Genetic Algorithm for Knapsack Problem

Fazal Mahmud Niloy
Master of Data Science
University of Canberra
Canberra, Australia
u3228358@uni.canberra.edu.au

***Abstract*—This document briefly discusses how Genetic Algorithm works using the classic Knapsack Problem as an example.**

***Index Terms*—genetic, algorithm, knapsack, ga**

I. Introduction

The Knapsack Problem, a classic optimization challenge, requires selecting items with given weights and values to maximize total value without exceeding a knapsack's weight capacity. Its NP-hard nature makes it a prime candidate for heuristic approaches, particularly Genetic Algorithms (GAs). GAs, inspired by natural selection, iteratively evolve potential solutions, balancing exploration and exploitation in the search space. This paper explores the application of GAs to the Knapsack Problem, demonstrating their efficacy in finding optimal or near-optimal solutions efficiently, particularly for large problem instances where exact algorithms are impractical. Through a concise analysis, we aim to underscore the utility, performance considerations, and potential improvements of this metaheuristic approach in solving the Knapsack Problem.

II. Literature Review

Genetic Algorithm (GA) and Genetic Programming (GP) are both established paradigms within evolutionary computing, drawing inspiration from natural evolutionary processes. These paradigms share a foundational framework yet differ in solution representation and problem applicability.

A. Genetic Algorithm (GA)

Genetic Algorithms, conceptualized in the 1960s by Holland [1], encode potential solutions as fixed-length strings, often in binary format, to tackle optimization and search challenges. Goldberg's comprehensive overview [2] solidified GA's theoretical basis and practical utility. Research has since evolved, focusing on adaptive parameter tuning, hybridization with other optimization techniques, and parallel implementations. Empirical analyses, such as those by De Jong [3], have been instrumental in developing parameter setting guidelines, showcasing GA's effectiveness across diverse problem domains.

B. Genetic Programming (GP)

Genetic Programming extends GA principles, evolving variable-size, tree-structured programs or symbolic expressions. Introduced by Koza in the 1990s [4], GP has demonstrated its prowess in symbolic regression, automated theorem proving, and control structure evolution for robotics. Research in GP has delved into challenges like code bloat, the importance of modularity, and the development of domain-specific adaptations, strengthening its problem-solving capabilities.

C. Comparative Studies and Hybrid Approaches

Comparative analyses between GA and GP underscore their respective strengths and context-dependent trade-offs. GAs are lauded for their simplicity and wide applicability, whereas GP excels in evolving complex, structured solutions. Hybrid approaches, merging GA and GP elements, aim to capitalize on both paradigms' advantages, promising robust and scalable evolutionary algorithms.

In summary, GA and GP have made significant strides in evolutionary computing, each with distinct strengths and challenges. Their continued evolution promises further advancements in automated problem-solving and optimization.

III. Code Structure

The code to solve the GA problem follows a simple structure. After initializing the items and weight limit I have defined the GA parameters such as population size, crossover rate, mutation rate and max generation.

After that, I have defined 4 functions that are fundamentally required for a typical knapsack problem. Those are:

- **Fitness:** This function takes a chromosome from a population and evaluates the fitness on the basis of the weight constraint. If the total weight is more than the constraint then it will return zero indicating that the weight limit exceeds and if the total weight is within the limit then it returns the value of that chromosome.
- **Mutate:** It is used to introduce random variations into a chromosome, which is a binary list representing a potential solution in a genetic algorithm. The purpose of mutation in genetic algorithms is to maintain diversity in the population of solutions and to prevent premature convergence to suboptimal solutions. The function, it loops through each gene in the chromosome then using `'random.random()'` it compares the mutation rate

against the random value and changes it to either 0 or 1.

- **Crossover:** This function is a key component of genetic algorithms, performing the recombination of genetic material from two parent individuals to create two new offspring. This process is inspired by biological crossover observed in nature. Based on the crossover rate it splits the 2 parents from the population and creates 2 new children.
- **Selection:** This function is designed to select individuals from a population to create the next generation in a genetic algorithm. This selection is guided by the fitness of the individuals, with a higher chance of selection for individuals with higher fitness values.
- **Genetic Algorithm:** This is the base function. By taking binary values randomly we get our first population of 100 chromosomes. Then we perform selection, mutation etc step by step. We record the best, average and lowest fitness in each generation to plot the data.

IV. Results

In fig:1 we can see that the algorithm finds the best solution shortly after starting the program. Although, it tries different other combinations to find a better solution the best value remains over 200 on average. So, we can say that the algorithm is really efficient.

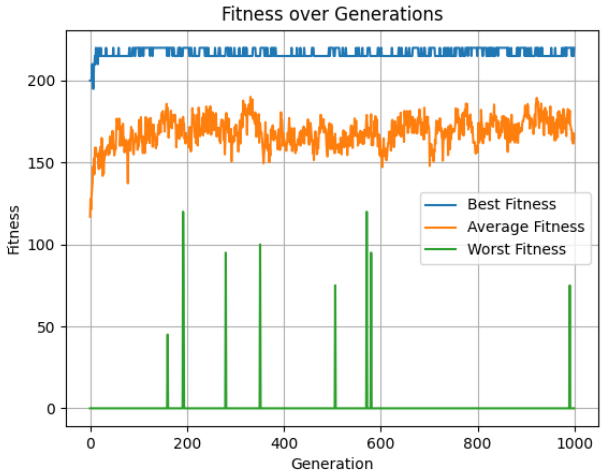

Fig. 1. Fitness through generations

V. Interesting Findings

Previously, I had solved the knapsack problem in an advanced algorithm class using dynamic programming and memoization. Although dynamic programming might be more computationally efficient, Genetic Algorithm introduced me to a new way of solving similar problems.

References

[1] Holland, J. H. (1975). Adaptation in Natural and Artificial Systems. University of Michigan Press.

[2] Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley.

[3] De Jong, K. A. (1975). An Analysis of the Behavior of a Class of Genetic Adaptive Systems. PhD thesis, University of Michigan.

[4] Koza, J. R. (1992). Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press.