

KNAPSACK PROBLEM

Let us now discuss how we can apply the branch-and-bound technique to solving the knapsack problem. Given n items of known weights w_i and values $v_i, i = 1, 2, \dots, n$, and a knapsack of capacity W , find the most valuable subset of the items that fit in the knapsack. It is convenient to order the items of a given instance in descending order by their value-to-weight ratios. Then the first item gives the best payoff per weight unit and the last one gives the worst payoff per weight unit, with ties resolved arbitrarily:

$$v_1/w_1 \geq v_2/w_2 \geq \dots \geq v_n/w_n.$$

It is natural to structure the state-space tree for this problem as a binary tree constructed as follows. Each node on the i th level of this tree, $0 \leq i \leq n$, represents all the subsets of n items that include a particular selection made from the first i ordered items. This particular selection is uniquely determined by the path from the root to the node: a branch going to the left indicates the inclusion of the next item, and a branch going to the right indicates its exclusion. We record the total weight w and the total value v of this selection in the node, along with some upper bound ub on the value of any subset that can be obtained by adding zero or more items to this selection.

Item	Weight	Value	value / weight	capacity
1	4	\$40	10	W = 10
2	7	\$42	6	
3	5	\$25	5	
4	3	\$12	4	
	w=19	v=119	$v_{i+1}/w_{i+1}=25$	

A simple way to compute the upper bound ub is to add to v , the total value of the items already selected, the product of the remaining capacity of the knapsack $W - w$ and the best per unit payoff among the remaining items, which is v_{i+1}/w_{i+1} :

$$ub = v + (W - w)(v_{i+1}/w_{i+1}).$$

$$= 0 + (10 - 0)(10) = 100$$

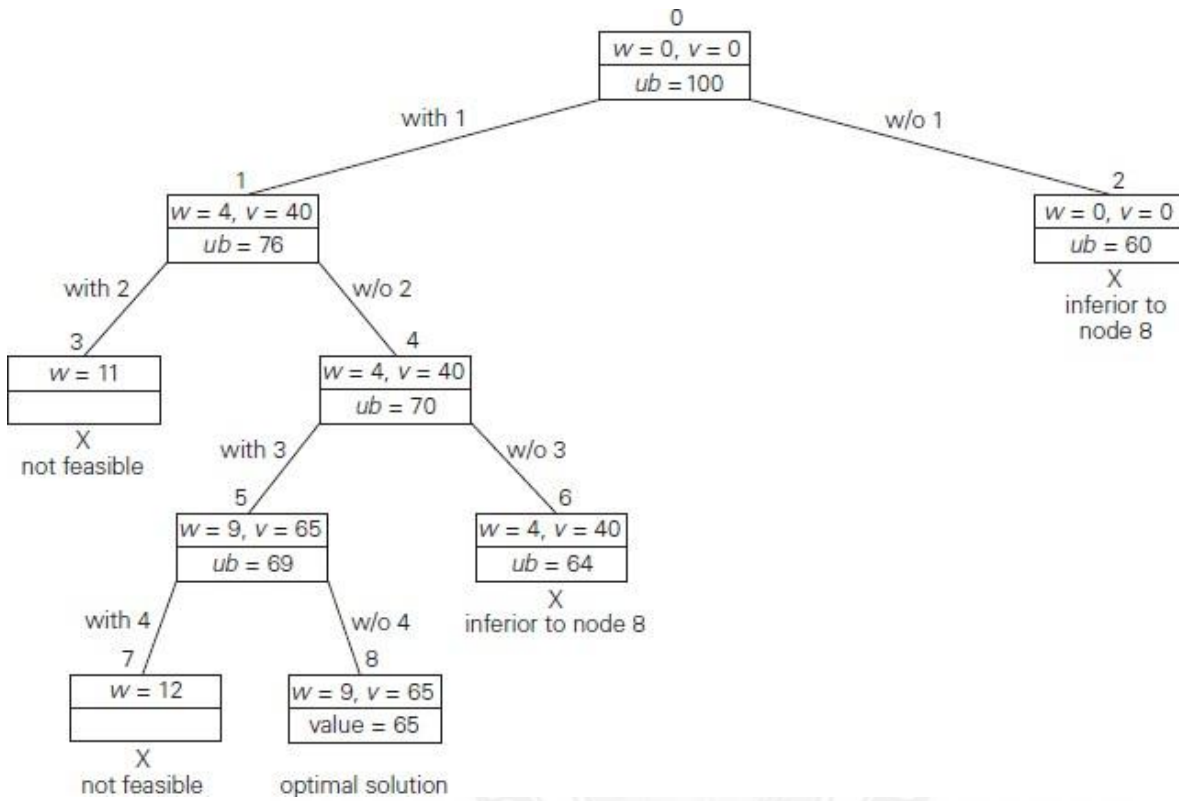


FIGURE State-space tree of the best-first branch-and-bound algorithm for the instance of the knapsack problem.

At the root of the state-space tree no items have been selected as yet. Hence, both the total weight of the items already selected w and their total value v are equal to 0. The value of the upper bound computed by formula (12.1) is \$100. Node 1, the left child of the root, represents the subsets that include item 1. The total weight and value of the items already included are 4 and \$40, respectively; the value of the upper bound is $40 + (10 - 4) * 6 = \$76$. Node 2 represents the subsets that do not include item 1. Accordingly, $w=0, v=\$0$, and $ub=0+(10-0) * 6 = \$60$. Since node 1 has a larger upper bound than the upper bound of node 2, it is more promising for this maximization problem, and we branch from node 1 first. Its children—nodes 3 and 4—represent subsets with item 1 and with and without item 2, respectively.

Since the total weight w of every subset represented by node 3 exceeds the knapsack's capacity, node 3 can be terminated immediately. Node 4 has the same values of w and v as its parent; the upper bound ub is equal to $40 + (10 - 4) * 5 = \$70$. Selecting node 4 over node 2 for the next branching (why?), we get nodes 5 and 6 by respectively including and excluding item 3. The total weights and values as well as the upper bounds

for these nodes are computed in the same way as for the preceding nodes. Branching from node 5 yields node 7, which represents no feasible solutions, and node 8, which represents just a single subset $\{1, 3\}$ of value \$65. The remaining live nodes 2 and 6 have smaller upper-bound values than the value of the solution represented by node 8. Hence, both can be terminated making the subset $\{1, 3\}$ of node 8 the optimal solution to the problem.

Solving the knapsack problem by a branch-and-bound algorithm has a rather unusual characteristic. Typically, internal nodes of a state-space tree do not define a point of the problem's search space, because some of the solution's components remain undefined. If we had done this for the instance investigated above, we could have terminated nodes 2 and 6 before node 8 was generated because they both are inferior to the subset of value \$65 of node 5.

