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 1 indicating inclusion of the subset and a 0 indicating exclusion.

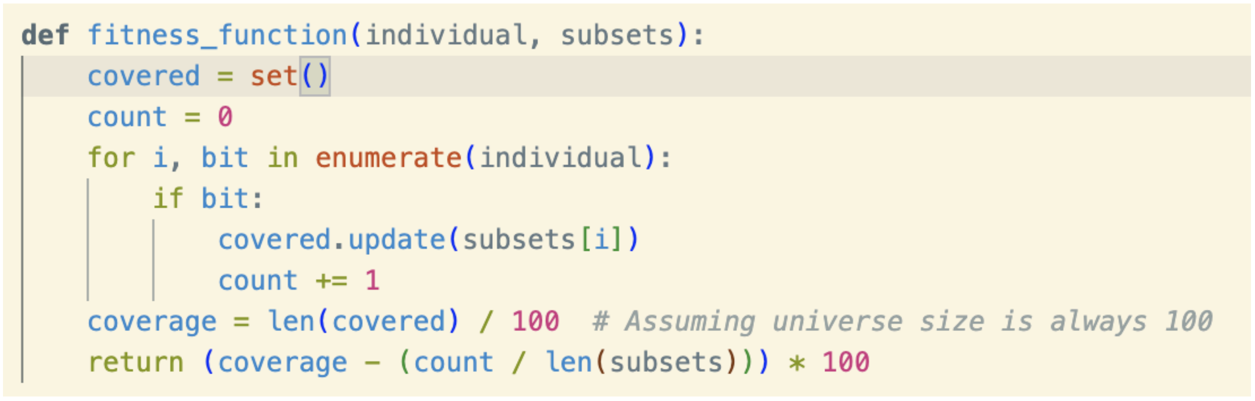
The algorithm starts by initializing a population of random individuals. Each individual is evaluated using a fitness function that balances two goals: maximizing the coverage of the universe (i.e., ensuring that all elements from the universe are covered by the selected subsets) and minimizing the number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

Selection is based on fitness, where fitter individuals are more 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 a specified number of 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.

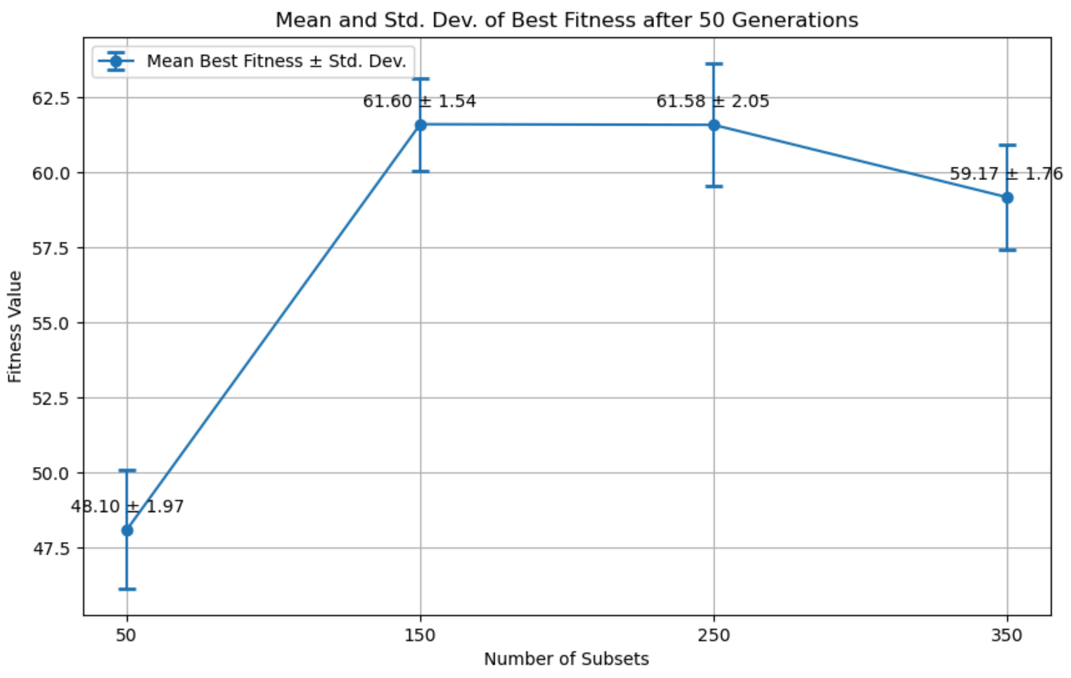


Figure 1: Mean Best Fitness Value over 50 generations

The graph shows the mean and standard deviation of the best fitness values after 50 generations for different subset sizes (50, 150, 250, 350) in the genetic algorithm for the Set Covering Problem (SCP). As the number of subsets increases, the mean fitness improves significantly from 48.10 for 50 subsets to 61.60 for 150 subsets, reaching a peak of 61.58 for 250 subsets. However, further increasing the subsets to 350 slightly reduces the mean fitness to 59.17. The standard deviation decreases with more subsets up to 150, indicating more consistent results, but slightly increases beyond 250 subsets, suggesting reduced consistency.

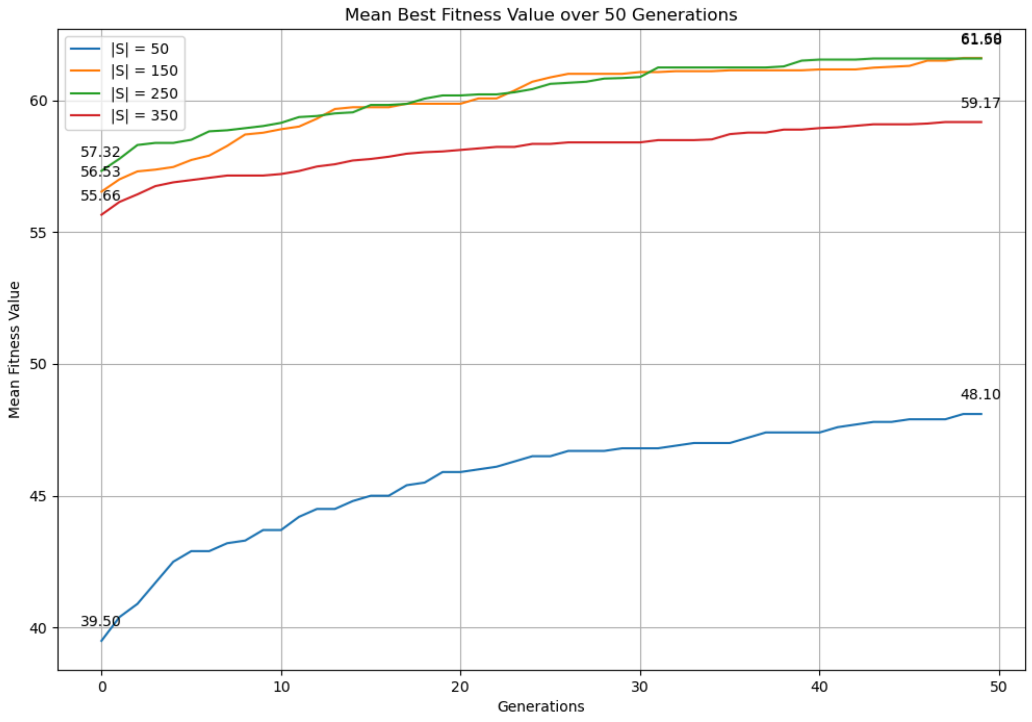


Figure 2: Mean and Standard Deviation of best Fitness values after 50 generations.

displays the mean and standard deviation of the best fitness values after 50 generations for different collection sizes. Notable findings:

1. The mean best fitness improves as the collection size increases from 50 to 150 subsets, then plateaus and slightly decreases for larger collections.
2. The highest mean fitness is achieved with |S| = 150 subsets (61.60 ± 1.54), suggesting this may be an optimal problem size for the algorithm.
3. The standard deviation is relatively small for all collection sizes, indicating consistent performance across multiple runs.
4. The fitness for |S| = 50 is significantly lower (48.10 ± 1.97), likely due to insufficient subsets to optimally cover the universe.

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 further enhance performance, especially for larger problem instances.

Figure 2

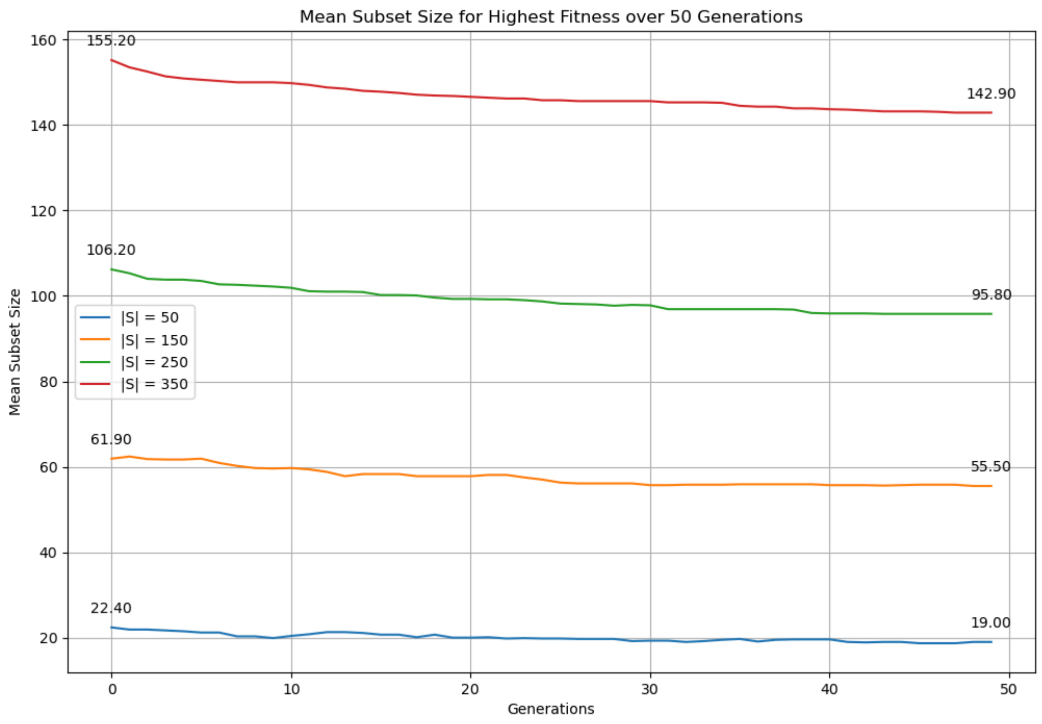


Figure 2: Mean Subset size over a generation

Analysis of Genetic Algorithm Performance on Set Covering Problem

We implemented a genetic algorithm to solve randomly generated instances of the Set Covering Problem (SCP) with a universe set U containing integers from 1 to 100, and varying numbers of subsets in the collection S. The algorithm was run for 50 generations with a population size of 50 for each problem instance.

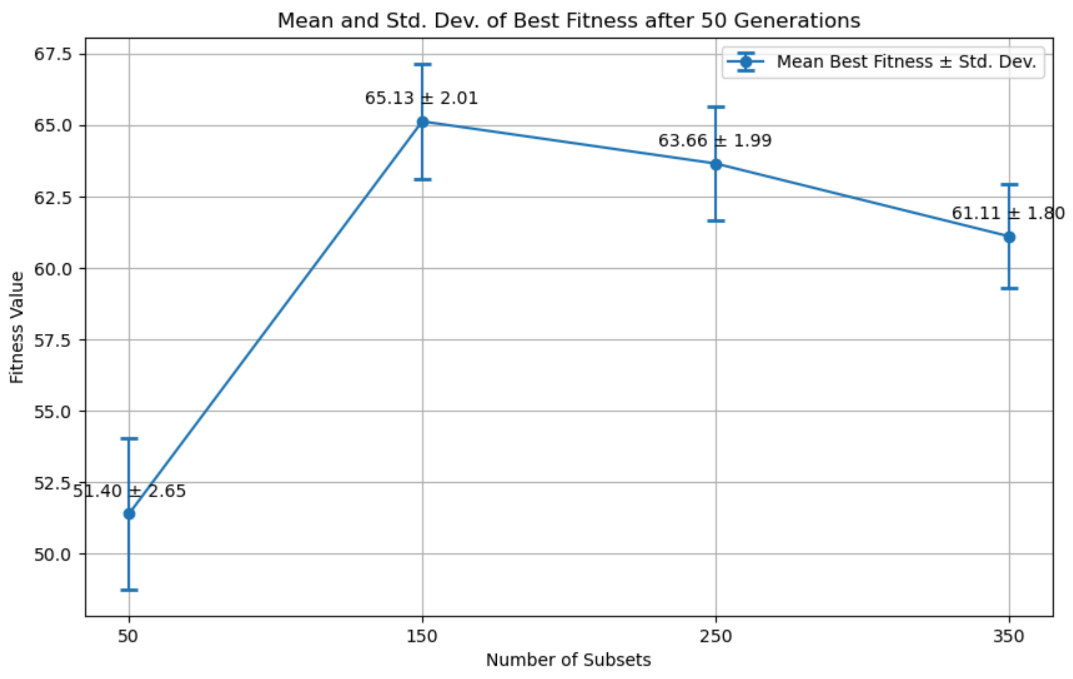
Figure 1 shows how the mean subset size for the highest fitness solution changes over 50 generations, for different collection sizes |S| of 50, 150, 250, and 350 subsets. Some key observations:

1. Larger collection sizes (|S| = 250, 350) result in solutions with larger subset sizes, likely because there are more subsets to choose from to cover the universe.
2. The mean subset size decreases for all collection sizes over generations, indicating the algorithm is optimizing to find smaller covering sets.
3. The rate of improvement is fastest in the early generations and slows down later, suggesting potential convergence.
4. Smaller collection sizes (|S| = 50, 150) show more stability in subset size across generations.

Question 2 :
Improving the Algorithm

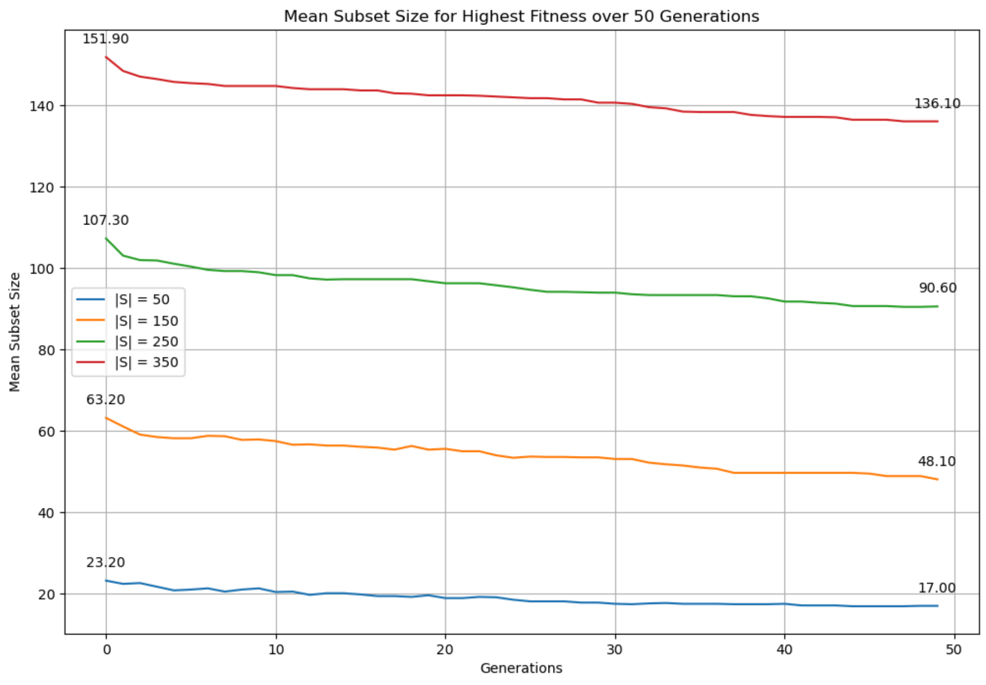
Inserting both Child

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.



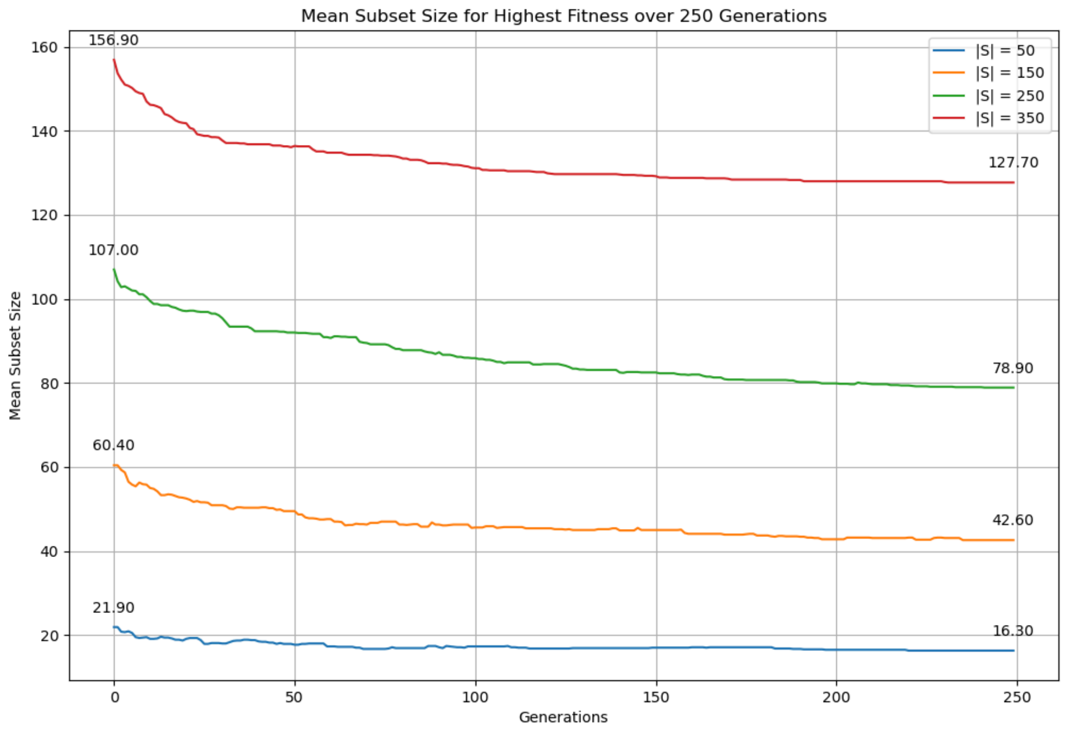
The modification to the genetic algorithm, which now returns two children instead of one during the crossover operation, has a 2.5% increase in the mean fitness values. The image shows that the mean best fitness value increased by 3 units after implementing the change. This enhancement demonstrates the 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%.

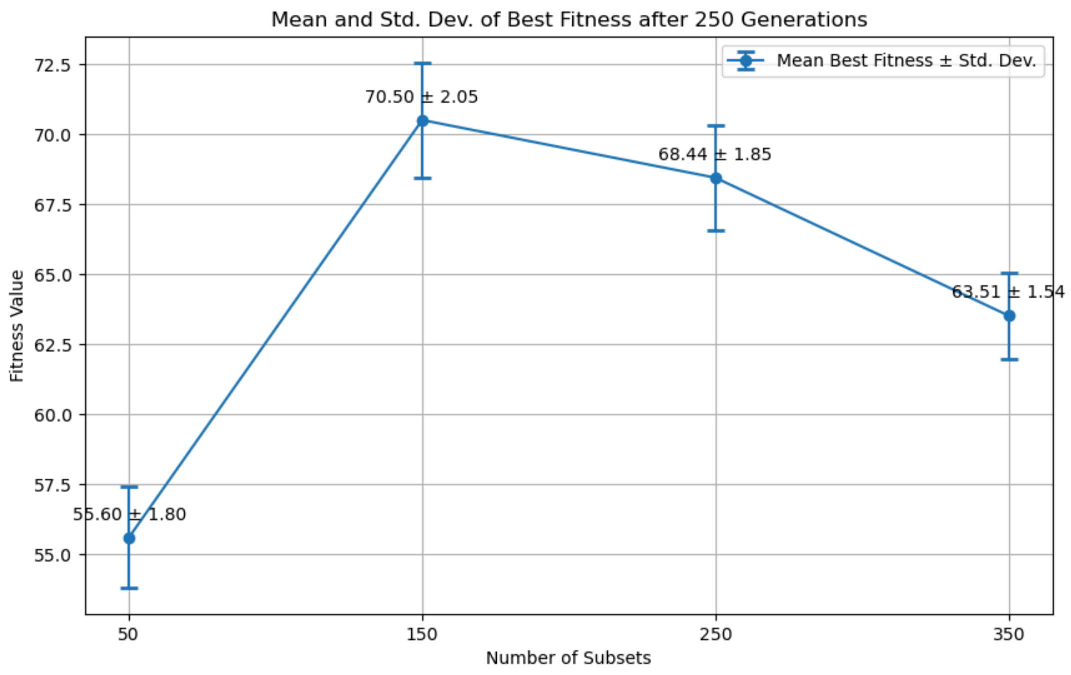


Changing Mutation Rate and Number of Generation

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.



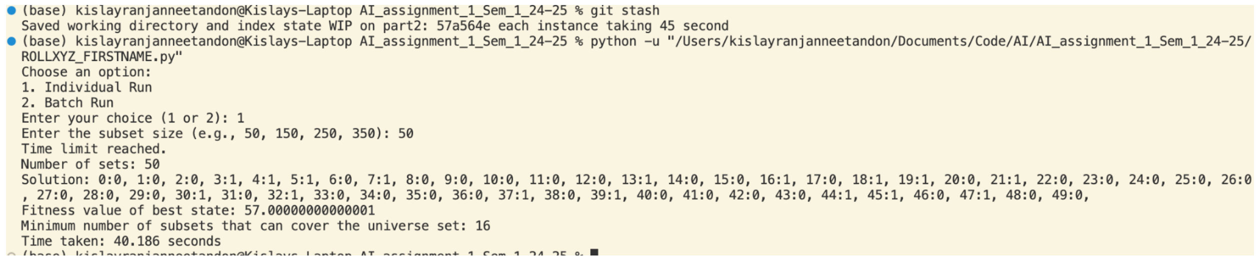
The new graph with 250 generations and a decreasing mutation rate from 0.3 to 0.01 shows significant changes compared to the earlier values observed over 50 generations. Initially, the mean subset sizes were 136 for 350 generations, 90 for 250 generations, 48 for 150 generations, and 18 for 50 generations. 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, leading to improved 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.



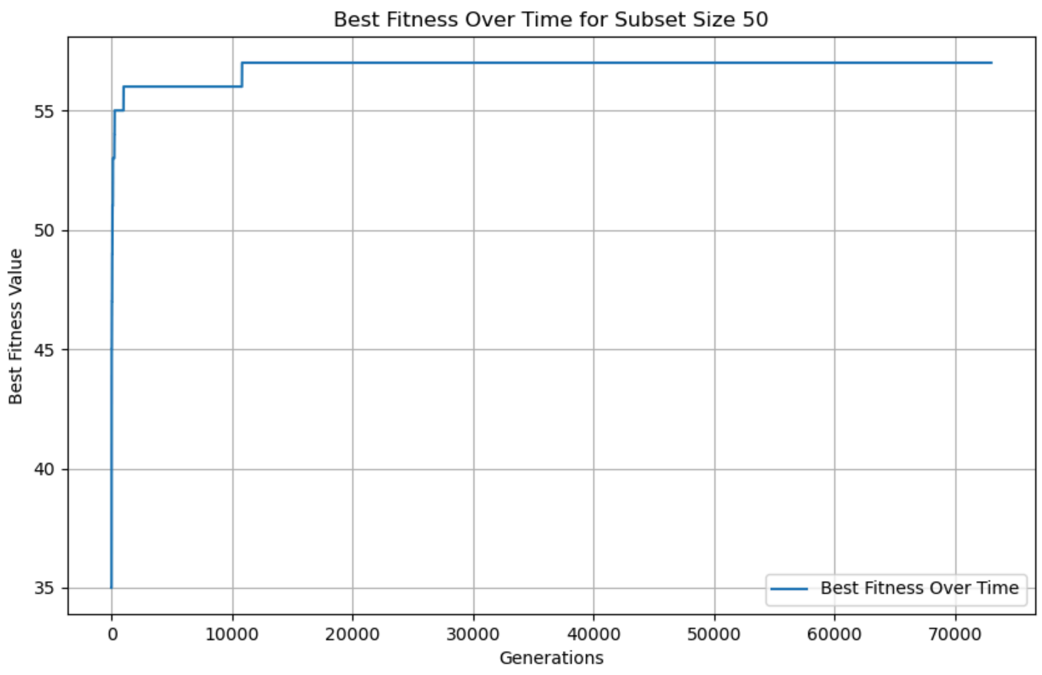
Also, there was a significant increase in mean fitness values by approx 2.5%.

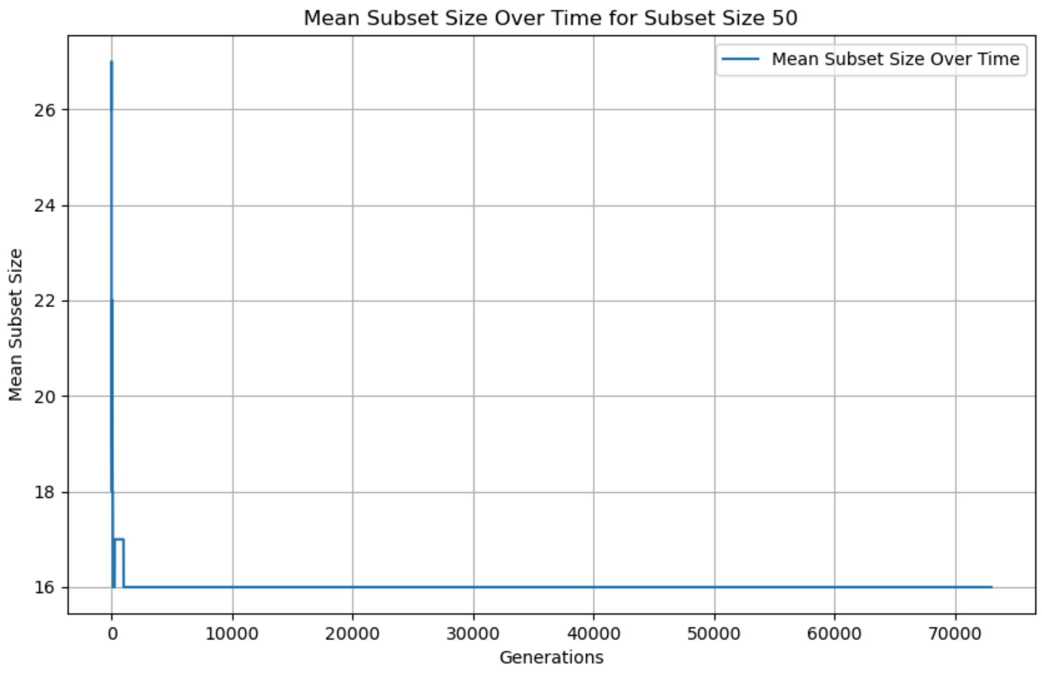
Utilizing time limit of 40 second

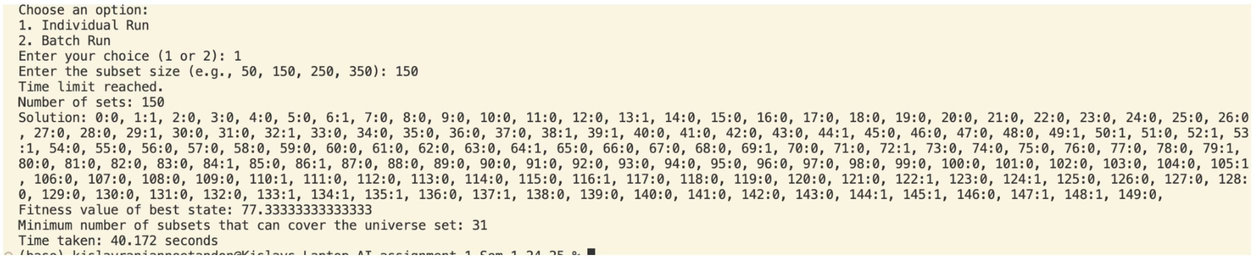
In the experiment, the genetic algorithm was modified to run each instance for a fixed duration of 40 seconds. This adjustment aimed to evaluate the impact of a time constraint on the algorithm’s performance. It was observed that the 40-second runtime influenced the quality of the solutions. Giving us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



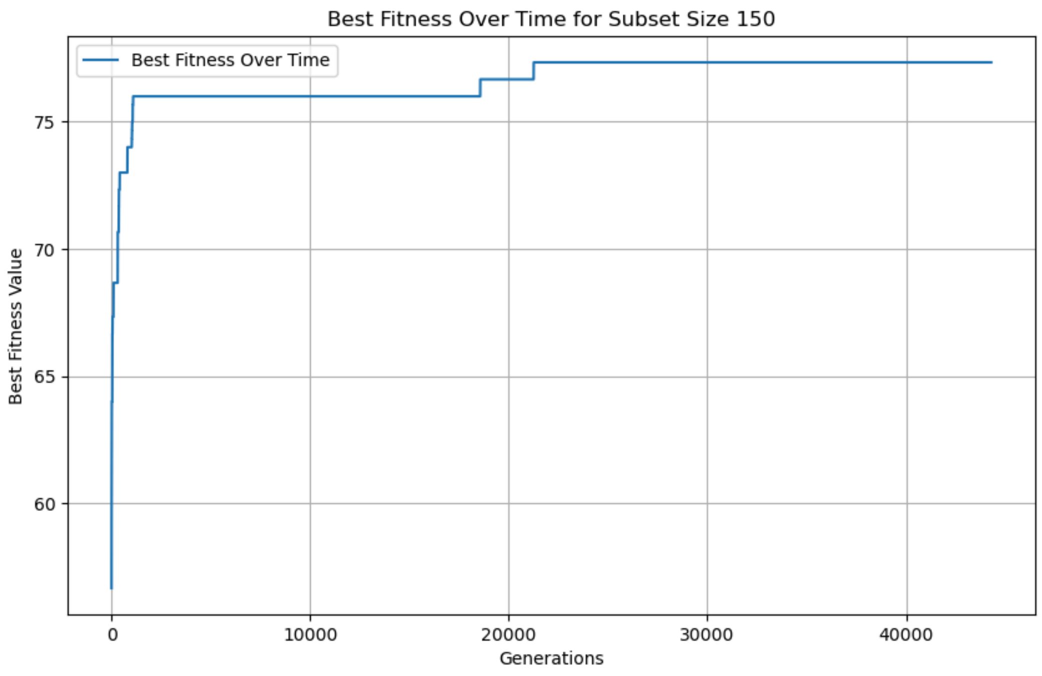
```
● (base) kislayranjanneetandon@Kislays-Laptop AI_assignment_1_Sem_1_24-25 % git stash
  Saved working directory and index state WIP on part2: 57a564e each instance taking 45 second
● (base) kislayranjanneetandon@Kislays-Laptop AI_assignment_1_Sem_1_24-25 % python -u "/Users/kislayranjanneetandon/Documents/Code/AI/AI_assignment_1_Sem_1_24-25/
  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
```

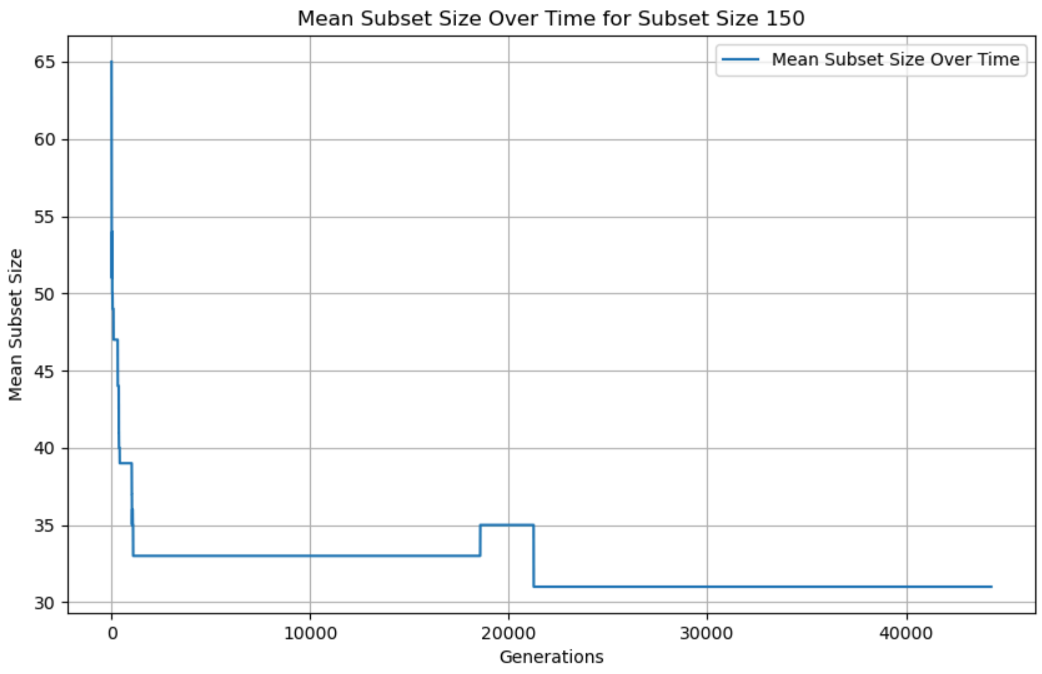

Best Fitness Over Time for Subset Size 50
Best Fitness Value
Generations
Best Fitness Over Time
35
40
45
50
55
0
10000
20000
30000
40000
50000
60000
70000

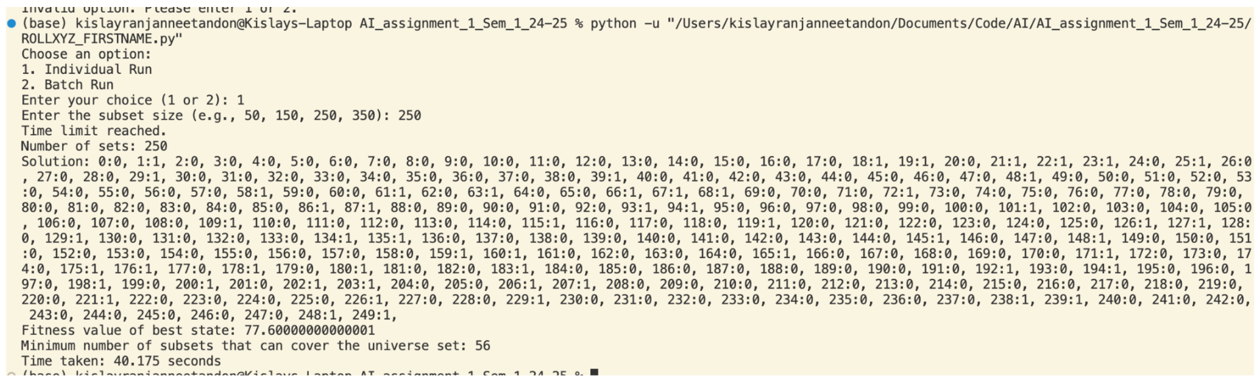

Mean Subset Size Over Time for Subset Size 50
Mean Subset Size
Generations
Mean Subset Size Over Time
16
18
20
22
24
26
0
10000
20000
30000
40000
50000
60000
70000



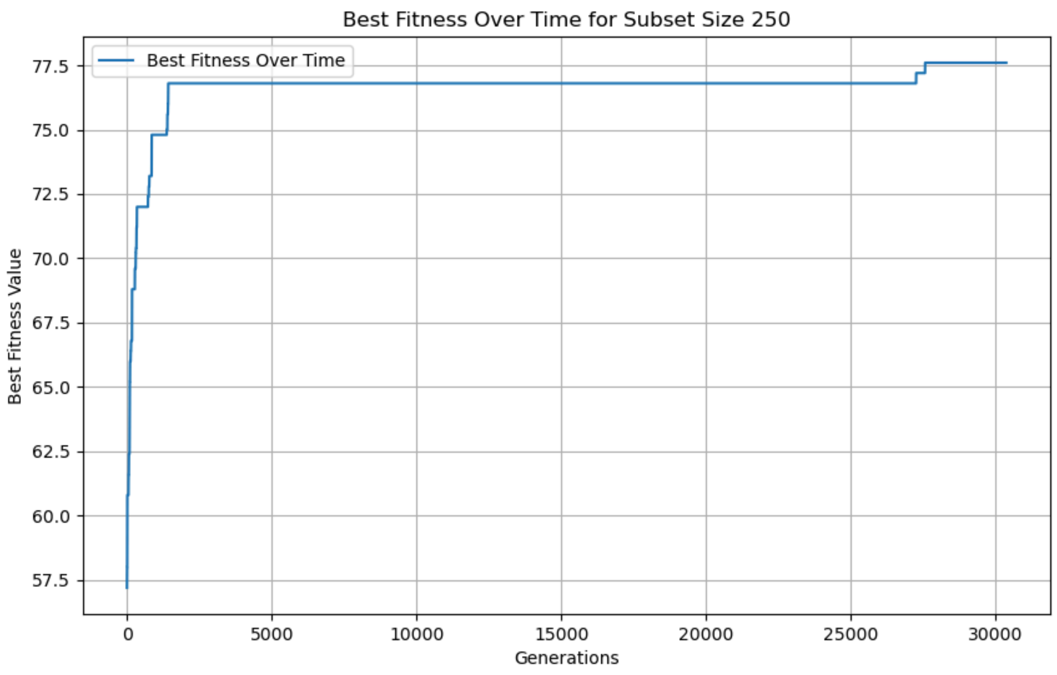
```
Choose an option:
1. Individual Run
2. Batch Run
Enter your choice (1 or 2): 1
Enter the subset size (e.g., 50, 150, 250, 350): 150
Time limit reached.
Number of sets: 150
Solution: 0:0, 1:1, 2:0, 3:0, 4:0, 5:0, 6:1, 7:0, 8:0, 9:0, 10:0, 11:0, 12:0, 13:1, 14:0, 15:0, 16:0, 17:0, 18:0, 19:0, 20:0, 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
(base) kislayranjannootandon@Kislays-Laptop AI assignment 1 Sem 1 24-25 %
```

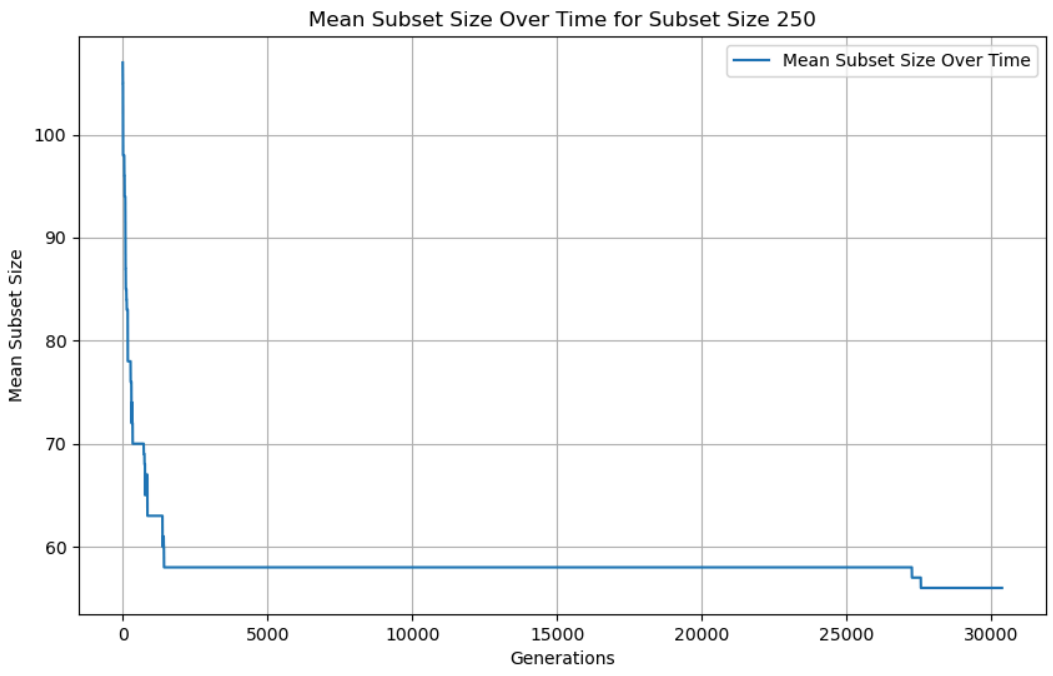


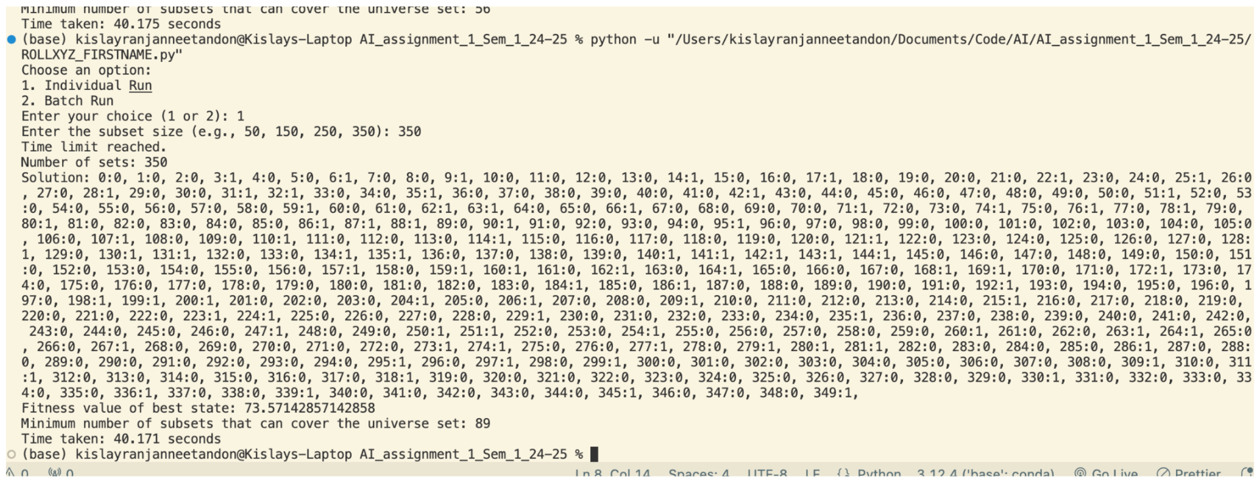




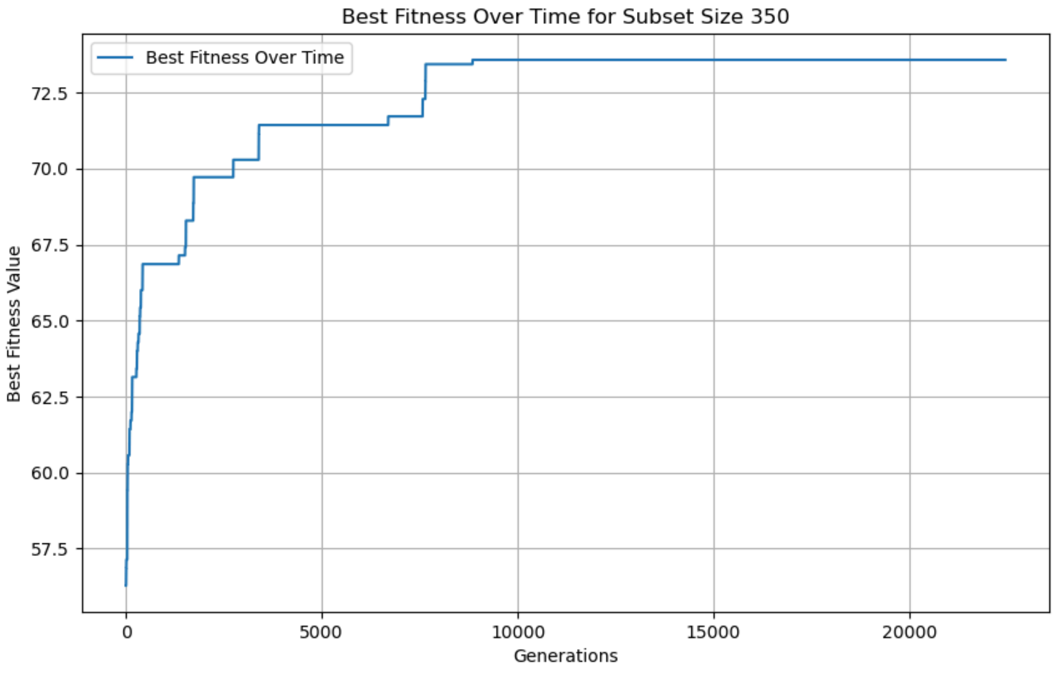
```
Invalid option. Please enter 1 or 2.
● (base) kislayranjanneetandon@Kislays-Laptop AI_assignment_1_Sem_1_24-25 % python -u "/Users/kislayranjanneetandon/Documents/Code/AI/AI_assignment_1_Sem_1_24-25/
  ROLLXYZ_FIRSTNAME.py"
  Choose an option:
  1. Individual Run
  2. Batch Run
  Enter your choice (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 % █
```

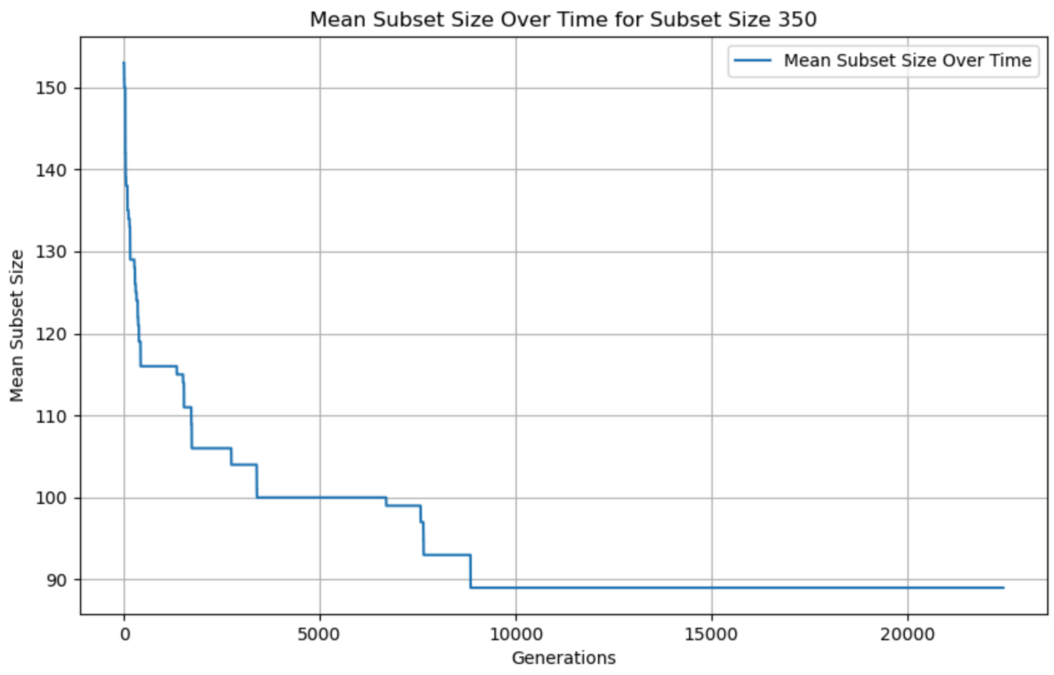

Best Fitness Over Time for Subset Size 250
Best Fitness Over Time
Best Fitness Value
Generations
77.5
75.0
72.5
70.0
67.5
65.0
62.5
60.0
57.5
0
5000
10000
15000
20000
25000
30000


Mean Subset Size Over Time for Subset Size 250
Mean Subset Size Over Time
Mean Subset Size
Generations
100
90
80
70
60
0
5000
10000
15000
20000
25000
30000



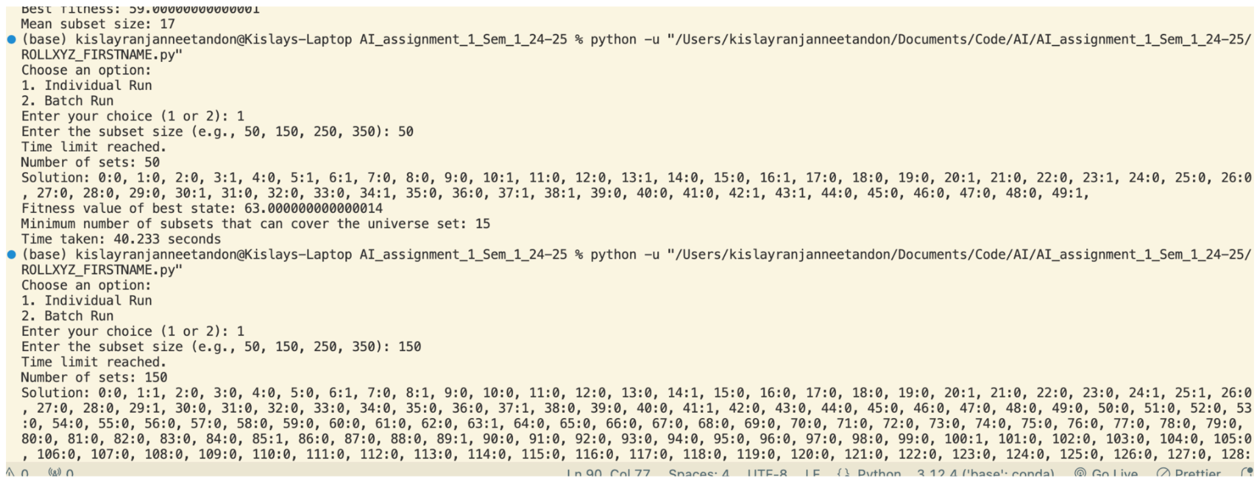
```
Minimum number of subsets that can cover the universe set: 56
Time taken: 40.175 seconds
(base) kislayranjanneetandon@Kislays-Laptop AI_assignment_1_Sem_1_24-25 % python -u "/Users/kislayranjanneetandon/Documents/Code/AI/AI_assignment_1_Sem_1_24-25/
ROLLXYZ_FIRSTNAME.py"
Choose an option:
1. Individual 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 %
```





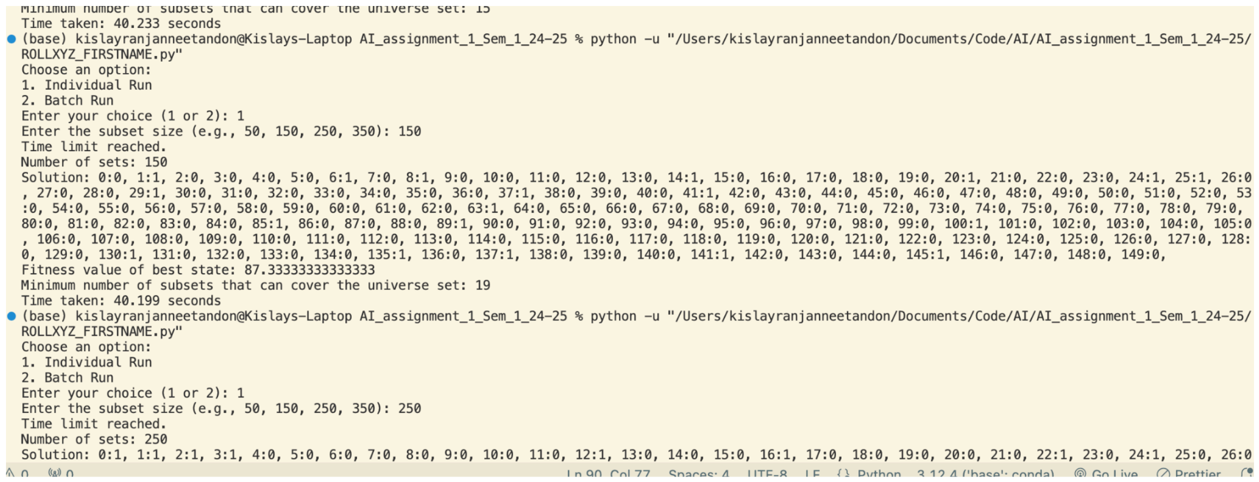
Introducing Elitism

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 them to the next generation, while the rest of the population undergoes genetic operations like crossover and mutation.



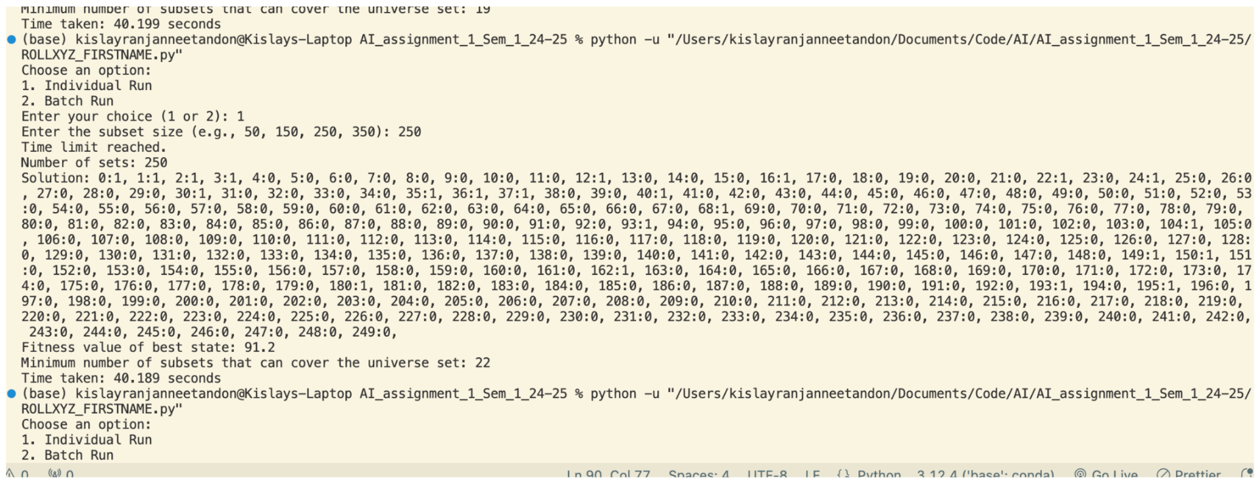
```
Best fitness: 59.00000000000001
Mean subset size: 17
● (base) kislayranjanneetandon@Kislays-Laptop AI_assignment_1_Sem_1_24-25 % python -u "/Users/kislayranjanneetandon/Documents/Code/AI/AI_assignment_1_Sem_1_24-25/
ROLLXYZ_FIRSTNAME.py"
Choose an option:
1. Individual Run
2. Batch Run
Enter your 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, 4:0, 5:1, 6:1, 7:0, 8:0, 9:0, 10:1, 11:0, 12:0, 13:1, 14:0, 15:0, 16:1, 17:0, 18:0, 19:0, 20:1, 21:0, 22:0, 23:1, 24:0, 25:0, 26:0
, 27:0, 28:0, 29:0, 30:1, 31:0, 32:0, 33:0, 34:1, 35:0, 36:0, 37:1, 38:1, 39:0, 40:0, 41:0, 42:1, 43:1, 44:0, 45:0, 46:0, 47:0, 48:0, 49:1,
Fitness value of best state: 63.000000000000014
Minimum number of subsets that can cover the universe set: 15
Time taken: 40.233 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, 3:0, 4:0, 5:0, 6:1, 7:0, 8:1, 9:0, 10:0, 11:0, 12:0, 13:0, 14:1, 15:0, 16:0, 17:0, 18:0, 19:0, 20:1, 21:0, 22:0, 23:0, 24:1, 25:1, 26:0
, 27:0, 28:0, 29:1, 30:0, 31:0, 32:0, 33:0, 34:0, 35:0, 36:0, 37:1, 38:0, 39:0, 40:0, 41:1, 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, 81:0, 82:0, 83:0, 84:0, 85:1, 86:0, 87:0, 88:0, 89:1, 90:0, 91:0, 92:0, 93:0, 94:0, 95:0, 96:0, 97:0, 98:0, 99:0, 100:1, 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:
```

Ln 90, Col 77 Spaces: 4 UTF-8 LF {} Python 3.12.4 ('base': conda) Go Live Prettier



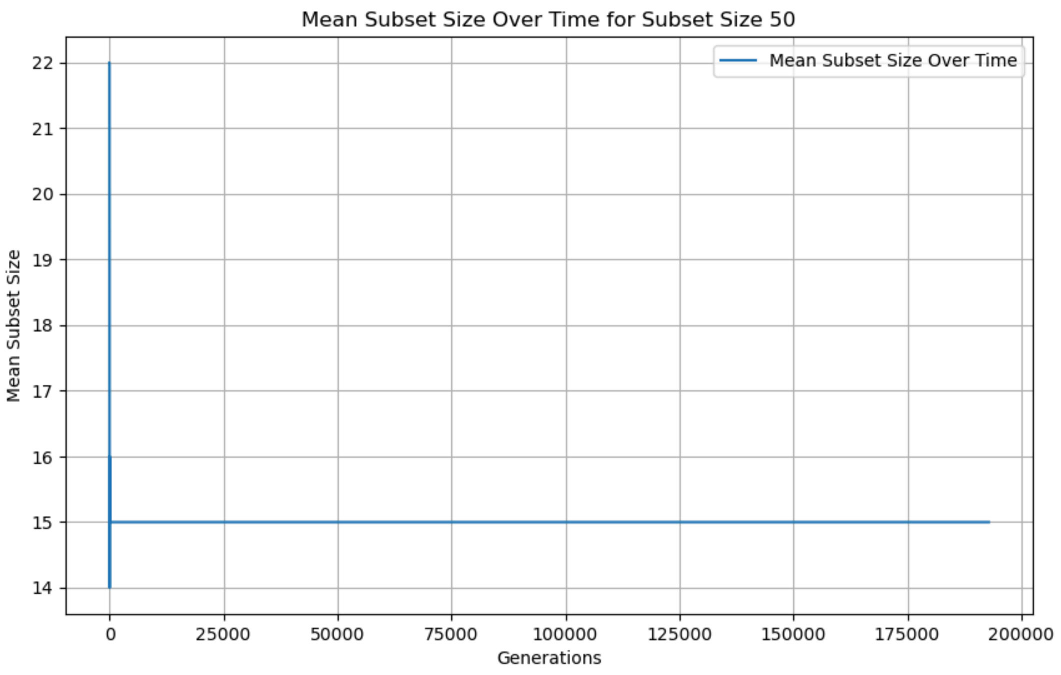
```
Minimum number of subsets that can cover the universe set: 15
Time taken: 40.233 seconds
● (base) kislayranjanneetandon@Kislays-Laptop AI_assignment_1_Sem_1_24-25 % python -u "/Users/kislayranjanneetandon/Documents/Code/AI/AI_assignment_1_Sem_1_24-25/
ROLLXYZ_FIRSTNAME.py"
Choose an option:
1. Individual 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, 3:0, 4:0, 5:0, 6:1, 7:0, 8:1, 9:0, 10:0, 11:0, 12:0, 13:0, 14:1, 15:0, 16:0, 17:0, 18:0, 19:0, 20:1, 21:0, 22:0, 23:0, 24:1, 25:1, 26:0
, 27:0, 28:0, 29:1, 30:0, 31:0, 32:0, 33:0, 34:0, 35:0, 36:0, 37:1, 38:0, 39:0, 40:0, 41:1, 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, 81:0, 82:0, 83:0, 84:0, 85:1, 86:0, 87:0, 88:0, 89:1, 90:0, 91:0, 92:0, 93:0, 94:0, 95:0, 96:0, 97:0, 98:0, 99:0, 100:1, 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, 130:1, 131:0, 132:0, 133:0, 134:0, 135:1, 136:0, 137:1, 138:0, 139:0, 140:0, 141:1, 142:0, 143:0, 144:0, 145:1, 146:0, 147:0, 148:0, 149:0,
Fitness value of best state: 87.33333333333333
Minimum number of subsets that can cover the universe set: 19
Time taken: 40.199 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): 250
Time limit reached.
Number of sets: 250
Solution: 0:1, 1:1, 2:1, 3:1, 4:0, 5:0, 6:0, 7:0, 8:0, 9:0, 10:0, 11:0, 12:1, 13:0, 14:0, 15:0, 16:1, 17:0, 18:0, 19:0, 20:0, 21:0, 22:1, 23:0, 24:1, 25:0, 26:0
```

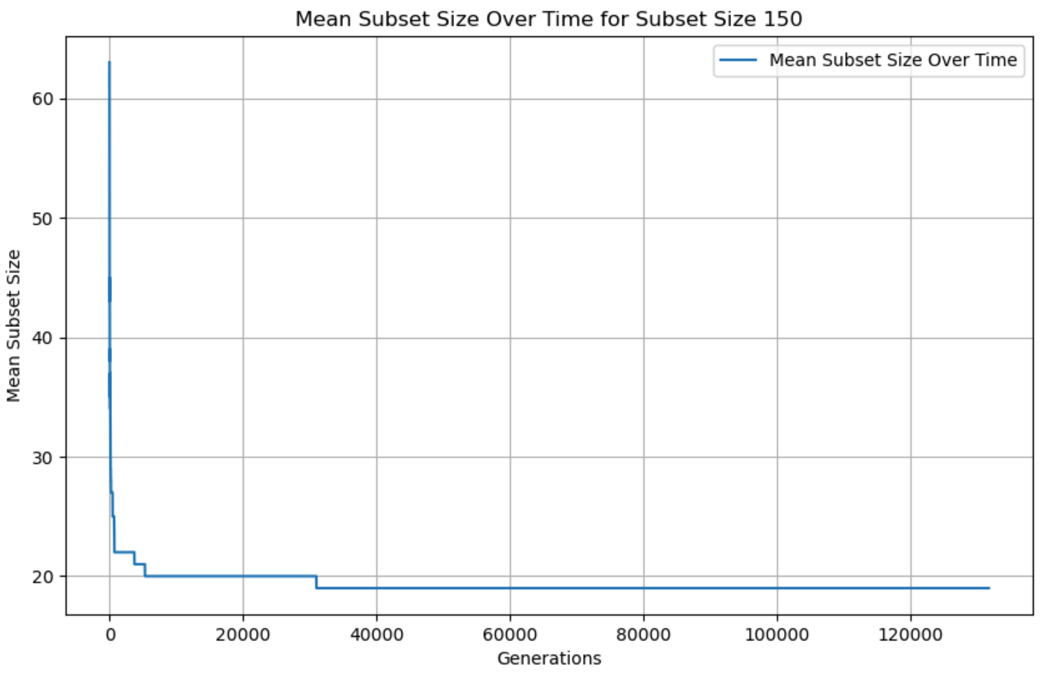
Ln 90, Col 77 Spaces: 4 UTF-8 LF {} Python 3.12.4 ('base': conda) Go Live Prettier

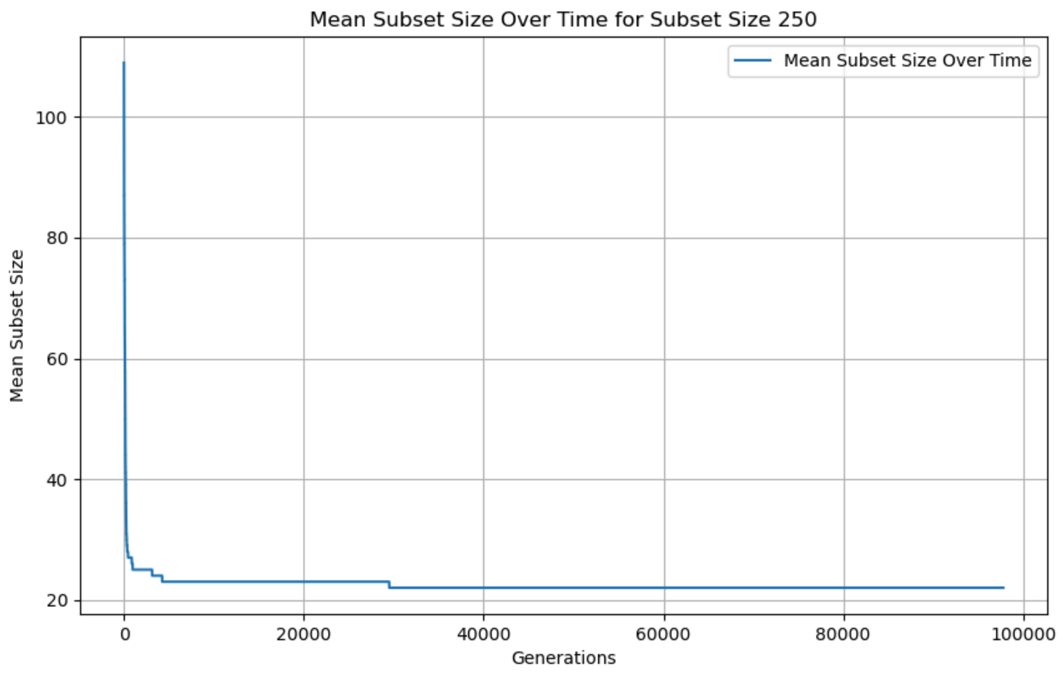


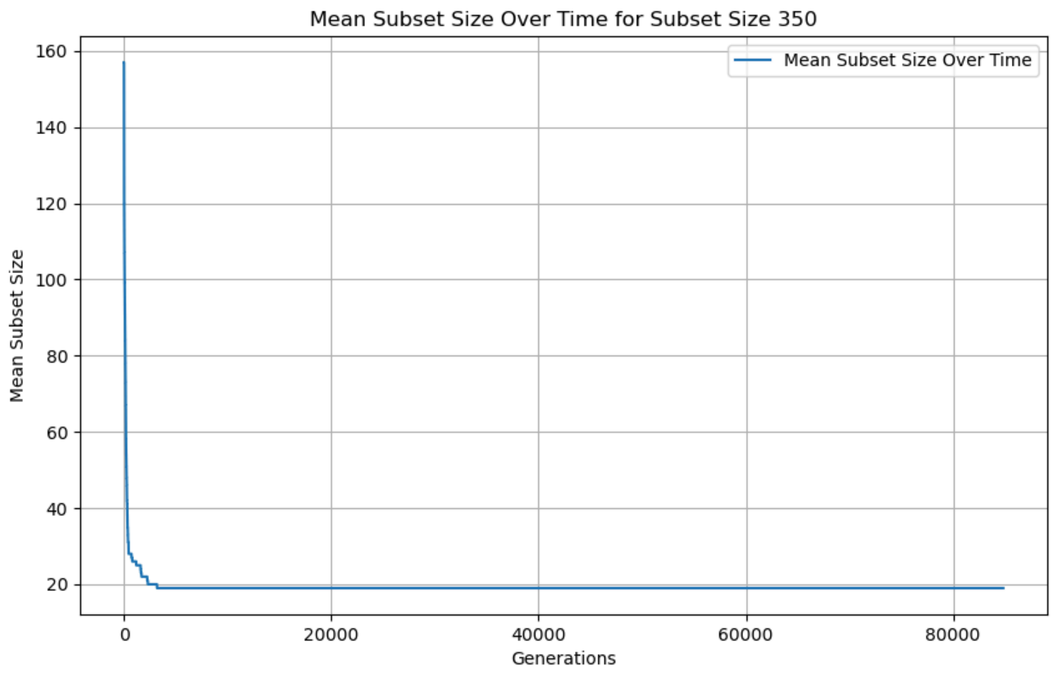
```
Minimum number of subsets that can cover the universe set: 19
Time taken: 40.199 seconds
● (base) kislayranjanneetandon@Kislays-Laptop AI_assignment_1_Sem_1_24-25 % python -u "/Users/kislayranjanneetandon/Documents/Code/AI/AI_assignment_1_Sem_1_24-25/
ROLLXYZ_FIRSTNAME.py"
Choose an option:
1. Individual Run
2. Batch Run
Enter your choice (1 or 2): 1
Enter the subset size (e.g., 50, 150, 250, 350): 250
Time limit reached.
Number of sets: 250
Solution: 0:1, 1:1, 2:1, 3:1, 4:0, 5:0, 6:0, 7:0, 8:0, 9:0, 10:0, 11:0, 12:1, 13:0, 14:0, 15:0, 16:1, 17:0, 18:0, 19:0, 20:0, 21:0, 22:1, 23:0, 24:1, 25:0, 26:0
, 27:0, 28:0, 29:0, 30:1, 31:0, 32:0, 33:0, 34:0, 35:1, 36:1, 37:1, 38:0, 39:0, 40:1, 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, 63:0, 64:0, 65:0, 66:0, 67:0, 68:1, 69:0, 70:0, 71:0, 72:0, 73:0, 74:0, 75:0, 76:0, 77:0, 78:0, 79:0,
80:0, 81:0, 82:0, 83:0, 84:0, 85:0, 86:0, 87:0, 88:0, 89:0, 90:0, 91:0, 92:0, 93:1, 94:0, 95:0, 96:0, 97:0, 98:0, 99:0, 100:0, 101:0, 102:0, 103:0, 104:1, 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, 144:0, 145:0, 146:0, 147:0, 148:0, 149:1, 150:1, 151
:0, 152:0, 153:0, 154:0, 155:0, 156:0, 157:0, 158:0, 159:0, 160:0, 161:0, 162:1, 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, 180:1, 181:0, 182:0, 183:0, 184:0, 185:0, 186:0, 187:0, 188:0, 189:0, 190:0, 191:0, 192:0, 193:1, 194:0, 195:1, 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: 91.2
Minimum number of subsets that can cover the universe set: 22
Time taken: 40.189 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
```

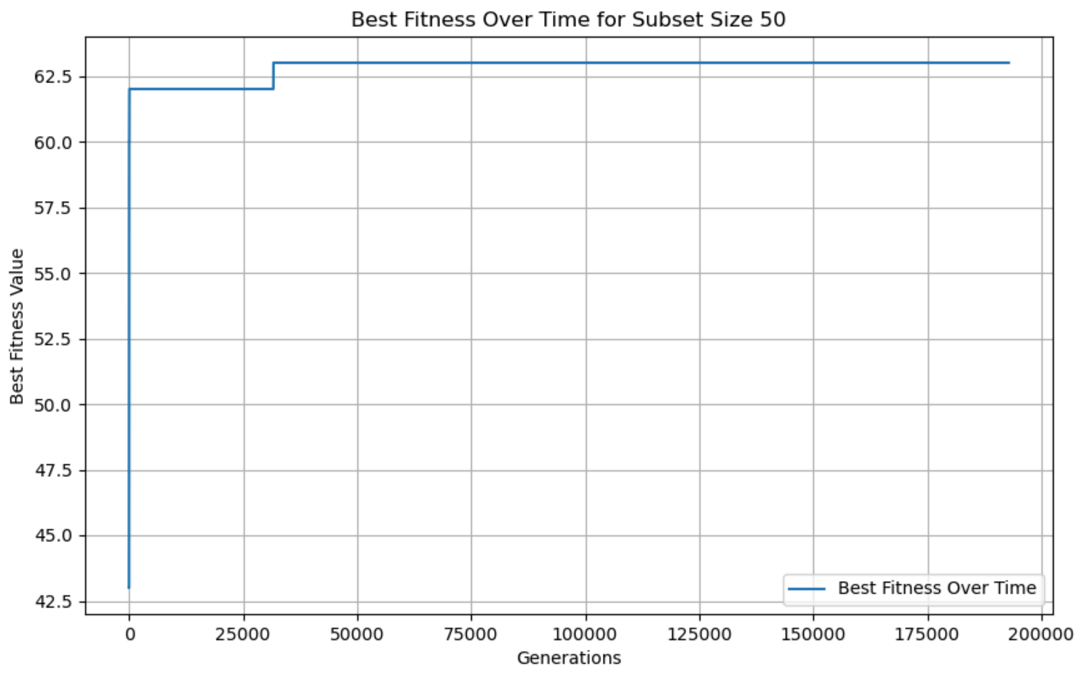
Ln 90, Col 77 Spaces: 4 UTF-8 LF {} Python 3.12.4 ('base': conda) Go Live Prettier

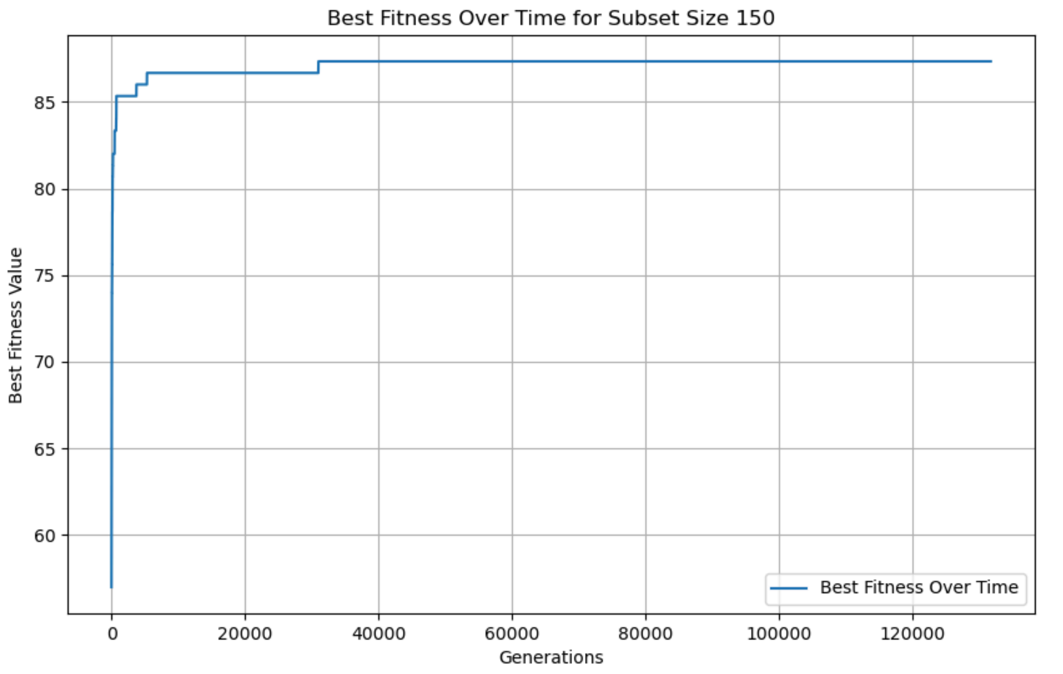

Mean Subset Size Over Time for Subset Size 50
Mean Subset Size Over Time
Mean Subset Size
Generations
14
15
16
17
18
19
20
21
22
0
25000
50000
75000
100000
125000
150000
175000
200000

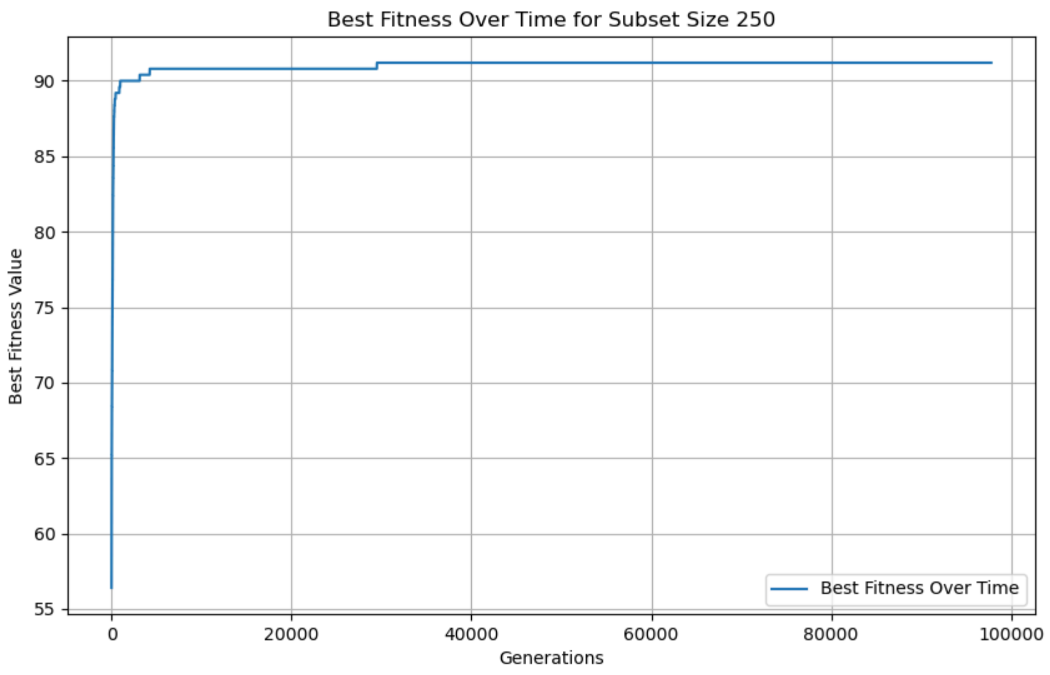

Mean Subset Size Over Time for Subset Size 150
Mean Subset Size Over Time
Mean Subset Size
Generations
20
30
40
50
60
0
20000
40000
60000
80000
100000
120000

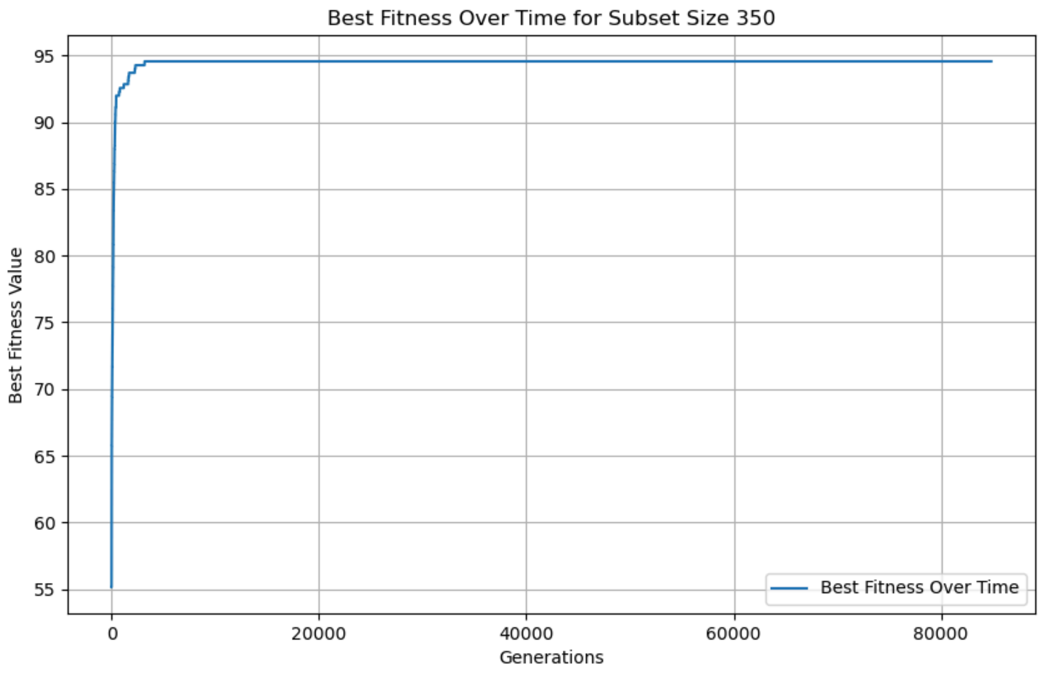

Mean Subset Size Over Time for Subset Size 250
Mean Subset Size Over Time
Mean Subset Size
Generations
100
80
60
40
20
0
20000
40000
60000
80000
100000


Mean Subset Size Over Time for Subset Size 350
Mean Subset Size Over Time
Mean Subset Size
Generations
160
140
120
100
80
60
40
20
0
20000
40000
60000
80000


Best Fitness Over Time for Subset Size 50
Best Fitness Value
Generations
Best Fitness Over Time
62.5
60.0
57.5
55.0
52.5
50.0
47.5
45.0
42.5
0
25000
50000
75000
100000
125000
150000
175000
200000


Best Fitness Over Time for Subset Size 150
Best Fitness Value
Generations
Best Fitness Over Time
85
80
75
70
65
60
0
20000
40000
60000
80000
100000
120000


Best Fitness Over Time for Subset Size 250
Best Fitness Value
Generations
Best Fitness Over Time
0
20000
40000
60000
80000
100000
55
60
65
70
75
80
85
90


Best Fitness Over Time for Subset Size 350
Best Fitness Value
Generations
Best Fitness Over Time
0
20000
40000
60000
80000
55
60
65
70
75
80
85
90
95

Tournament Selection

Tournament Selection is a popular and effective method for choosing parents in genetic algorithms. In this approach, a subset of individuals from the population is randomly selected, and the best individual from this subset is chosen to be a parent. This process is repeated to select another parent, potentially using a different subset. The size of the tournament can be adjusted based on the problem requirements. Tournament selection helps maintain diversity within the population and reduces the likelihood of premature convergence, as it allows for the selection of high-quality individuals without requiring a strict ranking of the entire population.