Knapsack Problem using Genetic Algorithm

Shriya Chepur
October 12, 2023

1 IMPLEMENTATION

Implementing the knapsack problem using a Genetic Algorithm (GA) involves creating a genetic representation for chromosomes, defining the fitness function, and applying selection, crossover, and mutation, and iterating through generations to find an optimal or near-optimal solution. In the context of the knapsack problem, a chromosome represents a potential set of items to be placed in the knapsack, where each gene in the chromosome corresponds to an item (selected or not). My fitness function evaluates each chromosome based on the total value of the selected items while ensuring the total weight does not exceed the knapsack capacity. The goal is to maximize the total value of the selected items given the weight constraint.

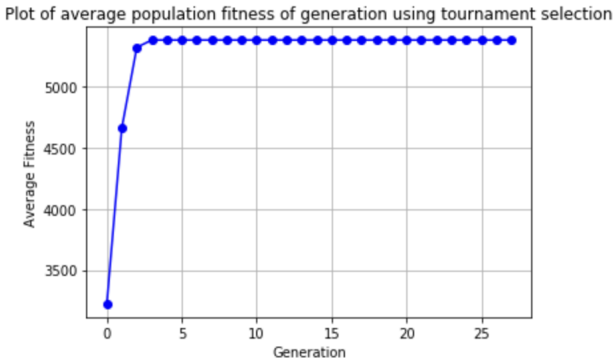
The implementation begins with initializing the initial population from both the config files. The functions for the selection operation, fitness calculation, crossover and mutation are defined.
Fitness function: If the weight in the knapsack exceeds the max. weight, fitness value is -1, else the value itself is the fitness value of that instance.
Selection function: It has two ways of selecting an instance - tournament selection and roulette wheel selection. The functions take the population as the input parameter (tournament selection has an additional parameter of tournament size. In this case, I've assigned it to 5) and return the selected instance.
Crossover: It takes in two instances and produces two instances which are a combination of their parents. Mutation randomly selects an instance, index and swaps the value at that index. The output of this function is a mutated population. We repeat these steps for 28 and 45 generations, which are the stop values given in config file 1 and config file 2 respectively.

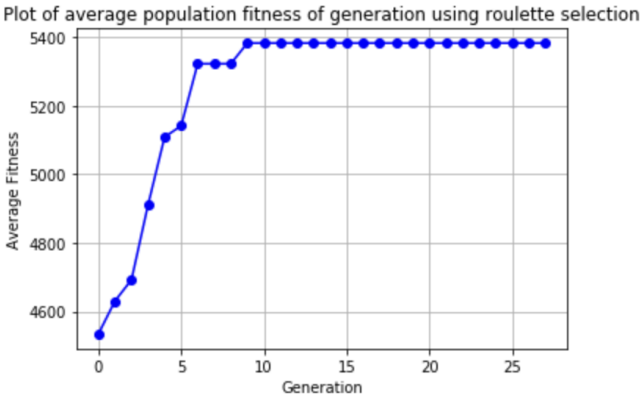
The user is asked to choose if the crossover and mutation functions are to be activated and then the code is executed accordingly. From my observations, the fitness function initially increases and then remains constant after a few generations.

2 Selection Alone

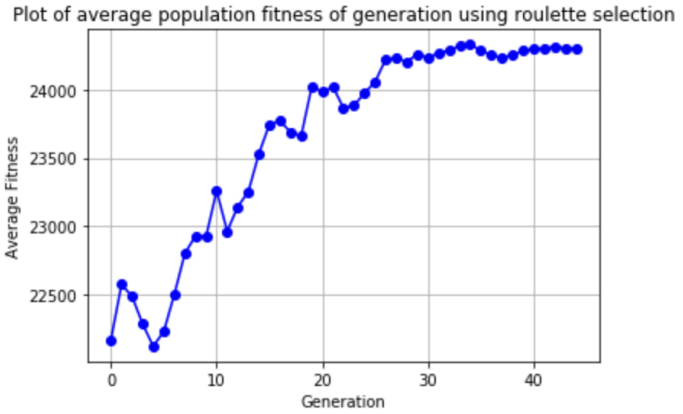
2.1 Average population fitness

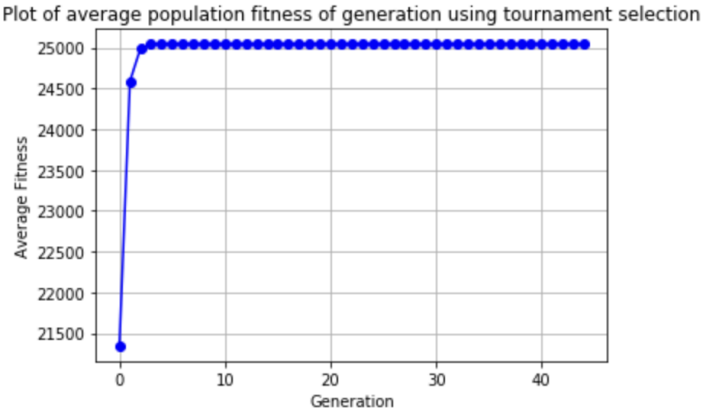
Config 1





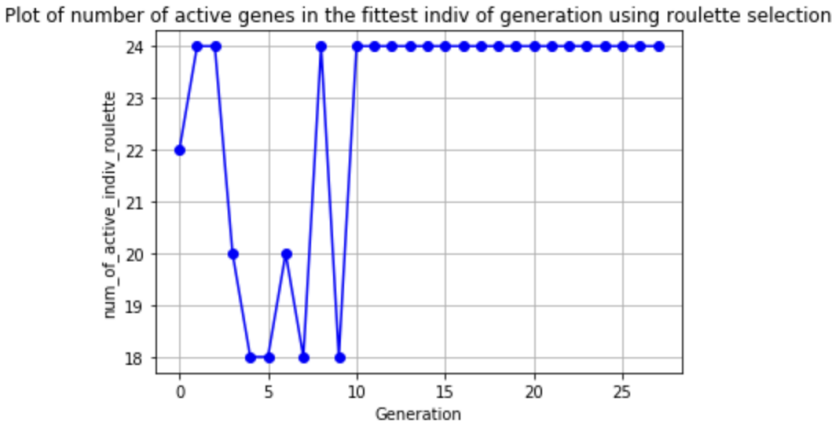
Config 2

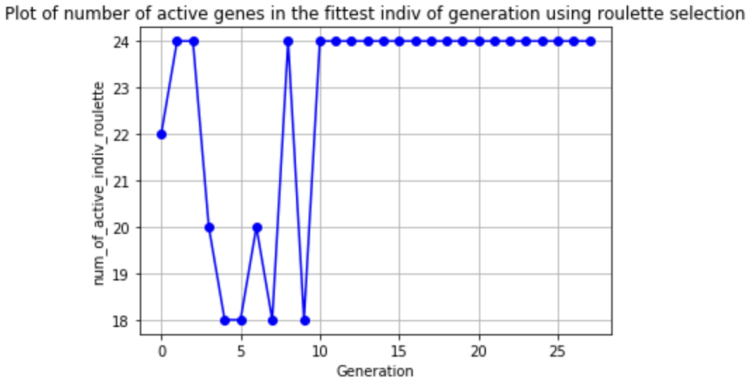




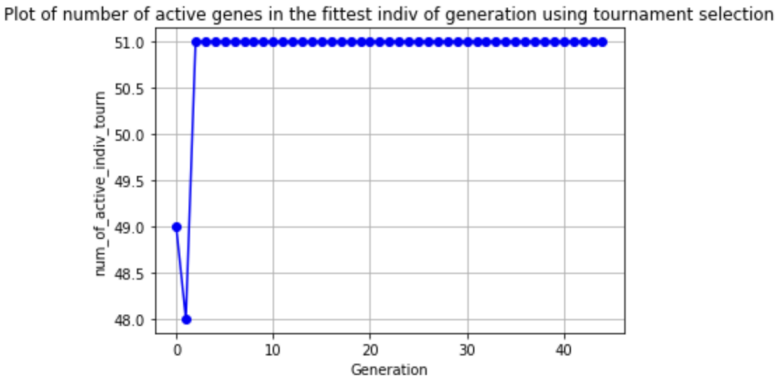
2.2 NUMBER OF ACTIVE GENES OF THE FITTEST INDIVIDUAL

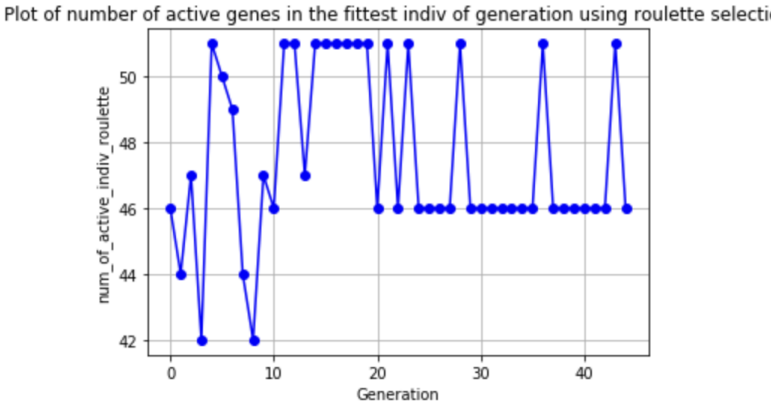
Config 1





Config 2





2.3

The number of active genes at the most fittest individual is 24. It is the same for both tournament and roulette wheel selection.

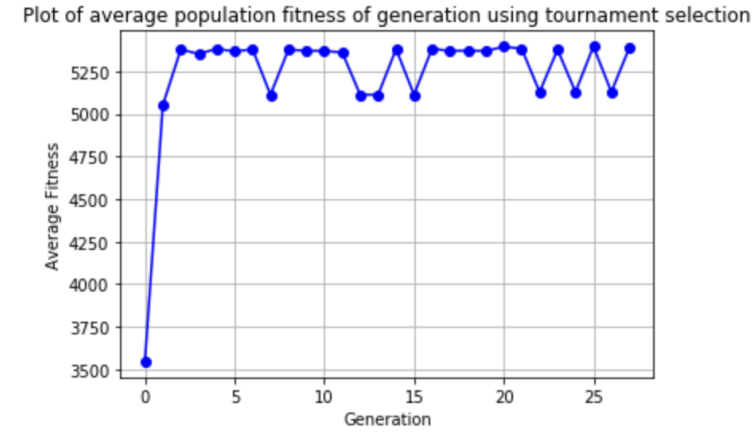
2.4

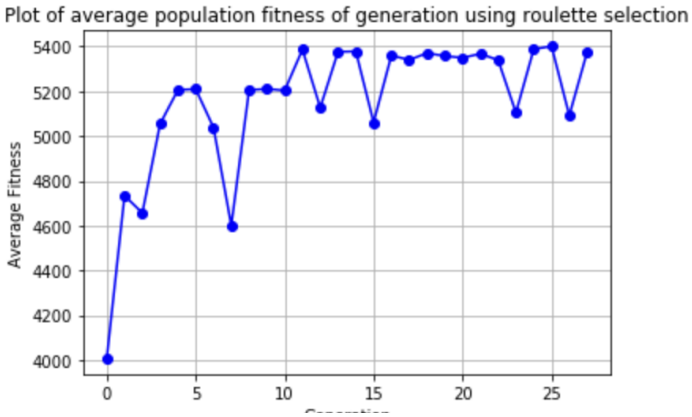
If the fittest chromosomes in "Pstop" consistently improve in quality (higher fitness values) compared to earlier generations, it indicates that the GA has effectively converged towards better solutions. If the fittest chromosomes consistently adhere to the knapsack weight constraint while maximizing value, it indicates that the GA has successfully converged to an optimal or near-optimal solution.

3 Crossover and Mutation

3.1

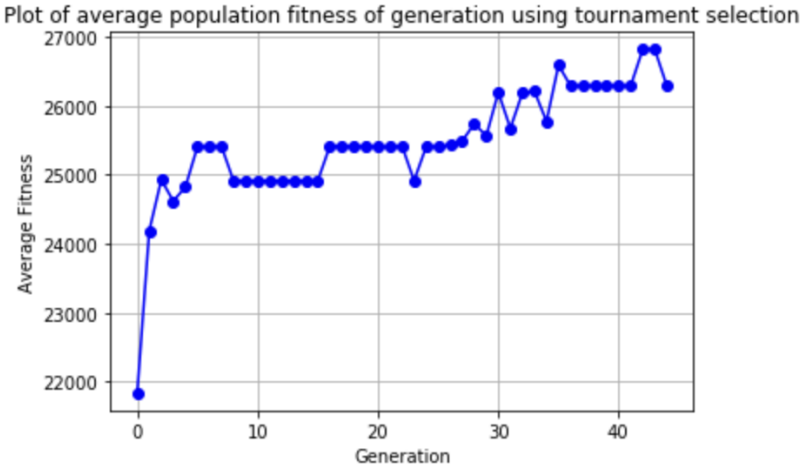
```
display_tourn()
```

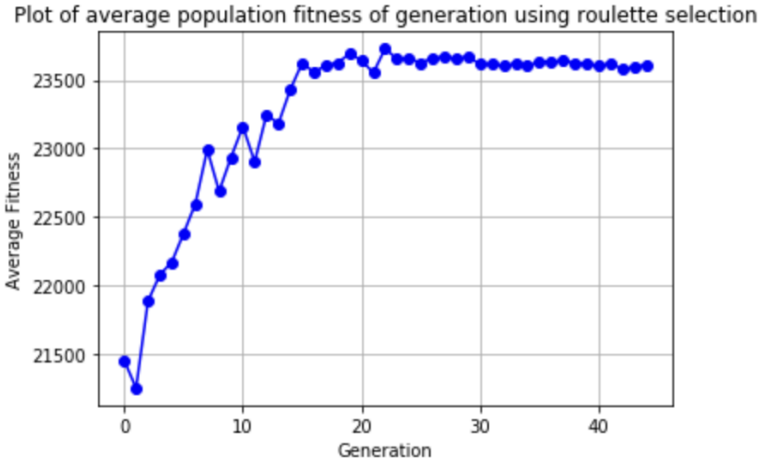




Config 1

3.2


Plot of average population fitness of generation using tournament selection
Average Fitness
Generation
27000
26000
25000
24000
23000
22000
0
10
20
30
40


Plot of average population fitness of generation using roulette selection
Average Fitness
Generation
23500
23000
22500
22000
21500
0
10
20
30
40

Config 2

pop size	total value	max weight	no. of active
2	-1	0	0
10	3934+34	546	22
12	5382+44	840	24
14	5382+44	840	24
20	5382+44	847	24
40	6101+-3	840	24
50	6101+-3	840	24
60	6101+-3	840	28
75	6101+-3	840	24
100	6101+-3	840	24

Table 4.1: Config 1

pop size	total value	max weight	no. of active
10	25569+10	2329	47
20	25569+12	2331	40
30	26380+9	2319	45
40	26705+10	2319	43
50	25662+3	2318	47
60	25662+3	2308	47
70	26471+21	2334	47
80	26490+21	2334	47
90	26810	2327	47
100	27468	2324	47

Table 4.2: Config 2

4 DIFFERENT POPULATION SIZES
5 COMPARISON OF RESULTS

Strength Comparison

- **Performance:**
  - For smaller instances or instances where a greedy algorithm or dynamic programming can provide an optimal solution efficiently, non-genetic algorithms may outperform GAs in terms of computational efficiency and accuracy.
  - However, as the problem size or complexity increases, GAs often show their strength in finding good (if not optimal) solutions within a reasonable time frame.
- **Solution Quality**:
  - GAs tend to find good solutions that may not always be optimal but are close to

the optimal solution. The diversity of solutions generated by GAs can be beneficial in finding globally good solutions.

  - Non-genetic algorithms, if capable of providing optimal solutions, guarantee the highest quality solutions for the problem instances they are designed for.

- **Adaptability**:
  - GAs are more adaptable and applicable to a wide range of optimization problems beyond the knapsack problem, making them a versatile choice for various domains.
  - Non-genetic algorithms like greedy or dynamic programming approaches are often problem-specific and may not be easily adaptable to other types of optimization problems.

Genetic Algorithm (GA):

- *Strengths:*
  - GAs are capable of finding good solutions to the knapsack problem, especially when dealing with a large number of items or complex problem instances.
  - GAs use evolutionary principles (selection, crossover, mutation) that enable effective exploration of the solution space, potentially leading to better solutions.
  - GAs are versatile and adaptable to various optimization problems, making them suitable for a wide range of applications.
- *Weaknesses:*
  - GAs may require a larger number of iterations (generations) to converge to an optimal or near-optimal solution, making them computationally intensive compared to certain non-genetic algorithms.
  - The effectiveness of a GA heavily depends on parameter tuning, such as population size, mutation rate, and crossover strategy.

Non-Genetic Algorithm (e.g., Greedy Algorithm, Dynamic Programming):

- *Strengths:*
  - Greedy algorithms, like the Fractional Knapsack or 0/1 Knapsack greedy approach, often provide optimal solutions for the knapsack problem with reasonable time complexity, especially for smaller problem instances.
  - Dynamic programming algorithms, like the 0/1 Knapsack using dynamic programming, can provide optimal solutions for a wide range of knapsack instances.
- *Weaknesses:*
  - Greedy algorithms may not always produce an optimal solution for the 0/1 knapsack problem, and the Fractional Knapsack solution may not be applicable in cases where only integer items are allowed.

– Dynamic programming approaches may have higher time and space complexity, making them less efficient for larger problem instances.

6 DESIGN OF A PROBLEM

DNA sequence alignment where each chromosome represents a potential alignment of DNA sequences. It is usually encoded as a string of characters where gaps, matches, and mismatches are represented. A higher fitness score indicates a better alignment.

Fitness function:

It can consider the number of matches, mismatches, and the positions of gaps in the alignment. The initial population is generated with a set of random alignments. Each alignment is represented as a chromosome.

Crossover operations:

They mimic genetic recombination and help generate new alignments. Different crossover techniques can be employed, such as onepoint, two-point, or uniform crossover. Mutation: introduces small random changes in the alignment, simulating genetic mutations. This helps in exploring and generating diversity in the population.

Chromosomes:

They are selected for reproduction based on their fitness scores. Higher fitness chromosomes have a greater chance of being selected and passing their genes to the next generation.

Stop Criterion:

If the fitness score doesn't show any significant changes/increase after 4-5 generations, we can stop the iteration