0-1 Knapsack Problem: BFS and DFS

Nemanja Antonic

January 2022

1 The problem

1.1 Representation of a solution

1.1.1 Variables

The set of variables X={X_1,...,X_i,...X_n} where n is the number of total items and each variable represents an item to be put in the knapsack.

1.1.2 Domain

The domain of the i-th variable D_i={*true*, *false*} as an item can either be in the knapsack or not.

1.2 Objective function

To define the objective function, the parameter b_i needs to be defined as the benefit of the variable X_i. Hence: max $\sum_{i=1}^{n} X_{\mathrm{i}} * b[i]$

1.3 Constraint

The total capacity of the knapsack K shall never be exceeded by the items selected as a solution. Each item has a weight w_i.
$\sum_{i=1}^{n} X_{\mathrm{i}} * w_{\mathrm{i}} <= K$

2 Comparison of the algorithms

2.1 Time complexity

The theoretical time complexity of BFS is O(b^d) where b is the maximum branching factor and d is the depth of the shallowest solution, while for DFS it's O(b^m) where m is the maximum depth. In my case b = n = 11 (since on the first expansion of the tree n children are generated) and d = 8 (practical result from running the algorithm), while m = 11 (the maximum depth is equal to n as the left-most node will be expandend from 1 to n). The corresponding time usages of the algorithms are, respectively: 0.06s and 0.09s which is in line with the theoretical outcome.

2.2 Space complexity

The theoretical space complexity of BFS is O(b^d) while for DFS it's O(b*m). It's difficult to assess the actual space usage but theoretically it is lower for the latter: $11^8 = 214358881$ for BFS and $11 * 11 = 121$ for DFS.