ProblemSet2

Charitha Madhamsetty

October 2024

Problem 1: Genetic Algorithm Implementation

Q1. Implementation Description

Problem Description and Initialization:
Here, the main aim of the knapsack problem is to maximize the value of the items added in the knapsack by keeping the total weight less than the constraint W given. I have used a population-based genetic algorithm involving several steps like fitness calculation, selection, crossover, and mutation to implement this. First, An initial population of a certain size is randomly generated - 'P' with size 'popsize'. This population contains individuals (chromosomes), which are the sequences of 0s and 1s of length 'n' that represent whether a particular item is included or not (0 - not included, 1 - included).
Fitness Function and Selection methods:
I have implemented the fitness function where the fitness represents the summation of values of active genes and zero if the total weight of the active genes exceeds the weight constraint W. I have also implemented the custom fitness function i.e. fitness with penalty. So, in this case, to ensure that the maximum weight included in the knapsack does not exceed W, a penalty value is reduced from the actual value which is proportional to the difference in the total weight and the weight constraint W instead of zero. This penalty method helps enforce the weight constraint by reducing the fitness of infeasible solutions. But this method can ensure that even the infeasible solutions that might contain valuable genetic information (high-value combination of items) are explored. penalty = (Weight - W)/W value = tvalue - penalty * tvalue After that, I used the Roulette wheel and Tournament selection methods. In roulette wheel selection, a probability of selection is assigned to each individual which is proportional to fitness, with better solutions having higher chances of selection. In the tournament method, a small subset of individuals is chosen, and the one with the highest fitness in that subset is selected, encouraging competition between individuals based on their fitness. Using any of the two selection methods, two individuals with the highest selection probability are selected.
Crossover and Mutation:
After selection, a crossover is performed to generate offspring. Here, I have used single point crossover which generated two off-springs. In this method, segments of parents are swapped after a randomly chosen cross-over point producing offspring. Then, in mutation, more than one randomly chosen genes are altered in the offspring introducing diversity. Along with mutation, I have implemented a repair method where the number of active genes is forcefully reduced to ensure the total weight of the offspring is less than the knapsack capacity. This way I can make sure that individuals with weights greater than knapsack capacity are not produced in each generation. I have explored different mutation rates in the range of (0.05, 0.2) and chose the best one i.e., the one that generated the

highest average fitness.

Stopping Criterion:

As a stopping criterion, a specified number of generations is used - "stop". The algorithm is stopped after going through this number of generations. I implemented my stopping criterion which is "convergence". If there is no improvement in the average fitness values across the generations, then the algorithm is stopped. Here, patience represents how many generations are to be ignored if there is no improvement and tolerance represents the threshold for the acceptable amount of improvement between the generations. During the process, the average fitness and the best fitness of each generation are stored. The best solution with the highest fitness values across all generations is recorded for each selection method.

Q2. Selection alone

a, b.

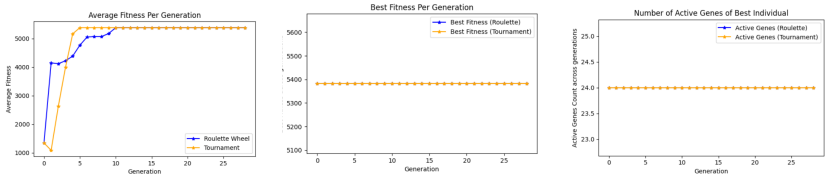

Figure 1: Plots for Config 1

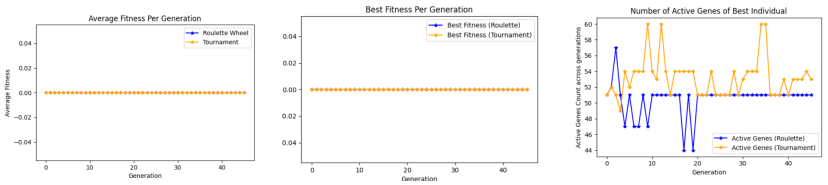

Figure 2: Plots for Config 2

c. **Results using config 1:**

Selection Method	No. of items	Total Value	Generation
Roulette	24	5382.00	0
Tournament	24	5382.00	0

Table 1: Best Individual across all generations

Results using config 2:

Selection Method	No. of items	Total Value	Generation
Roulette	51	0.00	0
Tournament	51	0.00	0

Table 2: Best Individual across all generations

d. From the results of config1, the best fitness is obtained in generation 0 (since best fitness is the same across all generations, 1st generation is considered the best) itself with a total value of 5382. Also, the best solution remained the same i.e., the best fitness across all the generations (till Pstop) is the same. This is because there is no crossover performed which means no new individuals are not generated. So, the average fitness and the best fitness remain the same across all the generations. The same is the case with config2 when the penalty is not used. The total fitness values remained zero across all the generations because individuals are the same, so the fitness values are the same and the probabilities of selection are also the same resulting in the same parents in each generation.

Q2. Extra Credit

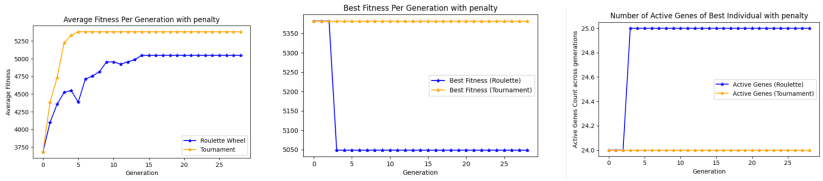

Figure 3: Plots for Config 1

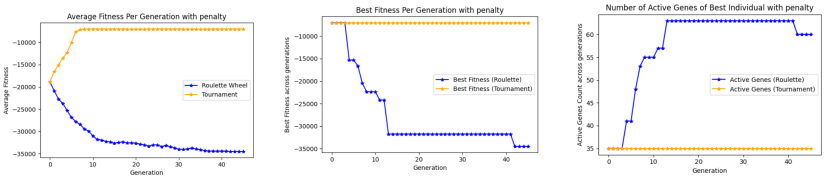

Figure 4: Plots for Config 2

Results using config 1 with penalty:

Selection Method	No. of items	Total Value	Generation
Roulette	24	5382.00	0
Tournament	24	5382.00	0

Table 3: Best Individual across all generations

Results using config 2 with penalty:

Selection Method	No. of items	Total Value	Generation
Roulette	35	-7004.92	0
Tournament	35	-7004.92	0

Table 4: Best Individual across all generations

Observations and Comparison with the fitness function without penalty:
When the penalty is used, fitness values change, so the probabilities of selection change across the generations. In the case of config1, the best solution is produced in generation 0 itself with the fitness value of - 5382 in the case of the Roulette wheel selection method, but it got reduced to 5048 at Pstop. This means that a significant number of individuals at Pstop have a total weight greater than W, resulting in a decrease in the total value.
The same is the case for config2, the best solution occurred in generation 0 and the total value is reduced when Pstop is reached.

Q3. Integrate Cross-over and Mutation

Without penalty:
,

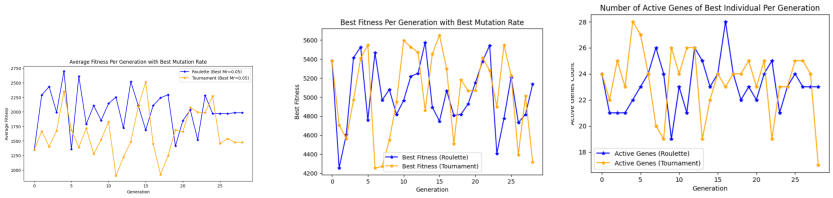

Figure 5: Plots for Config 1

Results for config1:

Selection Method	No. of items	Total Value	Total Weight	Best Mutation Rate
Roulette	23	4084.00	752	0.05
Tournament	17	2556.00	511	0.05

Table 5: Results at Pstop

Selection Method	No. of items	Total Value	Generation
Roulette	25	5574.00	13
Tournament	24	5650.00	15

Table 6: Best Individual across all generations

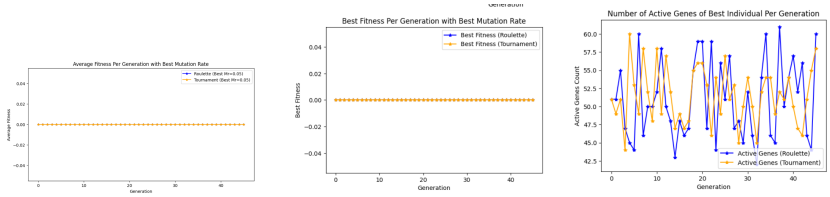

Figure 6: Plots for Config 2

Results for config2:

Selection Method	No. of items	Total Value	Total Weight	Best Mutation Rate
Roulette	60	0	29353.00	0.05
Tournament	58	0	26596.00	0.05

Table 7: Results at Pstop

Selection Method	No. of items	Total Value	Generation
Roulette	51	0.00	0
Tournament	24	0.00	0

Table 8: Best Individual across all generations

With penalty:
Results for config1:

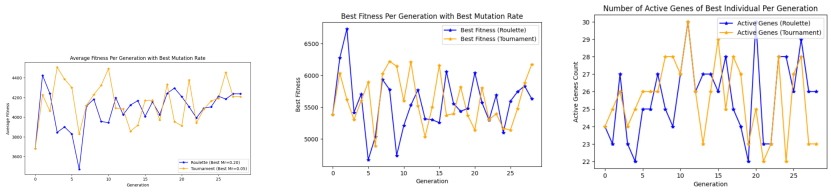

Figure 7: Plots for Config 1 using penalty

Selection Method	No. of items	Total Value	Total Weight	Best Mutation Rate
Roulette	26	5630.00	807	0.20
Tournament	23	4593.00	804	0.05

Table 9: Results at Pstop

Selection Method	No. of items	Total Value	Generation
Roulette	27	6734.00	2
Tournament	28	6219.00	8

Table 10: Best Individual across all generations

Results for config2:

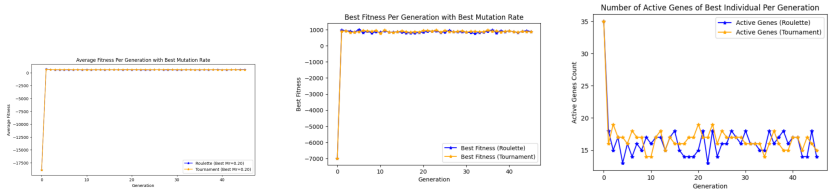

Figure 8: Plots for Config 2

Results for config2:

Selection Method	No. of items	Total Value	Total Weight	Best Mutation Rate
Roulette	14	741	2259.00	0.05
Tournament	15	561	2287.00	0.05

Table 11: Results at Pstop

Selection Method	No. of items	Total Value	Generation
Roulette	16	1015.00	5
Tournament	19	985.00	23

Table 12: Best Individual across all generations

Discussion and Comparison: As shown in the table1, for config 1, at Pstop, the value is maximized using the Roulette selection method. The best individual across all generations is seen in the 15th generation with a total value of 5650.00 and using the tournament selection method. When

compared with the results obtained in selection alone (5382.00), there is an improvement in the total value of the best individual across all the generations. In the case of config 2, the total value remained zero in both cases. Also, the total value in both cases is greater than the total capacity of the knapsack. This indicates that there are no individuals whose weight is less than or equal to knapsack capacity. So, the fitness value is always zero.

To overcome the problem: Penalty function and Repair solution: To overcome this problem, the penalty function along with forcible repair is used. In penalty function, the total value is reduced proportionally to the difference in total weight and the knapsack capacity when the total weight exceeds the knapsack capacity. In repair function, the number of active genes is reduced forcefully after mutation. This ensures that the produced off-springs are feasible solutions. so, after repairing, the total value improved and the total weight of the best individual at Pstop was less than the knapsack capacity.

Q3 Extra Credit

Using Convergence as stopping criteria:
Results for config1:

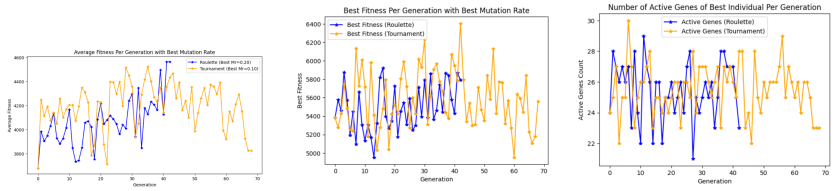

Figure 9: Plots for Config 1 using convergence

Selection Method	No. of items	Total Value	Total Weight	Best Mutation Rate
Roulette	23	3639.00	728	0.20
Tournament	23	3639.00	728	0.20

Table 13: Results at Pstop

Selection Method	No. of items	Total Value	Generation
Roulette	26	5921.00	16
Tournament	26	5921.00	16

Table 14: Best Individual across all generations

So, when convergence is applied, in the case of config1, the value is increased to 5921.00 which was 5650.00 without convergence. From the graph, the algorithm converged at the 42nd generation in the case of Roulette and at 69th generation in the case of the Tournament selection method. Also, using convergence helps in finding the nearly optimal solution since we are stopping the algorithm when there is no improvement in the fitness values. However, like every genetic algorithm, using

convergence as a stopping criterion also does not guarantee an optimal solution.

Q4. Explore Population sizes - different population sizes and number of trails

Report : Each experiment of specific population size is performed for 'num-trails' times

Config1: Roulette Wheel Selection method

Popsize	Knapsack total value	knapsack max weight	Mean no. of items in the best solution
20	2258.83 ± 1798.54	1090	22
30	2149.30 ± 1867.87	1078	22
40	2882.70 ± 2026.71	1275	22
50	2446.83 ± 1839.45	1132	22
60	1090.37 ± 1697.97	1131	24
80	2227.17 ± 1765.27	1230	23
90	2217.20 ± 1888.26	1079	23
100	1689.43 ± 1827.58	1007	23
110	2275.77 ± 2000.95	1214	22
120	1934.83 ± 1750.10	1156	22

Table 15: Best Individual selected from all the trails

Config2: Roulette Wheel Selection method

Popsize	Knapsack total value	knapsack max weight	Mean no. of items in the best solution
20	0.00 ± 0.00	30357	48
30	0.00 ± 0.00	29445	51
40	0.00 ± 0.00	31094	51
50	0.00 ± 0.00	30752	51
60	0.00 ± 0.00	27949	50
80	0.00 ± 0.00	27462	50
90	0.00 ± 0.00	30694	50
100	0.00 ± 0.00	29128	51
110	0.00 ± 0.00	28026	50
120	0.00 ± 0.00	29245	51

Table 16: Best Individual selected from all the trails

The reason for total value to be zero for all the population sizes in the case of config 2 is already discussed above. So, During the random generation that is performed to initialize the population, the individuals are generated in such a way that there is no individual whose total weight is less than or equal to knapsack capacity. To overcome this, we have used penalty and repair function as described above.

Problem 2: Compare GA with a Non-population-based Search Algorithm

Describe what you did and your results. Compare with the GA you implemented yourself.

Implementation : I used Dynamic Programming approach to solve the knapsack problem without using population.

1. The main objective of the knapsack problem is to maximize the total value of items added in the knapsack, while keeping the total weight less than or equal to the weight constraint W.
2. create a 2D DP table dp[i][W] - i represents the index of the item and w represents the current weight of the knapsack.
3. Initialize the table dp[0][w] = 0 for all w
4. Fill up the table
   if w[i] ¿ w,
   dp[i][w]=dp[i1][w]
   else
   dp[i][w]=v[i]+dp[i1][ww[i]]
   Take the maximum of the both
   dp[i][w]=max(dp[i1][w],v[i]+dp[i1][ww[i]])
5. dp[n][w] will give the maximum value obtained.

Results obtained using DP:

Config File	No. of items	Total weight	Total Value
Config1	32	850	7534.00
Config2	120	2335.00	1266.00

Table 17: Results using DP

Comparison of results obtained in genetic and non-genetic algorithms: In the genetic algorithm, the most feasible solution is not obtained. Both in config 1 and config2, the value of the items included in the knapsack is greater when a non-genetic algorithm is used. We can observe the DP produces the same result in every trail for config1 and config 2 but it is not the case with GA because we have random population initialization.

Q5. Empirical Comparison

Advantages of using genetic algorithm:

1. searches the entire search space more likely to find global optimum rather than local optimum.
2. GAs are applied to the population of solutions, enabling parallel processing

3. GAs can maintain a good balance in exploration and exploitation through crossover and Mutation.

Disadvantages of using genetic algorithm:

1. GAs heavily depend on population size, crossover rate and mutation rate. Improper tuning leads to bad performance.
2. GAs do not guarantee finding the optimum solution, but only the approximate optimum solution.

Advantages of using non-genetic algorithm:

1. These algorithms exhibit faster convergence.
2. They have deterministic outcomes i.e. they have the same result for the same input, unlike genetic algorithms.
3. They can find the exact global optimum.

Dis-advantages of using non-genetic algorithm:

1. They are prone to getting trapped in local optima.
2. They often focus on exploitation i.e. improving existing solutions ignoring exploration of the entire search space.

Problem 3: Genetic Algorithm Formulation

Q6 Problem Description and GA Design

Problem Description: Machine Learning Feature Selection
In machine learning, we have to select the best subset of features, improving accuracy and reducing overfitting.
Chromosomal Representation:
In each chromosome, o represents that the feature is not selected and 1 represents that it is selected.
Population Initialization:
Just like in the case of knapsack, randomly generated sequences of 0s and 1s of length equal to the number of features.
Stop Criterion:
The algorithm is stopped when there is convergence i.e., the fitness function is not improving over the generations.
Fitness function:
Fitness Function can be based on the model accuracy for each combinations of features. Also, the fitness can be penalized by the number of features selected in order to avoid over-fitting.
Implementation:
Objective Function: Maximize the accuracy and reduce the number of features selected to avoid over-fitting

$$Maximize : f(X) = Performance(S) - \lambda \times |S| \tag{1}$$

Where:

- S is the selected subset of features.
- $f(X)$ is the objective function.
- λ is a regularization parameter controlling the trade-off between model performance and the number of features.
- $|S|$ is the number of selected features.

Fitness Function:

$$Fitness(S) = ModelAccuracy(S) - \lambda \times |S|$$

Selection:
For selection, either Roulette wheel or tournament method can be used just as in the case of knapsack problem.
Cross-over and Mutation: These two operations would be similar to that of the ones used in knapsack problem. Here, off-springs are generated and mutated.

In the final generation, best chromosome with highest fitness values is selected and the number of active genes in that chromosome represent the number of features selected.

Problem 4: Exploring Another Technique

Q7 Exploring Another Nature-inspired Search/Optimization Technique

I have explored Hippopotamus optimization technique.

Hippopotamus Optimization Algorithm: The Hippopotamus Optimization Algorithm is a new addition to nature-inspired metaheuristics. It aims to simulate the rare behaviors of hippopotamuses in aquatic environments. It captures three basic survival behaviors: position updating, wherein hippos search for an improved location in water according to visibility and availability of food; second, defensive behavior, which models the defense mechanism of hippos in response to any form of threat; and third, escaping behavior that models their capability to avoid predators. These mechanisms act like balances between exploration, in finding new areas, and exploitation, in refining known good solutions, so that HO turns out to be robust for complex optimization tasks. None of these is new; rather, the novelty of HO lies in how these behaviors have been modeled mathematically to guide the optimization process across diverse domains.

The performances of the HO algorithm are rigorously tested on various benchmark problems, which include unimodal functions for testing the exploitation ability, multimodal functions for testing the exploration ability, and real-world engineering optimization problems. It performed very well compared with other established algorithms like PSO, GA, and DE in terms of convergence speed, solution quality, and escaping from local optima. HO has shown its potential in handling high-dimensional complex constrained optimization problems in real-world applications, rendering it fairly promising for practical applications, such as engineering design, scheduling, and training neural networks where precision and computational efficiency are of significance.

References

[1] Mohammad Hussein Amiri, Nastaran Mehrabi Hashjin, Mohsen Montazeri, Seyedali Mirjalili, and Nima Khodadadi, *Hippopotamus Optimization Algorithm: A Novel Nature-Inspired Optimization Algorithm.* Shahid Beheshti University, 2023, DOI: 10.21203/rs.3.rs-3503110/v1.