KNAPSACK USING FIFO

7.5 0/1 Knapsack Problem

Problem statement :

The 0/1 Knapsack problem states that - There are 'n' objects given and capacity of Knapsack is 'm'. Then select some objects to fill The Knapsack in such a way that it should not exceed the capacity of Knapsack and maximum profit can be earned. The Knapsack problem is a maximization problem. That means we will always seek for maximum $P_i x_i$(where P_i represents profit of object x_i). We can also get $\sum P_i x_i$ maximum iff $-\sum P_i x_i$ is minimum.

Minimize profit $-\sum_{i=1}^{n} P_i x_i$

Subject to $\sum_{i=1}^{n} W_i x_i$

Such that $\sum W_i x_i \leq m$ and

$x_i = 0$ or 1 where $1 \leq i \leq n$

Example

Consider the knapsack instance: $n = 4$;

$(pi, p2, p3, p4) = (10, 10, 12, 18)$;

$(wi. w2, w\ 3, w4) = (2, 4, 6, 9)$ and $M = 15$.

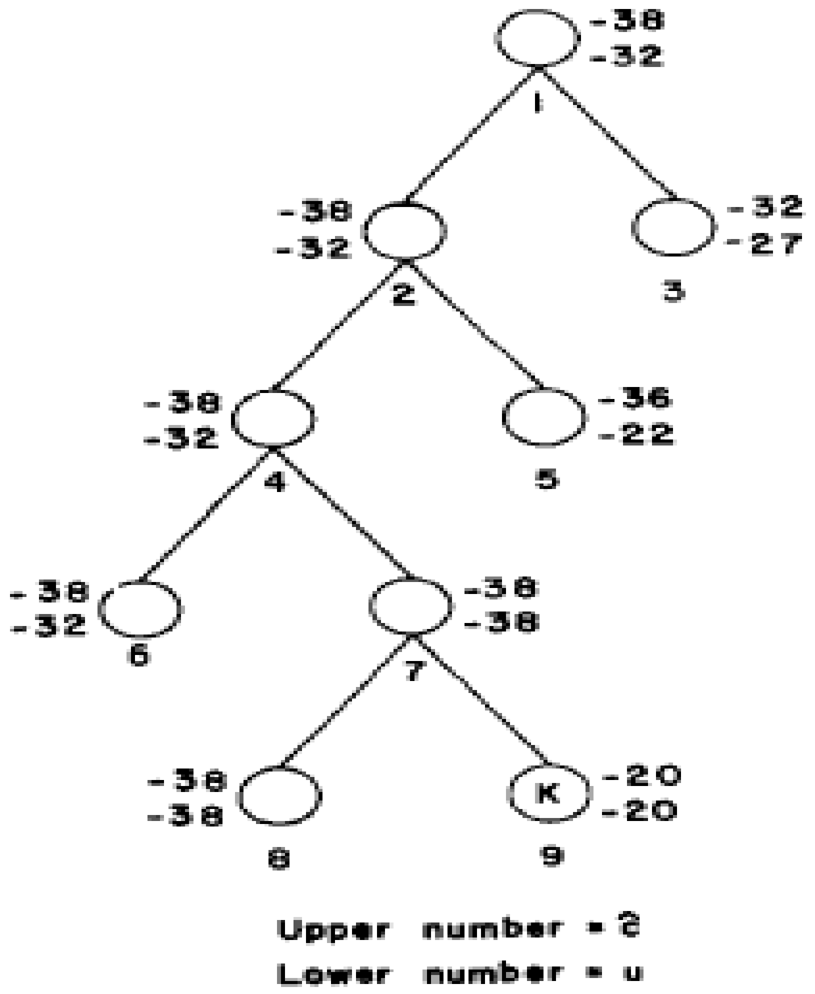

Figure 8.9 LC Branch-and-bound tree for Example 8.2

```
procedure UBOUND (p, w, k, M)
   //p, w, k and M have the same meaning as in Algorithm 7.11//
   //W(i) and P(i) are respectively the weight and profit of the ith object//
     global W(1:n), P(1:n); integer i, k, n
     b ← p; c ← w
     for i ← k + 1 to n do
        if c + W(i) ≤ M then c ← c + W(i); b ← b − P(i) endif
     repeat
     return (b)
  end UBOUND
```

Algorithm 8.5 Function $u(\cdot)$ for knapsack problem

7.5.2 FIFO Branch-and-Bound Solution

The space tree with variable tuple size formulation can be drawn and c^(.) and u(.) is computed (We have considered the same Knapsack problem which is discussed in section 7.5.1).

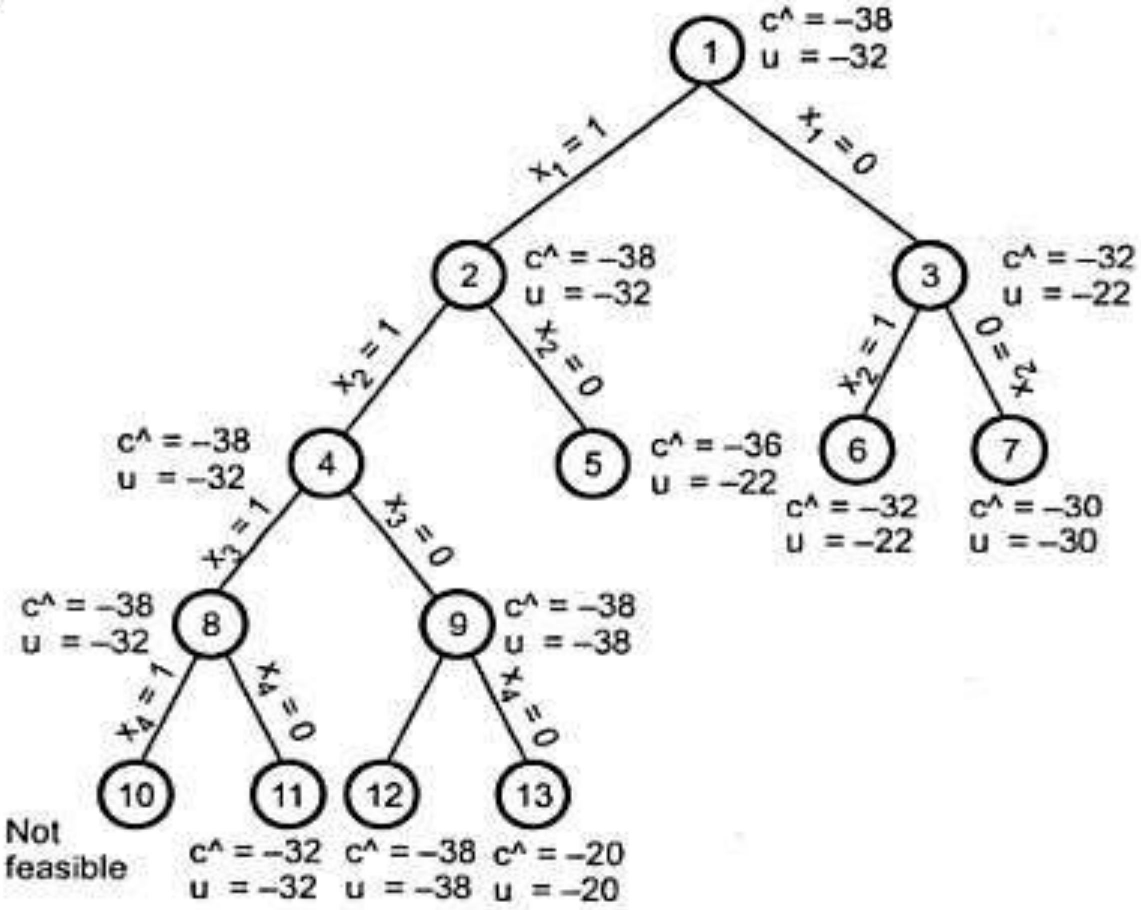

Fig. 7.7 FIFOBB space tree

Initially upper = u(1) = – 32. Then children of node 1 are generated. Node 2 becomes E-node and hence children 4 and 5 are generated. Node 4 and 5 are added in the list of live nodes. Next, node 3 becomes E-node and children 6 and 7 are generated. As c^(7) > upper we will kill node 7. Hence node 6 will be added in the list of live nodes. Node 4 is E-node and children 8 and 9 are generated. The upper is updated and it is now upper = u(9) = – 38. Nodes 8 and 9 are added in the list of live nodes. Node 5 and 6 becomes the next E-node but as c^(5) > upper and c^(6) >upper, kill nodes 5 and 6. Node 8 becomes next E-node and children 10 and 11 are generated. As node 10 is infeasible do not consider it. c^(11) > upper. Hence kill node 11. Node 9 becomes next E-node and upper = –38. Children 12 and 13 are generated. But c^(13) > upper. So kill node 13. Finally node 12 becomes an answer node. Therefore solution is $x_1 = 1$, $x_2 = 1$, $x_3 = 0$ and $x_4 = 1$.

FIFO branch and bound solution for knapsack problem is derived as follows :

1. Derive state space tree.
2. Compute lower bound : hatc(.) and upper bound : u(.) for each node.
3. If lower bound is greater than upper bound than kill that node.
4. Else, both the children of siblings are inserted in list and most promising node is selected as new E node.
5. Repeat step 3 and 4 until all nodes are examined.
6. The node with minimum lower bound value hatc(.) is the answer node. Trace the path from leaf to root in the backward direction to find the solution tuple.

LC branch and bound solution for knapsack problem is derived as follows :

1. Derive state space tree.
2. Compute lower bound : hatc(.) and upper bound : u(.) for each node.
3. If lower bound is greater than upper bound than kill that node.
4. Else select node with minimum lower bound as E-node.
5. Repeat step 3 and 4 until all nodes are examined.
6. The node with minimum lower bound value hatc(.) is the answer node. Trace the path from leaf to root in the backward direction to find the solution tuple.

Sample Questions:

1. In Branch and Bound, 0/1 Knapsack problem is a minimization problem. Explain?
2. With suitable example explain LIFOBB
3. Solve 0/1 Knapsack problem using FIFOBB

THANK YOU

- As discussed earlier, the goal of knapsack problem is to maximize

$$sum_i p_i x_i$$

given the constraints

$$sum_i w_i x_i le M$$

, where M is the size of the knapsack. A maximization problem can be converted to a minimization problem by negating the value of the objective function.

- The modified knapsack problem is stated as,

minimize

$$-sum_i p_i x_i$$

subjected to

$$sum_i w_i x_i le M$$

Where,

$$x_i in 0, 1, 1 le i le n$$

- Node satisfying the constraint

$$sum_i w_i x_i le M$$

in state space tree is called the answer state, the remaining nodes are infeasible.

LC branch and bound solution for knapsack problem is derived as follows :

1. Derive state space tree.
2. Compute lower bound : hatc(.) and upper bound : u(.) for each node.
3. If lower bound is greater than upper bound than kill that node.
4. Else select node with minimum lower bound as E-node.
5. Repeat step 3 and 4 until all nodes are examined.
6. The node with minimum lower bound value hatc(.) is the answer node. Trace the path from leaf to root in the backward direction to find the solution tuple.

FIFO branch and bound solution for knapsack problem is derived as follows :

1. Derive state space tree.
2. Compute lower bound : hatc(.) and upper bound : u(.) for each node.
3. If lower bound is greater than upper bound than kill that node.
4. Else, both the children of siblings are inserted in list and most promising node is selected as new E node.
5. Repeat step 3 and 4 until all nodes are examined.
6. The node with minimum lower bound value hatc(.) is the answer node. Trace the path from leaf to root in the backward direction to find the solution tuple.

u(1) =

$$-sump_i$$

such that

$$sumw_ileM$$

$$hatc(1) = u(1) - fracM - Weight; of; selected; itemsWeight; of; remaining; items * Profit; of; remaining; item$$

The bounding function is a heuristic computation. For the same problem, there may be different bounding functions. Apart from the above-discussed bounding function, another very popular **bounding function for knapsack is,**

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

where,

v is value/profit associated with selected items from the first i items.

W is the capacity of the knapsack.

w is the weight of selected items from first i items