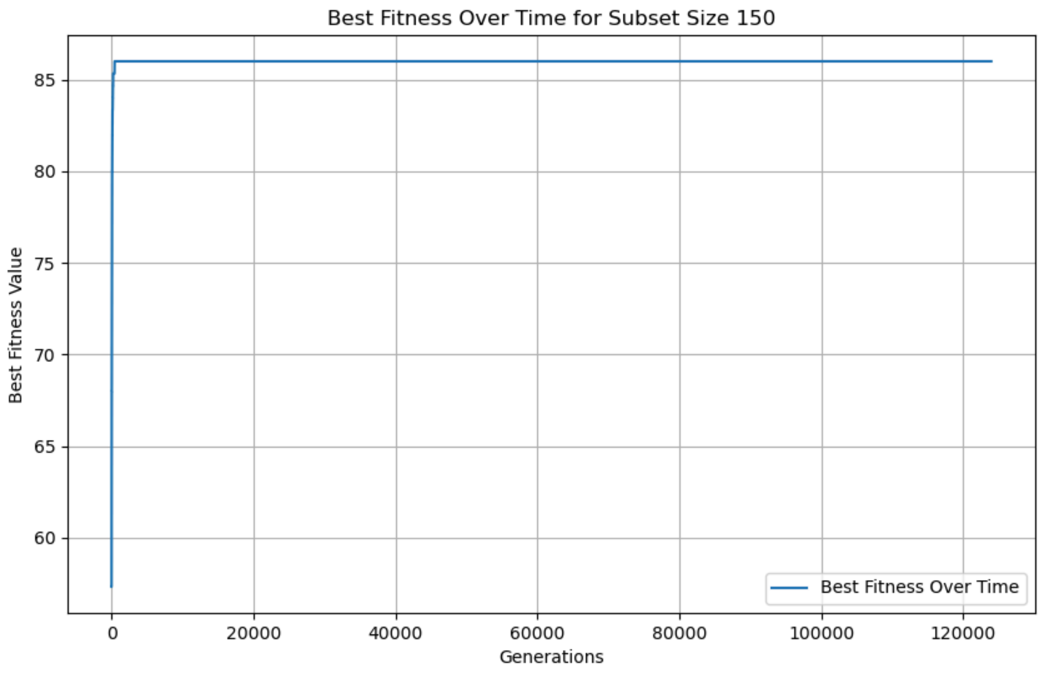

Best Fitness Over Time for Subset Size 50
Best Fitness Value
Generations
Best Fitness Over Time
35
40
45
50
55
0
25000
50000
75000
100000
125000
150000
175000
200000

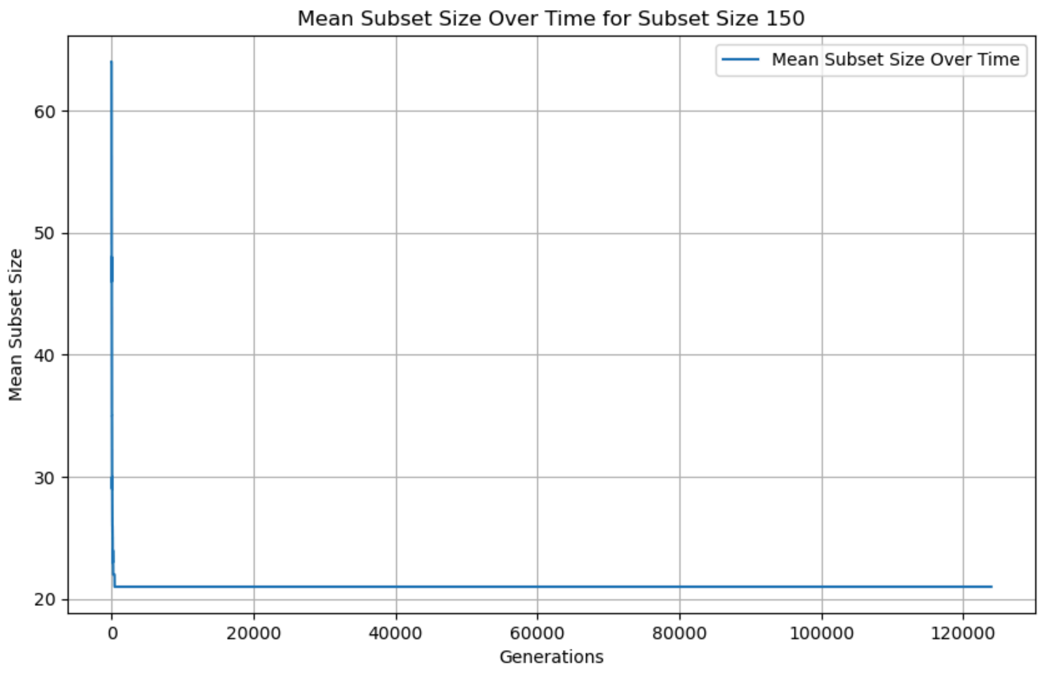

Mean Subset Size Over Time for Subset Size 50
Mean Subset Size
Generations
Mean Subset Size Over Time
16
18
20
22
24
0
25000
50000
75000
100000
125000
150000
175000
200000

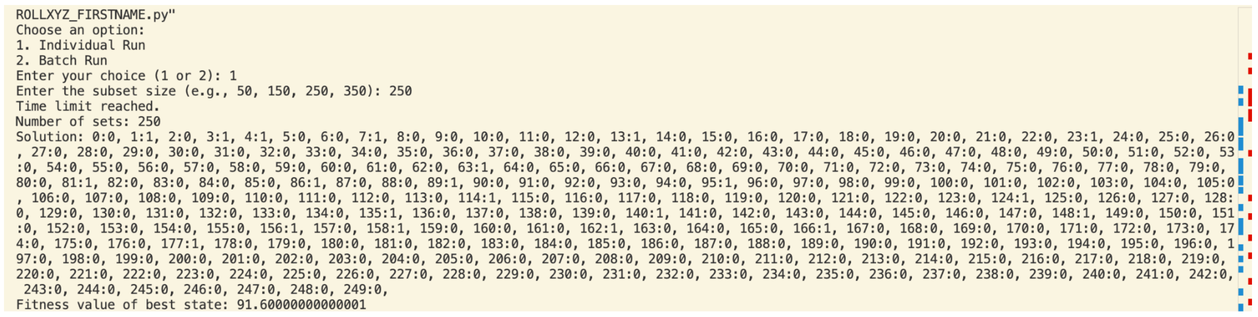


```
Time taken: 40.228 seconds
(base) kislayranjanneetandon@Kislays-Laptop AI_assignment_1_Sem_1_24-25 % python -u "/Users/kislayranjanneetandon/Documents/Code/AI/AI_assignment_1_Sem_1_24-25/
ROLLXYZ_FIRSTNAME.py"
Choose an option:
1. Individual Run
2. Batch Run
Enter your choice (1 or 2): 1
Enter the subset size (e.g., 50, 150, 250, 350): 150
Time limit reached.
Number of sets: 150
Solution: 0:0, 1:1, 2:0, 3:1, 4:0, 5:0, 6:0, 7:0, 8:0, 9:0, 10:0, 11:0, 12:0, 13:0, 14:0, 15:0, 16:0, 17:0, 18:1, 19:0, 20:0, 21:0, 22:0, 23:0, 24:0, 25:0, 26:1
, 27:1, 28:0, 29:0, 30:0, 31:0, 32:0, 33:0, 34:0, 35:0, 36:0, 37:0, 38:1, 39:0, 40:0, 41:0, 42:0, 43:0, 44:0, 45:0, 46:0, 47:1, 48:0, 49:0, 50:0, 51:0, 52:0, 53
:0, 54:0, 55:0, 56:1, 57:0, 58:0, 59:0, 60:1, 61:0, 62:0, 63:0, 64:1, 65:0, 66:0, 67:0, 68:0, 69:0, 70:0, 71:0, 72:0, 73:0, 74:0, 75:0, 76:0, 77:1, 78:1, 79:0,
80:0, 81:0, 82:0, 83:0, 84:0, 85:0, 86:0, 87:0, 88:0, 89:0, 90:0, 91:0, 92:0, 93:0, 94:0, 95:0, 96:1, 97:1, 98:0, 99:1, 100:0, 101:0, 102:0, 103:0, 104:0, 105:0
, 106:0, 107:1, 108:0, 109:0, 110:0, 111:0, 112:0, 113:0, 114:0, 115:1, 116:0, 117:0, 118:0, 119:0, 120:0, 121:0, 122:0, 123:0, 124:0, 125:1, 126:0, 127:0, 128:
0, 129:0, 130:0, 131:0, 132:1, 133:1, 134:0, 135:0, 136:0, 137:0, 138:0, 139:0, 140:0, 141:0, 142:0, 143:0, 144:0, 145:1, 146:0, 147:0, 148:0, 149:0,
Fitness value of best state: 86.0
Minimum number of subsets that can cover the universe set: 21
Time taken: 40.214 seconds
(base) kislayranjanneetandon@Kislays-Laptop AI_assignment_1_Sem_1_24-25 % python -u "/Users/kislayranjanneetandon/Documents/Code/AI/AI_assignment_1_Sem_1_24-25/
```

For a subset size of 150, the new fitness value increased to 86, and the subset size used was reduced to 21. These results show a notable improvement over earlier performance. Elitism seems to have significantly enhanced the algorithm's capability to optimize both fitness and subset efficiency, reflecting a more refined approach to solution selection.

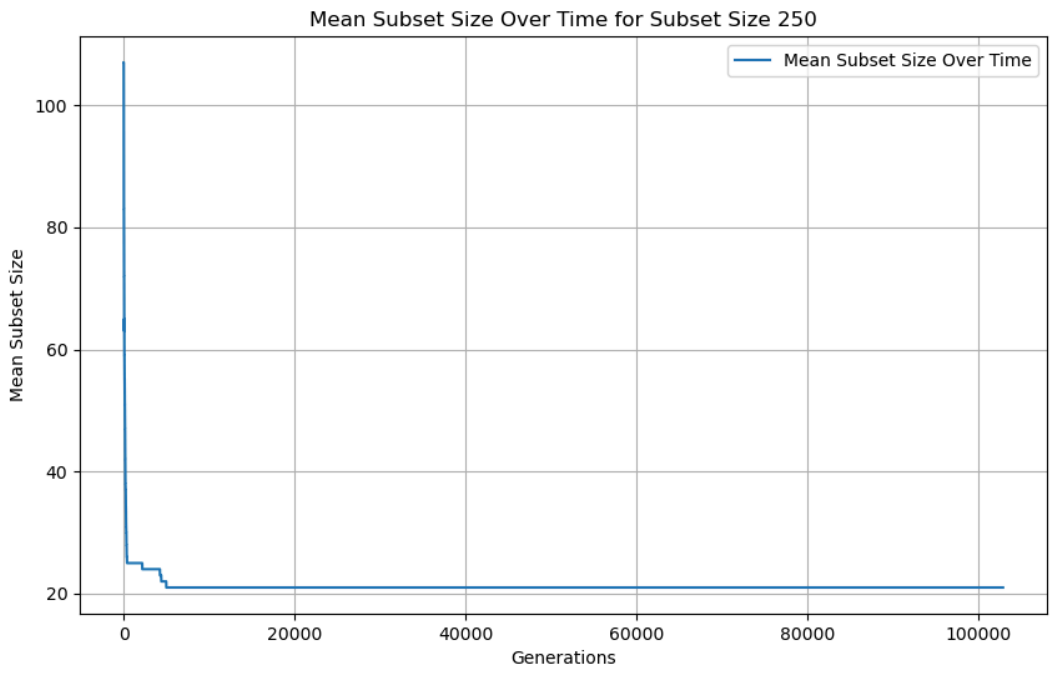


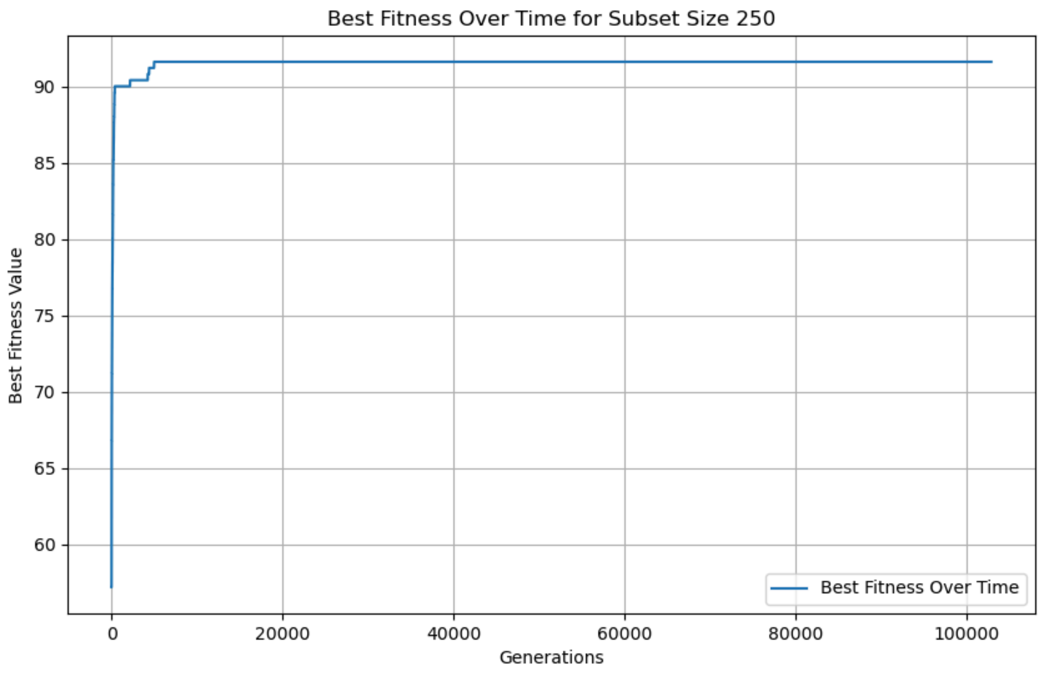


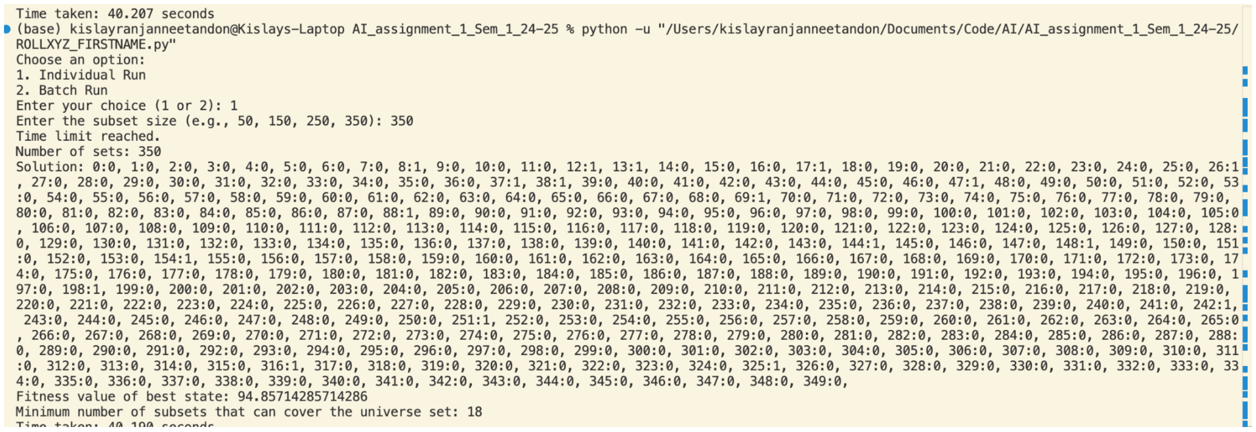


```
ROLLXYZ_FIRSTNAME.py"
Choose an option:
1. Individual Run
2. Batch Run
Enter your choice (1 or 2): 1
Enter the subset size (e.g., 50, 150, 250, 350): 250
Time limit reached.
Number of sets: 250
Solution: 0:0, 1:1, 2:0, 3:1, 4:1, 5:0, 6:0, 7:1, 8:0, 9:0, 10:0, 11:0, 12:0, 13:1, 14:0, 15:0, 16:0, 17:0, 18:0, 19:0, 20:0, 21:0, 22:0, 23:1, 24:0, 25:0, 26:0
, 27:0, 28:0, 29:0, 30:0, 31:0, 32:0, 33:0, 34:0, 35:0, 36:0, 37:0, 38:0, 39:0, 40:0, 41:0, 42:0, 43:0, 44:0, 45:0, 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, 62:0, 63:1, 64:0, 65:0, 66:0, 67:0, 68:0, 69:0, 70:0, 71:0, 72:0, 73:0, 74:0, 75:0, 76:0, 77:0, 78:0, 79:0,
80:0, 81:1, 82:0, 83:0, 84:0, 85:0, 86:1, 87:0, 88:0, 89:1, 90:0, 91:0, 92:0, 93:0, 94:0, 95:1, 96:0, 97:0, 98:0, 99:0, 100:0, 101:0, 102:0, 103:0, 104:0, 105:0
, 106:0, 107:0, 108:0, 109:0, 110:0, 111:0, 112:0, 113:0, 114:1, 115:0, 116:0, 117:0, 118:0, 119:0, 120:0, 121:0, 122:0, 123:0, 124:1, 125:0, 126:0, 127:0, 128:
0, 129:0, 130:0, 131:0, 132:0, 133:0, 134:0, 135:1, 136:0, 137:0, 138:0, 139:0, 140:1, 141:0, 142:0, 143:0, 144:0, 145:0, 146:0, 147:0, 148:1, 149:0, 150:0, 151
:0, 152:0, 153:0, 154:0, 155:0, 156:1, 157:0, 158:1, 159:0, 160:0, 161:0, 162:1, 163:0, 164:0, 165:0, 166:1, 167:0, 168:0, 169:0, 170:0, 171:0, 172:0, 173:0, 17
4:0, 175:0, 176:0, 177:1, 178:0, 179:0, 180:0, 181:0, 182:0, 183:0, 184:0, 185:0, 186:0, 187:0, 188:0, 189:0, 190:0, 191:0, 192:0, 193:0, 194:0, 195:0, 196:0, 1
97:0, 198:0, 199:0, 200:0, 201:0, 202:0, 203:0, 204:0, 205:0, 206:0, 207:0, 208:0, 209:0, 210:0, 211:0, 212:0, 213:0, 214:0, 215:0, 216:0, 217:0, 218:0, 219:0,
220:0, 221:0, 222:0, 223:0, 224:0, 225:0, 226:0, 227:0, 228:0, 229:0, 230:0, 231:0, 232:0, 233:0, 234:0, 235:0, 236:0, 237:0, 238:0, 239:0, 240:0, 241:0, 242:0,
 243:0, 244:0, 245:0, 246:0, 247:0, 248:0, 249:0,
Fitness value of best state: 91.60000000000001
```

With a subset size of 250, the fitness value rose to 91, while the subset size used remained at 21. The improved fitness value indicates a more effective exploration and exploitation of potential solutions, thanks to the inclusion of elitism. The results suggest that elitism has driven the algorithm toward higher fitness values while maintaining a consistent subset size.

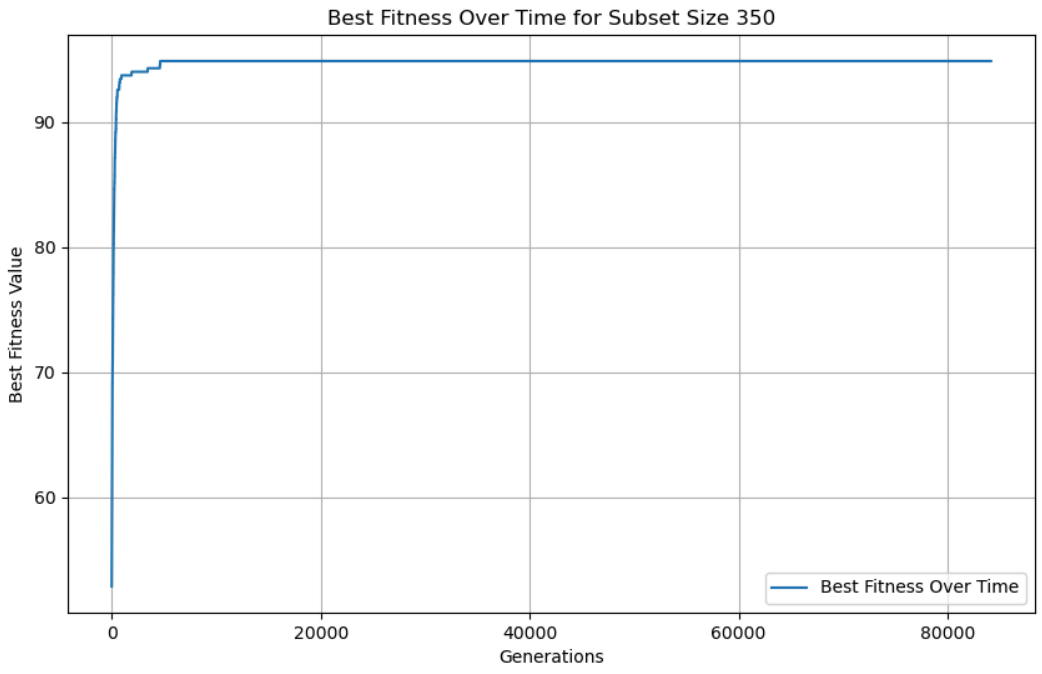

Mean Subset Size Over Time for Subset Size 250
Mean Subset Size Over Time
Mean Subset Size
Generations
100
80
60
40
20
0
20000
40000
60000
80000
100000

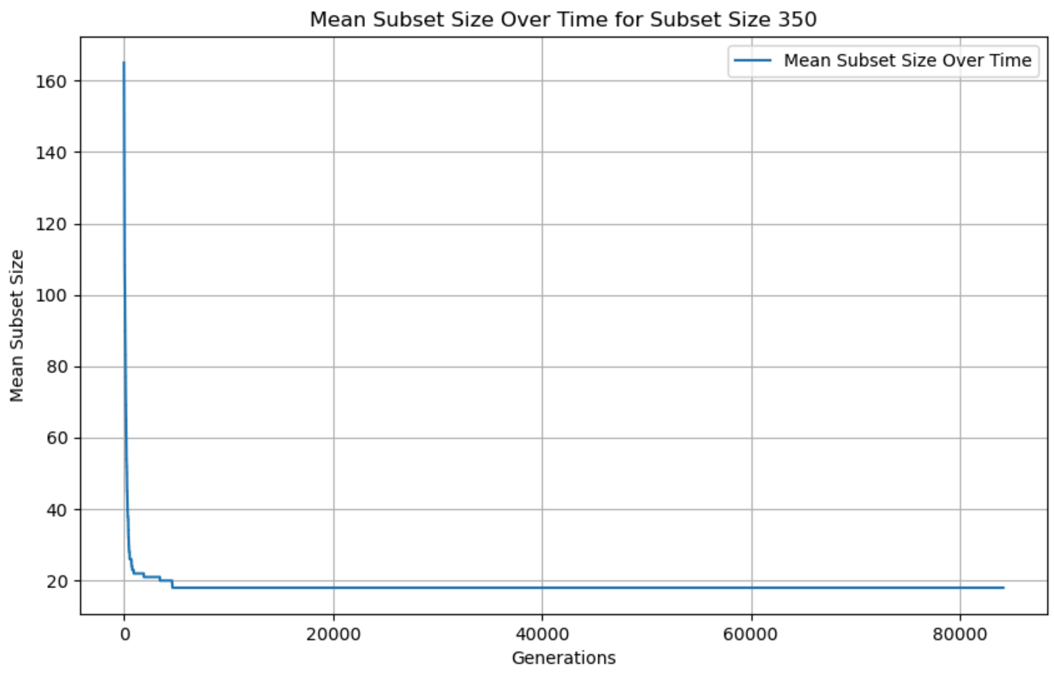

Best Fitness Over Time for Subset Size 250
Best Fitness Over Time
Best Fitness Value
Generations
90
85
80
75
70
65
60
0
20000
40000
60000
80000
100000



```
Time taken: 40.207 seconds
(base) kislayranjanneetandon@Kislays-Laptop AI_assignment_1_Sem_1_24-25 % python -u "/Users/kislayranjanneetandon/Documents/Code/AI/AI_assignment_1_Sem_1_24-25/
ROLLXYZ_FIRSTNAME.py"
Choose an option:
1. Individual Run
2. Batch Run
Enter your choice (1 or 2): 1
Enter the subset size (e.g., 50, 150, 250, 350): 350
Time limit reached.
Number of sets: 350
Solution: 0:0, 1:0, 2:0, 3:0, 4:0, 5:0, 6:0, 7:0, 8:1, 9:0, 10:0, 11:0, 12:1, 13:1, 14:0, 15:0, 16:0, 17:1, 18:0, 19:0, 20:0, 21:0, 22:0, 23:0, 24:0, 25:0, 26:1
, 27:0, 28:0, 29:0, 30:0, 31:0, 32:0, 33:0, 34:0, 35:0, 36:0, 37:1, 38:1, 39:0, 40:0, 41:0, 42:0, 43:0, 44:0, 45:0, 46:0, 47:1, 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, 62:0, 63:0, 64:0, 65:0, 66:0, 67:0, 68:0, 69:1, 70:0, 71:0, 72:0, 73:0, 74:0, 75:0, 76:0, 77:0, 78:0, 79:0,
80:0, 81:0, 82:0, 83:0, 84:0, 85:0, 86:0, 87:0, 88:1, 89:0, 90:0, 91:0, 92:0, 93:0, 94:0, 95:0, 96:0, 97:0, 98:0, 99:0, 100:0, 101:0, 102:0, 103:0, 104:0, 105:0
, 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:
0, 129:0, 130:0, 131:0, 132:0, 133:0, 134:0, 135:0, 136:0, 137:0, 138:0, 139:0, 140:0, 141:0, 142:0, 143:0, 144:1, 145:0, 146:0, 147:0, 148:1, 149:0, 150:0, 151
:0, 152:0, 153:0, 154:1, 155:0, 156:0, 157:0, 158:0, 159:0, 160:0, 161:0, 162:0, 163:0, 164:0, 165:0, 166:0, 167:0, 168:0, 169:0, 170:0, 171:0, 172:0, 173:0, 17
4:0, 175:0, 176:0, 177:0, 178:0, 179:0, 180:0, 181:0, 182:0, 183:0, 184:0, 185:0, 186:0, 187:0, 188:0, 189:0, 190:0, 191:0, 192:0, 193:0, 194:0, 195:0, 196:0, 1
97:0, 198:1, 199:0, 200:0, 201:0, 202:0, 203:0, 204:0, 205:0, 206:0, 207:0, 208:0, 209:0, 210:0, 211:0, 212:0, 213:0, 214:0, 215:0, 216:0, 217:0, 218:0, 219:0,
220:0, 221:0, 222:0, 223:0, 224:0, 225:0, 226:0, 227:0, 228:0, 229:0, 230:0, 231:0, 232:0, 233:0, 234:0, 235:0, 236:0, 237:0, 238:0, 239:0, 240:0, 241:0, 242:1,
 243:0, 244:0, 245:0, 246:0, 247:0, 248:0, 249:0, 250:0, 251:1, 252:0, 253:0, 254:0, 255:0, 256:0, 257:0, 258:0, 259:0, 260:0, 261:0, 262:0, 263:0, 264:0, 265:0
, 266:0, 267:0, 268:0, 269:0, 270:0, 271:0, 272:0, 273:0, 274:0, 275:0, 276:0, 277:0, 278:0, 279:0, 280:0, 281:0, 282:0, 283:0, 284:0, 285:0, 286:0, 287:0, 288:
0, 289:0, 290:0, 291:0, 292:0, 293:0, 294:0, 295:0, 296:0, 297:0, 298:0, 299:0, 300:0, 301:0, 302:0, 303:0, 304:0, 305:0, 306:0, 307:0, 308:0, 309:0, 310:0, 311
:0, 312:0, 313:0, 314:0, 315:0, 316:1, 317:0, 318:0, 319:0, 320:0, 321:0, 322:0, 323:0, 324:0, 325:1, 326:0, 327:0, 328:0, 329:0, 330:0, 331:0, 332:0, 333:0, 33
4:0, 335:0, 336:0, 337:0, 338:0, 339:0, 340:0, 341:0, 342:0, 343:0, 344:0, 345:0, 346:0, 347:0, 348:0, 349:0,
Fitness value of best state: 94.85714285714286
Minimum number of subsets that can cover the universe set: 18
Time taken: 40.190 seconds
```

For a subset size of 350, the new fitness value reached 95, and the subset size used decreased to 18. This significant improvement highlights the effectiveness of elitism in enhancing the genetic algorithm’s performance. The increased fitness value and reduced subset size demonstrate the benefits of elitism in optimizing solution quality and efficiency.





The study revealed that introducing elitism into the genetic algorithm significantly improved solution quality across various subset sizes. Specifically, elitism enhanced fitness values and reduced subset sizes for subset sizes of 50, 150, 250, and 350, demonstrating its effectiveness in optimizing the algorithm's performance.

For subset sizes of 50, 150, 250, and 350, the algorithm reached stable states at approximately 20,000 generations. This stability indicates that elitism improved the fitness values and subset sizes and contributed to efficient convergence. The consistent performance observed at this generation count across different subset sizes underscores the role of elitism in facilitating convergence and refining the selection process, thereby achieving high-quality solutions within a practical number of generations.

Conclusion

The genetic algorithm effectively solves the Set Covering Problem (SCP), with notable improvements in performance due to modifications in the crossover, mutation rates, and the introduction of elitism. Optimal performance is observed with a collection size of around 150 subsets. Enhancements like returning two children per crossover and adjusting mutation rates contribute to better fitness values and reduced subset sizes. Introducing elitism further refines solution quality and efficiency, proving its effectiveness in preserving high-quality solutions. The algorithm’s performance benefits from allowing more runtime, with significant fitness and efficiency improvements observed across various subset sizes. Future work could focus on tuning algorithm parameters and exploring alternative genetic operators to optimize the solution for more significant problem instances.