- The Method
- 0/1 Knapsack Problem
- Traveling Salesperson Problem
- Job Sequencing with Deadlines

THE METHOD

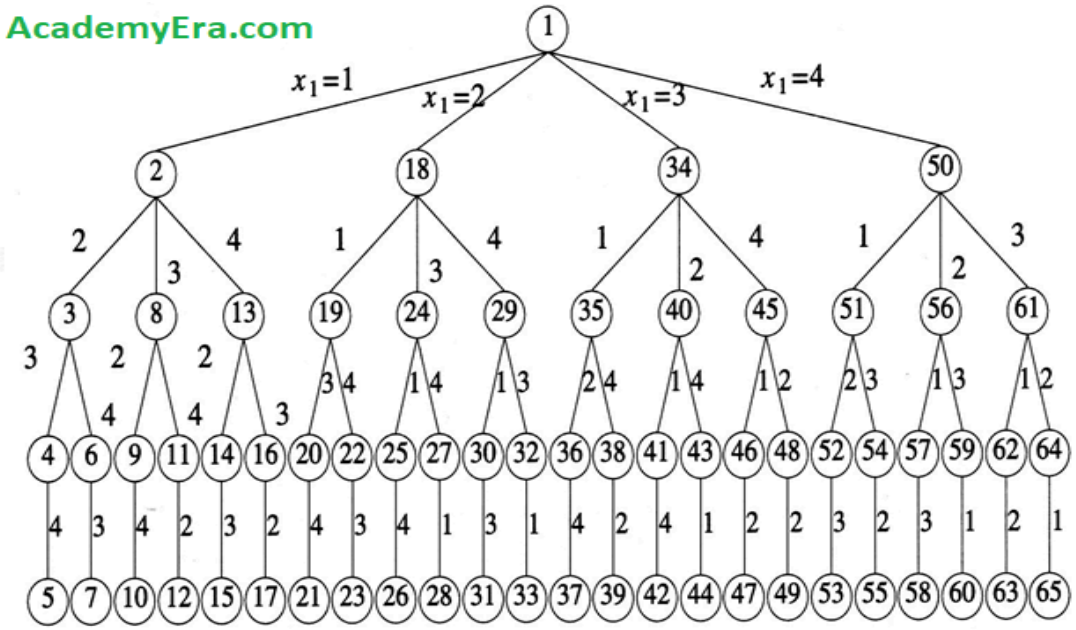
- The tree organization of the solution space is referred to as the ***state space tree.***
- A node which has been generated and all of whose children have not yet been generated is called a***live node.***
- The ***live node*** whose children are currently being generated is called the *E*-node (node being expanded).
- A ***dead node***is a generated node, which is not to be expanded further or all of whose children have been generated.
- ***Bounding functions*** are used to kill live nodes without generating all their children.
- The term branch-and-bound refers to all state space search methods in which all children of the E-node are generated before any other live node can become the E-node.
- In branch-and-bound terminology **Breadth first Search(BFS)**- like state space search will be called FIFO (First In First Output) search as the list of live nodes is a first -in-first -out list(or queue).
- A ***D-search*** (depth search) state space search will be called LIFO (Last In First Out) search, as the list of live nodes is a list-in-first-out list (or stack).
- Depth first node generation with bounding function is called backtracking. State generation methods in which the *E*-node remains the *E*-node until it is dead lead to***branch-and-bound method.***

Bounding functions are used to help avoid the generation of sub trees that do not contain an answer node.

The branch-and-bound algorithms search a tree model of the solution space to get the solution. However, this type of algorithms is oriented more toward optimization. An algorithm of this type specifies a real -valued cost function for each of the nodes that appear in the search tree.

Usually, the goal here is to find a configuration for which the cost function is minimized. The branch-and-bound algorithms are rarely simple. They tend to be quite complicated in many cases.

Example:[4-queens] Let us see how a FIFO branch-and-bound algorithm would search the state space tree for the 4-queens problem.



Initially, there is only one live node, node1. This represents the case in which no queen has been placed on the chessboard. This node becomes the E-node.

It is expanded and its children, nodes2, 18, 34and 50are generated.

These nodes represent a chessboard with queen1 in row 1and columns 1, 2, 3, and 4 respectively.

The only live nodes 2, 18, 34, and 50.If the nodes are generated in this order, then add node 2, 18, 34, and 50 to the queue. The next E-node are node 2.

2	18	34	50						

It is expanded and the nodes 3, 8, and 13 are generated. Node 3 is immediately killed using the bounding function. Nodes 8 and 13 are live nodes added to the queue as live nodes and remove node 2 from the queue.

18	34	50	8	13					

Node 18 becomes the next *E*-node. Nodes 19, 24, and 29 are generated. Nodes 19 and 24 are killed as aresult of the bounding functions. Node 29 is added to the queue of live node

and removes the node 18 from the queue.

34	50	8	13	29					

Node 34 becomes the next *E*-node. Nodes 35, 40, and 45 are generated. Nodes 40 and 45 are killed as a result of the bounding functions. Node 35 is added to the queue of live nodes and removes the node 34 from the queue.

50	8	13	29	35					

Node 50 becomes the next *E*-node. Nodes 51, 56, and 61 are generated. Nodes 61is killed as a result of the bounding functions. Node 51 and 56 is added to the queue of live nodes and removes the node 50 from the queue.

8	13	29	35	51	56				

Node 8 becomes the next *E*-node. Nodes 9, and 11 are generated. Nodes 9 and 11 are killed as a result of the bounding functions. No node is added to the queue of live nodes and removes the node 8 from the queue.

13	29	35	51	56				

Node 13 becomes the next *E*-node. Nodes 14, and 16 are generated. Nodes 16 is killed as a result of the bounding functions. Node 14 is added to the queue of live nodes and removes the node 13 from the queue.

29	35	51	56	14			

Node 29 becomes the next *E*-node. Nodes 30, and 32 are generated. Nodes 32is killed as a result of the bounding functions. Node 30 is added to the queue of live nodes and removes the node 29 from the queue.

35	51	56	14	30		

Node 35 becomes the next *E*-node. Nodes 36, and 38 are generated. Nodes 36 is killed as a result of the bounding functions. Node 38 is added to the queue of live nodes and removes the node 35 from the queue.

51	56	14	30	38	

Node 51 becomes the next *E*-node. Nodes 52, and 54 are generated. Nodes 52 is killed as a result of the bounding functions. Node 54 is added to the queue of live nodes and

removes the node 51 from the queue.

56	14	30	38	54

Node 56 becomes the next *E*-node. Nodes 57, and 59 are generated. Nodes 57 and 59 are killed as a result of the bounding functions. No node is added to the queue of live nodes and removes the node 56 from the queue.

14	30	38	54

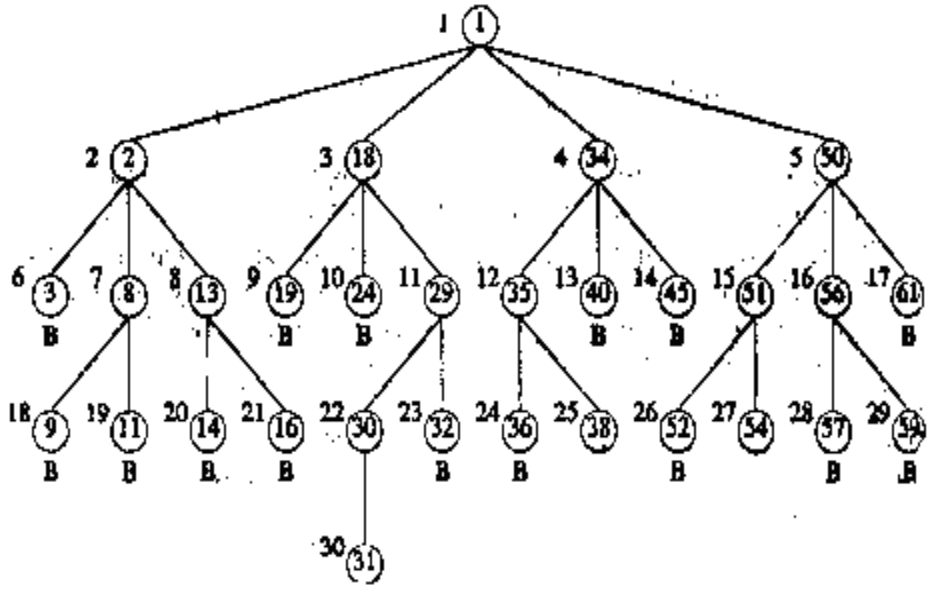
Node 14 becomes the next *E*-node. Node 15 is generated. Node 15 is killed as a result of the bounding functions. No node is added to the queue of live nodes and removes the node 14 from the queue.

30	38	54

Node 30 becomes the next *E*-node. Node 31 is generated and remove the node 30.

At the time the answer node, node 31, is reached, the only live nodes remaining are nodes 38 and 54.

Numbers inside the nodes correspond to the numbers in Figure.Numbers outside the nodes give the order in which the nodes are generated by FIFO branch-and-bound.



Least Cost (LC) Search:

BRANCH-AND-BOUND

In both LIFO and FIFO branch-and-bound the selection rule for the next E-node is rather rigid and in a sense blind. The selection rule for the next E-node does not give any preference to a node that has a very good chance of getting the search to an answer node quickly.

The search for an answer node can often be speeded by using an "intelligent" ranking function $\hat{c}(.)$ for live nodes. The next *E*-node is selected on the basis of this ranking function.

If in the 4-queens example we use a ranking function that assigns node 30 a better rank than all other live nodes, then node 30 will become *E*-node.The remaining live nodes will never become *E*-nodes as the expansion of node 30 results in the generation of an answer node (node 31).

Let $\hat{g}(x)$ be an estimate of the additional effort needed to reach an answer node from x. node x is assigned a rank using a function (.) such that $\hat{c}(x) = f(h(x)) + \hat{g}(x)$, where h(x) is the cost of reaching x from the root and f(.) is any non-decreasing function.

A search strategy that uses a cost function $\hat{c}(x) = f(h(x)) + \hat{g}(x)$, to select the next e-node would always choose for its next e-node a live node with least (.).Hence, such a strategy is called an LC-search (least cost search).

Cost function c (.) is defined as, if x is an answer node, then c(x) is the cost (level, computational difficulty,etc.) of reaching x from the root of the state space tree. If x is not an answer node, then c(x) =infinity, providing the sub-tree x contains no answer node; otherwise c(x) is equals the cost of a minimum cost answer node in the sub-tree x.

It should be easy to see that $\hat{c}(.)$ with f (h(x)) =h(x) is an approximation to c (.). From now on (x) is referred to as the cost of x.

The 15 – puzzle problem

The 15-puzzle problem consists of 15 numbered tiles on a square frame with a capacity of 16 tiles. We are given an initial arrangement of the tiles, and the objective is to transform the arrangement into the goal arrangement through a series of legal moves.

1	3	4	15
2		5	12
7	6	11	14
8	9	10	13

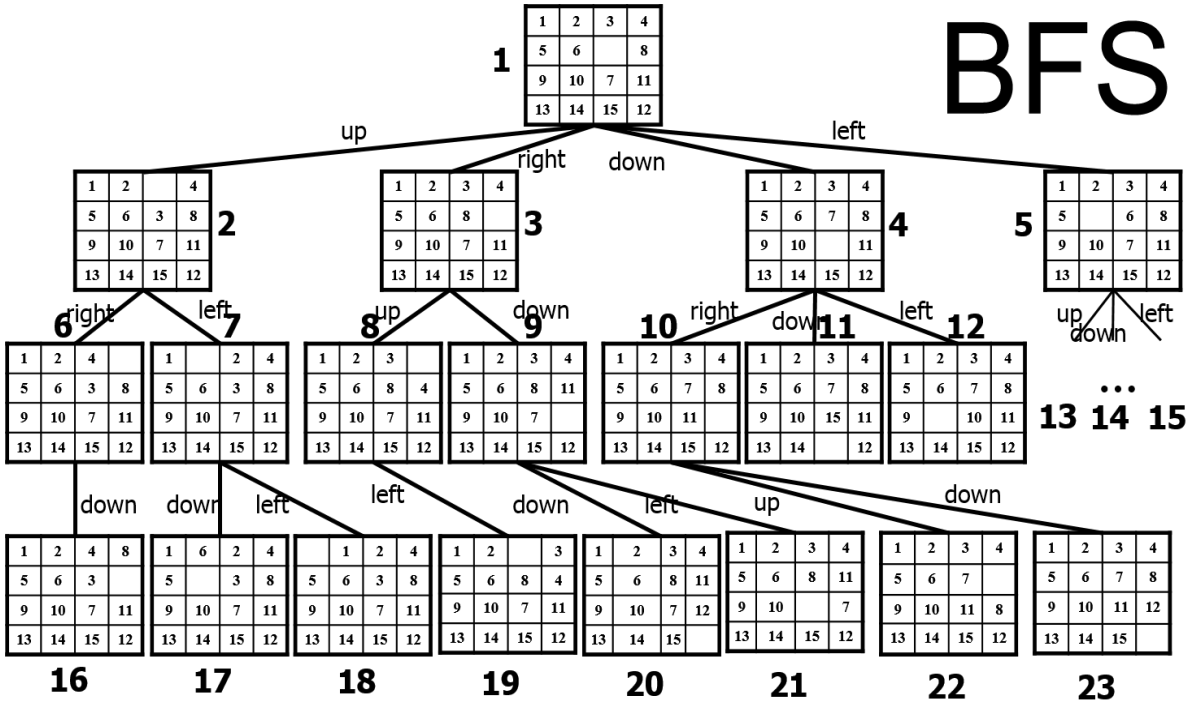
(a) Initial arrange

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

(b) Target arrange

The only legal moves are ones in which a tile adjacent to the empty spot (ES). Thus from the initial arrangement four legal moves are possible. We can move any one of the tiles numbered 2, 3, 5, or 6 to the empty slot (ES).

Part of the state space tree for the 15-puzzle problem



In a FIFO search of the state space tree the nodes will be generated in the ordered number. A breadth first search will always find a goal node nearest to the root.

In least cost search, we compute

C(x) = f(x) + g(x)

Where f(x) = the length of the path from the root node to node x.

g(x) = number of non-blank tiles not in their goal position.

We begin with node 1 as a E-node. All it'schildren nodes 2, 3, 4, and 5 are generated, and kill node 1. So live nodes are 2, 3, 4, 5.

C(2) = 1 + 4 = 5 C(3) = 1 + 4 = 5

C(4) = 1 + 2 = 3 C(5) = 1 + 4 = 5

Since node 4 has the least cost value, the next E-node is node 4. All it's children nodes 10, 11, 12 are generated, and kill node 4. So live nodes are 2, 3, 5, 10, 11, 12.

C(10) = 2 + 1 = 3 C(11) = 2 + 3 = 5 C(12) = 2 + 3 = 5

Since node 10 has the least cost value, the next E-node is node 10. All it's children nodes 22, 23 are generated, and kill node 10. So live nodes are 2, 3, 5, 11, 12, 22, 23.

C(22) = 3+ 2 = 5 C(23) = 3 + 0 = 3

Since node 23 has the least cost value, the next E-node is node 23.

This is the goal node.

Control Abstractions for LC-Search

```
listnode = record
{
        float cost;
        listnode *next, *parent;
}

Algorithm LCSearch(t)
//Search t for an answer node
{
        if (*t is an answer node) then output *t and return;
        E = t;   //E-node
        Initialize the list of live nodes to be empty;
        repeat
        {
                for (each child x of E) do
                {
                        if (x is an answer node) then output the path from x to t and return;
                        add(x);             // x is a new live node.
                        (x → parent) = E;       // pointer for path to root
```

```
                }
                if (there are no more live nodes) then
                {
                        write("No answer node");
                        return;
                }
                E = Least();
        }until(false);
}
```

Bounding:

A branch -and-bound searches the state space tree using any search mechanism in which all the children of the E-node are generated before another node becomes the E-node.

We assume that each answer node x has a cost c(x) associated with it and that a minimum-cost answer node is to be found. Three common search strategies are FIFO,LIFO,and LC.

A cost function $\hat{c}$ (.) such that $\hat{c}$ (x) <=c(x) is used to provide lower bounds on solutions obtainable from any node x.If upper is an upper bound on the cost of a minimum-cost solution, then all live nodes x with $\hat{c}$ (x)>upper may be killed as all answer nodes reachable from x have cost c(x)>= $\hat{c}$ (x)>upper. The starting value for upper can be set to infinity.

Clearly, so long as the initial value for upper is no less than the cost of a minimum-cost answer node, the above rule to kill live nodes will not result in the killing of a live node that can reach a minimum-cost answer node.Each time a new answer is found ,the value of upper can be updated.

BRANCH-AND-BOUND

As an example optimization problem, consider the problem of job scheduling with deadlines. We generalize this problem to allow jobs with different processing times. We are given n jobs and one processor. Each job i has associated with it a three tuple (P_i d_i t_i).job i requires t_i units of processing time .if its processing is not completed by the deadline d_i , and then a penalty P_i is incurred.

The objective is to select a subset j of the n jobs such that all jobs in j can be completed by their deadlines. Hence, a penalty can be incurred only on those jobs not in j.The subset j should be such that the penalty incurred is minimum among all possible subsets j. such a j is optimal.

Consider the following instances: n=4,(P_1 , d_1 , t_1)=(5,1,1),(P_2 , d_2 , t_2)=(10,3,2),(P_3 , d_3 , t_3)=(6,2,1),and(P_4 , d_4 , t_4)=(3,1,1).The solution space for this instances consists of all possible subsets of the job index set{1,2,3,4}. The solution space can be organized into a tree by means of either of the two formulations used for the sum of subsets problem.

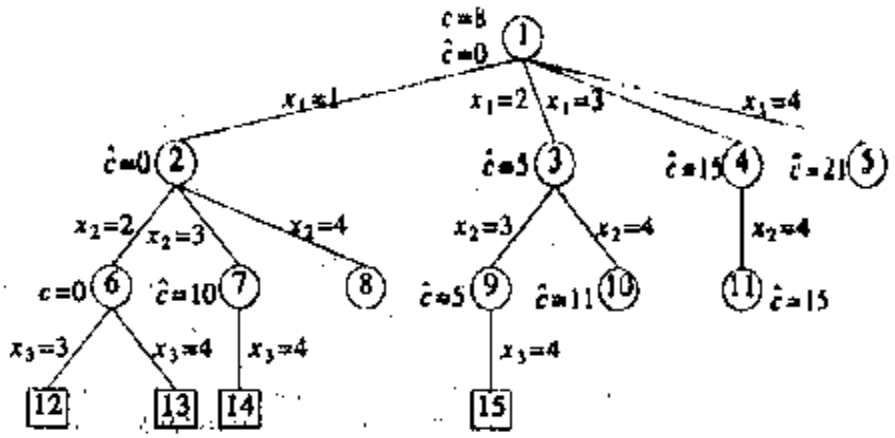


Figure 8.6 State space tree corresponding to variable tuple size formulation

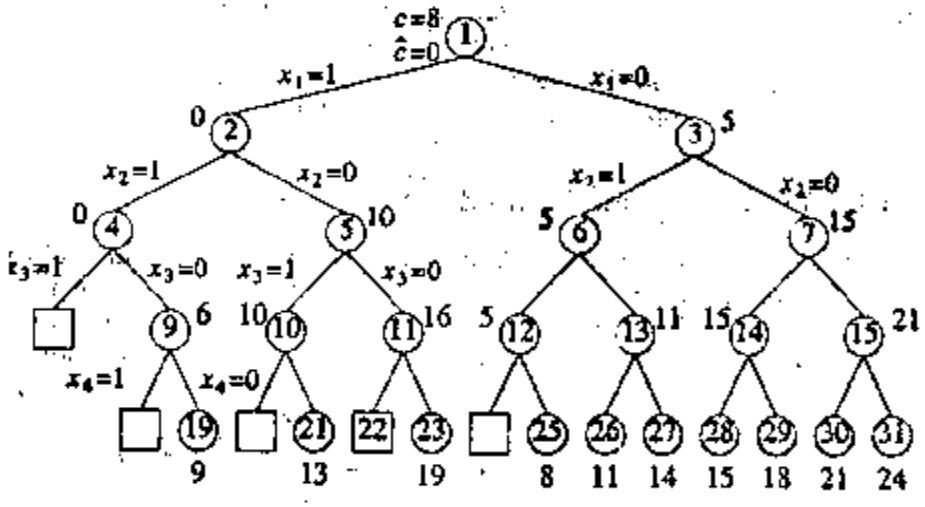


Figure 8.7

Figure 8.6 corresponds to the variable tuple size formulations while figure 8.7 corresponds to the fixed tuple size formulation. In both figures square nodes represent infeasible subsets. In figure 8.6 all non-square nodes are answer nodes. Node 9 represents an optimal solution and is the only minimum-cost answer node. For this node j= {2,3} and the penalty (cost) is 8. In figure 8.7 only non-square leaf nodes are answer nodes. Node 25 represents the optimal solution and is also a minimum-cost answer node. This node corresponds to j={2,3} and a penalty of 8. The costs of the answer nodes of figure 8.7 are given below the nodes.

0/1 Knapsack Problem

- To use the Branch-and-Bound technique to solve any problem.
- It is first necessary to conceive of a state space tree for the problem.
- Firstwe convert the maximization knapsack problem to minimize knapsack problem
- The modified knapsack problem is

$$\text{Minimize} \quad -\sum_{i=1}^{n} PiXi$$

$$\text{Subject to} \quad \sum_{i=1}^{n} WiXi \leq m \qquad xi = 0 \text{ or } 1$$

Assume a fixed tuple size formulation for the solution space.

Define $C(x) = -\sum_{i=1}^{n} PiXi$ for every answer node x.

$C(x) = \infty$ for infeasible leaf nodes.

We need two function C^(x) and U(x) such that $C^(x) \leq C(x) \leq U(x)$ for every node x.

$C^(x) = -\sum_{i=1}^{n} PiXi$ with fractions

$U(x) = -\sum_{i=1}^{n} PiXi$ without fractions

Initially upper bound (upper) = ∞

for every node x, if upper > U(x) then upper = U(x).
if C^(x) > upper then kill the node x.

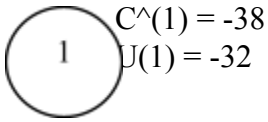
Example:Solve the following 0/1 knapsack problem using LCBB. n = 4, (P1, P2, P3, P4) =(10, 10, 12, 18) and (W1, W2, W3, W4) = (2, 4, 6, 9) and m = 15.

Solution:

BRANCH-AND-BOUND

We solve the problem using LCBB. Search using C^(x) and U(x).
Initially upper = ∞.
The search begin with the root (node 1) as a E-node.
C^(1) = -(10+10+12+(18/9)*3) = -(10+10+12+6) = -38
U(1) = -(10+10+12) = -32

Since upper >U(1) = -32, So upper = -32.

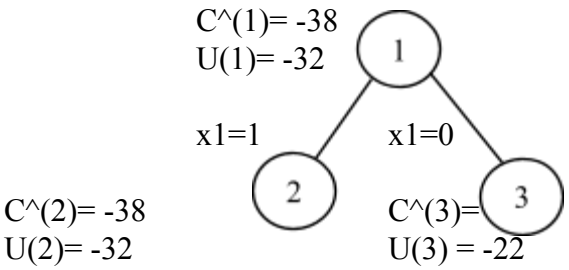


The possibility of object 1 is included or not.
The E-node (node 1) is expanded and it's children's node 2 and 3 are generated and both are put into the list of live nodes.

C^(2) = -(10+10+12+(18/9)*3) = -(10+10+12+6) = -38
U(2) = -(10+10+12) = -32

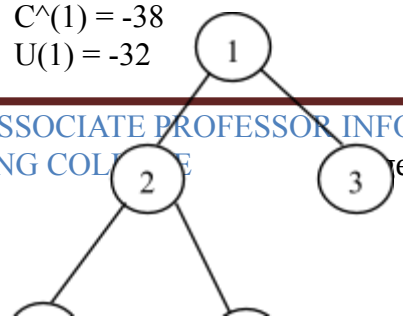
C^(3) = -(10+12+(18/9)*5) = -(10+12+10) = -32
U(3) = -(10+12) = -22

Upper = -32



Node 2, 3 are lives nodes. Node 2 has the least cost, so node 2 is the next E-node. The E-node (node 2) is expanded and it's children's node 4 and 5 are generated and both are put into the list of live nodes.
C^(4) = -(10+10+12+(18/9)*3) = -(10+10+12+6) = -38
U(4) = -(10+10+12) = -32

C^(5) = -(10+12+(18/9)*7) = -(10+12+14) = -36
U(5) = -(10+12) = -22
Upper = -32



x1 = 1 x1 = 0

C^(2) = -38
U(2) = -32

C^(3) = -32
U(3) = -22

x2 =1 x2 = 0

C^(4) = -38
U(4) = -32

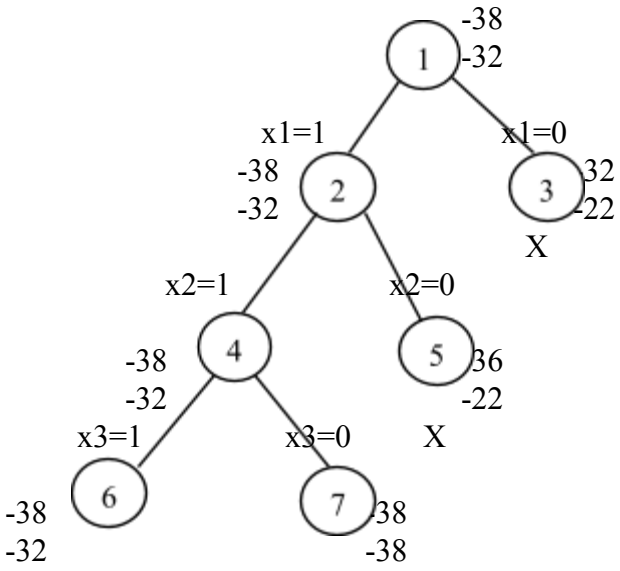
C^(5) = -36
U(5) = -22

Node 3, 4, 5 are lives nodes. Node 4 has the least cost, so node 4 is the next E-node. The E-node (node 4) is expanded and it's children's node 6 and 7 are generated and both are put into the list of live nodes.

C^(6) = -(10+10+12+(18/9)*3) = -(10+10+12+6) = -38
U(6) = -(10+10+12) = -32

C^(7) = -(10+10+18) = -38
U(7) = -(10+10+18) = -38

Since upper = -32 >U(7) = -38, So upper = -38.



Since C^(5) = -36 > upper = -38, So kill the node 5.
Since C^(3) = -32 > upper = -38, So kill the node 3.

Node 6, 7 are lives nodes. Node 6, 7 has the same cost, So assume that node 7 is the next E-node. The E-node (node 7) is expanded and it's children's node 8 and 9 are generated and both are put into the list of live nodes.

BRANCH-AND-BOUND

C^(8) = -(10+10+18) = -38
U(8) = -(10+10+18) = -38

C^(9) = -(10+10) = -20
U(7) = -(10+10) = -20

upper = -38.

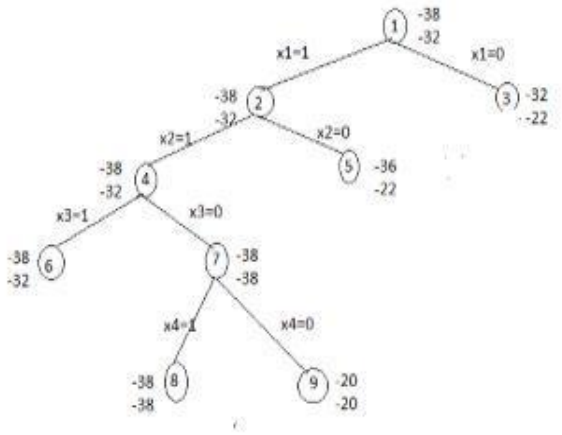


Figure2: SPACE STATE TREE FOR KNAPSACK PROBLEM USING LC-Branch and Bound

Since C^(9) = -20 > upper = -38, So kill the node 9. Node 6 and 8 are two live nodes with least cost.
Regardless of which become the next E-node, C^(E) ≥ upper and the search terminates with node 8 the answer node. At this time, the value -38 together with the path 8, 7, 4, 2, 1 is print out and the algorithm terminates.
The solution is (1, 1, 0, 1)and profit = 38.

Function U(x)

```
Algorithm UBound (cp, cw, k, m)
{
        // cp is current profit and cw is current weight
        b = cp;
        c = cw;
        for i = k+1 to n do
        {
                if (c+W[i] ≤ m) then
                {
```

```
                    c = c + W[i];
                    b = b – p[i];
                }
        }
        return b;
}
```

Function C^(x)

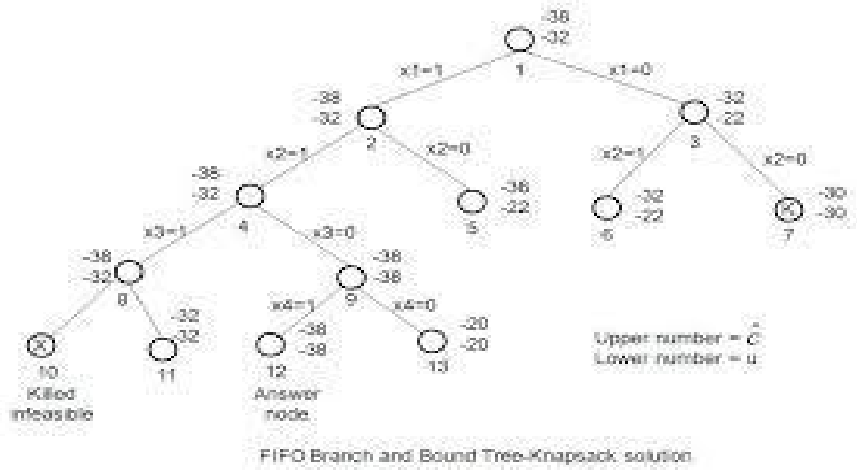
```
Algorithm Bound (cp, cw, k)
{
        // cp is current profit and cw is current weight
        b = cp;
        c = cw;
        for i = k+1 to n do
        {
                if (c+W[i] ≤ m) then   b = b + p[i];
                else                   b = b +((p[i]/w[i]) *(m-c));

        }
        return b;
}
```

Example: Solve the following 0/1 knapsack problem using FIFOBB. n = 4, (P1, P2, P3, P4) =(10, 10, 12, 18) and (W1, W2, W3, W4) = (2, 4, 6, 9) and m = 15.

Solution:

BRANCH-AND-BOUND



FIFO Branch and Bound Tree-Knapsack solution

We solve the problem using FIFOBB. Search using C^(x) and U(x).
Initially upper = ∞.
The search begins with the root (node 1) as a E-node.
C^(1) = -(10+10+12+(18/9)*3) = -(10+10+12+6) = -38
U(1) = -(10+10+12) = -32

Since upper >U(1) = -32, So upper = -32.

The possibility of object is included or not.
The E-node (node 1) is expanded and it's children's node 2 and 3 are generated.
C^(2) = -(10+10+12+(18/9)*3) = -(10+10+12+6) = -38
U(2) = -(10+10+12) = -32
C^(3) = -(10+12+(18/9)*5) = -(10+12+10) = -32
U(3) = -(10+12) = -22
Node 2, 3 are put into the list of live nodes and added to the queue.

2	3								

upper = -32.
Node 2, 3 is lives nodes. Node 2 is the next E-node. The E-node (node 2) is expanded and it's children's node 4 and 5 are generated.
C^(4) = -(10+10+12+(18/9)*3) = -(10+10+12+6) = -38
U(4) = -(10+10+12) = -32
C^(5) = -(10+12+(18/9)*7) = -(10+12+14) = -36
U(5) = -(10+12) = -22
Node 4, 5 are put into the list of live nodes and added to the queue and remove node 2 from the queue.

3	4	5						

Upper = -32
Node 3, 4, 5 is lives nodes. Node 3 is the next E-node. The E-node (node 3) is expanded and it's children's node 6 and 7 are generated.
C^(6) = -(10+12+(18/9)*5) = -(10+12+10) = -32
U(6) = -(10+12) = -22
C^(7) = -(12+18) = -(12+18) = -30
U(7) = -(12+18) = -30
Since C^(7) = -30 > upper = -32, So kill the node 7.
Node 6 put into the list of live nodes and added to the queue and remove node 3 from the queue.

4	5	6					

Upper = -32
Node 4, 5, 6, is lives nodes. Node 4 is the next E-node. The E-node (node 4) is expanded and it's children's node 8 and 9 are generated.
C^(8) = -(10+10+12+(18/9)*3) = -(10+10+12+6) = -38
U(8) = -(10+10+12) = -32
C^(9) = -(10+10+18) = -38
U(9) = -(10+10+18) = -38
Since upper = -32 >U(9) = -38, So upper = -38.
Since C^(5) = -36> upper = -38, So kill the node 5.
Since C^(6) = -32 > upper = -38, So kill the node 6.

Node 8, 9 are put into the list of live nodes and added to the queue and remove node 2 from the queue.

8	9						

Upper = -38.
Node 8, 9 is lives nodes. Node 8 is the next E-node. The E-node (node 8) is expanded and it's children's node 10 and 11 are generated.
Node 10 is an infeasible solution(sum of weights is exceeds the capacity of knapsack), So kill the node 10.
C^(11) = -(10+10+12) = -32
U(11) = -(10+10+12) = -32
Since C^(11) = -32 > upper = -38, So kill the node 11.
Remove node 8 from the queue.

9						

upper = -38.
Node 9 is live node. Node 9 is the next E-node. The E-node (node 9) is expanded and it's

children's node 12 and 13 are generated.

C^(12) = -(10+10+18) = -(10+10+18) = -38
U(12) = -(10+10+18) = -38
C^(13) = -(10+10) = -20
U(13) = -(10+10) = -20
Since C^(13) = -20> upper = -38, So kill the node 13.

Node 12 put into the list of live nodes and added to the queue and remove node 9 from the queue.

12						

The only remaining live node is node 12. It has no children's, so the search terminates.
The path from node12 to root is the solution.
The solution is (1, 1, 0, 1) and profit = 38.

TRAVELLING SALESPERSON PROBLEM

- The time complexity of traveling salesperson problem using Dynamic programming is $O(n^2 2^n)$.
- The worst case time complexity of traveling salesperson problem using Branch-and-Bound is also $O(n^2 2^n)$.
- Let G = (V, E) be a directed graph defining an instance of the traveling salesperson problem. Let $c_{ij} = \infty$ if $(i, j) \notin E$, and let |V| = n.
- Without loss of generality, we can assume that every tour starts and ends at vertex 1.
- The solution space S is given by S = { 1π1 / π is a permutation of {2, 3, …,n}}
- Then |S| = (n-1)! and s can be organized into a state space tree.

To search traveling salesperson state space tree using LCBB, we need to define cost function C(x) and two other function C^(x) and U(x) such that C^(x) ≤ C(x) ≤ U(x) for all nodes x.

C(A)= length of tour defined by the path from the root to A if A is a leaf
Cost of a minimum-cost leaf in the sub tree A if A is not a leaf

BRANCH-AND-BOUND

Reduced matrix:

A row (column) is said to be *reduced*iff it contains at least one zero and all remaining entries are non-negative.

A matrix is said to be reduced iff every row and column is reduced.

C^(A) = reduced cost of the matrix A.

We can associate a reduced cost matrix with every node in the traveling salesperson state space tree.

- A is the reduced cost matrix for node R
- S is a child of R such that the tree edge (R, S) corresponds to including edge (i, j) in the tour.
- If S is not a leaf, then the reduced cost matrix for S may be obtain as follows
  - Change all entries in row i and column j of A to ∞ (This prevent the use of any more edges leaving vertex i or entering vertex j).
  - Set A(j, 1) = ∞ (This preventsthe use of edge (j, 1)).
  - Reduced all rows and columns in the resulting matrix except for rows and columns containing only ∞. Resulting matrix is B.
  - If r is the total amount subtracted value, then C^(S) =C^(R) + A(i, j)+ r

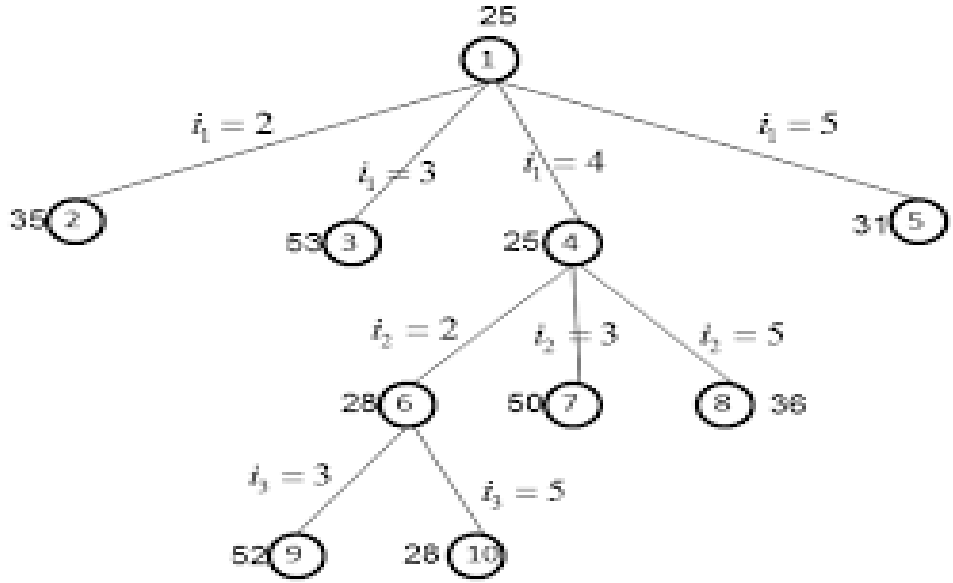
Example:

∞	20	30	10	11
15	∞	16	4	2
3	5	∞	2	4
19	6	18	∞	3
16	4	7	16	∞

Cost matrix

∞	10	17	0	1
12	∞	11	2	0
0	3	∞	0	2
15	3	12	∞	0
11	0	0	12	∞

Reduced cost matrix L=25



∞	∞	`∞	∞	∞
∞	∞	11	2	0
0	∞	∞	0	2
15	∞	12	∞	0
11	∞	0	12	∞

∞	∞	∞	∞	∞
1	∞	∞	2	0
∞	3	∞	0	2
4	3	∞	∞	0
0	0	∞	12	∞

Node 2: path 1—2 : C^(2) = 25+10+0 = 35 Node 3 : path 1—3 : C^(3)=25+17+11=53

∞	∞	∞	∞	∞
12	∞	11	∞	0
0	3	∞	∞	2
∞	3	12	∞	0
11	0	0	∞	∞

∞	∞	∞	∞	∞
10	∞	9	0	∞
0	3	∞	0	∞
12	0	9	∞	∞
∞	0	0	12	∞

Node 4 : path 1—4 : C^(4)=25+0+0=25 Node 5 : path 1—5 : C^(5)=25+1+5=31

∞	∞	∞	∞	∞
∞	∞	11	∞	0
0	∞	∞	∞	2
∞	∞	∞	∞	∞
11	∞	0	∞	∞

∞	∞	∞	∞	∞
1	∞	∞	∞	0
∞	1	∞	∞	0
∞	∞	∞	∞	∞
0	0	∞	∞	∞

Node 6 path: 1—4—2 C^(6)=25+3+0=28 Node 7 path: 1—4—3 C^(7)=25+12+13=50

∞	∞	∞	∞	∞

∞	∞	∞	∞	∞

1	∞	0	∞	∞		∞	∞	∞	∞	∞
0	3	∞	∞	∞		∞	∞	∞	∞	0
∞	∞	∞	∞	∞		∞	∞	∞	∞	∞
∞	0	0	∞	∞		0	∞	∞	∞	∞

Node 8 path 1—4—5 C^(8)=25+0+11=36 Node 9 path: 1-4-2-3 C^(9)=28+11+13=52

∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
0	∞	∞	∞	∞
∞	∞	∞	∞	∞
∞	∞	0	∞	∞

Node 10 path: 1-4-2-5 C^(10) = 28+0+0=28.

Hence, LCBB terminates with 1, 4, 2, 5, 3, 1 as the shortest length tour and cost = 28.

JOB SEQUENCING WITH DEADLINES

- We solve this problem using FIFOBB.
- We generalize this problem to allow jobs with different processing times.
- We are given n jobs and one processor. Each job i has associated with it a tuple (pi, di, ti).

BRANCH-AND-BOUND

- Job i requires ti units of processing time. If it's processing is not completed by the deadline di, then a penalty pi is occurred.
- The objective is to select the subset J of the n jobs such that all jobs in J can be completed by their deadlines.

Example:

Solve the following Job Sequencing with deadlines problem using FIFOBB.
n = 4, (p1, d1, t1) = (5, 1, 1), (p2, d2, t2) = (10, 3, 2) (p3, d3, t3) = (6, 2, 1) and (p4, d4, t4) = (3, 1, 1).

Solution:

Jobs	1	2	3	4
Penalties	5	10	6	3
Deadlines	1	3	2	1
Times	1	2	1	1

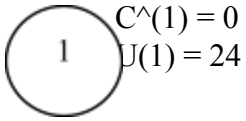
We need to find two functions C^(x) and U(x) such that C^(x) ≤ C(x) ≤ U(x) for all nodes x.

C^(x) = ∑ pi = sum of penalties till the last job considered.
U(x) = ∑ Pi = sum of all penalties except that include in solution.

Initially upper = ∞.
If upper > U(x) then upper = U(x).
If C^(x) >upper then kill node x.

C^(1) = 0.
U(1) = 5+10+6+3 = 24.

Since upper >U(1) = 24, So upper = 24.



The E-node (node 1) is expanded and it's children's node 2, 3, 4, and 5 are generated and they are put into the list of live nodes.

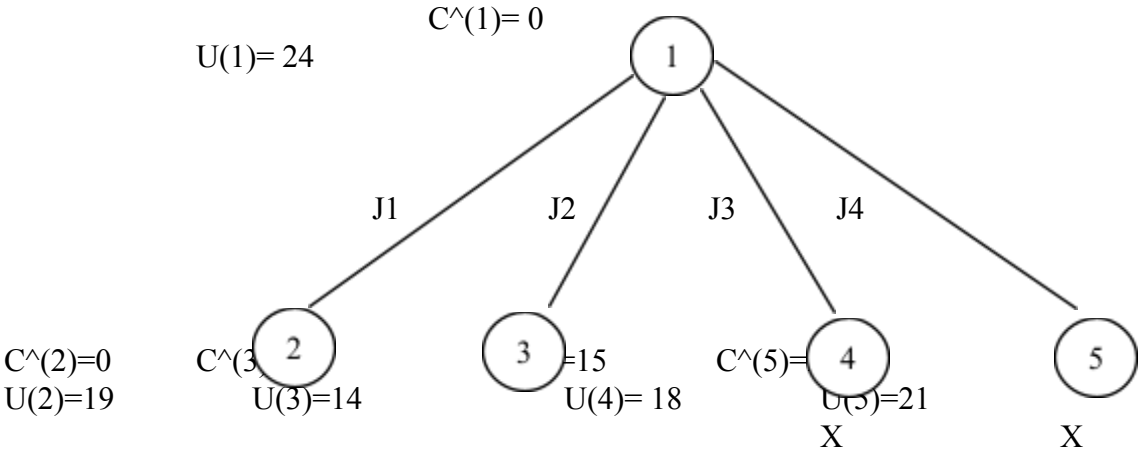
C^(2) = 0 C^(3) = 5
U(2) = 10+6+3=19 U(3) = 5+6+3 = 14

C^(4) = 5+10 = 15 C^(5) = 5+10+6 = 21
U(4) = 5+10+3=18 U(5) = 5+10+6 = 21

BRANCH-AND-BOUND

Since upper =24 >U(3) = 14, So upper = 14.
Since C^(4) = 15 > upper = 14, So kill the node 4.
Since C^(5) = 21 > upper = 14, So kill the node 5.

Upper = 14



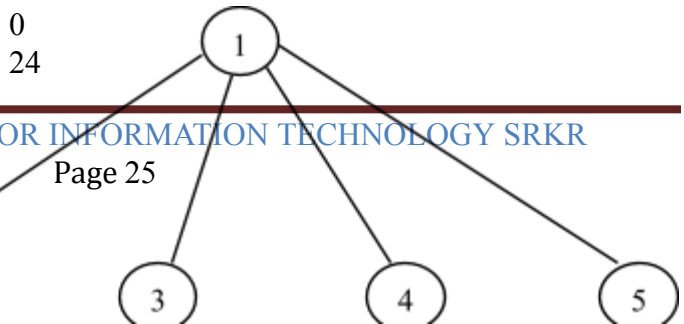
Node 2, 3 are lives nodes. So node 2 is the next E-node. The E-node (node 2) is expanded and it's children's node 6, 7 and 8 are generated and they are put into the list of live nodes.

C^(6) = 0
U(6) = 6+3 = 9

C^(7) = 10
U(7) = 10+3 = 13

C^(8) = 10+6 = 16
U(8) = 10+6 = 16
Since upper =14 >U(6) = 9, So upper = 9.
Since C^(7) = 10 > upper = 9, So kill the node 7.
Since C^(8) = 16 > upper = 9, So kill the node 8.

upper = 9



J1 J2 J3 J4

0 5 15 21
19 14 18 21
X X

J2 J3 J4

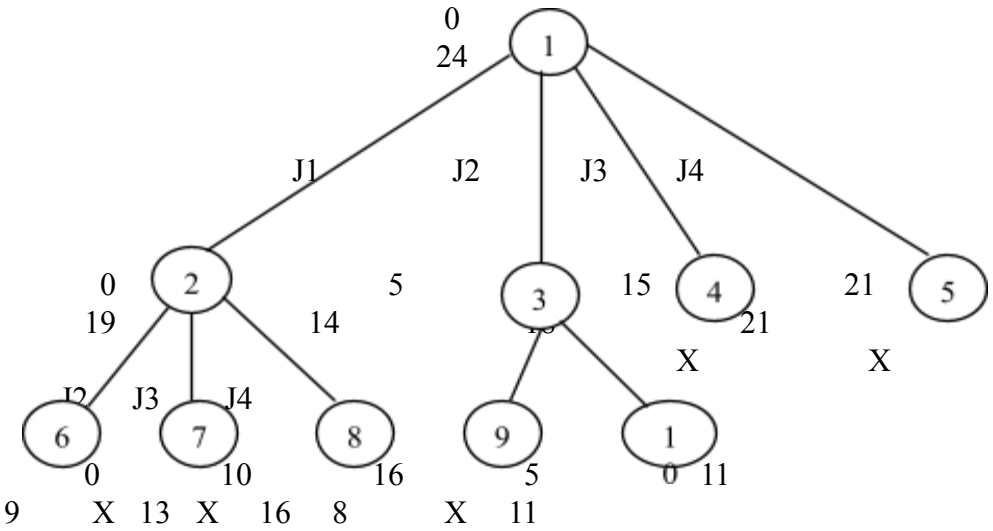
0 10 16
9 13 XX16

Node 3, 6 are lives nodes. So node 3 is the next E-node. The E-node (node 3) is expanded and it's children's node 9 and 10 are generated and both are put into the list of live nodes.

C^(9) = 5 C^(10) = 5+6 = 11
U(9) = 5+3 = 8 U(10) = 5+6 = 11

Since upper =9 >U(9) = 8, So upper = 8.
Since C^(10) = 11 > upper = 8, So kill the node 10.

upper = 8.

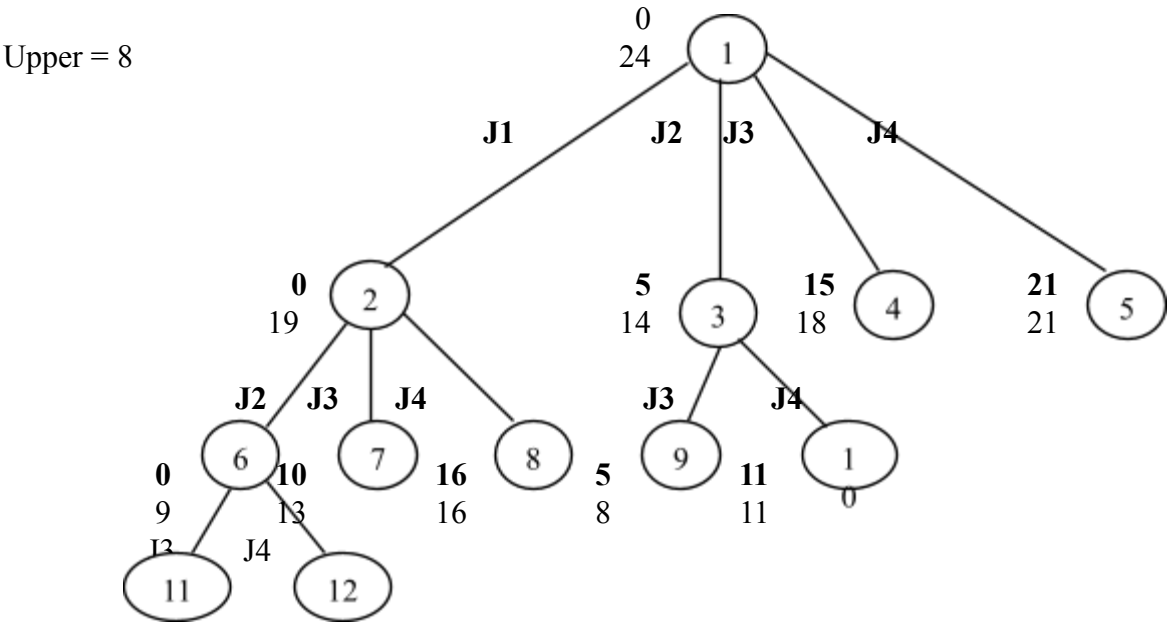


Node 6, 9 are lives nodes. Sonode 6 is the next E-node. The E-node (node 6) is expanded

and it's children's node 11 and 12 are generated and both are put into the list of live nodes.

Node 11, if include J1, J2, and J3 jobs, then they process 4 units of time but their deadlines maximum is 3 only. So, node 11 is infeasible.

Node 12, include J1, J2 and J4 jobs is not possible, because J1 and J4 deadlines are 1. Therefore Node 12 is infeasible.



Node 9 is live node. So node 9 is the next E-node. The E-node (node 9) is expanded and it's children's node 13 is generated and put into the list of live nodes.

Node 13, if include J2, J3, and J4 jobs, then they process 4 units of time but their deadlines maximum is 3 only. So, node 13 is infeasible.

So the answer node is node 9 and penalty is 8.

The solution is J = { 2, 3 } cost = 16 and penalty = 8.