UNIT - V

BRANCH AND BOUND

5.1 GENERAL STRATEGY

We have seen different algorithmic strategies. In this chapter, we are going to see branch-and-bound approach. We know that the graph can be searched using two ways :

(i) Depth First Search (DFS)

(ii) Breadth First Search (BFS)

In the branch and bound approach, at each node we calculate a bound on the possible value of solutions. Calculation of the bounds is combined with either DFS or BFS. The calculated bound is used to decide which node should be expanded first. The nodes which can be expanded (explored) are termed as live nodes.

5.2 GENERAL CHARACTERISTICS

The branch-and-bound terminology works in two ways :

1. If the live nodes are explored in the reverse order of their creation like DFS, then it is termed as Last In First Out (LIFO) search. It uses data structure stack.
2. If the live nodes are searched in the order of their creation like BFS, then it is termed as First In First Out (FIFO) search. It uses data structure queue.

Actually BFS and DFS are special cases of Least Cost (LC) search.

Branch-and-bound uses a priority list to store nodes that have been generated but not yet explored. A heap can better represent a priority list.

It uses bounding functions to avoid generation of subtrees that do not contain solution node (an answer node).

The technique is sufficiently powerful and is often used in practical applications.

The behaviour of this technique depends on the bound value. With a better bound, we may get an optimal solution more quickly. Some times, we may have to spend time at each node.

5.3 LEAST COST (LC) SEARCH

BFS and DFS are special cases of LC search. In BFS, the live nodes are considered in FIFO order. In DFS, the live nodes are considered in LIFO order. But in LC search, that node is considered immediately which has a very good chance of getting the search to a solution node quickly. This quick search can be done using proper **ranking function** $\hat{c}(.)$ for live nodes. The ranking function helps to select next live node. The ranking function considers the additional efforts needed to reach an answer node from the live node. The number of levels from the nearest answer node can be considered as **ranking**.

5.4 BOUNDING

Actually BFS, DFS and LC search differ only in the sequence in which they visit nodes in the tree. In branch-and-bound approach, all the children of a live node are generated before another live node is considered.

Let a cost function $\hat{c}(.)$ is such that $\hat{c}(x) \leq c(x)$ and it provides lower bounds on solutions which can be obtained from x.

Let u(x) is upper bound on minimum cost of solution, then all live nodes for which $c(x) \geq \hat{c}(x) > u(x)$ will be killed because they exceed upper bound. Each time a new answer node is found, the upper bound value should be updated.

5.5 0/1 KNAPSACK PROBLEM

- We already know the 0/1 knapsack problem. It is to maximize $\Sigma p_i x_i$ such that $\Sigma p_i x_i \leq M$. In other words,

$$\sum_{i=1}^{n} p_i x_i \quad \text{and}$$

$$\sum_{i=1}^{n} w_i x_i \leq M \text{ and}$$

$$x_i = 0/1, \text{ for } 1 \leq i \leq n$$

Every leaf node in the state space tree is an answer node for which $\sum_{i=1}^{n} w_i x_i \leq M$. For such answer nodes, $c(x) = \sum_{i=1}^{n} p$

All the remaining leaf nodes are obviously infeasible and for them $c(x) = \infty$. For all non-leaf nodes,

$$c(x) = \min[c(\text{Lchild}(x)), c(\text{Rchild}(x))]$$

For every node x, $\hat{c}(x) \leq c(x) \leq u(x)$.

Let x be a node at level j for $1 \leq j \leq n + 1$. For each node $1 \leq i \leq j$,

$c(x) \leq -\sum_{1 \leq i \leq j} p_i x_i$ and $u(x) = -\sum_{1 \leq i \leq j} p_i x_i$. When new answer node is found and $a = -\sum_{1 \leq i \leq j} p_i x_i$, then we can upd upper bound as

$$u(x) = \text{UpperBound}(a, \sum_{1 \leq i \leq j} w_i x_i, j-1, M)$$

- The algorithm UpperBound takes four input parameters: Current weight cw, current profit cp, current objec maximum capacity of knapsack M. The output of the function is modified upper bound. An array w[1...n] ho weights. An array p[1 ... n] holds profits.

Algorithm: UpperBound (cw, cp, k, M)

```
begin
    np = cp;
    nw = cw;
    for i=k+1 to n
    begin
        if (nw+w[i]≤M)
        then
            nw = nw + w[i]
            np = np - p[i]
        end if
    end for
    return np
end
```

- Algorithm BBKnapsack takes three inputs : object k, current weight cw, current profit cp. Assume that $(p[i]/w[i] \geq (p[i+1])$. Let tw is the final weight of knapsack and tp is the final maximum profit. The final solution will be availab in an array x[1 ... n]. If object k is in knapsack, x[k] = 1, otherwise x[k]= 0. Upper bound for a feasible left child of node N is same as that for N. But for right child, upper bound should be generated. The path tx[ ] is the path from firs node to the current node. Hence current weight $cw = \sum_{i=1}^{k-1} w_i tx_i$ and current profit $cp = \sum_{i=1}^{k-1} p_i tx_i$

- The path of current node is stored in x[ ] if needed.

Algorithm: BBKnapsack (k, cw, cp)

```
begin
    // Generation of left child
    if (cw+w[k]≤M)
    begin
        tx[k] = 1
        if (k<n)
            BBKnapsack (k + 1, cw+w[k], cp +p[k]);
        end if
    if ((cp+p[k]>tp) and (k=n))
    begin
        tp = cp + p[k]
        tw = cw + w[k]
        for j=1 to k
            x [j] = tx [j]
        end for
    end if
end if

// Generation of right child
if (UpperBound (cw, cp, k, M) ≥ tp)
  begin
    tx[k] = 0
    if (k<n)
        BBKnapsack (k + 1, cw, cp);
    end if
    if ((cp>tp) and (k=n))
    begin
        tp = cp
        tw = cw
        for j=1 to k
            x [j] = tx [j]
        end for
    end if
end
```

Initially the algorithm is called as BBKnapsack (1, 0, 0).

5.5.1 LC Branch-and-Bound

Let us solve one example :

SOLVED EXAMPLES

Example 5.1 : *Consider the knapsak instance n = 4, profits (p_1, p_2, p_3, p_4) = (10, 10, 12, 18), weights (w_1, w_2, w_3, w_4) = (2, 4, 6, 9), maximum capacity M = 15. Solve the problem using LC branch-and-bound approach.*

Solution : LCBB will generate the following tree in Fig. 5.1

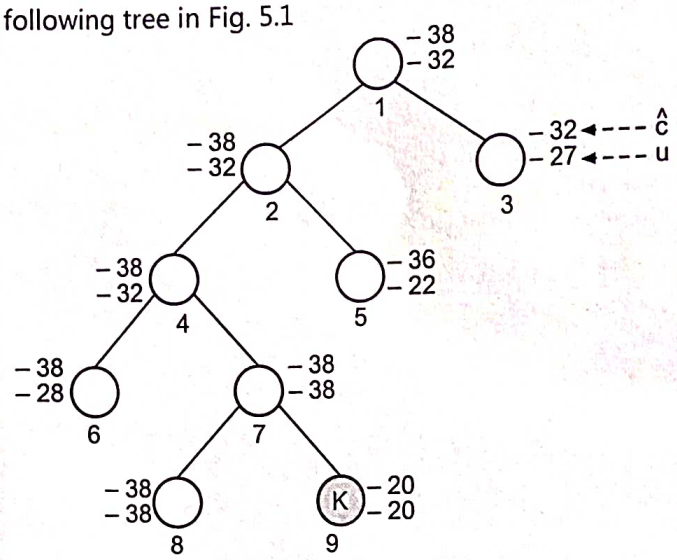


Fig. 5.1 : LCBB tree

For root node 1, $\hat{c}(1) = -38$, $u(1) = -32$

They are calculated as

$$u(1) = \text{UpperBound}(0, 0, 0, 15) = -10 - 10 - 12 = -32$$

$$\hat{c}(1) = -10 - 10 - 12 - \frac{3}{9} * 18 = -38$$

- Node 1 is not a solution node, hence answer = 0 and UpperBound (UB) = – 32. The E-node 1 is expanded which generates children 2 and 3. u(2) = – 32, $\hat{c}(2) = -38$, u(3) = – 27, $\hat{c}(3) = -32$. Add 2 and 3 to the list of live nodes. Next E-node is 2 which generates 4 and 5, add them to live nodes list. Next E-node 4 generates 6 and 7, add them to live nodes list. Node 7 changes UB to – 38, generates nodes 8 and 9, and adds them to live nodes list. Node 8 is a solution node. Node 9 has $\hat{c}(9)$ > UB and hence killed. Nodes 6 and 8 have least $\hat{c}$. For both, $\hat{c} \geq$ UB, hence the search terminates with node 8 as an answer node. Optimal solution is (x_4, x_3, x_2, x_1) = (1, 0, 1, 1). Though LCBB finds solution using less number of nodes, observe that it has to maintain heap. And insertions and deletions in heap are expensive.
- Hence FIFOBB can be used which uses FIFO queue which requires θ (1) time for insertions and deletions.

5.5.2 FIFO Branch-and-Bound

Example 5.2 : *Let us solve the above example 5.1 using FIFOBB:*

Solution : The FIFOBB tree is shown below in Fig. 5.2.

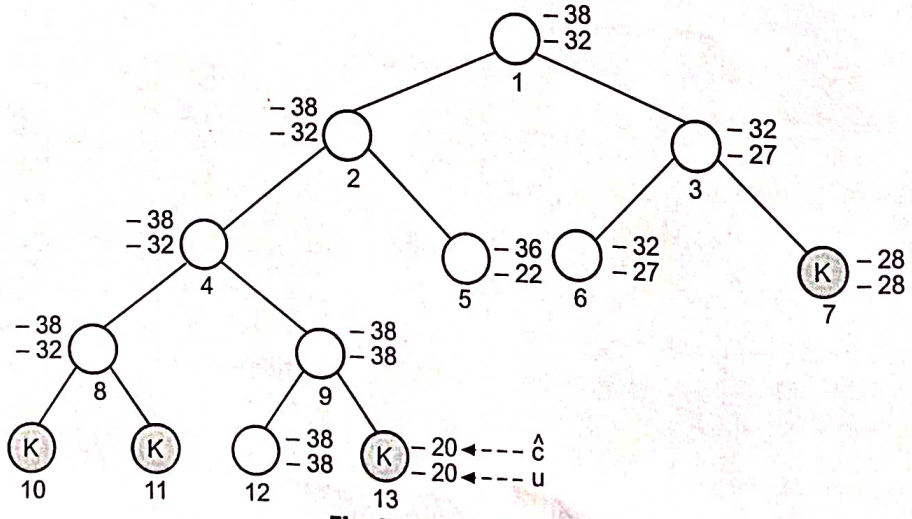


Fig. 5.2 : FIFOBB tree

- Initially queue is empty and root node 1 has u(1) = – 32, $\hat{c}(1)$ = – 38. Node 1 is not a solution node, it generates nodes 2 and 3, adds them to queue. UB remains same. Next node is 2 which generates nodes 4 and 5, and adds to queue. Then node 3 generates 6, 7 and adds 6 to queen. Node 7 is killed because $\hat{c}(7)$ > UB. Node 4 generates 8, 9 and adds to queue, changes u(9) = – 38, UB = – 38. Then nodes 5, 6 are considered. But their $\hat{c}$ > UB, hence not expanded. Node 8 generates nodes 10, 11, kills node 10 because it is infeasible, kills node 11 because $\hat{c}(11)$ > UB. Node 9 generates nodes 12, 13, adds 12 to queue, kills 13 because $\hat{c}(13)$ > UB, changes UB = – 38, answer = 12. Now there is only one node 12 in queue, but it has no children, hence the search terminates.
- We have solved the 0 / 1 knapsack problem using different approaches : greedy approach, dynamic programming, backtracking, LCBB, FIFOBB. Which approach is the best ?
- We can decide it by writing programs, run them using different data, and obtain computing times. But still the results will vary due to effectiveness of bounding functions.

5.6 TRAVELING SALESPERSON PROBLEM

- The travelling salesperson problem is to find a minimum cost tour in a directed graph which visits each vertex in a graph exactly once.

Application of Travelling Salesperson Problem :

- Consider a graph in which a vertex represents city and an edge represents distance between two cities. If a sales person wants to visit all the cities for selling items starting and ending from home city, then the route taken by a sales person is a tour and he is interested in finding a minimum length tour.
- A company wants to use a robot to pack a produced item in a box at each machine after every hour. There are many machines. If a machine denotes vertex of a graph, then time required to pack an item is same at each machine. But the time required to move from one machine to other is cost of the edge which connects those two machines. So the route taken by a robot is tour and company is interested in finding a tour which requires minimum time.

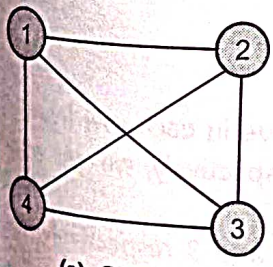
Using dynamic programming approach, the problem can be solved in $O(n^2 2^n)$ time, but it requires $O(n2^n)$ space. But if the problem instance is considered while solving, then lesser time is required if proper bounding functions are selected.

Let us solve the problem using branch-and-bound approach. Let G = (V, E) be a directed graph. Let c_{ij} denotes the cost of edge <i, j>. $c_{ij} = \infty$ if edge <i, j> $\notin$ E. Let the graph has n vertices, hence |V| = n. Assume that the tour of salesperson starts and ends at vertex 1. So the solution space S is given by path $(i_0, i_1, ..., i_{n-1}, i_n)$ where $i_0 = i_n = 1$. Obviously this is possible iff there is an edge $<i_k, i_{k+1}> \in E(1 <= k < n)$. For each adjacent pair of vertices in path S. The solution space S can be represented as a state space tree.

Example 5.3 : *Consider the following graph G in Fig. 5.3 (a).*

This is a graph with n = 4. For this graph, state space tree is as shown below in Fig. 5.3 (b).

Solution:



(a) Graph G

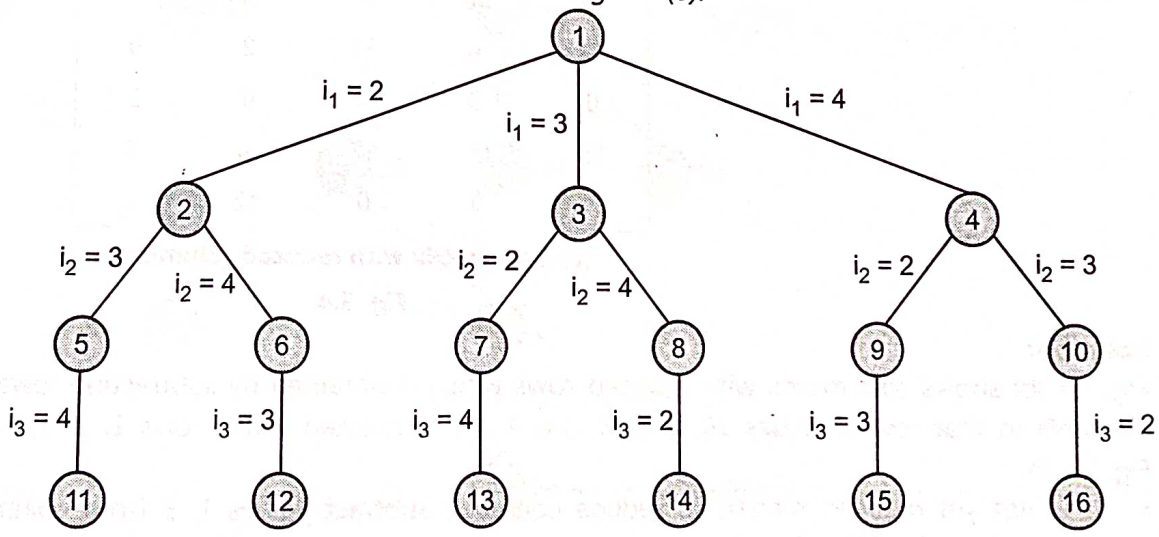


Fig. 5.3 : (b) State space tree for graph G

Here each leaf node is a solution node which defines a path from root to that node. For example, node 12 defines tour $(i_0, i_1, i_2, i_3, i_4) = (1, 2, 4, 3, 1)$.

We have already seen FIFOBB and LCBB techniques. Let us use LCBB here. We can use either static space tree dynamic space tree. Let us first solve the problem using static space tree by LCBB search.

5.6.1 LCBB Using Static State Space Tree

It requires c(.), $\hat{c}$(.) and u(.).

Definition of Cost Function c(.) :

- Let cost function c(.) is defined in the following way :
  - ➢ If R is a leaf node, then c(R) is a length of path from root to node R.
  - ➢ If R is a non-leaf node, then c(R) is a cost of minimum-cost leaf in the subtree of R.

Definition of $\hat{c}$(.) Function :

- $\hat{c}$(R) denotes the lower bound of node R. It can be obtained by using a **reduced cost matrix**. A matrix is reduce every row and column is reduced. A row (column) is **reduced** if it contains at least one zero value and all remaining entries in that row (column) are non-negative values.
- We can represent graph G as a cost matrix M. To find $\hat{c}$( ), do the following :
  1. Let k is the minimum value in row i.
  2. Subtract k from each value in row i.
  3. Repeat steps 1 and 2 for all rows and all columns until we obtain a reduced matrix.
  4. The total of all k values used for subtraction, say L, is $\hat{c}$(.) for root node in state space tree.
- Actually it tells the lowest value for all tours in graph G.

Let us solve one example :

Example 5.4 : *Consider the following cost matrix in Fig. 5.4 (a) which defines an instance of traveling salesperson.*

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

Fig. 5.4 : (a) Original cost matrix

∞	10	20	0	1
13	∞	14	2	0
1	3	∞	0	2
16	3	15	∞	0
12	0	3	12	∞

Fig. 5.4 : (b) Cost matrix with reduced rows

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

(c) Cost matrix with reduced columns

Fig. 5.4

Solution:

Fig. 5.4 (b) shows cost matrix with reduced rows which is obtained by subtracting lowest value in each row from all t elements in that row. If values 10, 2, 2, 3 and 4 are subtracted form rows 1, 2, 3, 4, 5 respectively, we get matrix Fig. 5.4 (b).

- It is not yet reduced matrix, so reduce columns. Subtract values 1, 3 from columns 1 and 3 respectively, we g reduced matrix of Fig. 5.4 (c).
- Total amount subtracted is 10 + 2 + 2 + 3 + 4 + 1 + 3 = 25. This is lower bound of root.

In this way, we have to find reduced cost matrix for each node in the state space tree. Let A is parent node of child B in the tree and M is a reduced cost matrix of node A. Let A and B are connected by an edge <x, y> from A to B. If B is a leaf node, then $\hat{c}(.) = c(.)$.

If B is a non-leaf node, then compute its reduced cost matrix as follows :

- Make all elements in row x and column y as ∞, because we don't want to consider edges <x, k> or <k, y>, that is, remaining edges going from vertex x and remaining edges coming to vertex y.
- Change M[y, 1] = ∞ to prevent use of edge <y, 1>.
- Except the rows and columns containing only ∞, all the remaining rows and columns should be reduced. Let L be the total amount subtracted to obtain reduced cost matrix in this step.
- Compute $\hat{c}(B) = \hat{c}(A) + M[x, y] + L$

mputation of u( ) :

For all the nodes A in state space tree, upper bound u(A) = ∞.

As we have defined c( ), $\hat{c}$( ) and u( ), we can apply LCBB algorithm now. Consider the same cost matrix again :

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

(a) Initial cost matrix

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

(b) Reduced cost matrix with L = 25

Fig. 5.5

This is reduced cost matrix of root node of state space tree with L = 25, hence $\hat{c}$ ( ) = 25 and upper = ∞. Using this, we can generate reduced cost-matrix of children of root node. We obtain the following state space tree in Fig. 5.6 for the given traveling salesperson instance with n = 5.

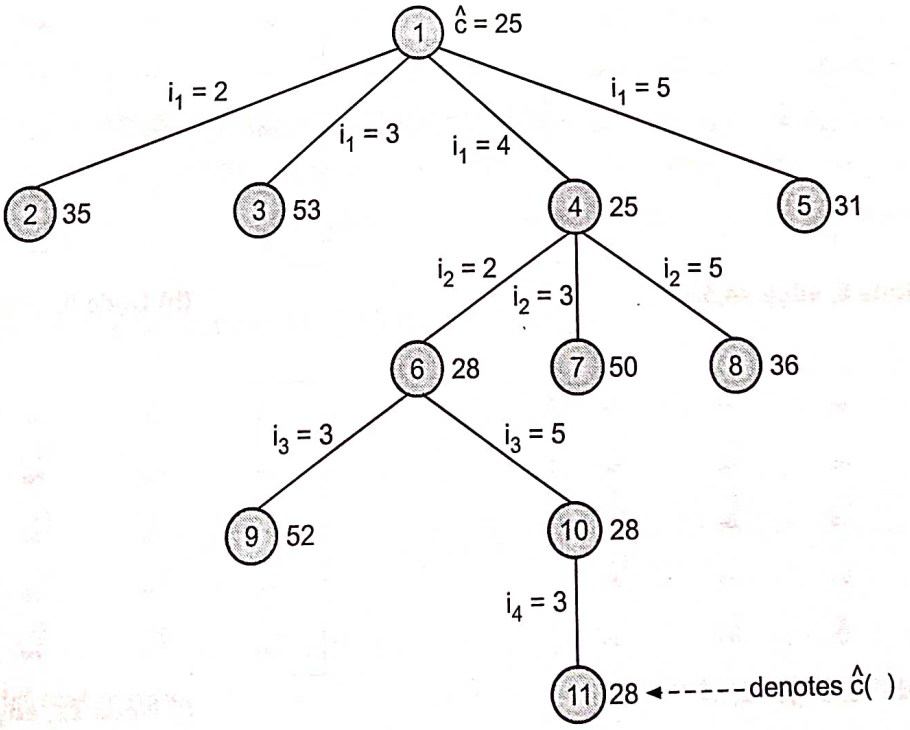


Fig. 5.6 : State space tree

- Reduced cost matrix should be computed for each node generated to find its $\hat{c}$ value. For all the non-leaf nodes except root, the reduced cost matrices are as shown below in Fig. 5.7.

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

(a) Node 2, edge <1,2>

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

(b) Node 3, edge <1,3>

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

(c) Node 4, edge <1,4>

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

(d) Node 5, edge <1,5>

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

(e) Node 6, edge <4,2>

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

(f) Node 7, edge <4,3>

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

(g) Node 8, edge <4,5>

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

(h) Node 9, edge <2,3>

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

(i) Node 10, edge <2, 5>

∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
∞	∞	∞	∞	∞

(j) Node 11, edge <5,3>

Fig. 5.7

- Let us see how reduced cost matrix of node 4 is obtained. Node 1 and 4 denote edge <1, 4>.

1. First all the elements of row 1 and column 4 of node 1 matrix are set to ∞.
2. Set M[4, 1] = ∞. We obtain

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ \infty & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & \infty & \infty \end{bmatrix}$$

3. It is already a reduced matrix, hence L = 0.
4. Lower bound of node 4 is

$$\hat{c}(4) = \hat{c}(1) + M[1, 4] + L = 25 + 0 + 0 = 25$$

Note that M is a reduced cost matrix of parent node 1.

Similarly,

$$\hat{c}(11) = \hat{c}(10) + M[5, 3] + L = 28 + 0 + 0 = 28$$

In this way, we can find $\hat{c}$ values of all the nodes .

- Root node 1 has 4 children 2, 3, 4, 5 having $\hat{c}$ values 35, 53, 25, 31 respectively. Since node 4 has lowest $\hat{c}$ value, it will be expanded further to have children 6,7, 8. Again node 6 has lowest $\hat{c}$ value among nodes 2, 3, 6, 7, 8, 5. Hence node 6 will be expanded further to have children 9 and 10. Again node 10 has lowest $\hat{c}$ value, hence it is expanded to have child node 11. Among all leaf nodes node 11 has lowest $\hat{c}$ value = 28, hence upper is modified to 28. Tour length of node 11 is 28. Now among the remaining children, node 5 has the lowest $\hat{c}$ value = 31, but 3 > upper. Hence LCBB search terminates here. The minimum cost tour is 1, 4, 2, 5, 3, 1.

5.6.2 LCBB Using Dynamic State Space Tree

- In this approach, a state space tree is generated which is dynamic binary tree, in which a left branch represents inclusion of an edge and a right branch represents exclusion of an edge. Depending on the problem to be solved, the order in which edges are considered differs. The method to compute $\hat{c}$ is same as in the previous section.
- If an edge <x, y> and right subtree represent all paths not including edge<x, y> at each node, that edge should be selected which has the highest probability to be in a minimum cost tour. One common approach for edge selection is to select that edge which will result in a right subtree having highest $\hat{c}$ value. We can achieve this by selecting an edge with reduced cost 0.

Example 5.5 : *Consider again the same problem instance shown in Fig. 5.8 (a) :*

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix}$$

(a) Initial cost matrix

$$\begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$

(b) Reduced cost matrix with L = 25

Fig. 5.8

Solution:

- Because total amount of subtraction L = 25, all tours in the graph will have minimum length 25. Hence $\hat{c}$ of root =
  The dynamic state space tree is shown below in Fig. 5.9.

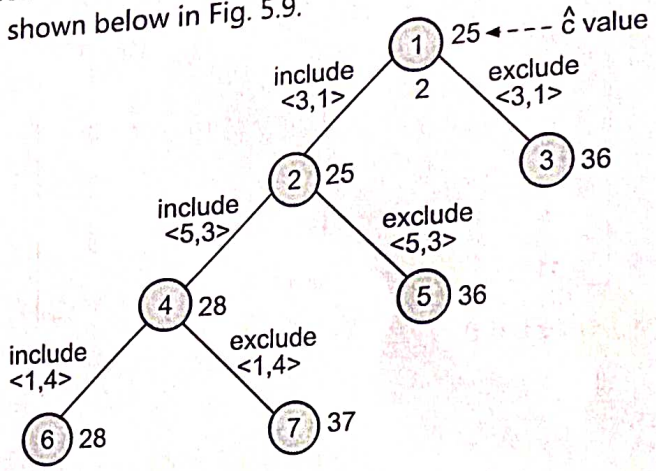


Fig. 5.9 : Dynamic state space tree

- At root node, we have to select an edge which results in a right child having highest ĉ value. This can be don selecting an edge with reduced cost 0. If the next edge is one of <1, 4>, <2, 5>, <3, 1>, <3, 4>, <4, 5>, <5, 2>, <5 then the resultant cost matrix B will have ∞ at that B[a, b] position for edge <a, b>. Matrix B should be convert reduced cost matrix, which will increase reduced cost of right child.
  - ➢ If an edge <1, 4> is included, then B[1, 4] = ∞, reduce row 1 by subtracting 1, column 4 remains reduced. He of right child will increase by 1.
  - ➢ If edge <3, 1> is included, then B[3, 1] = ∞, reduce column 1 by subtracting 11, row 3 remains reduced. He of right child will increase by 11.
  - ➢ In this way if the included edge is one of <1, 4>, <2, 5>, <3, 1>, <3, 4>, <4, 5>, <5, 2>, <5, 3>, then $\hat{c}$ of child will increase by 1, 2, 11, 0, 3, 3, 11 respectively. In general, if M is the reduced matrix of parent A inclusion of edge <a, b> where M [a, b] = 0 will increase $\hat{c}$ value of right child by

$$\Delta = \min_{k \neq y} \{M(x, k)\} + \min_{k \neq x} \{M(k, y)\}$$

  which should be subtracted from row x and column y to reduce them.

- Here edges <3, 1> and <5, 3> both will increase $\hat{c}$ of right child by 11. Let us select <3, 1>. Hence row 3 and co 1 will have all ∞. B[1, 3] = ∞ to avoid cycle. The resultant matrix B is shown in Fig. 5.10 (a) below. Like this we car reduced cost matrix when edge <3, 1> is excluded as shown in Fig. 5.10 (b).

$$\begin{bmatrix} \infty & 10 & \infty & 0 & 1 \\ \infty & \infty & 11 & 2 & 0 \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 3 & 12 & \infty & 0 \\ \infty & 0 & 0 & 12 & \infty \end{bmatrix}$$

(a) Node 2, edge <3, 1> included, L = 0

$$\begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix} \quad \text{reduced matrix} \Longrightarrow \quad \begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 1 & \infty & 11 & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 4 & 3 & 12 & \infty & 0 \\ 0 & 0 & 0 & 12 & \infty \end{bmatrix}$$

(b) Reduced cost matrix for node3, edge <3, 1> excluded, L = 11

Fig. 5.10

Node 2 has the lowest $\hat{c}$ value among nodes 2 and 3, hence node 2 will be expanded next. E-node = node 2. For selection of next edge, consider following order for edges having reduced cost 0 in Fig (a) : edge <1, 4>, <2, 5>, <4, 5>, <5, 2>, <5, 3>. For these edges Δ = (1 + 2 = 3), 2, 3, 3, 11 respectively. So next edge will be <5, 3> which has maximum Δ.

Fig. 5.10 (c) is obtained when <5, 3> included, row 5 and column 3 changed to ∞. M [3, 5]= ∞. Also M[1, 5] = ∞, as <3, 1>, <5, 3> are already included and inclusion of edge <1, 5> will create cycle.

$$\begin{bmatrix} \infty & 10 & \infty & 0 & \infty \\ \infty & \infty & \infty & 2 & 0 \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 3 & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix} \xrightarrow{\text{reduced matrix}} \begin{bmatrix} \infty & 7 & \infty & 0 & \infty \\ \infty & \infty & \infty & 2 & 0 \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

(c) Reduced cost matrix for node 4, edge <5, 3> included, L = 3, $\hat{c}$ (4) = 25 + 3 = 28

$$\begin{bmatrix} \infty & 10 & \infty & 0 & 1 \\ \infty & \infty & 11 & 2 & 0 \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 3 & 12 & \infty & 0 \\ \infty & 0 & \infty & 12 & \infty \end{bmatrix} \xrightarrow{\text{reduced matrix}} \begin{bmatrix} \infty & 10 & \infty & 0 & 1 \\ \infty & \infty & 0 & 2 & 0 \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 3 & 1 & \infty & 0 \\ \infty & 0 & \infty & 12 & \infty \end{bmatrix}$$

M[5, 3] = ∞

(d) Reduced cost matrix for node 5, edge <5. 3> excluded, L = 11, $\hat{c}$ (5) = 25 + 11 = 36

Fig. 5.10

Again node 4 has lowest $\hat{c}$ value, hence it will be expanded next. E-node = node 4. Consider the edges in the following order from Fig. 5.10 (c) : edge <1, 4>, <2, 5>, <4, 2>, <4, 4>. They give Δ = (7 + 2 = 9), 2, 7, 0 respectively. As edge <1, 4> has maximum Δ, it will be selected next.

Fig. 5.10 (e) shows reduced cost matrix when edge <1, 4> is included. M[4, 1] = ∞, row 1 and column 4 set to ∞. So now path includes <3, 1>, <5, 3>, <1, 4>. Also M[4, 5] = ∞, as inclusion of edge <4, 5> will form cycle.

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

Fig. 5.10 : (e) Reduced cost matrix for node 6, edge <1, 4> included, L = 0, $\hat{c}$ (6) = 28 + 0 = 28

$$\begin{bmatrix} \infty & 7 & \infty & \infty & \infty \\ \infty & \infty & \infty & 2 & 0 \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix} \xrightarrow{\text{reduced matrix}} \begin{bmatrix} \infty & 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & 0 & 0 \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

Fig. 5.10 : (f) Reduced cost matrix for node 7, edge <1, 4> excluded, L = 7 + 2 = 9, $\hat{c}$ (7) = 28 + 9 = 37

- Node 6 has lowest $\hat{c}$ value. Hence it will be expanded next. E-node = Node 6. Consider edges in the following order from Fig. 5.10 (e) <2, 5>, <4, 2>. They give Δ = 0, 0 respectively. So both edges can be selected directly. Hence edges in path are <3, 1>, <5, 3>, <1, 4>, <4, 2>, <2, 5>, which give path 1, 4, 2, 5, 3, 1 with length 28. Hence upper is modified to 28. In the remaining nodes, node 3 and node 5 have smaller $\hat{c}$ value = 36. Let node 3 becomes next E-node, whose $\hat{c}$ value = 36 > upper. Hence LCBB search terminates here.
- We can conclude that the LCBB search using dynamic state space tree works better for large subtrees. When a small subtree is reached, then it is evaluated without using the bounding functions.

Let us solve few examples :

Example 5.6 : *Consider the following cost matrix for a traveling salesperson instance. Compute reduced cost matrix and draw the dynamic state space tree obtained using LCBB. Also write the reduced cost matrices of all the nodes.*

$$\begin{bmatrix} \infty & 7 & 3 & 12 & 8 \\ 3 & \infty & 6 & 14 & 9 \\ 5 & 8 & \infty & 6 & 18 \\ 9 & 3 & 5 & \infty & 11 \\ 18 & 14 & 9 & 8 & \infty \end{bmatrix}$$

Solution : Subtract 3, 3, 5, 3, 8 from the row 1, 2, 3, 4, 5 respectively. We get first matrix in Fig. 5.11 (a).

$$\begin{bmatrix} \infty & 4 & 0 & 9 & 5 \\ 0 & \infty & 3 & 11 & 6 \\ 0 & 3 & \infty & 1 & 13 \\ 6 & 0 & 2 & \infty & 8 \\ 10 & 6 & 1 & 0 & \infty \end{bmatrix} \quad \text{Reduced column 5 =====>} \quad \begin{bmatrix} \infty & 4 & 0 & 9 & 0 \\ 0 & \infty & 3 & 11 & 1 \\ 0 & 3 & \infty & 1 & 8 \\ 6 & 0 & 2 & \infty & 3 \\ 10 & 6 & 1 & 0 & \infty \end{bmatrix}$$

Fig. 5.11 : (a) Reduced cost matrix of root node 1, $\hat{c}$ (1) = 3 + 3 + 5 + 3 + 8 + 5 = 27

Initially upper = ∞.

- Consider edges with reduced cost = 0 in the following order : <1, 3>, <1, 5>, <2, 1>, <3,1>, <4,2>, <5, 4>. They give Δ = 1, 1, 1, 1, (2 + 3 = 5), (1 + 1 = 2) respectively. Hence edge <4, 2> giving maximum Δ is selected next. Reduced cost matrix when <4, 2> is included is shown in Fig. 5.11 (b). It has M[4, 2] = ∞, row 4 and column 2 is set to ∞. Also M[2, 4] = ∞ to avoid cycle.

$$\begin{bmatrix} \infty & \infty & 0 & 9 & 0 \\ 0 & \infty & 3 & \infty & 1 \\ 0 & \infty & \infty & 1 & 8 \\ \infty & \infty & \infty & \infty & \infty \\ 10 & \infty & 1 & 0 & \infty \end{bmatrix}$$

Fig. 5.11 : (b) Node 2, edge <4, 2> included, L = 0, $\hat{c}$ (2) = 27 + 0 = 27

- The dynamic state space tree is as shown below in Fig. 5.11 (c).

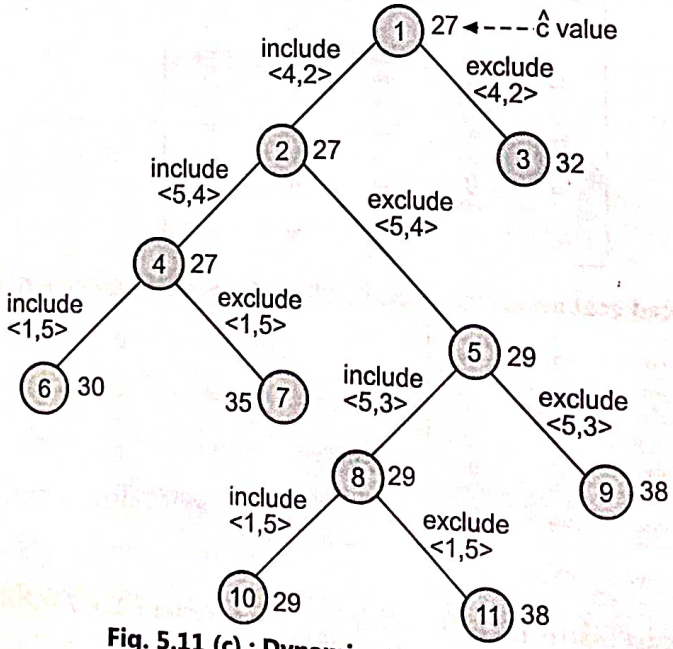


Fig. 5.11 (c) : Dynamic state space tree

$$\begin{bmatrix} \infty & 4 & 0 & 9 & 0 \\ 0 & \infty & 3 & 11 & 1 \\ 0 & 3 & \infty & 1 & 8 \\ 6 & \infty & 2 & \infty & 3 \\ 10 & 6 & 1 & 0 & \infty \end{bmatrix}$$ Reduced row 4 and column 2 =====> $$\begin{bmatrix} \infty & 1 & 0 & 9 & 0 \\ 0 & \infty & 3 & 11 & 1 \\ 0 & 0 & \infty & 1 & 8 \\ 4 & \infty & 0 & \infty & 1 \\ 10 & 3 & 1 & 0 & \infty \end{bmatrix}$$

Fig. 5.11 : (d) Node 3, edge <4, 2> excluded. L = 2+3 = 5, $\hat{c}$ (3) = 27 + 5 = 32

As node 2 has lowest $\hat{c}$ value, next E-node = node 2. Consider the edges from Fig. 5.11 (b) in the order : <1, 3>, <1, 5>, <2, 1>, <3, 1>, <5, 4>. They give Δ = 1, 1, 1, 1, (1 + 1 = 2) respectively. As edge <5, 4> has maximum Δ, it is selected next. Reduced cost matrix when edge <5, 4> is selected is as shown in Fig. 5.11 (e). It has M[4, 5]= ∞, row 5 and column 4 set to ∞. So now path includes <4, 2>, <5, 4> edges. As inclusion of edge <2, 5> creates cycle, set M[2, 5] = ∞.

$$\begin{bmatrix} \infty & \infty & 0 & \infty & 0 \\ 0 & \infty & 3 & \infty & \infty \\ 0 & \infty & \infty & \infty & 8 \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

Fig. 5.11 : (e) Node 4, edge <5, 4> included, L = 0, $\hat{c}$ (4) = 27 + 0 = 27

$$\begin{bmatrix} \infty & \infty & 0 & 9 & 0 \\ 0 & \infty & 3 & \infty & 1 \\ 0 & \infty & \infty & 1 & 8 \\ \infty & \infty & \infty & \infty & \infty \\ 10 & \infty & 1 & \infty & \infty \end{bmatrix}$$ Reduced row 5 column 4 =====> $$\begin{bmatrix} \infty & \infty & 0 & 8 & 0 \\ 0 & \infty & 3 & \infty & 1 \\ 0 & \infty & \infty & 0 & 8 \\ \infty & \infty & \infty & \infty & \infty \\ 9 & \infty & 0 & \infty & \infty \end{bmatrix}$$

M [5, 4] = ∞

Fig. 5.11 : (f) Node 5, edge <5, 4> excluded, L = 2, $\hat{c}$ (5) = 27 + 2 = 29

As node 4 has lowest $\hat{c}$ value, it is next expanded. E-node = node 4. From Fig. 5.11 (e), consider edges with reduced cost 0 in the following order : <1, 3>, <1, 5>, <2, 1>,<3, 1>. They give Δ = 3, 8, 3, 8 respectively. Edges <1, 5> and <3, 1> give maximum Δ. Let us select <1, 5> as next edge. Hence reduced cost matrix when edge <1, 5> is included, is as shown in Fig. 5.11 (g). Set M[5, 1] = ∞. Set row 1 and column 5 to ∞. So now path includes <4, 2>, <5, 4>, <1, 5>. As inclusion of edge <2, 1> creates cycle. Set M[2, 1] = ∞

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 3 & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$ Reduced matrix =====> $$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

Fig. 5.11 : (g) Node 6, edge <1, 5> included, L = 3, $\hat{c}$ (6) = 27 + 3 = 30

$$\begin{bmatrix} \infty & \infty & 0 & \infty & \infty \\ 0 & \infty & 3 & \infty & \infty \\ 0 & \infty & \infty & \infty & 8 \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$ Reduced column 5 =====> $$\begin{bmatrix} \infty & \infty & 0 & \infty & \infty \\ 0 & \infty & 3 & \infty & \infty \\ 0 & \infty & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

M[1, 5] = ∞

Fig. 5.11 : (h) Node 7, edge <1, 5> excluded, L = 8, $\hat{c}$ (7) = 27 + 8 = 35

- As node 5 has lowest $\hat{c}$ value, it is expanded next. E-node = node 5. form Fig. 5.11 (f), consider edges with reduced cost 0 in the order : <1, 3>, <1, 5>, <2, 1>, <3, 1>, <3, 4>, <5, 3>. They give Δ = 0, 1, 1, 0, 8, 9 respectively. As edge <5, 3> gives maximum Δ, it is selected next. Reduced cost matrix when edge <5, 3> is included, is as shown in Fig. 5.11 (i) below. Set M[3, 5] = ∞. Set row 5 and column 3 to ∞. Now path includes <4, 2>, <5, 3>.

$$\begin{bmatrix} \infty & \infty & \infty & 8 & 0 \\ 0 & \infty & \infty & \infty & 1 \\ 0 & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

Fig. 5.11 : (i) Node 8, edge <5, 3> excluded, L = 8, $\hat{c}(8) = 29 + 0 = 29$

$$\begin{bmatrix} \infty & \infty & 0 & 8 & 0 \\ 0 & \infty & 3 & \infty & 1 \\ 0 & \infty & \infty & 0 & 8 \\ \infty & \infty & \infty & \infty & \infty \\ 9 & \infty & \infty & \infty & \infty \end{bmatrix}$$

Reduced row 5 ====>

$$\begin{bmatrix} \infty & \infty & 0 & 8 & 0 \\ 0 & \infty & 3 & \infty & 1 \\ 0 & \infty & \infty & 0 & 8 \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \end{bmatrix}$$

M [5, 3] = ∞

Fig. 5.11 : (j) Node 9, edge <5, 3> excluded, L = 9, $\hat{c}(9) = 29 + 9 = 38$

- Among nodes 6, 7, 8, 9, 3 only node 8 has lowest $\hat{c}$ value, hence E-node = node 8. From Fig. 5.11 (i), consider edge with reduced cost 0 in the order : <1, 5>, <2, 1>, <3, 1>, <3, 4>. They give Δ = (8 + 1 = 9), 1, 0, 8 respectively. As edg <1, 5> gives maximum Δ. It is selected next. Reduced cost matrix when edge <1, 5> included is shown in Fig. 5.11 (k) Set row 1 and column 5 to ∞. Also set M [5, 1]= ∞ to avoid cycle. Now path contains edges <4, 2>, <5, 3>, <1, 5>. A inclusion of edge <3, 1> creates cycle {<5,3>, <3, 1>, <1, 5>}, set M[3, 1] = ∞.

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

Fig. 5.11 : (k) Node 10, edge <1, 5> included, L = 0, $\hat{c}$ (10) = 27 + 0 = 29

$$\begin{bmatrix} \infty & \infty & \infty & 8 & \infty \\ 0 & \infty & \infty & \infty & 1 \\ 0 & \infty & \infty & 0 & 8 \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

Reduced row 1 and column 5 ====>

$$\begin{bmatrix} \infty & \infty & \infty & 0 & \infty \\ 0 & \infty & \infty & \infty & 0 \\ 0 & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

M[1, 5} = ∞

Fig. 5.11 : (l) Node 11, edge <1, 5> included, L = 8 +1 = 9, $\hat{c}$ (4) = 29 + 9 = 38

- Among nodes 6, 7, 3, 10, 11, 9 node 10 has lowest $\hat{c}$ value, hence next E-node = node 10. From the reduce cost matrix of node 10 in Fig. 5.11 (k), if is clear that only 2 edges are remained : <2, 1> and <3, 4>. They can be selected directly. Now path contains edges <4, 2>, <5, 3>, <1, 5>, <2, 1>, <3, 4> which give tour of 1, 5, 3, 4, 2, 1 of length 29. Hence upper is modified to 29. In the remaining nodes, node 6 has lowest $\hat{c}$ value. But $\hat{c}$ (6) = 30 > upper. Hence LCBB terminates here. The minimum-cost tour is 1, 5, 3, 4, 2, 1 of length 29.

Example 5.7 : *Reduce the following cost matrix which gives a traveling salesperson instance. Using LCBB, generate dynamic state space tree, find reduced cost matrices for all the nodes in the tree, find the minimum-cost tour.*

$$\begin{bmatrix} \infty & 11 & 10 & 9 & 6 \\ 8 & \infty & 7 & 3 & 4 \\ 8 & 4 & \infty & 4 & 8 \\ 11 & 10 & 5 & \infty & 5 \\ 6 & 9 & 5 & 5 & \infty \end{bmatrix}$$

Solution : Subtract 6, 3, 4, 5, 5 from rows 1, 2, 3, 4, 5 respectively. We get first matrix in Fig. 5.12.

$$\begin{bmatrix} \infty & 5 & 4 & 3 & 0 \\ 5 & \infty & 4 & 0 & 1 \\ 4 & 0 & \infty & 0 & 4 \\ 6 & 5 & 0 & \infty & 0 \\ 1 & 4 & 0 & 0 & \infty \end{bmatrix}$$

Reduced column 1 =====>

$$\begin{bmatrix} \infty & 5 & 4 & 3 & 0 \\ 4 & \infty & 4 & 0 & 1 \\ 3 & 0 & \infty & 0 & 4 \\ 5 & 5 & 0 & \infty & 0 \\ 0 & 4 & 0 & 0 & \infty \end{bmatrix}$$

Fig. 5.12 : Reduced cost matrix of root node 1, $\hat{c}(1) = 6 + 3 + 4 + 5 + 5 + 1 = 24$

- From second matrix of Fig. 6.12 consider edges with reduced cost 0 in the order : <1, 5>, <2, 4>, <3, 2>, <3, 4>, <4, 3>, <4, 5>, <5, 1>, <5, 3>, <5, 4>. They give Δ = 3, 1, 4, 0, 0, 0, 3, 0, 0 respectively. Edge <3, 2> gives maximum Δ, hence it is selected next. The dynamic state space tree for this example is as shown below in Fig. 5.13.

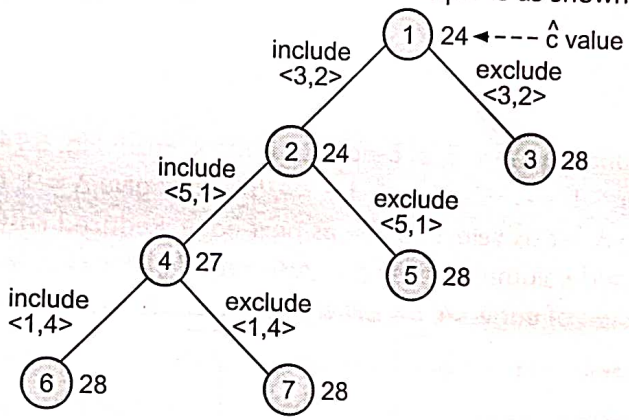


Fig. 5.13 : Dynamic state space tree

- Reduced cost matrix for node 2 when edge <3, 2> included is shown in Fig. 5.14 (a). Set row 3 and column 2 to ∞. Set M[2, 3] = ∞ to avoid creation of cycle.

$$\begin{bmatrix} \infty & \infty & 4 & 3 & 0 \\ 4 & \infty & \infty & 0 & 1 \\ \infty & \infty & \infty & \infty & \infty \\ 5 & \infty & 0 & \infty & 0 \\ 0 & \infty & 0 & 0 & \infty \end{bmatrix}$$

Fig. 5.14 : (a) Node 2, edge <3, 2> included, L = 0, $\hat{c}(2) = 24 + 0 = 24$

$$\begin{bmatrix} \infty & 5 & 4 & 3 & 0 \\ 4 & \infty & 4 & 0 & 1 \\ 3 & \infty & \infty & 0 & 4 \\ 5 & 5 & 0 & \infty & 0 \\ 0 & 4 & 0 & 0 & \infty \end{bmatrix}$$

Reduced column 2 =====>

$$\begin{bmatrix} \infty & 1 & 4 & 3 & 0 \\ 4 & \infty & 4 & 0 & 1 \\ 3 & \infty & \infty & 0 & 4 \\ 5 & 1 & 0 & \infty & 0 \\ 0 & 0 & 0 & 0 & \infty \end{bmatrix}$$

M [3, 2] = ∞

Fig. 5.14 : (b) Node 3, edge <3, 2> included, L = 4, $\hat{c}(3) = 24 + 4 = 28$

- As node 2 has lowest $\hat{c}$ value, it is expanded next. E-node = node 2. From Fig. 5.14 (a), consider edges with reduced cost 0 in the order : <1, 5>, <2, 4>, <4, 3>, <4, 5>, <5, 1>, <5, 3>, <5, 4>. They give Δ = 3, 1, 0, 0, 4, 0, 0 respectively. As edge <5, 1> fives maximum Δ, it is selected next. Reduced cost matrix when edge <5, 1> included is as shown in Fig. 5.14 (c), row 5 and column 1 is set to ∞. Also M[1, 5] = ∞ to avoid cycle, now path includes edges <3, 2>, <5, 1>.

$$\begin{bmatrix} \infty & \infty & 4 & 3 & \infty \\ \infty & \infty & \infty & 0 & 1 \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix} \xrightarrow{\text{Reduced row 1}} \begin{bmatrix} \infty & \infty & 1 & 0 & \infty \\ \infty & \infty & \infty & 0 & 1 \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

Fig. 5.14 : (c) Node 4, edge <5, 1> included, L = 3, $\hat{c}(4) = 24 + 3 = 27$

$$\begin{bmatrix} \infty & \infty & 4 & 3 & 0 \\ 4 & \infty & \infty & 0 & 1 \\ \infty & \infty & \infty & \infty & \infty \\ 5 & \infty & 0 & \infty & 0 \\ \infty & \infty & 0 & 0 & \infty \end{bmatrix} \xrightarrow{\text{Reduced column 1}} \begin{bmatrix} \infty & \infty & 4 & 3 & 0 \\ 0 & \infty & \infty & 0 & 1 \\ \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & \infty & 0 \\ \infty & \infty & 0 & 0 & \infty \end{bmatrix}$$

Fig. 5.14 : (d) Node 5, edge <5, 1> excluded, L = 4, $\hat{c}(5) = 24 + 4 = 28$

- As node 4 has lowest value among nodes 4, 5, 3, E-node = node 4. From Fig. 5.14 (c) reduced matrix, consider edge with reduced cost 0 in the order: <1, 4>, <2, 4>, <4, 3>, <4, 5>. They give Δ = 1, 0, 1, 0 respectively. As edges <1, 4 and <4, 3> both give maximum Δ, let us select <1, 4> as next edge. Reduced cost matrix when edge <1, 4> include is shown in Fig. 5.14 (e). Row 1 and column 4 is set to ∞. Also set M[4, 1] = ∞ to avoid cycle. Now path contains edge <3, 2>, <5, 1>, <1, 4>. As inclusion of edge <4, 5> will create cycle {<1, 4>, <4, 5>, <5, 1>}, set M[4, 5] = ∞

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 1 \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix} \xrightarrow{\text{Reduced row 2}} \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

Fig. 5.14 : (e) Node 6, edge <1, 4> included, L = 1, $\hat{c}(6) = 27 + 1 = 28$

$$\begin{bmatrix} \infty & \infty & 1 & \infty & \infty \\ \infty & \infty & \infty & 0 & 1 \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix} \xrightarrow{\text{Reduced row 1}} \begin{bmatrix} \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & 0 & 1 \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

M[1, 4] = ∞

Fig. 5.14 : (f) Node 7, edge <1, 4> excluded, L = 1, $\hat{c}(7) = 27 + 1 = 28$

- As all the nodes 6, 7, 5, 3 have same $\hat{c}$ value, let next E-node = node 6. Now only two edges are remained: <2, 5> and <4, 3>. They can be selected directly. Now path contains edges <3, 2>, <5, 1>, <1, 4>, <2, 5>, <4, 3>. Hence upper is modified to 28. Among the remaining 9 nodes, all have same $\hat{c}$ value, which is equal to upper. Hence LCBB terminates here. Minimum-cost tour is 1, 4, 3, 2, 5, 1 of length 28.

SOLVED EXERCISE

Long Question Answer :

1. Explain search strategies in branch and bound.

Ans. : There are four search strategies in branch and bound.

(1) LIFO (2) FIFO

(3) LC (4) Heuristic

(1) LIFO search is also termed as DFS (Depth First search). It explores live nodes in the reverse order of their creation. It uses stack data structure.

(2) FIFO search is also termed as BFS (Breadth First Search). It explores live nodes in the order of their creation. It uses queue data structure.

(3) DFS and BFS are special cases of LC search. LC search explores that node immediately which has a very good chance of getting the search to a solution node quickly. It uses priority queue as data structure.

(4) Heuristic search

2. Explain heuristic search in brief.

ns. : Heuristic search consists of the following characteristics : It helps to designs an algorithm with provably good run times and provably good or optimal solution quality. But there is no proof that it will always satisfy both these characteristics.

(a) Use of heuristic search is very common in real world implementations.

(b) A heuristic search might not find the best solution always, but it is guaranteed to find a good solution in reasonable time.

(c) It is useful to solve problems which require an infinite time for solving by other methods.

(d) A classic example of heuristic search is travelling salesperson problem.

- The heuristic search performs the following steps :

(a) It generates a possible solution which can either be a point in the problem state or a path from the initial state.

(b) It tests to see if this possible solution is a real solution by comparing the state reached with set of goal states

(c) If the possible solution is a real solution, them return, otherwise repeat step (a) again.

3. Write control abstraction for LC search.

ns. : It uses the following structure :

```
typedef struct ListNode
{
   ListNode *next, *parent;
   float cost;
*t, *x, *E-node;
```

lgorithm: LCSearch (t)

```
egin
   // Search t for an answer node.
   if *t is an answer node then
   begin
      output *t
      return
   endif
```

```
    E-node = t // Set E-node.

//Initialize list of live nodes to empty.
LiveNodes = ϕ
repeat
{
    for (each child x of E-node)
    begin
        if x is an answer node then
        begin
            output the path (x, t)
            return
        endif
    //Add x to list of live nodes
    LiveNodes = LiveNodes ∪ x
    // Set E-node as parent of node x
        x → parent = E-node
        end for

    if (there are no more live nodes) then
    begin
        print "No answer node"
        return
     endif
    E-node = Least ( )
}
until (false);
end
```

Function Least( ) searches LiveNodes list and returns a node having least cost $\hat{c}$.

EXERCISE

1. Write a short note on Branch and bound approach.
2. Explain in brief :
   (a) FIFO Search
   (b) LIFO Search
   (c) LC Search
   (d) Heuristic Search
3. List and explain various search strategies of branch and bound approach.
4. Write control abstraction for LC search.
5. Explain the term : bounding.
6. Define reduced cost matrix. Explain how we can obtain reduced cost matrix.
7. Define minimum cost tour.

UNIVERSITY QUESTION BANK

1. Discuss the use and the particular characteristics of the following algorithmic strategies :
   (i) Divide and conquer
   (ii) Backtracking
   Give explanation for each of the above strategies with the help of following examples respectively.
   (i) Merge-sort
   (ii) Eight Queen's problem
2. Consider the traveling sales person instance defined by the cost matrix:

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix}$$

   (a) Obtain the reduced cost matrix.
   (b) Using the space tree formulation, obtain the tree that will be generated by LCBB. Label each node by its value. Write out the reduced matrices corresponding to each of nodes.
   (c) What is path and minimum cost of tour for traveling sales person in above problem?
   (d) What do you mean by function?
3. Devise LC branch and bound algorithm for traveling salesman problem.
4. (a) Describe following w.r.t Branch and Bound :
      (i) The method
      (ii) Least-Cost search (LC-Search).
      (iii) Control-abstraction for LC-search.
      (iv) Bounding.
   (b) Write LCBB algorithm for knapsack problem using the fixed tuple size formulation and the state space tree.
5. (a) Develop an algorithm based on the FIFO approach to solve the knapsack problem.
   (b) Write a Reduction Algorithm, which defectively reduced a knapsack instance with large n to an equivalent instance with smaller n.
6. Develop the LCBB algorithm for the 0/1 Knapsack problem using a fixed tuple size formulation.
   State the assumptions made. The algorithm should include procedures to compute lower and upper bounds, creating a new node and output the answer.
7. Describe in brief "Branch and Bound strategy".
8. Consider the traveling salesman instance defined by the following cost matrix.

$$\begin{bmatrix} \infty & 7 & 3 & 12 & 8 \\ 3 & \infty & 6 & 14 & 9 \\ 5 & 8 & \infty & 6 & 18 \\ 9 & 3 & 5 & \infty & 11 \\ 18 & 14 & 9 & 8 & \infty \end{bmatrix}$$

   (a) Obtain reduced cost matrix.
   (b) Obtain the portion of the state space tree by LCBB. Label each node by its $\hat{c}$ value.

(c) Solve the above to compute the shortest tour?

(d) What do you mean by heuristic search?

9. Write any TWO of the following algorithms.

(a) Develop an algorithm based on the FIFO approach to solve the knapsack problem.

(b) Write an algorithm for implementing queues so that the following operations can be executed on line.

(i) ENQUE (i, A) : Add an integer i to queue A. Distinct integers or the same integers are permitted.

(ii) DEQUE (A) : Extract from queue A that element which has been there longest.

✠ ✠ ✠