MI-PAA: Assignment #1

Petr Hanzl
hanzlpe2@fit.cvut.cz

Czech Technical University - Faculty of Information Technology — October 29, 2018

Tasks

1. Naprogramujte řešení 0/1 problému batohu hrubou silou (tj. nalezněte skutečné optimum). Na zkušebních datech pozorujte závislost výpočetního času na n.
2. Naprogramujte řešení problému batohu heuristikou podle poměru cena/váha. Pozorujte:
   (a) závislost výpočetního času na n. Grafy jsou vítány (i pro exaktní metodu),
   (b) průměrnou a maximální relativní chybu (tj. zhoršení proti exaktní metodě) v závislosti na n.

Introduction

The knapsack problem is a basic algoritmic problem, taught at universities all around the world. Therefore, it is considered to be the basic knowledge of every computer science graduate. The knapsack problem has various modifications, the most common version is known as 0/1 knapsack problem.

0/1 knapsack problem

Given the knapsack weight capacity of W kilograms and a set of N items, each has its own unique index $i \in N^+$, a value of $v_i \in R^+$ money and weight of $w_i \in R^+$ kilograms. Determine which items should be put into the knapsack to maximize its value so that the sum of the weights is less than or equal to the weight capacity.

$$\text{maximize} \sum_{i=1}^{n} v_i x_i \tag{1}$$

$$\text{subject to} \sum_{i=1}^{n} w_i x_i \leq W \text{ and } x_i \in \{0, 1\} \tag{2}$$

Info: x_i represents whether the i-th item is present in the knapsack or not.

1 Bruteforce

To find the optimal solution for 0/1 knapsack problem, it means finding such a combination of binary elements x_i of a vector, such that no other vector with different elements has higher value.

Apparently, every vector of n binary values has 2^n possible combinations, thus naive brute force algorithm can find optimal solution for worst case in exponential time of 2^n, because every combination has to be checked by the algorithm. Even though some algorithms have lower computation times in average, none are discussed in this assignment as it is outside the scope of this problem.

Algorithm 1: `01Knapsack-Bruteforce`

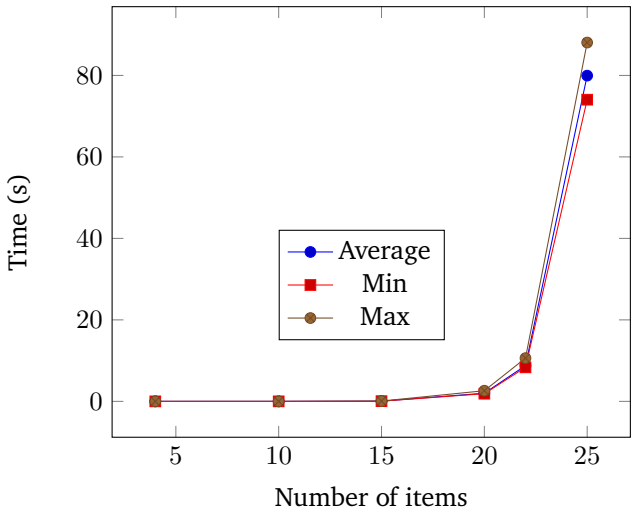
Input: $(maxWeight, items[N])$
$maxWeight$ - a weight capacity;
$items[N]$ - a numbered set of N items, each with $value$ and $weight$
Result: $(maxValue, maxConfiguration)$
$maxValue$ - an optimal value of the knapsack;
$maxConfiguration$ - an integer whose bits represent presence of an i-th item in the knapsack

---

$maxValue \leftarrow 0$;
$maxConfiguration \leftarrow 0$;
for $configuration \leftarrow 0..2^N$ **do**
  $tmpValue \leftarrow 0$;
  $tmpWeight \leftarrow 0$;
  for *every* i*-th bit of* $configuration$ **do**
    if i*-th bit is set* **then**
      $tmpValue \leftarrow tmpValue + items[i].value$;
      $tmpWeight \leftarrow tmpWeight + items[i].weight$;
    end
  end
  if $(tmpWeight \leq maxWeight) \wedge (tmpValue \geq maxValue)$ **then**
    $maxValue \leftarrow tmpValue$;
    $maxConfiguration \leftarrow configuration$;
  end
end
return $(maxValue, maxConfiguration)$;

1.1 Results

Graph 1 - Time series of 01Knapsack-Bruteforce algorithm



N items (Hz)	Avg. time (s)	Min. time (s)	Max. time (s)
4	0.000064	0.000039	0.0003
10	0.001513	0.001041	0.005197
15	0.050413	0.046177	0.076715
20	1.993293	1.881023	2.613521
22	8.771025	8.349322	10.6071
25	79.94936	74.04401	88.06801

2 Heuristic approach

Finding the optimal solution might be impractical for applications where the goal is to find a satisfactory but immediate solution. Heuristic based algorithms can be used to speed up the process of finding these solutions. A good example of heuristic approach for 0/1 knapsack problem is to put most valuable items into knapsack at first.

The required heuristic for this assigment is to use ratio of value to weight. The algorithm below continously puts items into knapsack until the sum of weights of all the items in knapsack plus new possibly added item, is not greater than knapsack weight capacity. From the given results of heuristic algorithm, the average and the maximal approximation errors of the heuristic and bruteforce method are calculated and plotted onto a separated graph.

Algorithm 2: `01Knapsack-Heuristic`

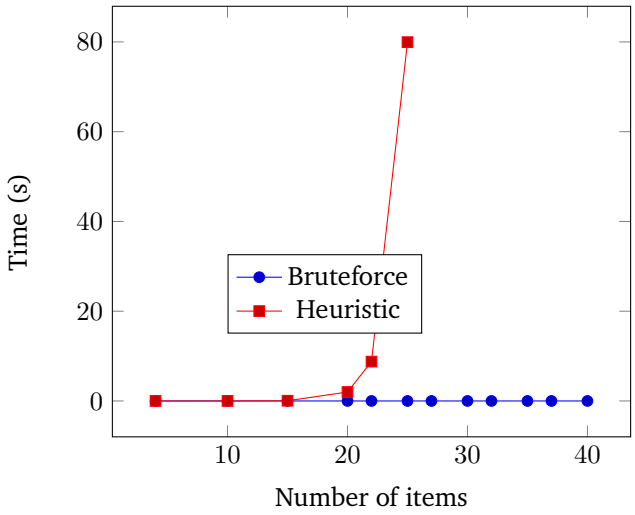
Input: $(maxWeight, items[N])$
$maxWeight$ - a weight capacity;
$items[N]$ - a numbered set of N items, each with $value$ and $weight$
Result: $(maxValue, maxConfiguration)$
$maxValue$ - a suboptimal value;
$maxConfiguration$ - an integer whose bits represent presence of an i-th item in the knapsack

---

$sortedItems \leftarrow sortItemsByRatioOfValueToWeight(items)$;
$maxValue \leftarrow 0$;
$maxConfiguration \leftarrow arrayOfSize(N)$;
$maxConfiguration.fillWith(0)$;
$tmpWeight \leftarrow 0$
for $i = 0..N$ **do**
  if $(tmpWeight + sortedItems[i].weight \leq maxWeight)$ **then**
    $maxValue \leftarrow maxValue + sortedItems[i].value$;
    $maxConfiguration[i] \leftarrow 1$;
  end
end
return $(maxValue, maxConfiguration)$;

2.1 Results

Graph 2 - Comparison of bruteforce and heuristic method



N items	Bruteforce Avg. time time (s)	Heuristic Avg. time (s)
4	0.000064	0.00003
10	0.001513	0.000033
15	0.050413	0.000049
20	1.993293	0.000055
22	8.771025	0.000053
25	79.94936	0.00008
27	NaN	0.000065
30	NaN	0.000084
32	NaN	0.000087
35	NaN	0.000073
37	NaN	0.000089
40	NaN	0.000084

3 Conclusion

The operating time of the bruteforce algorithm has exponential growth that is also shown on the graph 1.

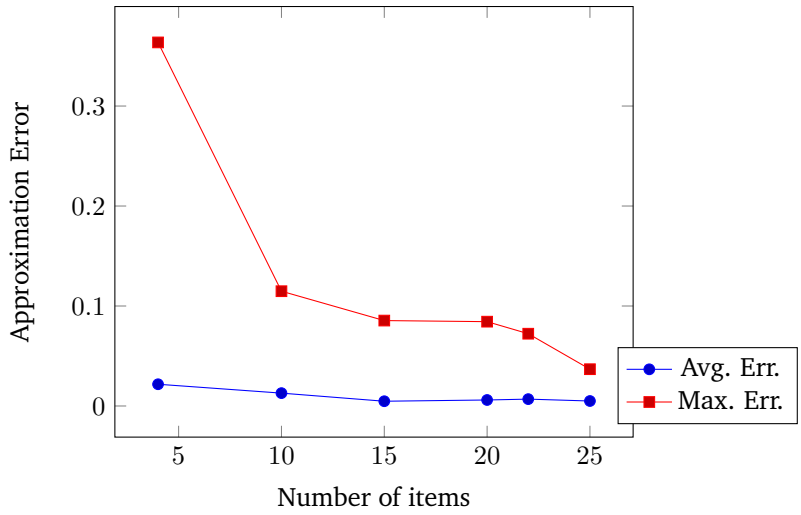
From the observations of the graph 2 and the table underneath, it can be see that the computation time for the heuristic method almost did not grow for larger instances at least in the comparison with the bruteforce method. Hence, its time complexity is linear.

Also, it seems that the heuristic algorithm with higher number of available items tends to return solutions that are closer to optimal solution, which results from lower average and also lower maximal approximation error that is plotted in the graph 3.

Because of the time complexity of the larger instances, that take approximately whole days to compute, computations for these instances are not finished yet. This is also the reason why these instances are not included in this assigment.

All solutions, times and approximation errors are stored in **/problems/data.json**. How to run the solver is described in **README.md**.

Graph 3 - Approximation errors of the heuristic method (value/weight ratio)



N items	Max. Error	Avg. Error
4	0.363636	0.021745
10	0.1148	0.012861
15	0.085427	0.004758
20	0.084337	0.006
22	0.072289	0.006866
25	0.036789	0.004983