**Report: Simulated Annealing for Knapsack Problem**

**Introduction**

The goal of this report is to analyze and explain the implementation of a simulated annealing algorithm for solving the knapsack problem. The algorithm aims to find the best combination of items to maximize the total value while respecting the knapsack's weight capacity. The implementation is done in Java and utilizes a greedy approach to find an initial feasible solution. The algorithm then uses simulated annealing to improve upon this initial solution by exploring neighboring solutions and accepting changes based on a probability.

**Implementation Details**

**a. Initializing the Solution State Variables:**

The algorithm initializes the solution state variables, namely currentSolution, bestSolution, currentValue, and bestValue. These variables are used to keep track of the current and best solutions found during the algorithm's execution.

**b. Finding the Initial Greedy Solution:**

Before starting the simulated annealing process, the algorithm finds an initial feasible solution using a greedy approach. It calculates the ratio of value to weight for each item and selects items with the highest ratios until reaching the knapsack's weight capacity. This greedy solution serves as the starting point for the simulated annealing algorithm.

**c. Simulated Annealing Process:**

The algorithm performs the simulated annealing process to improve upon the initial greedy solution.

It starts by setting the initial temperature to a predefined maximum temperature and defines a cooling rate.

Inside the main loop, the algorithm generates a new neighbor solution by randomly selecting an item and toggling its inclusion status.

It then calculates the value and capacity of the neighbor solution and checks if the capacity constraint is violated.

If the capacity is violated, the algorithm reverts the change to the neighbor solution.

Next, the algorithm calculates the acceptance probability based on the current value, neighbor value, and temperature. If the acceptance probability is greater than a randomly generated number, the current solution is updated to the neighbor solution.

The algorithm also keeps track of the best solution found so far.

After each iteration, the current and best values are printed, and the temperature is decreased according to the cooling rate.

The process continues until the temperature reaches a threshold value.

**Execution Time Analysis**

To analyze the execution time of the algorithm, it is important to measure how it changes with different problem sizes. This can be achieved by running the algorithm on various datasets and recording the execution times. The datasets should vary in terms of the number of items and their associated values and weights.

**Execution Time and Solution Quality Analysis**

In addition to measuring the execution time, it is also essential to analyze how the execution time and solution quality change when the difference between the starting temperature and stopping temperature is increased. This can be done by running the algorithm on the same datasets with different temperature configurations and comparing the execution times and the quality of the solutions obtained.

**a. Clone the Given Repository:**

Begin by cloning the given repository using Git. This will provide access to the necessary code and files for running the algorithm.

**b. Code the Algorithm:**

Implement the algorithm based on the provided classes and methods. Ensure that the implementation accurately reflects the steps and logic described in this report.

**c. Push the Project Code to a Private Repository:**

Once the implementation is complete, push the project code to a private repository. Ensure that the repository includes all the required files and explanations of the parts written by you.

**d. Share the Repository with the Course TA's:**

Share the private repository with the course TA's to allow them access to the code and files for evaluation and feedback.

**e. Dataset Execution and Analysis:**

Run the algorithm on various datasets with different problem sizes.

Measure and record the execution times for each dataset.

Analyze the data to observe how the execution time changes as the problem size increases.

**f. Temperature Configuration Execution and Analysis:**

Run the algorithm on the same datasets with different temperature configurations.

Vary the difference between the starting temperature and stopping temperature.

Measure and record the execution times for each temperature configuration.

Compare the execution times and solution qualities to identify trends and patterns.

**Conclusion**

In conclusion, the implemented simulated annealing algorithm offers a method for solving the knapsack problem by iteratively improving upon an initial greedy solution. The algorithm explores neighboring solutions and accepts changes based on a probability determined by the acceptance criterion. By analyzing the execution time with varying problem sizes and temperature configurations, it is possible to gain insights into the algorithm's scalability and the impact of temperature on the solution quality.

**Gökçe Büyükköprü – S021730**

**Zeynep Ada Yalçın – S024491**