COS 314 Assignment 2

Tayla Orsmond u21467456

# Comparing the Effectiveness of Utilizing a Genetic Algorithm (GA) and Ant Colony Optimization (ACO) in Solving the Knapsack Problem

## Introduction

The 1-0 knapsack problem is an optimization problem where a subset of n items, each with a weight and value, must be placed into a knapsack such that the total weight of the items in the knapsack does not exceed the capacity, and such that their combined value is maximized. In the simplest form of the problem the items are one dimensional and as such their shape / volume does not have to be considered. Items also cannot be placed more than once into the knapsack, nor can only a portion of the item exist in the knapsack.

This assignment aims to compare the effectiveness of using a Genetic Algorithm (GA) against Ant Colony Optimization (ACO) in solving instances of the knapsack problem.

## Genetic Algorithm (GA)

A genetic algorithm is a meta-heuristic that applies Darwin's principle of natural selection to produce optimal solutions. The algorithm begins by generating an initial population of individuals (chromosomes) and, through several generations, modifies and updates the population to reflect the evolving individuals. Offspring are produced from selected parents who form part of the new generation that is expected to be more fit than the previous one. The fitness of the individuals is evaluated by a fitness function, and the algorithm is biased towards fitter individuals such that, over time, the average fitness of the population increases, and the population converges towards an optimal solution. The algorithm stops when a specified criterion is met.

When designing a GA, several different methods and parameters must be carefully chosen and developed. In the context of the knapsack problem, these are discussed below.

### Individual Representation

An individual is represented as a chromosome made up of several genes. In this instance, individuals are represented as binary fixed-length strings (chromosomes) where each bit (gene) represents whether an item from the instance set is selected. The number of bits in the string equals the total number of items in the instance. For example, 10110 would indicate that the 1st, 3rd, and 4th items from the instance set are selected to be placed in the knapsack.

Individuals are initially randomly generated and can initially have a weight that exceeds the capacity of the knapsack. **The size of the population is controlled by the populationSize parameter**. This is dealt with in the fitness function.

### Fitness function

The knapsack problem aims to maximize the total weight of the knapsack without exceeding its capacity. As such, the fitness function sums the values of all the items in the knapsack with the best fitness being the maximum sum value. If the combined weight of the items exceeds the capacity of the knapsack, the individual is given a fitness of 0.0.

### Selection

Tournament selection is used. A random number of individuals are selected from the population (**defined by parameter tournamentSize**) that form the tournament. The best individual is then selected from the tournament to be a parent to produce offspring for the next generation. This is done for both parents.

Two perturbation operations are used (below) to produce offspring from the selected parents.

### Crossover

Single point crossover is used to recombine parents into children. **This is controlled by the parameter crossoverRate**. If the random number rolled is less than the crossoverRate, crossover occurs. A random point from 0 to the parent’s size is used as the crossover point. The parents then swap genes after that point to produce offspring. Single point crossover is effective in exploiting current best solutions while being simple to implement.

### Mutation

Bit-flip mutation is used to modify children further. **This is controlled by the mutationRate parameter**. If the random number rolled is less than the mutationRate, mutation occurs. A bit (gene) is randomly selected from the individual, and this is flipped (1 to 0 or 0 to 1) to either add or remove an item from the knapsack. This is effective in exploring the search space and allowing for diversity in individuals. The mutation rate is typically lower than the crossover rate, so the search does not become stochastic.

### Elitism

This algorithm uses elitism which allows a few (fittest) individuals from the old generation to replicate in the new generation unchanged. This allows for better exploitation of fitter individuals and is **controlled by the numElite parameter**.

### Replacement

Generational replacement is used. A new generation replaces the older generation if the average fitness of the newer generation is better. Generational replacement allows the algorithm to converge faster.

### Termination

The algorithm terminates after a fixed number of iterations (**numGenerations** **parameter**).

### Parameters

The initial parameters were taken from literature and are shown in the table below (Julstrom, 2005). These were then fine-tuned to improve each algorithm’s performance.

This was done offline, by starting with initial values for each parameter and slowly adjusting the parameters incrementally until there was no longer an improvement in the algorithm. The values were both increased and decreased to attempt to improve performance. “Improvement” was considered as an average enhancement in both algorithms across all the instances over multiple runs.

Below is a list of parameters and their initial and final values after fine-tuning:

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Initial Value** | **Final Value** |
| populationSize | Number of items |  |
| tournamentSize | 2 |  |
| crossoverRate | 0.7 |  |
| mutationRate | 0.3 |  |
| numElite | 1 |  |
| numGenerations | 10 \* Number of items |  |

The parameters….

## Ant Colony Optimization (ACO)

Use the A1 loader and summarizer and tweak

1. Develop a GA to solve the Knapsack problem
2. Develop ACO to solve the Knapsack problem
3. Measure Number of Optima
4. Measure Time
5. Present Results
6. Statistics
7. Conclusion

## References

Julstrom, B.A. (2005) “Greedy, genetic, and greedy genetic algorithms for the quadratic knapsack problem,” Proceedings of the 7th annual conference on Genetic and evolutionary computation, pp. 607–614. Available at: https://doi.org/10.1145/1068009.1068111.