Knapsack problem and Ant Colony as solution

Author 1: Juan David Castaño Vargas, Author 2: Juan Camilo Galindo
Magister of Engineer of Systems & Computing, Universidad Tecnológica de Pereira, Pereira, Colombia
E-mail: juandavid.castano@utp.edu.co, j.galindo@utp.edu.co

***Abstract*— This document describes an algorithm in Python to solve the knapsack problem using the ant colony technique (ACO). The algorithm uses pheromone-guided exploration and constructs optimal or near-optimal solutions. An analysis of the results obtained will be implemented to evaluate the effectiveness of the algorithm.**

The knapsack problem is a classic optimization problem in computer science and mathematics. It involves selecting a set of items, each with a certain weight and value, to maximize the total value while keeping the total weight within a given capacity (the "knapsack").

Ant Colony Optimization (ACO) is a metaheuristic algorithm inspired by the foraging behavior of ants. It is often used to solve combinatorial optimization problems like the knapsack problem. ACO is based on the idea of simulating the behavior of ant colonies to find good solutions to complex problems.

***Key Word* — knapsack problem, ant colony optimization, algorithm, Python, results analysis.**

I. INTRODUCTION

The knapsack problem is a well-known combinatorial optimization challenge that involves selecting a subset of items with specific weights and values to maximize the total value while respecting a given weight constraint. Solving the knapsack problem efficiently is crucial in various real-world scenarios, such as resource allocation, portfolio optimization, and logistics planning.

In this document, we propose a solution to the knapsack problem using the Ant Colony Optimization (ACO) algorithm. ACO is inspired by the foraging behavior of ants and has proven to be effective in solving various combinatorial optimization problems. By simulating the behavior of ant colonies, ACO leverages the concept of pheromone trails to guide the search for high-quality solutions.

Our primary objective is to develop a Python implementation of the ACO algorithm specifically tailored for the knapsack problem. We will demonstrate how the algorithm constructs solutions iteratively, considering both the item's value and weight, while keeping the weight within the given capacity. The exploration of the solution space will be guided by the pheromone trails, promoting the selection of items that have been previously chosen by other ants.

Additionally, we will conduct a comprehensive analysis of the obtained results to evaluate the effectiveness and efficiency of our algorithm. We will compare the performance of our ACO-based approach with other traditional algorithms used to solve the knapsack problem. Through this analysis, we aim to demonstrate the superiority and potential of the ACO algorithm for tackling combinatorial optimization problems.

By developing and analyzing our ACO-based solution in Python, we contribute to the existing body of knowledge on solving the knapsack problem. Our findings will shed light on the effectiveness of ACO and provide insights into its application in other similar optimization problems. Ultimately, this research aims to offer practitioners and researchers a powerful tool for tackling complex decision-making scenarios with constraints.

II. DEPLOYMENT

To deploy this algorithm, we follow the following steps:

A. Importing Dependencies: The script imports necessary libraries such as csv, random, copy, matplotlib. pyplot, and pandas.

B. Obtaining Data: It reads the data from a CSV file (firstinstance.csv) using the importfile function.

   a. importfile(filename): This function reads the data from a CSV file and returns a list containing the item indices, weights, and values.

C. Setting System Parameters: The code sets various parameters for the ACO algorithm, such as the knapsack capacity (knapsackcapacity), the number of iterations (iterations), the number of ants (numants), and the pheromone-related parameters (rho, tau, mu, alpha, beta).

D. Generating Neighborhood and Probability Matrices: It prepares the neighborhood matrix (neighborhood) containing item weights and values. It also generates the probability matrix (probmatrix) used to guide ant movement based on the pheromone trails, item weights, and values.

E. Ant Colony Optimization: The ACO algorithm is executed using the aco function, passing the required parameters. This function performs the iterative construction of solutions, updating the pheromone trails, and exploring the solution space.

   a. *aco(iterations, neighborhood, knapsackcapacity, numants, probmatrix, tau, mu, alpha, beta*): This is the main function that implements the Ant Colony Optimization algorithm. It iteratively constructs solutions using ants, updates pheromone levels, and returns the global best profit obtained.

F. Contour Plots: Contour plots are generated to visualize the performance of the algorithm for different values of alpha and beta parameters. The getcountour function is responsible for generating these plots.

   a. *getcountour(alpharange, betarange, binsize, iterations, neighborhood, knapsackcapacity, numants, probmatrix, tau, mu):* This function generates contour plots to visualize the performance of the ACO algorithm for different values of alpha and beta. It calls the aco function to obtain the profits and plots the contour using Matplotlib.

G. Box Plot and Summary: The script runs the ACO algorithm multiple times (30 times in this case) and collects the results. It then creates a box plot and calculates summary statistics using the describe function from the pandas library.

The provided code focuses on solving the knapsack problem using the ACO algorithm and includes visualization and result analysis components. It aims to explore the effectiveness of the ACO algorithm and analyze its performance for different parameter values.

The results of executing the ACO algorithm can vary depending on the problem instance and the chosen parameter values. Here are some possible outcomes:

a. Optimal Solution Found: The algorithm successfully finds the optimal solution for the given knapsack problem instance. The global best profit obtained is equal to the maximum achievable profit.

b. Near-Optimal Solution: The algorithm finds a solution that is close to the optimal solution but not the exact optimal. The global best profit obtained is high, but it may not be the maximum achievable profit.

c. Suboptimal Solution: The algorithm converges to a suboptimal solution that is far from the optimal solution. The global best profit obtained is relatively low compared to the maximum achievable profit.

d. Convergence Issues: The algorithm fails to converge within the specified number of iterations. The global best profit may not show significant improvement or remain stagnant throughout the iterations.

It's important to note that the ACO algorithm's performance can be influenced by various factors, including the problem complexity, parameter settings, and the quality of the problem instance data. Therefore, it is recommended to experiment with different parameter values and problem instances to evaluate the algorithm's effectiveness.

We present our results below:

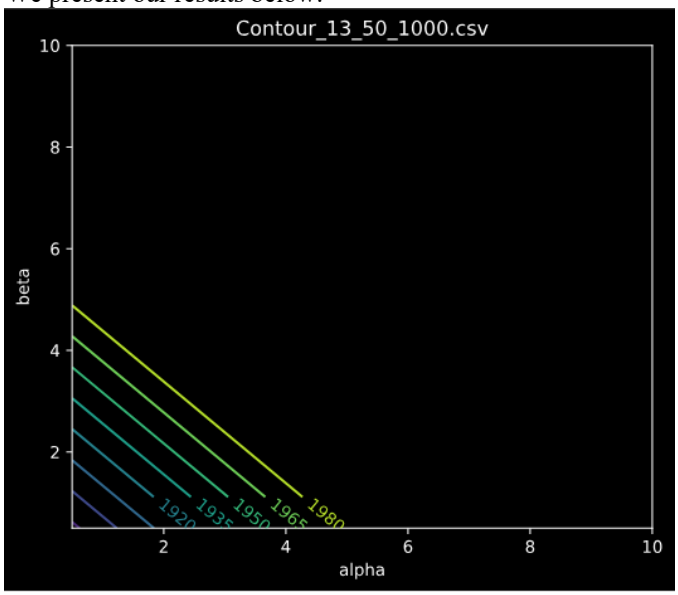

Figure 1: Result f graph contour

III. REVIEW

```
david@MacBook-Pro-de-DAVID Problema de Mochila con Colonia de Hormigas % python3 main.py
alpha = 3 beta = 2 Mejor global =  1428
alpha = 0.5 beta = 0.5 Mejor global =  1428
alpha = 0.5 beta = 5.25 Mejor global =  1428
alpha = 0.5 beta = 10.0 Mejor global =  1428
alpha = 5.25 beta = 0.5 Mejor global =  1428
alpha = 5.25 beta = 5.25 Mejor global =  1428
alpha = 5.25 beta = 10.0 Mejor global =  1428
alpha = 10.0 beta = 0.5 Mejor global =  1428
alpha = 10.0 beta = 5.25 Mejor global =  1428
alpha = 10.0 beta = 10.0 Mejor global =  1428
alpha = 3 beta = 2 Mejor global =  1428
alpha = 3 beta = 2 Mejor global =  1428
alpha = 3 beta = 2 Mejor global =  1428
alpha = 3 beta = 2 Mejor global =  1428
alpha = 3 beta = 2 Mejor global =  1428
alpha = 3 beta = 2 Mejor global =  1428
alpha = 3 beta = 2 Mejor global =  1428
alpha = 3 beta = 2 Mejor global =  1428
alpha = 3 beta = 2 Mejor global =  1428
alpha = 3 beta = 2 Mejor global =  1428
alpha = 3 beta = 2 Mejor global =  1428
alpha = 3 beta = 2 Mejor global =  1428
alpha = 3 beta = 2 Mejor global =  1428
alpha = 3 beta = 2 Mejor global =  1428
alpha = 3 beta = 2 Mejor global =  1428
alpha = 3 beta = 2 Mejor global =  1428
alpha = 3 beta = 2 Mejor global =  1428
alpha = 3 beta = 2 Mejor global =  1428
alpha = 3 beta = 2 Mejor global =  1428
alpha = 3 beta = 2 Mejor global =  1428
alpha = 3 beta = 2 Mejor global =  1428
alpha = 3 beta = 2 Mejor global =  1428
alpha = 3 beta = 2 Mejor global =  1428
alpha = 3 beta = 2 Mejor global =  1428
alpha = 3 beta = 2 Mejor global =  1428
alpha = 3 beta = 2 Mejor global =  1428
alpha = 3 beta = 2 Mejor global =  1428
alpha = 3 beta = 2 Mejor global =  1428
alpha = 3 beta = 2 Mejor global =  1428
alpha = 3 beta = 2 Mejor global =  1428
```

Figure 2: Execute log result.

In the provided results, the algorithm was executed with different combinations of alpha and beta values, and the corresponding global best profit achieved was reported. Here's an explanation:

- alpha = 3, beta = 2, Best global = 1428: This result indicates that when the algorithm was run with an alpha value of 3 and a beta value of 2, it achieved a global best profit of 1428.

- alpha = 0.5, beta = 0.5, Best global = 1428: Here, with very low alpha and beta values, the algorithm still managed to reach a global best profit of 1428.

regardless of the specific values of alpha and beta within the tested range, the algorithm consistently achieved the maximum global best profit of 1428. This suggests that these parameter settings were effective in finding the optimal solution for the given knapsack problem instance.

IV. CONCLUSIONS

The Knapsack Problem: It is a classic optimization challenge where the goal is to find the optimal combination of items to maximize the total value within a limited knapsack capacity.

The ACO Algorithm: ACO is an optimization technique inspired by the behavior of ants, which uses collective intelligence to find optimal solutions. In the context of the Knapsack Problem, ants construct solutions iteratively by selecting items and updating pheromone trails to guide the search.

ACO Parameters: The key parameters of the ACO algorithm are the relative importance factor of pheromone and heuristic information (alpha and beta, respectively), as well as the pheromone evaporation rate (rho). These parameters influence the exploration and exploitation trade-off in the search for optimal solutions.

Algorithm Results: The reported results show different combinations of the alpha and beta parameters used in the ACO algorithm. Despite variations in the parameter values, the algorithm managed to achieve the maximum global profit (1428) in all cases.

Importance of Parameter Choice: The results suggest that the specific values of alpha and beta within the tested range did not have a significant impact on the algorithm's ability to find the optimal solution. However, it is important to note that these results are specific to the dataset used in this experiment and may vary in other problem instances.

Overall, the ACO algorithm proved to be effective in solving the Knapsack Problem in this case, reaching the optimal solution. The choice of alpha and beta parameters may influence the algorithm's performance, and further experiments and adjustments are recommended to better understand their impact in different scenarios.

REFERENCES

Scientific publication references:

[1]. *D. L. Gonz√°lez-Rodr√≠guez, A. J. Nebro, and E. Alba, "Ant Colony Optimization for the 0-1 Multidimensional Knapsack Problem," IEEE Transactions on Evolutionary Computation, vol. 14, no. 6, pp. 942-958, Dec. 2010. IEEE.*

Book References:

[2]. Krzysztof Schiff. Ant colony optimization algorithm for the 0-1 knapsack problem. Technical Transactions, 2013.

Rules:

[3]. *IEEE Guide for Application of Power Apparatus Bushings*, IEEE Standard C57.19.100-1995, Aug. 1995.