Università della Svizzera italiana

Comparative Analysis of Algorithmic Strategies For the 0/1 Knapsack Problem

Student: Valentino Belotcaci

Advisor: Antonio Carzaniga

Motivation

This study focuses on implementing and comparing various algorithmic approaches to solving the 0/1 Knapsack Problem.
The Knapsack problem has many important applications in various areas in engineering and logistics. However, given the NP-hard nature of the Knapsack Problem, finding efficient and scalable solutions is likely impossible in general. We therefore have to resort to various heuristics, that may be effective for some classes of problems. Comparing these different algorithms provides information of their performance, helping identify the most suitable methods for different scenarios. This is important when dealing with large-scale benchmarks and the need for a statistical evaluation of algorithmic efficiency.

Mathematical formulation

- Let n denote the number of items available.
- Each item i has a value $v_i > 0$ and a weight $w_i > 0$.
- The capacity of the knapsack is W.
- Define x_i as a binary decision variable where:

$$x_i = \begin{cases} 1 & \text{if item } i \text{ is included in the knapsack,} \\ 0 & \text{otherwise.} \end{cases}$$

The objective is to maximize the total value of items included in the knapsack, subject to the weight constraint:

$$\text{Maximize } Z = \sum_{i=1}^{n} v_i x_i$$

$$\text{Subject to: } \sum_{i=1}^{n} w_i x_i \leq W, \quad x_i \in \{0,1\}, \forall i \in \{1,2,\ldots,n\}.$$

Benchmarks

The study was conducted by generating **5000** benchmarks of different sizes and retrieving other **3000** from a research paper.
Each benchmark was ran using dynamic programming, greedy algorithm, branch and bound, Martello-Toth, Simulated Annealing, ACO and Integer Linear Programming.
After running each algorithm we registered important information such as memory usage, time taken and best value found.
Subsequently we conducted a statistical analysis on each algorithm computing the mean, median, standard deviation, error %,total time or memory…
The created benchmarks, were structured in different sizes(**from 10 to 10,000**) and different capacities(**from 10 to 1,000**) choosen randomically.
Each benchmark line was created following the structure of the research paper's benchmarks, having for each line the id as first value, weight as second (**from 1 to half knapsack's capacity**) and profit as third(**from 1 to 100**).

Greedy pseudocode

```
1. Define an array of structures for items, including profit, weight, and ratio.
2. Calculate value-to-weight ratio for each item.
3. Sort items by ratio in descending order using a comparator function.
4. Initialize total_value = 0 and current_capacity = W (knapsack capacity).
5. For each item (in sorted order):
      If weight <= current_capacity:
          Add item to knapsack.
          Update total_value and current_capacity.
6. Return total_value.
```

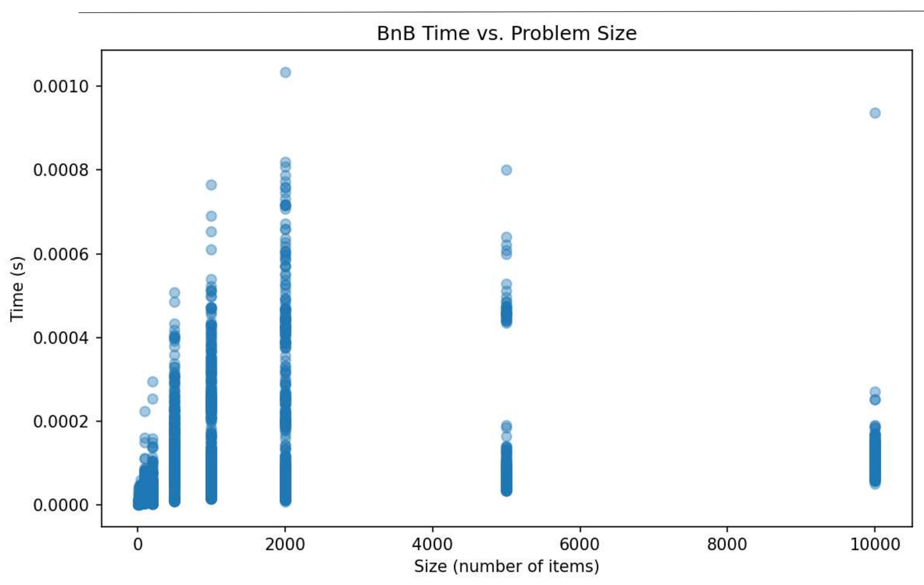
Simulated annealing pseudocode

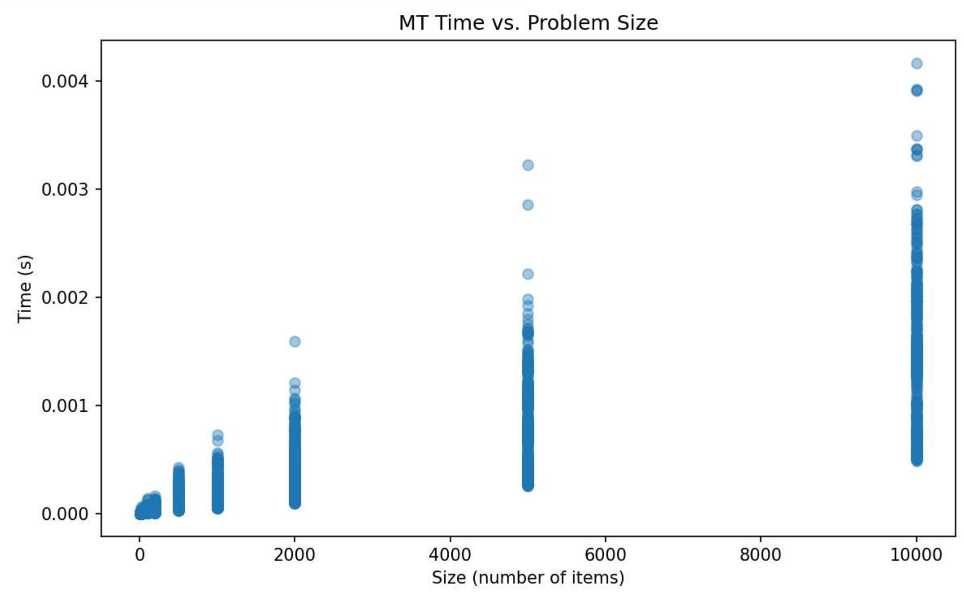
```
1. Initialize the solution using a greedy heuristic and temperature T.
2. While T > stopping_temperature:
      For a fixed number of neighbors:
          Generate a neighbor solution by flipping one item's inclusion.
          Calculate the change in profit (ΔP).
          If ΔP > 0 or exp(ΔP / T) > random(0, 1):
              Accept the neighbor solution as the current solution.
              Update the best solution if the current solution improves it.
      Decrease temperature using a cooling rate.
3. Return the best solution found.
```

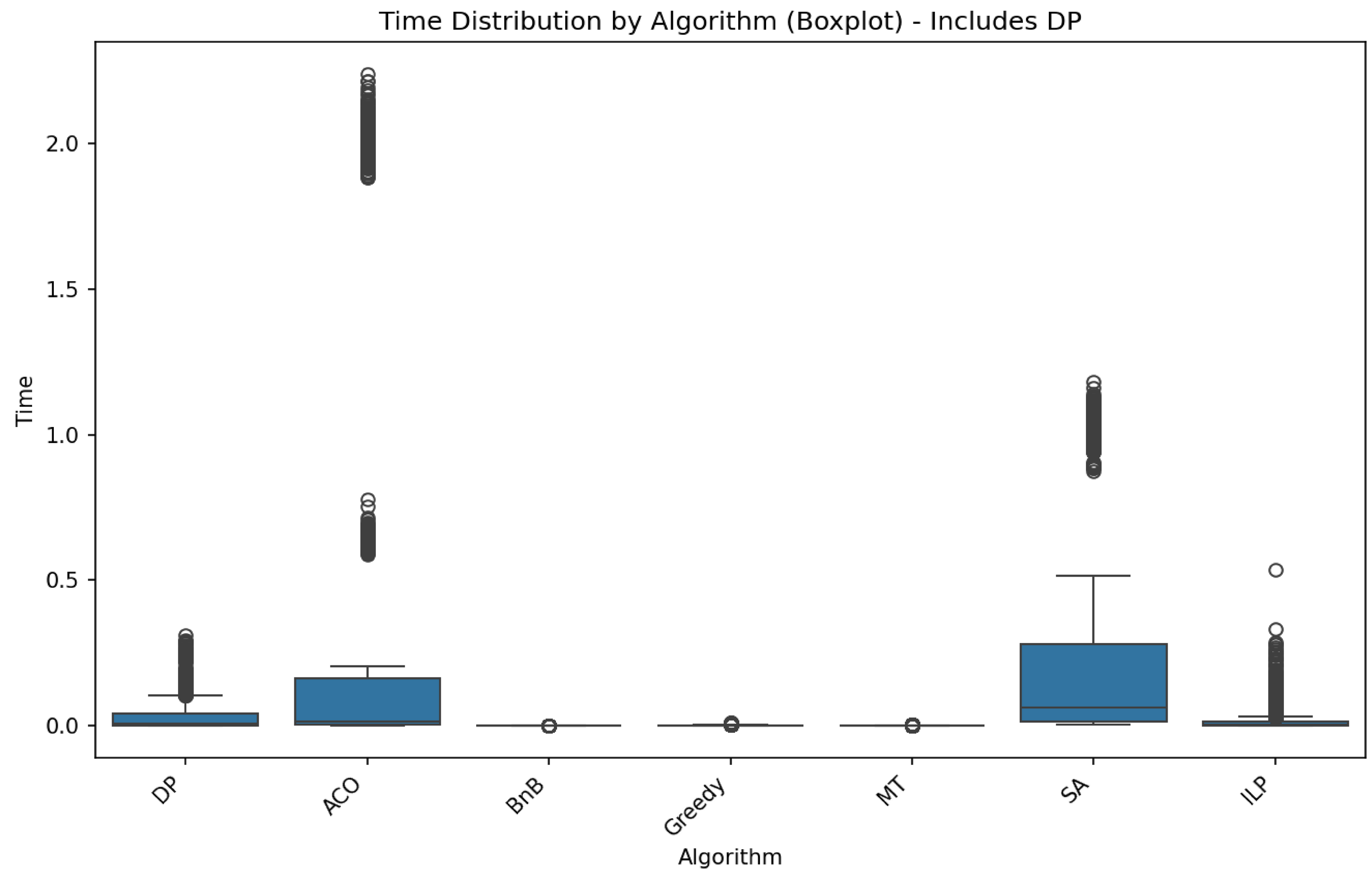
ILP pseudocode

```
1. Read the number of items (n), weights, values, and knapsack capacity from input.
2. Create a GLPK problem instance and set it to maximize the objective function.
3. Add one row (constraint):
       a. Total weight of selected items <= capacity.
4. Add n columns (decision variables):
       a. Set bounds for each variable as 0 <= x[i] <= 1.
       b. Set the variable type to binary (0/1).
       c. Set the profit of each item as the objective coefficient.
5. Define the constraint matrix:
       a. Each item's weight contributes to the weight constraint.
6. Load the constraint matrix into the GLPK problem.
7. Run the simplex method to solve the relaxed problem.
8. Run the integer optimizer to solve the ILP.
9. If no timeout occurs, retrieve the optimal solution and profit.
10. Return the best solution and profit.
```

Results







Greedy and dp results, hard benchmarks

Metric	Time results (s)	Memory Results(KB)	Error Results (%)
Total	0.627559	**6.18 GB**	-
Mean	**0.000211**	2175.96	**0.854**
Median	0.000194	2176	0.0072
Standard Deviation	0.000101	2347	2.048
Minimum	0.000082	2048	0
Maximum	0.001026	2176	**11.32**
25th Percentile	0.000146	-	0.000018
75th Percentile	0.000258	-	0.2515

Metric	Time results (s)	Memory Results(KB)	Error Results (%)
Total	20691	**3220 GB**	-
Mean	**19.176**	3129328	0
Median	14.756	3127936	0
Standard Deviation	15.072	1105378	0
Minimum	1.815	1564416	0
Maximum	81.999	4691200	0
25th Percentile	6.531	-	0
75th Percentile	29.088	-	0