UNIT - V

7 Branch and Bound

Syllabus

The method, Control abstractions for Least Cost Search, Bounding, FIFO branch and bound, LC branch and bound, 0/1 Knapsack problem - LC branch and bound and FIFO branch and bound solution, Traveling salesperson problem - LC branch and bound

Contents

SPPU : May-15, Marks 5

7.1 Basic Concept

- Branch and bound is a general algorithmic method for finding optimal solutions of various optimization problems.
- Branch and bounding method is a general optimization technique that applies where the greedy method and dynamic programming fail. However, it is much slower.
- It often leads to exponential time complexities in the worst case. On the other hand, if applied carefully, it can lead to algorithms that run reasonably fast on average.

7.1.1 Comparison between Backtracking and Branch and Bound

Sr. No.	Backtracking	Branch and bound
1.	Solution for backtracking is traced using depth first search.	In this method, it is not necessary to use depth first search for obtaining the solution, even the Breadth first search, best first search can be applied.
2.	Typically decision problems can be solved using backtracking.	Typically optimization problems can be solved using branch and bound.
3.	While finding the solution to the problem bad choices can be made.	It proceeds on better solutions. So there cannot be a bad solution.
4.	The state space tree is searched until the solution is obtained.	The state space tree needs to be searched completely as there may be chances of being an optimum solution any where in state space tree.
5.	Applications of backtracking are - M coloring, Knapsack.	Applications of branch and bound are - Job sequencing, TSP.

ReviewQuestion

Q.1 Explain the term : Compare backtracking and branch and bound method.

SPPU : May-15, End Sem, Marks 5

7.2 The Method

SPPU : May-07,09, Dec.-07, Marks 8

- In branch and bound method a state space tree is built and all the children of E nodes (a live node whose children are currently being generated) are generated before any other node can become a live node.

- For exploring new nodes either a **BFS** or **D-search** technique can be used.
- In Branch and bound technique, **BFS-like** state space search will be called **FIFO** (First In First Out) search. This is because the list of live node is First In First Out list (**queue**). On the other hand the **D-search** like state space search will be called **LIFO** search because the list of live node is Last in First out list (**stack**).
- In this method a space tree of possible solutions is generated. Then partitioning (called as branching) is done at each node of the tree. We compute lower bound and upper bound at each node. This computation leads to selection of answer node.
- **Bounding functions** are used to **avoid the generation of subtrees** that do not contain an answer node.

7.2.1 General Algorithm for Branch and Bound

The algorithm for branch and bound method is as given below.

```
Algorithm Branch_Bound()
{
//E is a node pointer;
 E ← new(node);        // This is the root node which is the
                       // dummy start node
//H is  heap for all the live nodes.
// H is a min-heap for minimization problems,
//H is a max-heap for maximization problems.
while (true)
{
if (E is a final leaf) then
   {
        //E is an optimal solution
        write( path from E to the root);
        return;
   }
 Expand(E);
 if (H is empty) then //if no element is present in heap
        {
             write(" there is no solution");
             return;
   }
 E ← delete_top(H);
}
}
```

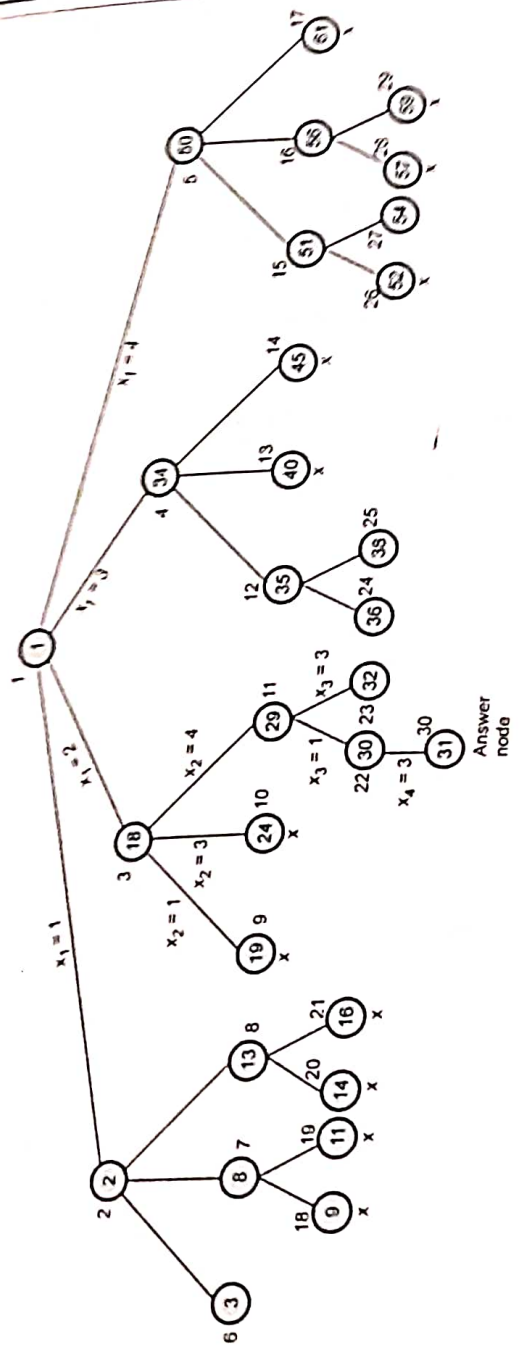


Fig. 7.2.1 Portion of state space tree using 4-queens

Following is an algorithm named Expand is for generating state space tree -

```
Algorithm Expand(E)
{
Generate all the children of E;
Compute the approximate cost value of each child;
Insert each child into the heap H;
}
```

For Example

- Consider the 4-queens problem using branch and bound method.
- In branch and bound method a bounding function is associated with each node.
- The node with-**optimum bounding function** becomes an **E-node**. And children of such node get expanded. The state space tree for the same is as given Fig. 7.2.1.
- In Fig. 7.2.1 based on bounding function the nodes will be selected and answer node can be obtained.
- The numbers that are written outside the node indicates the order in which evaluation of tree takes place. (Refer Fig. 7.2.1 on previous page)

Review Questions

Q.1 Write a short note on branch and bound method. **SPPU : May-07, 09, Marks 8**

Q.2 Describe in brief the general strategy used in branch and bound method. **SPPU : Dec.-07, Marks 4**

7.3 Control Abstractions for Least Cost Search

SPPU : Dec.-08, 10, May-12, 15, 17, 18, Marks 8

- In branch and bound method the basic idea is selection of E-node. The selection of E-node should so perfect that we will reach to answer node quickly.
- Using FIFO and LIFO branch and bound method the selection of E-node is very complicated and somewhat blind.
- For speeding up the search process we need to intelligent ranking function for live nodes. Each time, the next E-node is selected on the basis of this ranking function. For this ranking function additional computation (normally called as cost) is needed to reach to answer node from the live node.
- The Least Cost (LC) search is a kind of search in which least cost is involved for reaching to answer node. At each E-node the probability of being an answer node is checked.
- BFS and D-search are special cases of LC search.

- Each time the next E-Node is selected on the basis of the ranking function (smallest c^(x)). Let g^(x) be an estimate of the additional effort needed to reach an answer node from x. Let h(x) to be the cost of reaching x from the root and f(x) to be any non-decreasing function such that

$$c^(x) = f(h(x)) + g^(x)$$

- If we set g^(x) = 0 and f(h(x)) to be level of node x then we have BFS.
- If we set f(h(x)) = 0 and g^(x) g^(y) whenever y is a child of x then the search is a D-search.
- An LC search with bounding functions is known as LC Branch and Bound search.
- In LC search, the cost function c(.) can be defined as
  - i) If x is an answer node then c(x) is the cost computed by the path from x to root in the state space tree.
  - ii) If x is not an answer node such that subtree of x node is also not containing the answer node then c(x) = ∞.
  - iii) Otherwise c(x) is equal to the cost of minimum cost answer node in subtree x.
- c^(.) with f(h(x)) = h(x) can be an approximation of c(.).

7.3.1 Control Abstraction for LC Search

The control abstraction for Least cost search is given as follows.

```
Algorithm LC_search()
//tr is a state space tree and x be the node in the tr
//E represents the E-node
//initialize the list of live nodes to empty
{
        if( tr is answer node) then
        {
            write(tr);   //output the answer node
            return;
        }
        E←tr             //set the E-node
        repeat
        {
            for(each child x of E) do
            {
                if(x is answer node) then
                {
                    output the path from x to root;
                    return;
                }
                //the new live node x will be added in the list of live nodes
                Add_Node(x);
                x ← parent ← E;  //pointing the path from x to root;
            }
            if(no more live nodes) then
            {
                write("Can not have answer node||");
                return;
            }
                                 // E points to current E-node
            E ← Least_cost();    //finds the least cost node to set
                                 //next E-node
        }until(false);
}
```


In above algorithm if x is in tr then c(x) be the minimum cost answer node in tr. The algorithm uses two functions Least_cost and Add_Node() function to delete or add the live node from the list of live nodes. Using above algorithm we can obtain path from answer node to root. We can use parent to trace the parent of node x. Initially root is the E node. The for loop used in the algorithm examines all the children of E-node for obtaining the answer node. The function Least_cost() looks for the next possible E-node.

Review Questions

Q.1 Explain in detail control abstraction for LC search.

Q.2 What is LC search ? How does it help in finding a solution for branch and bound algorithm? SPPU : Dec.-08, Marks 8; Dec.-10, Marks 6

Q.3 Explain the term : Least cost branch and bound. SPPU : May-12, Marks 8

Q.4 What is LC search ? Explain in detail control abstraction for LC search. SPPU : May-15, End Sem, Marks 5

Q.5 Explain the term LC search. SPPU : May-17, End Sem, Marks 8 SPPU : May-18, End Sem, Marks 2

7.4 Bounding

SPPU : Dec.-12, May-14, 18, Marks 18

- As we know that the bounding functions are used **to avoid the generation of sub trees that do not contain the answer nodes.** In bounding lower bounds and upper bounds are generated at each node.
- A **cost function c^(x)** is such that c^(x) ≤ c(x) is used to provide the lower bounds on solution obtained from any node x.

- Let *upper* is an upper bound on cost of minimum-cost solution. In that case, all the live nodes with c^(x)> upper can be killed.
- At the start the upper is usually set to ∞. After generating the children of current E-node, *upper* can be updated by minimum cost answer node. Each time a new answer node can be obtained.
- Heuristic function : The heuristic is the one, which is used to decide which node is the best in the search tree to expand next. The heuristic function is specific to the problem under consideration. The heuristic function or the evaluation function is denoted by f'. The heuristic function or the evaluation function decides the next move to be made in the search space. The convention used could be a smaller value of the evaluation function leads earlier to the goal state. The next node to be expanded in the search tree will be the one having lowest f' value.

7.4.1 Job Sequencing with Deadlines

We will consider an example of job sequencing with deadlines to understand the computation of c^(x) and upper. Let us see the problem statement first.

Problem statement :

Let there be n jobs with different processing times. With only one processor the jobs are executed on or before given deadlines. Each job i is given by a tuple (P_i, d_i, t_i) where t_i is the processing time required by job i. If processing of job i is not completed by deadline d_i then penalty P_i will occur. The objective of this problem is to select a subset J such that penalty will be minimum among all possible subsets. Such a J should be optimal subset.

Ex. 7.4.1 *Let n = 4*

Job index	P_i	d_i	t_i
1	5	1	1
2	10	3	2
3	6	2	1
4	3	1	1

Then select an optimal subset J with optimal penalty. What will be the penalty corresponding to optimal solution ? **SPPU : May-14, Marks 18**

Sol. : The state space tree can be drawn for proper selection of job i for creating J. There are two ways by which the state space tree can be drawn : Fixed tuple size formulation and variable tuple size formulation. The variable tuple size formulation can be as shown below.

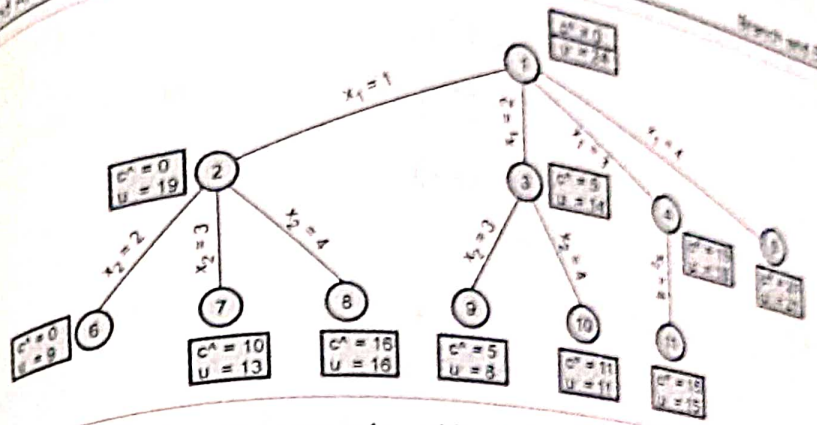


Fig. 7.4.1 State space tree with valable tuple size formulation

The function c^(x) can be computed for each node x as

$$c^{\wedge}(x) = \sum_{i < m \; i \notin S_x} P_i$$

where

$$m = \max \{i \mid i \in S_x\}$$

and S_x be subset of jobs selected for J at node x.

The upper bound can be computed using function u(x). Then,

$$u(x) = \sum_{i \notin S_x} P_i$$

Here u(x) corresponds to the cost of the solution S_x for node x.

For node 1 (root) there are 4 childrens. These children are for selection of either 1 or 2 or 3 or for job 4. Hence $x_1 = 2$ or $x_1 = 3$ or $x_1 = 4$.

For node 2 we have with $x_1 = 1$ and therefore we can select either 2, 3 or 4. Hence $x_2 = 2$ or $x_2 = 3$ or $x_2 = 4$ corresponds to node 6, 7 or 8. Continuing in this fashion, we have drawn the state space tree.

At node 1 :

c^ = 0

u = 24 $\because \sum_{i=1}^{n} P_i = 5 + 10 + 6 + 3$

At node 2 :

c^ = 0 $\because \sum P_i$ where $i < 1 = 0$

u = 19 $\because$ the sum of P_i excluding $x_1 = 1$, i.e. $P_1 = 5$.

At node 3 :

$c\hat{} = 5$ $\because \sum P_i$ where $i < 2$

$u = 14$ $\because \sum P_i$ without $x_1 = 2$ i.e. $24 - 10 = 14$

At node 4 :

$c\hat{} = 15$ $\because \sum P_i$ where $i < 3$

$u = 18$ $\because \sum P_i$ without P_3 i.e. $24 - 6 = 18$

At node 5 :

$c\hat{} = 21$ $\because \sum P_i$ where $i < 4$

$u = 21$ $\because \sum P_i$ without P_4 i.e. $24 - 3 = 21$

At node 6 :

$c\hat{} = 0$ $\because \sum_{i<2} P_i$ and not containing $x_1 = 1$ i.e. P_1

$u = 9$ $\because \sum_{i=1}^{n} P_i$ such that $i \neq 2$ and $i \neq 1$. Hence $6 + 3 = 9$

At node 7 :

$c\hat{} = 10$ $\because \sum_{i<3} P_i$ where $i \neq 1$ and $i \neq 3$. Hence $P_2 = 10$.

$u = 13$ $\because \sum_{i=1}^{n} P_i$ such that $i \neq 1$ and $i \neq 3$. Hence $P_2+P_4 = 13$

At node 8 :

$c\hat{} = 16$ $\because \sum_{i<4}^{n} P_i$ and $i \neq 1$ and $i \neq 4$. Hence $P_2 + P_3 = 16$.

$u = 16$ $\because \sum_{i=1}^{n} P_i$ and $i \neq 1$ and $i \neq 4$. Hence $P_2 + P_4 = 16$

At node 9 :

$c\hat{} = 5$ $\because \sum_{i<3}^{n} P_i$ but $i \neq 2$. Hence $P_1 = 5$.

$u = 8$ $\because \sum_{i=1}^{n} P_i$ but $i \neq 2$ and $i \neq 3$. That is $P_1 + P_4 = 8$

At node 10 :

$c\hat{} = 11$ $\because \sum_{i<4}^{n} P_i$ but $i \neq 2$. Hence $P_1 + P_3 = 5 + 6 = 11$.

$u = 11$

At node 11 :

$c\hat{} = 15$ $\because \sum_{i<4}^{n} P_i$ and $i \neq 3$. Hence $P_1 + P_2 = 5 + 10 = 15$

$u = 15$ $\because \sum_{i=1}^{n} P_i$ and $i \neq 3$ and $i \neq 4$. Hence $P_1 + P_2 = 5 + 10 = 15$

Now we will draw fixed tuple size formulation.

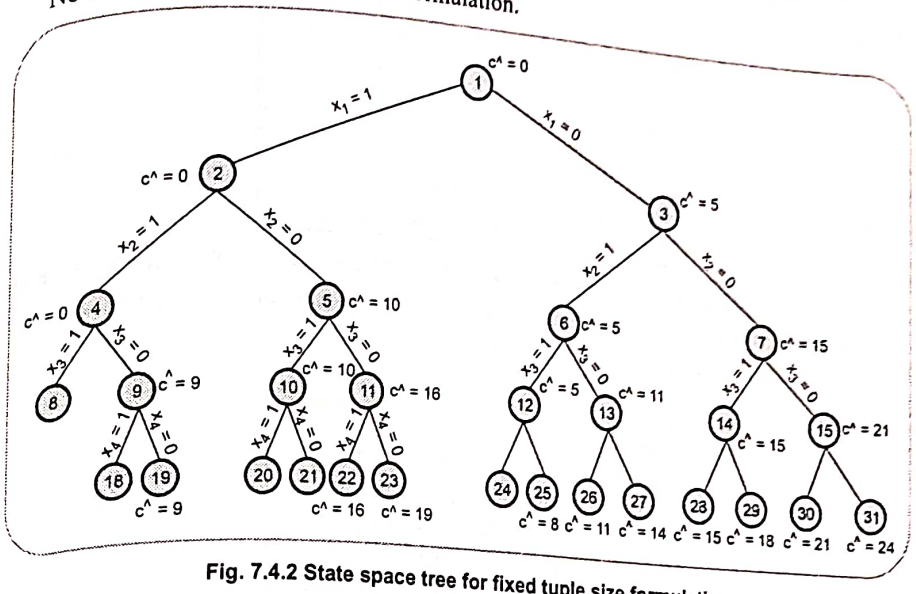


Fig. 7.4.2 State space tree for fixed tuple size formulation

In fixed tuple size formulation. $x_i = 1$ means job i is selected $x_i = 0$ means job i is not selected for formation of J (the optimal subset). Initially, $c\hat{} = 0$ for node 1 as no job is selected. For node 3, it indicates omission of job 1. Hence, penalty is 5. For node 5 there is omission of job 2. Hence $c\hat{}$ = penalty = 10. For node 7 there is omision of job 1 and job 2. Hence $c\hat{}$=15. For node 13 there is omission of job 1 and job 3. Hence penalty is 11. Therefore $c\hat{} = 11$. Continuing in this fashion $c\hat{}$ are computed.

Review Questions

Q.1 Explain the term :

a) Branch and bound b) Bounding function

c) Various searching techniques in branch and bound.

d) Heuristic function **SPPU : Dec.-12, Marks 8**

Q. 2 Explain the terms : Bounding function. **SPPU : May-18, End Sem. Marks 2**

7.5 FIFO Branch and Bound

SPPU : May18, Marks 8

To understand FIFO branch and bound, we will consider the variable tuple size formulation. The state space tree can be drawn as below. For job sequencing problem as -

For node u = 24, the upper is now 24. Then nodes 2, 3, 4 and 5 are generated u(2) = 19, u(3) = 14, u(4) = 18, u(5) = 21. The minimum value of u(x) will be upper. Hence upper will be updated to 14.

upper = 14
For node 4
c^(4) = 15
c^(4) > upper
i.e 15 > 14
Hence kill node 4.

upper = 14
For node 5
c^(5) = 21
c^(5) > upper
i.e 21 > 14
Hence kill node 5.

Node 2 and 3 become live nodes. So, now we will consider 2 and 3. We will expand node 2 (2 becomes E-node). u(6) = 9, u(7) = 13, u(8) = 16. Hence minimum of u(x) becomes upper. As u(6) = 9 is minimum. The upper will be updated to 9.

Hence,

upper = 9
For node 7
c^(7) = 10
c^(7) > upper
Hence kill node 7.

upper = 9
For node 8
c^(8) = 16
c^(8) > upper
Hence node 8
is not considered

The live node 3 being live node becomes E-node now. Now children 9 and 10 are generated. u(9) = 8 and u(10) = 11. Hence upper is updated to 8.

Fig. 7.5.1 Variable tuple sized formulation

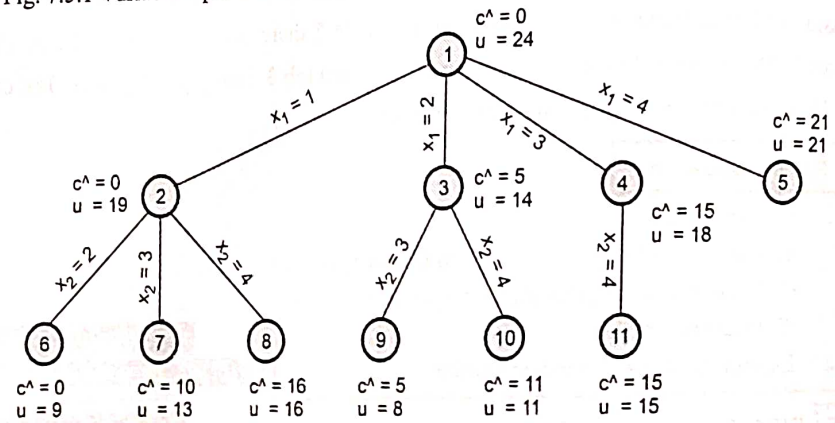


Fig. 7.5.1 Variable tuple sized formulation

Hence,

upper = 8
For node 10
c^(10) = 11
c^(10) > upper
Hence kill node 10.

Now node 6 becomes E-node. But as children of node 6 are infeasible we will not consider node 6. The only remaining live node is 9. The only child of 9 is infeasible. Now there is no live node remaining the minimum cost answer node is node 9. And it has a cost of 8. This method is referred as FIFO based branch and bound.

Algorithm

Algorithm for FIFO Branch and Bound is as given below

```
Algorithm FIFOBB()
//tr is a state space tree and x be the node in the tr
//E represents the E-node
//initialize the list of live nodes to empty
{
    if( tr is answer node) then
    {
        u:=min(cost[tr],(upper(tr)+E))
        write(tr);    //output the answer node
        return;
    }
    E←tr          //set the E-node
    repeat
    {
    for(each child x of E) do
    {
        if(x is answer node) then
        {
            output the path from x to root;
            return;
        } //the new live node x will be added in the list of          //live nodes
            Insert_Q(x);
            x.parent ←E;      //pointing the path from x to root;
        if((x is answer node) AND cost(x)<u) then
        {
            u←min(cost[tr],(upper(tr)+E))
        }
    }
```

```
    if(no more live nodes) then
    {
        write("No more Live nodes now");
        write("least Cost is =",u);
        return;
    }
    E← Del_Q();      // E points to current E-node
                     // deletes the least cost node from Q
                     // to set next E-node
    } until(false);
}
```

In above algorithm if x is in tr then c(x) be the minimum cost answer node in *tr*. The computation of u value is done. We always choose the minimum values of cost of the children generated from the current E node.

The algorithm uses two functions Del_Q and () and Insert_Q() function to delete or add the live node from the list of live nodes stored in queue. Using above algorithm we can obtain path from answer node to root. We can use parent to trace the parent of node x. Initially root is the E node. The for loop used in the algorithm examines all the children of E-node for obtaining the answer node.

Review Questions

Q.1 Present a program schema for a FIFO branch and bound search for a least cost answer node.

Q.2 Write an algorithm for FIFO branch and bound. **SPPU : May-18, End Sem. Marks 8**

7.6 LC Branch and Bound

SPPU : Dec.-13, 18, May-14, 19, Marks 12

In LC branch and bound method. For node 1 upper = 24. Now, node 1 becomes an E-node. Then node 2, 3, 4 and 5 are generated. The upper is now 14. As c^(4) > upper and c^5 > upper. Hence node 4 and 5 are killed. For node 2 and 3 the cost c^(2) is 0, node 2 becomes an E-node. Hence children node 6, 7 and 8 are generated. The upper is now 9 (because u(6) = 9). Node 7 and 8 are killed. Node 6 is selected as it has minimum cost. But both the children of node 6 are infeasible. So kill node 6. Now, node 3 becomes E-node. Then node 9 and 10 are generated. The upper = 8 now, since c^(10) > upper we will kill node 10. Now only remaining node is 9. Next E-node becomes node 9. Its only child is infeasible. As there are no live nodes remaining, we will terminate search with node 9 as an answer node. The principle idea in LC search is choosing of minimum cost node each time.

This method of LC based branch and bound with appropriate c^(.) and u(.) is called as LCBB (LIFOBB).

Algorithm

Algorithm for LIFO Branch and Bound is as given below

```
Algorithm LIFOBB()
//tr is a state space tree and x be the node in the tr
//E represents the E-node
//initialize the list of live nodes to empty
{
    if( tr is answer node) then
    {
        u←min(cost[tr],(upper(tr)+E))
        write(tr);   //output the answer node
        return;
    }
    E←tr        //set the E-node
    repeat
    {
        for(each child x of E) do
        {
            if(x is answer node) then
            {
                output the path from x to root;
                return;
            } //the new live node x will be added in the list of live nodes
            PUSH(x);
            x.parent← E;      //pointing the path from x to root;
            if( (x is answer node) AND cost(x)<u) then
            {
                u←min(cost[tr],(upper(tr)+E))
            }
        }
        if(no more live nodes) then
        {
            write("No more Live nodes now");
            write("least Cost is =",u);
            return;
        }      // E points to current E-node
        E ←POP();  // deletes the least cost node from stack to set next E-node
    } until(false);
}
```

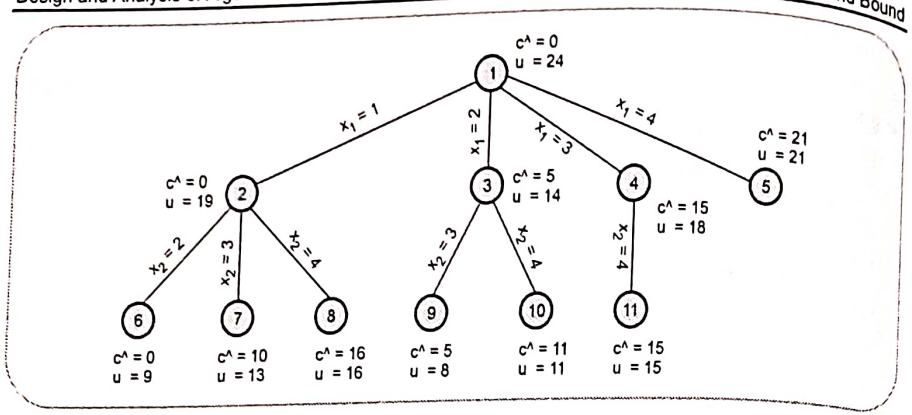


Fig. 7.6.1

In above algorithm if x is in tr then c(x) be the minimum cost answer node in tr. The computation of u value is done. We always choose the minimum values of cost of the children generated from the current E node.

The algorithm uses two functions POP () and PUSH() function to delete or add the live node from the list of live nodes stored in stack. Using above algorithm we can obtain path from answer node to root. We can use parent to trace the parent of node x. Initially root is the E node. The for loop used in the algorithm examines all the children of E-node for obtaining the answer node.

Review Questions

Q.1 Write a program schema for a LIFO branch and bound search for Least-cost answer node.

Q.2 Explain the terms :
Branch and Bound, LC, LIFO and Bounding function. How are LIFO and LC techniques different ? **SPPU : Dec.-13, Marks 9**

Q.3 Explain for Branch and Bound -
i) LIFO search ii) FIFO search iii) LC search **SPPU : May-14, Marks 12**

Q.4 What is least cost search ? Explain in detail control abstraction for LC search. **SPPU : Dec.-18, End Sem. Marks 8**

Q.5 Write an algorithm for least cost (LC) branch and bound. **SPPU : May-19, End Sem. Marck 5**

7.7 0/1 Knapsack Problem

SPPU : Dec.-10,12,16, 18, May-08,09,10,15,18, 19, Oct-16, Marks 8

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 $\sum P_i x_i \leq m$ and

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

We will discuss the branch and bound strategy for 0/1 Knapsack problem using fixed tuple size formulation. We will design the state space tree and compute c^(.) and u(.) where c^(x) represents the approximate cost used for computing the least cost c(x). Clearly u(x) denotes the upper bound. As we know upper bound is used to kill those nodes in the state space tree which can not lead to the answer node.

Let, x be the node at level j. Then we will draw the state space tree for fixed tuple formulation having levels $1 \leq j \leq n+1$.

Then we need to compute c^(x) and u(x). Such that $c^(x) \leq c(x) \leq u(x)$ for every node.

Algorithm

The algorithm for computing c^(x) is as given below.

```
Algorithm C_Bound(total_profit,total_wt,k)
//total_profit denotes the current total profit
//total_wt denotes the current total weight
// k is the index of last removed object
//w[i] represents the weight of object i
//p[i] represents the profit of object i
//m is the weight capacity of knapsack
{
    pt←total_profit;
    wt←total_wt;
    for(i←k+1 to n) do
    {
        wt←wt+w[i];
        if(wt<m) then pt←pt+p[i];
        else return (pt+(1 – (wt-m)/w[i])*p[i]);
    }
    return pt;
}
```

The algorithm for computing u(x) is as given below

```
Algorithm U_Bound(total_profit,total_wt,k,m)
//total_profit denotes the current total profit
//total_wt denotes the current total weight
// k is the index of last removed object
//w[i] represents the weight of object i
//p[i] represents the profit of object i
//m is the weight capacity of knapsack
{
    pt←total_profit;
    wt←total_wt;
    for(i←k+1 to n) do
    {
        if(wt+w[i]<=m) then
        {
            pt←pt – p[i];
            wt←wt+w[i];
        }
    }
    return pt;
}
```

7.7.1 LC Branch and Bound Solution

The LC branch and bound solution can be obtained using fixed tuple size formulation.

The steps to be followed for LCBB solution are

1. Draw state space tree.
2. Compute c^(.) and u(.) for each node.
3. If c^(x) > upper kill node x.
4. Otherwise the minimum cost c^(x) becomes E-node. Generate children for E-node.
5. Repeat step 3 and 4 until all the nodes get covered.
6. The minimum cost c^(x) becomes the answer node. Trace the path in backward direction from x to root for solution subset.

Ex. 7.7.1 *Consider Knapsack instance n = 4 with capacity m = 15 such that,*

Object i	P_i	W_i
1	10	2
2	10	4
3	12	6
4	18	9

SPPU : Dec.-16, 18 End Sem, Marks 18; May-17, 18, 19 End Sem, Marks 10

Sol. : Let us design state space tree using fixed tuple size formulation. The computation of c^(x) and u(x) for each node x is done.

In Fig. 7.7.1 at each node a structure is drawn in which computation of c^ and u is given. The tag field is useful for tracing the path.

Using algorithm U_Bound u(x) is computed and using algorithm C_Bound c^(x) is computed.

u(1) can be computed as -

for i = 1, 2 and 3

$$\therefore \quad \sum P_i = -(10 + 10 + 12)$$

$$\therefore \quad u(1) = -32$$

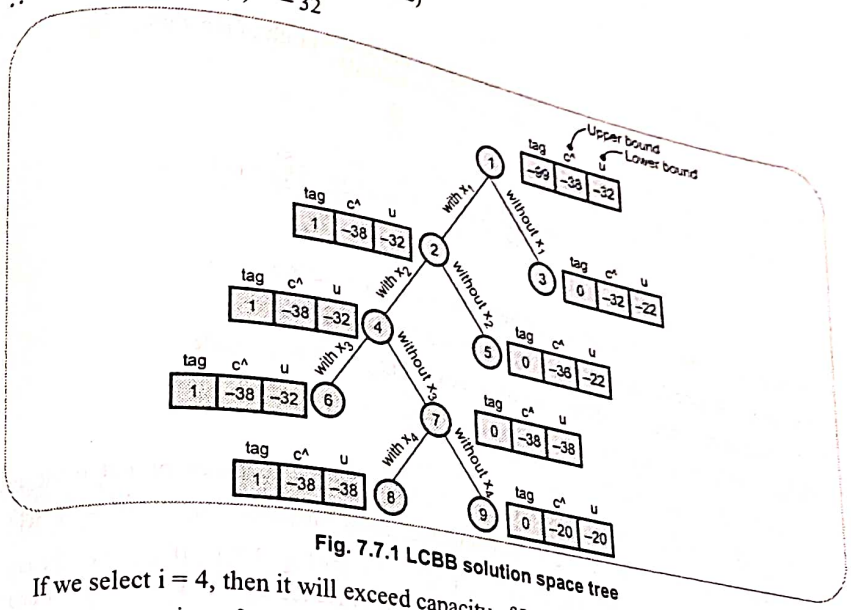


Fig. 7.7.1 LCBB solution space tree

If we select i = 4, then it will exceed capacity of Knapsack.

The computation of c^(1) can be done as follows :

$$c^{\wedge}1) = -\left[u(1) + \left[\frac{m - \text{Current total weight}}{\text{Actual weight of remaining object}} \right] * \left[\text{Actual profit of remaining object} \right] \right]$$

$$c^{\wedge}1) = -\left[32 + \left[\frac{15 - (2 + 4 + 6)}{9} \right] * 18 \right]$$

$$= -\left(32 + \frac{3}{9} * 18 \right)$$

$$c^{\wedge}1) = -38$$

Now at node 2,

Since the item 1 is selected

The values of c^ and u will be the same i.e. c^ = –38, u = – 32

Now at node 3

Do not select item 1.

∴ u(3) can be computed as

$$-\sum pi = -(10+12) \text{ i.e. with item 2 and 3.}$$

$$= 22$$

$$\therefore \quad u = 22$$

For computation of c^

$$c^{\wedge} = -\left[u + \left(\frac{m - \text{total selected wt}}{\text{remaining objects weight}} \right) * \text{profit of remaining object} \right]$$

Here remaining object is object 4.

It has P = 18 and W = 9

$$\therefore \quad c^{\wedge} = -\left[22 + \left(\frac{15-(4+6)}{9}\right) * 18\right] = -\left[22 + \left(\frac{5}{9}\right) * 18\right]$$

$$c^{\wedge} = -32$$

Now consider node 2. The difference between – 38 and – 32 is 6.

Similarly for node 3. The difference between – 32 and – 22 is 10.

The node having minimum difference between C^ and u becomes E node.

Hence node 2 becomes E node.

Therefore we expand node 2.

In this way considering each possibility of object being in Knapsack or not being in Knapsack with least cost c^(x) and u(x) is computed. Each time minimum $-\sum P_i x_i$ will become E-node an we will get the answer node as node 8. (Refer Fig. 7.7.1). If we trace the tag field we will get tag(2) – tag (4) – tag (7) – tag (8). i.e. 1101. Hence $x_4 = 1$, $x_3 = 0$, $x_2 = 1$ and $x_1 = 1$. We will select object x_4, x_2 and x_1 to fill up the Knapsack and gain maximum profit.

7.7.2 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 7.7.1).

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 $x_4 = 1$.

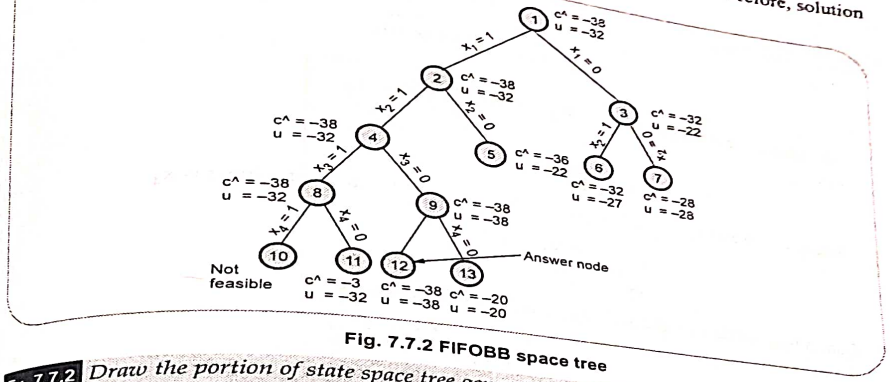


Fig. 7.7.2 FIFOBB space tree

Ex. 7.7.2 *Draw the portion of state space tree generated by LCKNAP for the Knapsack instances : n = 5, $(P_1, P_2, \ldots\ldots P_5) = (10, 15, 6, 8, 4)$, $(W_1, W_2, \ldots\ldots W_5) = (4,6,3,4,2)$ and $M = 12$.*

Sol. : We will compute u(x) and c^(x) for each node in a state space tree. The fixed tuple size formulation is considered to construct LCBB space tree. Following formulae will be used to compute u(.) and c^(.).

$$u(x) = -\sum P_i x_i$$

$$c^{\wedge}(x) = u(x) - \left[\frac{m - \text{Current total weight}}{\text{Actual weight of remaining object}} \right] * \left[\text{Actual profit of remaining object} \right]$$

For x = 1 i.e for root node.

$$u(x) = u(1) = -\sum P_i \text{ where } i = 1, 2 \text{ and } 5.$$

$$= -(P_1 + P_2 + P_5) = -(10 + 15 + 4) = -29$$

As by selecting P_1, P_2 and P_5 we get the total weight $W_1 + W_2 + W_5$ = Actual size of Knapsack. Hence,

$$c\hat{}(x) = c\hat{}(1) = u(1) - 0$$

$$c\hat{}(1) = -29$$

Again, for node 2 we select $x_1 = 1$. Hence computation of u(2) is,

$$u(2) = -\sum P_i = -(P_1 + P_2 + P_5)$$

$$u(2) = -29$$

$$c\hat{}(2) = -29.$$

Hence,

For node 3, when $x_1 = 0$. That means we want to find an upper bound u(x) without first object.

Then,

$$u(3) = -(P_2 + P_3 + P_5) = -(15 + 6 + 4)$$

$$u(3) = -25$$

The total current weight is $W_2 + W_3 + W_5 = 11$.

Hence,

$$c\hat{}(3) = u(3) - \left(\frac{m - 11}{W_4} * P_4\right)$$

When, $x_1 = 0$ and x_2, x_3 and x_5 are selected the only. Remaining object is x_4.

$$= -25 - \left(\frac{1}{4} * 8\right)$$

$$c\hat{}(3) = -27.$$

Continuing in this way the state space tree can be drawn as :

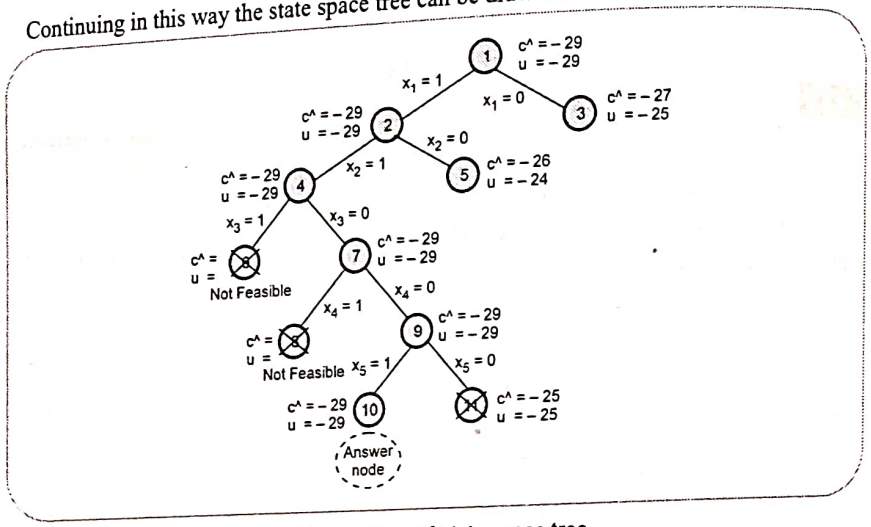


Fig. 7.7.3 Portion of state space tree

In Fig. 7.7.3, the upper is – 29. As node 2 has $c\hat{}(2) = -29$. Hence node 2 becomes an E node. Therefore generate children of node 2. Node 5 has $c\hat{}(5) >$ upper, hence we kill node 5 immediately. Node 4 becomes an E node. The children 6 and 7 of node 4 are generated. Node 6 does not form a feasible solution. Hence node 7 becomes an E-node. Continuing in this fashion the state space tree is generated. We get the solution to this Knapsack instance at node 10. Hence solution is $(x_1, x_2, x_3, x_4, x_5) = (1, 1, 0, 0, 1)$.

Ex. 7.7.3 *Find an optimal solution for the following 0/1 Knapsack instance using branch and bound method :*

Number of objects n = 5, capacity of knapsack m = 100

Profits = (10, 20, 30, 40, 50), weights = (20, 30, 66, 40, 60)

SPPU : Oct-16, (In Sem), Marks 10

Sol. : We will arrange all the items in such a way that $\frac{P_i}{W_i} > \frac{P_{i+1}}{W_{i+1}} > \frac{P_{i+2}}{W_{i+2}} \cdots$

The arrangement will be

Item	P_i	W_i
4	40	40
5	50	60
2	20	30
1	10	20
3	30	66

- Here we will select item 4, item 5. Now if we select item 2 then total weight > m. Hence by selecting item 4 and item 5 the upper bound for profit is obtained i.e. $40 + 50 = 90$ ∴ we will write ub = – 90 for root node, of BB tree. As by choosing item 4 and item 5 the total capacity of knapsack is fulfilled.

$$c\hat{} = ub = -90$$

Hence

- Now if we don't select item 4. Then consider selection of item 5 and item 2. Then upper bound for profit is ub = – 70. By this selection the total weight = $60 + 30 = 90$. Now we need only weight 10 to reach to maximum capacity of knapsack.

∴ Just consider next item for selection i.e. item 1 but we need its fractional weight

∴ $c\hat{} =$ ub + fractional profit of next item $= (70) + \left(\frac{10}{20} * 10\right) = 75$

∴ $c\hat{} = -75$

- The partial state space tree will be -

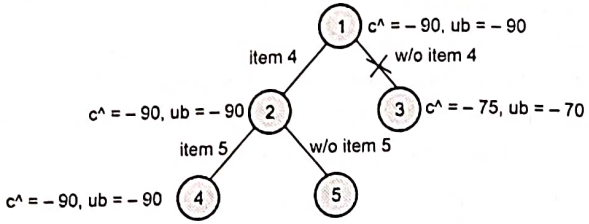


Fig. 7.7.4

We will choose node 2 because it is having less upper bound than node 3. The node 4 and node 5 has the same and ub as that of node 2 because it considers selection of item 4 and item 5.

Now consider node 5 i.e. select item 4 and item 2 ub = – (40 + 20) = – 60. The next remaining weight of item 1 is 20 which can be added as a whole.

$\therefore$ c^ = – (ub + weight of item 1) = – (60 + 10)

$\therefore$ c^ = – 70

$\therefore$ At node 5 c^ = – 70, ub = – 60

- But as node 4 is having lesser upper bound than node 5. Select node 4.

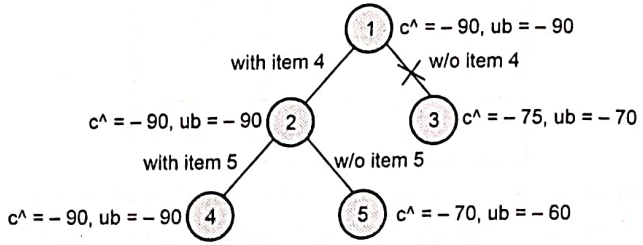


Fig. 7.7.4 (a)

- Now can not further select item 2, item 1 or item 3 because by selecting item 4 and item 5 the knapsack has reached to its capacity.
- Hence solution = {item 4, item 5} with maximum **profit = 90**.

Solution using Alternate Method : Let us solve the Knapsack problem using alternate method of branch and bound.

Ex. 7.7.4 *Consider 0/1 Knapsack instance n = 4 with capacity 10 kg such that*

Item	Profit (in ₹)	Weight (in kg)
1	40	4
2	42	7
3	20	5
4	12	3

Find maximum profit using first in first out branch and bound (FIFOBB) method. Use fixed size formation for state space tree.

SPPU : May-15, End Sem. Marks 8

Sol. : To fill the given Knapsack we select the object with some weight and having some profit. This selected object is put in the Knapsack. Thus the Knapsack is filled up with selected objects. Note that the Knapsack's capacity W should not be exceeded. Hence the first item gives best pay off per weight unit. And last one gives the worst pay off per weight unit.

$$v_1/w_1 \geq v_2/w_2 \geq v_3/w_3 \ldots\ldots v_n/w_n$$

- First of all we compute upperbound of the tree.
- We design a state space tree by inclusion or exclusion of some items.

The upper bound can be computed using following formula.

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

Consider 4 items as -

Item	Weight	Value	Value/Weight
1	4	$ 40	10
2	7	$ 42	6
3	5	$ 20	4
4	3	$ 12	4

W = Capacity of Knapsack = W = 10

We will first compute the upper bond by using above given formula -

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

Initially v = 0, w = 0 and $v_{i+1} = v_1 = 40$ and $w_{i+1} = w_1 = 4$. The capacity W = 10.

$\therefore$ $ub = 0 + (10 - 0)\ (40/4) = (10)\ (10)$

$ub = 100\ \$$

Now we will construct a state space tree by selecting different items.

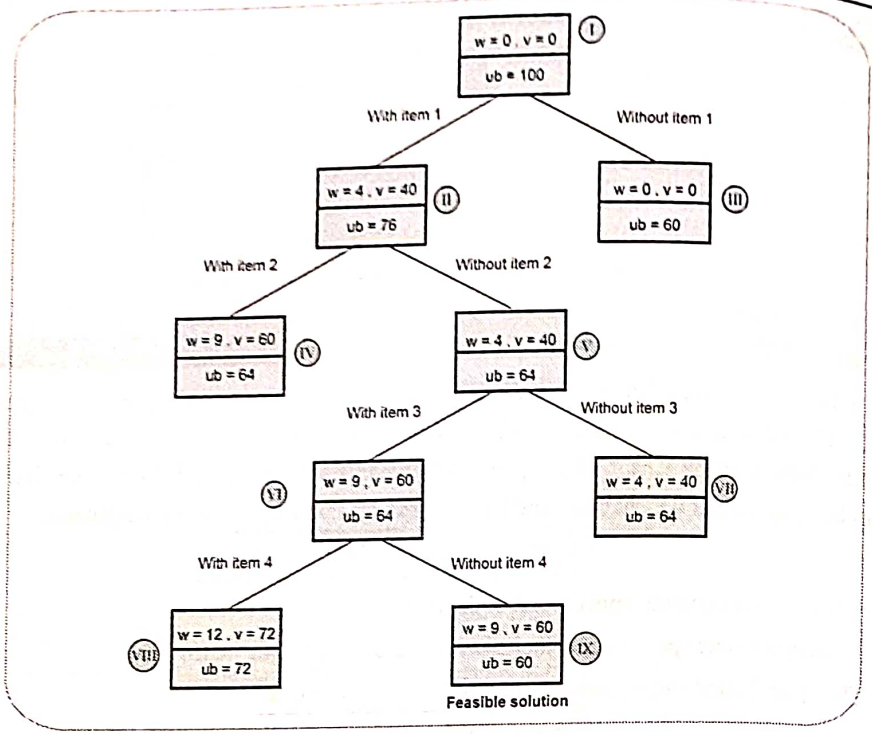


Fig. 7.7.5 State space tree for knapsack problem

Computation at node I

i.e. root of state space tree.

Initially, $w = 0$, $v = 0$ and $v_{i+1}/w_{i+1} = v_1/w_1 = 40/4 = 10$.

The capacity $W = 10$.

$\therefore \quad ub = v + (W - w)\, v_{i+1}/w_{i+1}$

$= 0 + (10 - 0)\,(10)$

$\therefore \quad ub = 100$

Computation at node II

At node II in state space tree we assume the selection of item 1.

$\therefore \quad v = 40$

$w = 4$

The capacity $W = 10$

Now $v_{i+1} / w_{i+1} \rightarrow$ means next item to item 1

i.e. $v_2 / w_2 = 6$

$\therefore \quad ub = v + (W - w)\, v_{i+1}/w_{i+1}$

$= 40 + (10 - 4) * 6$

$= 40 + 6 * 6$

$ub = 76$

Computation at node III

At node III in state space tree we assume that item 1 is not selected.

$\therefore \quad v = 0$

$w = 0$

The capacity $W = 10$

Next to item 1 is item 2

$\therefore \quad v_{i+1}/w_{i+1} \rightarrow$ means 2

i.e. $v_2/w_2 = 6$

$\therefore \quad ub = v + (W - w)\, v_{i+1}/w_{i+1}$

$= 0 + (10 - 0)\,(6)$

$ub = 60$

Computation at node IV

This is a node at which we have selected item 1 and item 2 already

$\therefore \quad v_{i+1}/w_{i+1} = v_3/w_3 = 20/5 = 4$

$\therefore \quad v = 40 + 20 = 60$

$w = 4 + 5 = 9$

The capacity $W = 10$

$\therefore \quad v_{i+1}/w_{i+1} = v_3/w_3 \rightarrow 4$

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

$= 60 + (10 - 9) * 4$

$= 60 + 4$

$= 64$

Computation at node V

At this node item 2 is not selected and only item 1 is selected.

$\therefore \quad v = 40$

$w = 4$

$W = 10$

Next item will be item 3

$\therefore \quad v_{i+1} / w_{i+1} = v_3 / w_3 = 4$

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

$= 40 + (10 - 4) * 4$

$= 40 + 6 * 4$

$ub = 64$

Computation at node VI

Node VI is an instance at which only item 1 and item 3 are selected. And item 2 is not selected.

$v = 40 + 20 = 60$

$w = 4 + 5 = 9$

The capacity W = 10.

The next item would be $v_{i+1} / w_{i+1} \rightarrow$ item 4

$\therefore \quad v_4 / w_4 = 4$

$\therefore \quad ub = v + (W - w) v_4 / w_4$

$= 60 + (10 - 9) * 4$

$= 60 + 4$

$ub = 64$

Computation at node VII

Node VII is an instant at which item 1 is selected, item 2 and item 3 are not selected.

$\therefore \quad v = 40$

$w = 4$

$W = 10$

The next item being selected is item 4.

$v_{i+1} / w_{i+1} = v_4 / w_4 = 4$

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

$= 40 + (10 - 4) * 4$

$= 40 + 24$

$ub = 64$

Computation at node VIII

At node VIII, we consider selection of item 1, item 3, item 4. There is no next item given problem statement.

$\therefore \quad v_{i+1} / w_{i+1} = 0$

$\therefore \quad w = 4 + 5 + 3 = 12$

→ But this is exceeding capacity W = 10.

$v = 40 + 20 + 12 = 72$

$W = 10$

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

$ub = 72 + (10 - 12) * 0$

But as weight of selected items exceed the capacity W this is **not a feasible solution.**

Computation at node IX

At node IX, we consider selection of item 1 and item 3. There is no next item given.

$\therefore \quad v_{i+1} / w_{i+1} = 0$

$\therefore \quad w = 4 + 5 = 9$

$v = 40 + 20 = 60$

$W = 10$

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

$= 60 + (10 - 9) * 0$

$ub = 60$

The node IX is a node indicating maximum profit of selected items with maximum weight of item = 9 i.e. < capacity of Knapsack (W = 10).

Thus for the given instance of Knapsack's problem we get **{item 1, item 4}** with maximum weight 9 and maximum profit gained = 60 $ as a solution.

Ex. 7.7.5 *Solve the following instance of the Knapsack problem by branch and bound algorithm.*

Item	*Weight*	*Values*
1	10	$ 100
2	7	$ 63
3	8	$ 56
4	4	$ 12
W = 16		

Ans : We can solve this problem using the branch and bound method by calculating bounding function by using the following formula :

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

We will draw the state space tree, for each node we will compute ub i.e. upper bound value and then expand the tree accordingly.

We will first arrange the items $v_1/w_1 \geq v_2/w_2 \geq \ldots v_i/w_i$

Item	Weight	Values	v_i/w_i
1	10	$ 100	10
2	7	$ 63	9
3	8	$ 56	7
4	4	$ 12	3

Computation at Node I

No item is selected initially.

Hence w = 0, v = 0, W = 16, $v_{i+1}/w_{i+1} = v_1/w_1 = 10$

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

$$ub = 0 + (16 - 0) * 10$$

$$ub = 160$$

Computation at Node II

Select item 1.

Hence w = 10, v = $ 100, $v_{i+1}/w_{i+1} = v_2/w_2 = 9$

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

$$= 100 + (16 - 10) * 9$$

$$= 100 + 54$$

$$ub = 154$$

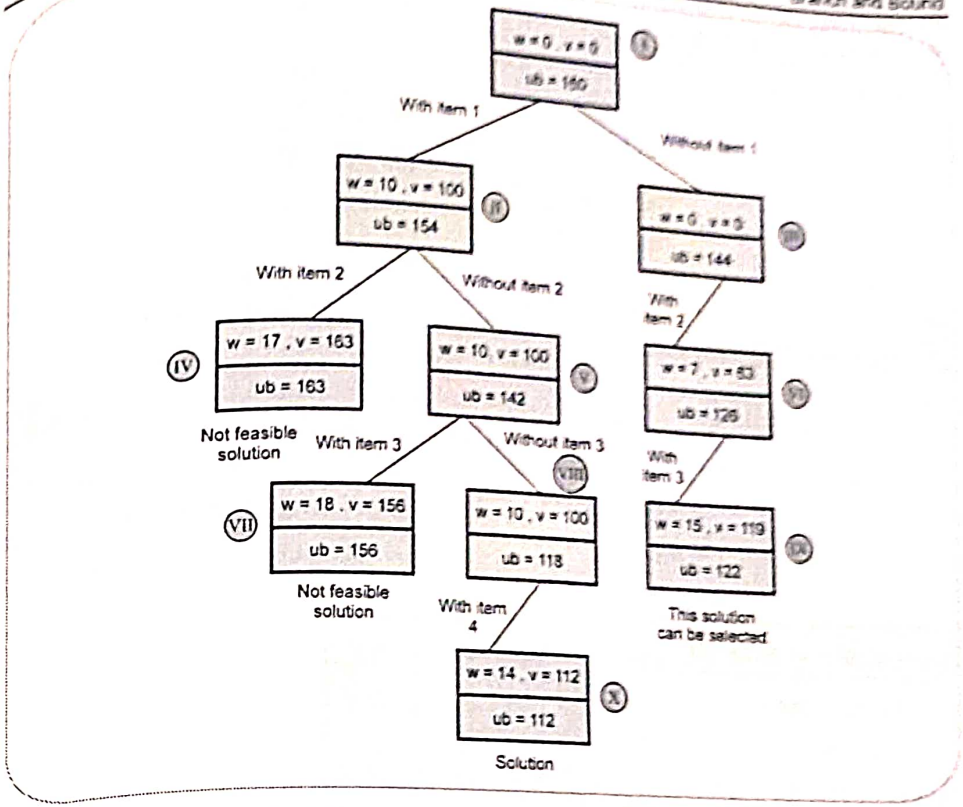


Fig. 7.7.6 State space tree for Knapsack problem

Computation at Node III

Select without item 1.

Hence w = 0, v = 0, $v_{i+1}/w_{i+1} = v_2/w_2 = 9$

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

$$= 0 + (16 - 0) * 9$$

$$ub = 144$$

Computation at Node IV

Select item 1 and item 2

$\therefore$ w = 10 + 7 = 17, v = 100 + 63 = $ 163.

But this exceeds the capacity W.

Computation at Node V

Select item 1, without item 2

$\therefore$ w = 10, v = 100, $v_{i+1}/w_{i+1} = v_3/w_3 = 7$

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

$= 100 + (16 - 10) * 7$

$= 100 + 42$

$ub = 142$

Computation at Node VI

Select without item 1, but select item 2

$\therefore$ w = 7, v = 63, $v_{i+1}/w_{i+1} = v_3/w_3 = 7$

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

$= 63 + (16 - 7) * 7$

$= 63 + 63$

$= 126$

Computation at Node VII

Select item 1, item 3

$\therefore$ w = 10 + 8 = 18, v = 100 + 56 = $ 156

As this exceeds the Knapsack's capacity. It is not feasible solution.

Computation at Node VIII

Select item 1, without item 2 and without item 3.

w = 10, v = 100, $v_{i+1}/w_{i+1} = 3$

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

$= 100 + (16 - 10) * 3$

$= 100 + 18$

$= 118$

Computation at Node IX

Select item 1 and item 4

$\therefore$ w = 10 + 4 = 14, v = 100 + 12 = 112, $v_{i+1}/w_{i+1} = 0$ as there is no next item for selection.

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

$= 112 + (16 - 14) * 0$

$= 112$

Computation at Node X

Select item 2 and item 3

$\therefore$ w = 7 + 8 = 15, v = 63 + 56 = 119,

$v_{i+1} / w_{i+1} = v_4 / w_4 = 3$

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

$= 119 + (16 - 15) * 3$

$= 119 + 3$

$ub = 122$

As we get maximum profit with selection of item 2 and 3. Also the ub is also greater. Hence the solution to this Knapsack problem is {item 2, item 3}.

Ex. 7.7.6 *Explain how branch and bound technique is used to solve 0/1 knapsack problem. Solve it for the example n = 4, m = 15, {p1 ... p4) = {45, 30, 45, 10), {w1 ...w4) = {3, 5, 9, 5}.*

Sol. : We will arrange the given values as -

p1/w1 ≥ p2/w2 ≥ p3/w3...

Let,

Item	pi	wi	pi/wi
1	45	3	15
2	30	5	6
3	45	9	5
4	10	5	2

We will compute the upper bound of the tree using the following formula -

$ub = p + (n - w) (pi + 1 / w i + 1)$

Computation at node I

$p = 0, w = 0$

$ub = 0 + (15 - 0) (45/3) = (15) (15)$

$= 225$

We will construct a state space tree by selecting different items. Refer Fig. 7.7.7

Computation at node II

Select item 1

$\therefore$ $p = 45$ and w = 3, the capacity m =15

$\therefore$ $ub = p + (m - w) (pi + 1 / wi + 1)$

$\therefore$ ub = 45 + (15 – 3) (30/5) = 45 + (12) (6) = 45 + 72

ub = 117

Computation at node III

At node III in state space tree we assume that item 1 is not selected.

Item 1 is not selected.

p = 0, w = 0, m = 15.

Item 2 is next to item 1 $\therefore$ pi + 1/wi + 1 = p2/w2

ub = p + (m – w) (pi + 1 /w i + 1) = 0 + (15 – 0) (30 / 5)

ub = 90

Computation at node IV

At node IV we assume that we have selected item 1 and item 2 already.

$\therefore$ p i + 1/w i + 1 = p3/w3 = 5

p = 45 + 30 = 70

w = 3 + 5 = 8 and m = 15

$\therefore$ ub = p + (m – w) (p i + 1/w i + 1) = 70 + (15 – 8) (5) = 70 + (7) (5)

ub = 105

Computation at node V

At node V, item 2 is not selected. Only item 1 is selected. Next item that can be selected is item 3. Hence i + 1 = 3.

$\therefore$ p = 45, w = 3, m = 15

p i + 1/w i + 1 = p3/w3 = 5

$\therefore$ ub = 45 + (15 – 3) = 45 + 12

ub = 57

Computation at node VI

Node VI is an instance at which only item 1 and item 3 are selected and item 2 is not selected.

$\therefore$ p = 45 + 45 = 90

w = 3 + 9 = 12

The capacity m = 15

The next item p i + 1/w i + 1 = p4/w4 = 2

ub = p + (m – w) (pi + 1 / w i + 1)

= 90 + (15 – 12) (2)

The state space tree can be –

= 90 + 6

ub = 96

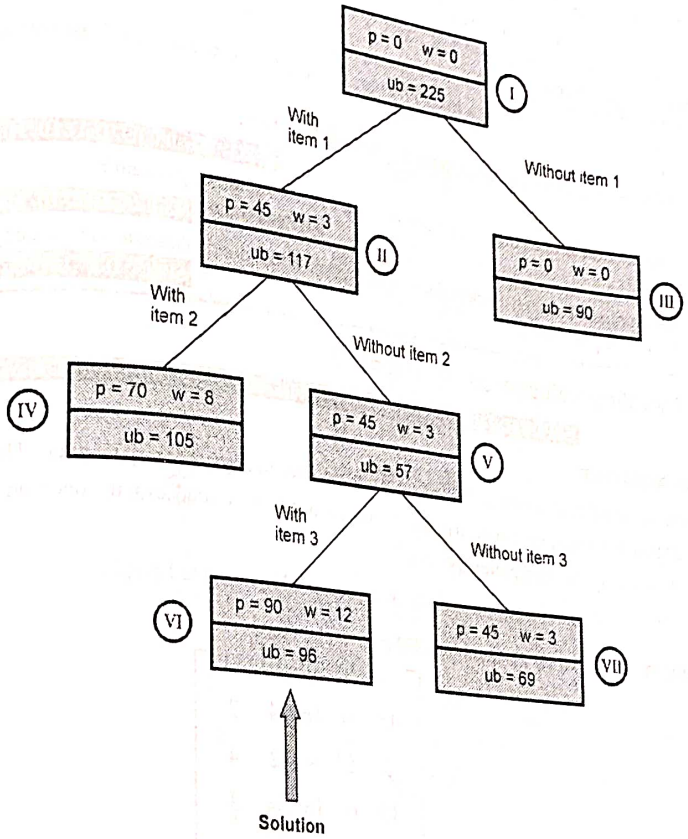


Fig. 7.7.7

Computation at node VII

Node VII is an instance at which item 1 is selected. Item 2 and 3 are not selected.

p = 45, w = 3, capacity m = 15.

The next item being selected is item 4.

pi + 1/wi + 1 = p4/w4 = 2

ub = p + (m – w) (pi + 1 / w i + 1) = 45 + (15 – 3) (2) = 45 + 24

ub = 69

Item 1 and **Item 3** are selected because total weight w < m i.e. 12 < 15 and maximum profit i.e. 90 is obtained with this selection.

Review Questions

Q.1 Present a program schema for a FIFO branch and bound search for a least cost answer node.

Q.2 Write an upper bound function for 0/1 Knapsack problem. **SPPU : May-08, 09, Dec.-10, Marks 6**

Q.3 Explain how branch and bound can be used to solve Knapsack problem ? **SPPU : May-10, Marks 8; Dec.-10, Marks 6**

Q.4 Differentiate between backtracking and branch and bound. Illustrate with example of knapsack problem. **SPPU : Dec.-12, Marks 16**

7.8 Traveling Salesperson Problem

SPPU : May-08,12,14,15, 17, 18, 19, Dec.-08,11,12,13,15,16, 18, Marks 18

Problem Statement

If there are n cities and cost of travelling from any city to any other city is given. Then we have to obtain the cheapest round-trip such that each city is visited exactly once and then returning to starting city, completes the tour.

Typically travelling salesperson problem is represented by weighted graph.

For example : Consider an instance for TSP is given by G as,

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

There n = 5 nodes. Hence we can draw a state space tree with 5 nodes as shown in Fig. 7.8.1. (Refer Fig. 7.8.1 on next page)

Tour(x) is the path that begins at root and reaching to node x in a state space tree and returns to root. In branch and bound strategy cost of each node x is computed. The travelling salesperson problem is solved by choosing the node with optimum cost. Hence,

c(x) = cost of tour (x) where x is a leaf node.

The c^(x) is the approximation cost along the path from the root to x.

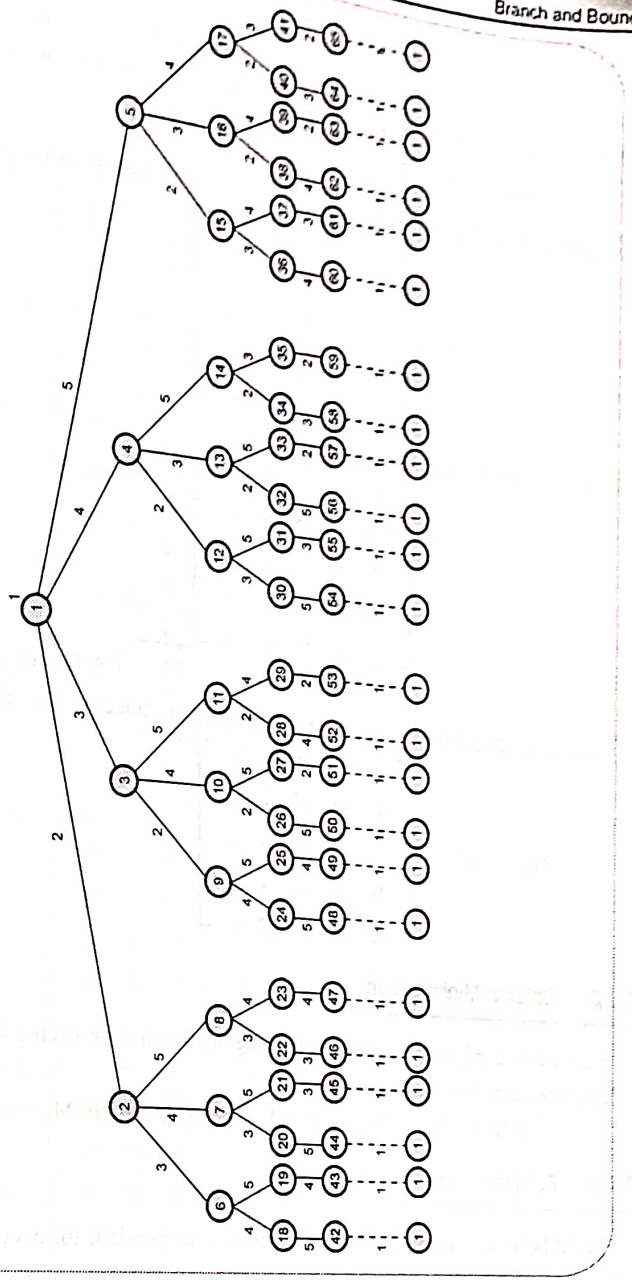


Fig. 7.8.1 State space tree for TSP

7.8.1 Row Minimization

To understand solving of travelling salesperson problem using branch and bound approach we will reduce the cost of the cost matrix M, by using following formula.

$$\text{Red\_Row}(M) = \left[M_{ij} - \min\left\{(M_{ij} \mid 1 \le j \le n)\right\}\right] \quad \text{where } M_{ij} < \infty$$

For example : Consider the matrix M representing cost between any two cities.

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

We will find minimum of each row.

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

21 Total reduced cost.

We will subtract the row_minimum value from corresponding row. Hence,

$$\text{Red\_Row(M)} = \begin{bmatrix} \infty & 10 & 20 & 0 & 1 \\ 13 & \infty & 14 & 2 & 0 \\ 1 & 3 & \infty & 0 & 2 \\ 16 & 3 & 15 & \infty & 0 \\ 12 & 0 & 3 & 12 & \infty \end{bmatrix}$$

7.8.2 Column Minimization

Now we will reduce the matrix by choosing minimum from each column. The formula for column reduction of matrix is

$$\text{Red\_Col}(M) = M_{ji} - \min\{M_{ji} \mid 1 \le j \le n\} \quad \text{where } M_{ji} < \infty$$

7.8.3 Full Reduction

Let M be the cost matrix for travelling salesperson problem for n vertices then M is called reduced if each row and each column consists of either entirely entries or else contain at least

one zero. The full reduction can be achieved by applying both row_reduction and column reduction.

For example : Consider, the matrix M as given below and reduce it fully.

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

Red_Row (M) can be obtained as,

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

21

$$\text{Red\_Row}(M) = \begin{bmatrix} \infty & 10 & 20 & 0 & 1 \\ 13 & \infty & 14 & 2 & 0 \\ 1 & 3 & \infty & 0 & 2 \\ 16 & 3 & 15 & \infty & 0 \\ 12 & 0 & 3 & 12 & \infty \end{bmatrix}$$

Red_Col(M) be obtained as,

$$\begin{bmatrix} \infty & 10 & 20 & 0 & 1 \\ 13 & \infty & 14 & 2 & 0 \\ 1 & 3 & \infty & 0 & 2 \\ 16 & 3 & 15 & \infty & 0 \\ 12 & 0 & 3 & 12 & \infty \end{bmatrix}$$

If row or column contains at least one zero ignore corresponding row or column.

$$\text{Red\_Col}(M) = \begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$

This total reduced cost will be

$$= \text{cost (Red\_Row (M))} + \text{cost (Red\_Col(M))}$$
$$= 21 + 4$$
$$= 25$$

$$\text{Thus} \begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 1 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix} \xrightarrow[\text{reduced matrix}]{\text{fully}} \begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$

7.8.4 Dynamic Reduction

We obtained the total reduced cost as 25. That means all tours in the original graph have a length at least 25.

Using dynamic reduction we can make the choice of edge i → j with optimum cost.

Steps in dynamic reduction technique

1. Draw a state space tree with optimum cost at root node.
2. Obtain the cost of matrix for path i → j by making i^{th} row and j^{th} column entries as ∞. Also set M[i] [j] = ∞.
3. Cost of corresponding node x with path i, j is optimum cost + reduced cost + M[i][j].
4. Set node with minimum cost as E-node and generate its children. Repeat step 1 to 4 for completing tour with optimum cost.

For example :

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

SPPU : May-15,17, 19, End Sem, Dec.-16, 18, End Sem, Marks 18

Fully reduced matrix is,

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

Optimum cost = 21 + 4 = 25.

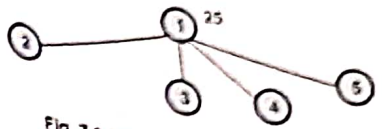


Fig. 7.8.2 Portion of state space tree

Consider path 1, 2. Make 1st row and 2nd column ∞. And set M [2] [1] = ∞.

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & 2 & 0 \\ 0 & \infty & \infty & 0 & 2 \\ 15 & \infty & 12 & \infty & 0 \\ 11 & \infty & 0 & 12 & \infty \end{bmatrix}$$

ignore, ignore, ignore, ignore, ignore

ignore ignore ignore ignore ignore

Cost of node 2 is

25 (optimum cost) + 0 (reduced cost) + 10 (old value of M[1] [2])

= 35

Consider path 1, 3. Make 1st row = ∞. 3rd column = ∞ and M[3] [1] = ∞.

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

11 ← This is only min value obtained from column

Cost of node 3 is

= 25 (optimum cost) + 11 (reduced cost) + 17 (M[1] [3])

= 53

Consider path 1, 4.

The 1st row = 4th column = M[1] [4] = ∞.

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

There is no min value from row or column.

Cost of node 4 is = 25 + 0 + 0 = 25

Consider path 1, 5.

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & 2 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 15 & 3 & 12 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{bmatrix} \begin{matrix} \\ \rightarrow 2 \\ \\ \rightarrow 3 \\ \\ \end{matrix}$$

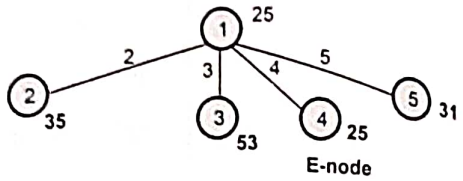


Fig. 7.8.3 Portion of state space tree

Cost of node 5 is = 25 + 5 + 1 = 31

Now, as cost of node 4 is optimum, we will set node 4 as E-node and generate its children nodes 6, 7, 8.

Consider path 1, 4, 2 for node 6. Set 1st row, 4th row to ∞. Set 2nd column to ∞. Also M[4] [1] = ∞, M[2] [1] = ∞.

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

Cost of node 6 = 25 + M [4, 2]
= 25 + 3 = 28

Consider path 1, 4, 3 for node 7. Set 1st row, 4th row to . Set 4th column, 3rd column to ∞. Set M [4] [1] = M[3] [1] = ∞.

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

↑
11

Cost of node 7 = 25 + 13 + M [4] [3]
= 25 + 13 + 12
= 50

Consider path 1, 4, 5 for node 8.

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

Cost of node 8 = 25 + 11
= 36

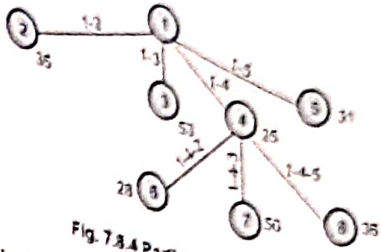


Fig. 7.8.4 Portion of state space tree

Now as cost of **node 6 is minimum**, node 6 becomes an E-node. Hence generate children for node 6. Node 9 and 10 are children nodes of node 6.

Consider **path 1, 4, 2, 3 for node 9**. Set 1st row, 4th row, 2nd row to ∞. Set 4th column, 2nd column and 3rd column to ∞.

Set M[1, 4] = M[1, 2] = M[1, 3] = ∞.

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & \infty & \infty & \infty & \infty \end{bmatrix} \rightarrow 2$$

↑
11

Cost of node 9

= 28 + 13 + 11
(28: optimum cost; 13: reduced cost; 11: M[1] [3])
= 52

Consider path **1, 4, 2, 5 for node 10.** Set 1st row, 4th row, 2nd row to . Set 4th column, 2nd column and 5th column to ∞. Set M [1, 4] = M[1, 2] = M [1, 5] = ∞.

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

There is no reduced cost

Cost of node 10 = 28 + M[2] [5]
= 28 + 0
= 28

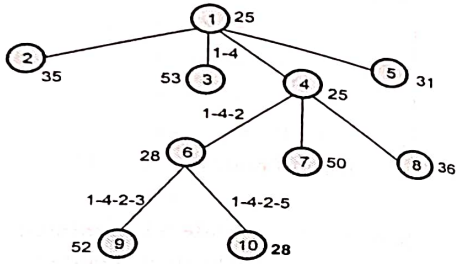


Fig. 7.8.5 Portion of state space tree

Node 10 becomes E-node now.

As for node 10 only child being generated is node 11. We set path as **1, 4, 2, 5, 3**. To complete the tour we return to 1. Hence the state space tree is,

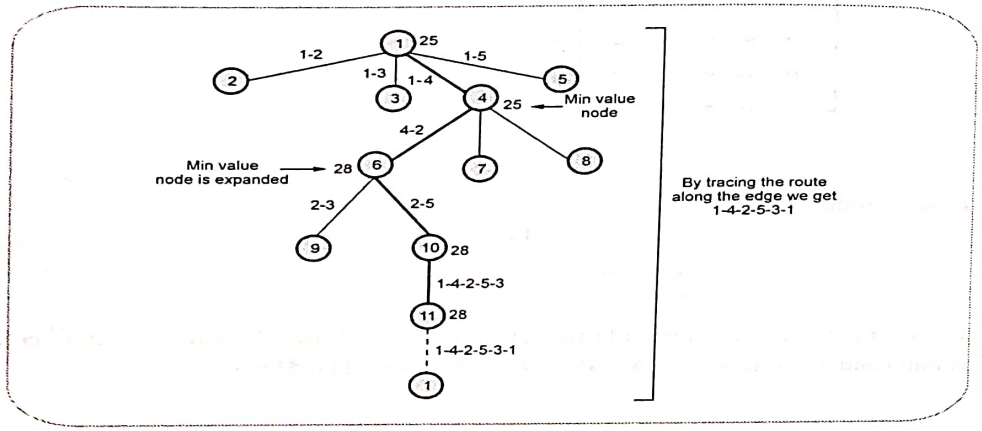


Fig. 7.8.6

Hence the optimum cost of the tour is 28.

Ex. 7.8.1 *Apply the branch and bound algorithm to solve the TSP for the following cost matrix.*

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

Sol. :

Step 1 : We will find the minimum value from each row and subtract the value from corresponding row.

∞	11	10	9	6	→ 6
8	∞	7	3	4	→ 3
8	4	∞	4	8	→ 4
11	10	5	∞	5	→ 5
6	9	5	5	∞	→ 5
					23

⇒

∞	5	4	3	0
5	∞	4	0	1
4	0	∞	0	4
6	5	0	∞	0
1	4	0	0	∞

reduced Matrix

Fig. 7.8.7

Now we will obtain minimuim value from each column. If any column contains 0 then ignore that column and a fully reduced matrix can be obtained.

∞	5	4	3	0
5	∞	4	0	1
4	0	∞	0	4
6	5	0	∞	0
1	4	0	0	∞
↑	↑	↑	↑	↑
1	Ignore	Ignore	Ignore	Ignore

Subtracting ⇒ 1 from 1st column

∞	5	4	3	0
4	∞	4	0	1
3	0	∞	0	4
5	5	0	∞	0
0	4	0	0	∞

Total reduced cost = Total reduced row cost + Total reduced column cost
= 23 + 1
= 24

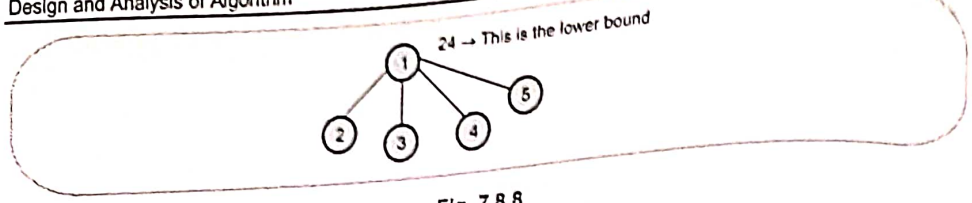


Fig. 7.8.8

Now we will set 24 as the optimum cost.

Step 2 : Now we will consider the paths [1, 2], [1, 3], [1, 4] and [1, 5] of state space tree as given above consider path [1, 2] make 1st row, 2nd column to ∞. Set M [2] [1] = ∞.

∞	∞	∞	∞	∞	→ ignore
∞	∞	4	0	1	→ ignore
3	∞	∞	0	4	→ ignore
5	∞	0	∞	0	→ ignore
0	∞	0	0		→ ignore

Now we will find min value from each corresponding column.

∞	∞	∞	∞	∞	
∞	∞	4	0	1	
3	∞	∞	0	4	There is no minimum value
5	∞	0	∞	0	from any column
0	∞	0	0	∞	
↑	↑	↑	↑	↑	
Ignore	Ignore	Ignore	Ignore	Ignore	

Hence total reduced cost for node 2 is -

= Optimum cost + old value of M [1] [2]

= 24 + 5

= **29**

Consider path (1, 3). Make 1st row, 3rd column to be ∞.

Set M [3] [1] = ∞.

∞	∞	∞	∞	∞	→ ignore
4	∞	∞	0	1	→ ignore
∞	0	∞	0	4	→ ignore
5	5	∞	∞	0	→ ignore
0	4	∞	0		→ ignore
↑	↑	↑	↑	↑	
Ignore	Ignore	Ignore	Ignore	Ignore	

There is no minimum value from any row and column

Hence total cost for node 3 is

= optimum cost + M [1] [3]

= 24 + 4

= **28**

Consider path (1, 4). Make 1st row, 4th column to be ∞. Set M [4] [1] = ∞.

∞	∞	∞	∞	∞	→ ignore
4	∞	4	∞	1	→ 1
3	0	∞	∞	4	→ ignore
∞	5	0	∞	0	→ ignore
0	4	0	∞	∞	→ ignore

Subtracting 1 from 2nd row Subtracting

∞	∞	∞	∞	∞
4	∞	3	∞	∞
3	0	∞	∞	0
∞	5	0	∞	4
0	4	0	∞	0
↑	↑	↑	↑	↑
Ignore	Ignore	Ignore	Ignore	Ignore

The total cost for node 4 is

= Optimum cost + M [1] [4] ÷ Minimum row cost

= 24 + 3 + 1

= **28**

Consider the path (1, 5). Make 1st row, 5th column to be ∞. Set M [5] [1] = ∞.

∞	∞	∞	∞	∞	
4	∞	4	0	∞	→ ignore
3	0	∞	0	∞	→ ignore
5	5	0	∞	∞	→ ignore
∞	4	0	0	∞	→ ignore
↑	↑	↑	↑	↑	
3	Ignore	Ignore	Ignore	Ignore	

Subtracting 3 from 1st column.

∞	∞	∞	∞	∞
1	∞	4	0	∞
0	0	∞	0	∞
2	5	0	∞	∞
∞	4	0	0	∞

The total cost at node 5 is

= Optimum cost + Reduced column cost + Old value M [1] [5]

= 24 + 3 + 0

= 27

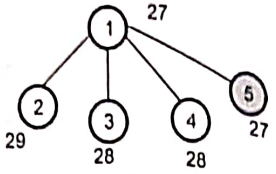


Fig. 7.8.9

The partial state space tree will be -

The node 5 shows minimum cost. Hence node 5 will be an E node. That means we will select node 5 for expansion.

Step 3 : Now we will consider the paths [1, 5, 2], [1, 5, 3] and [1, 5, 4], of above state space tree do the further computations. Consider path 1, 5, 2. Make 1st row, 5th row and second column as . Set M [5] [1] and M [2] [1] = ∞.

∞	∞	∞	∞	∞	→ ignore
∞	∞	4	0	1	→ ignore
3	∞	∞	0	4	→ ignore
5	∞	0	∞	0	→ ignore
∞	∞	∞	∞	∞	→ ignore
↑	↑	↑	↑	↑	
3	Ignore	Ignore	Ignore	Ignore	

Now subtracting 3 from 1st column, we get -

∞	∞	∞	∞	∞
∞	∞	4	0	1
0	∞	∞	0	4
2	∞	0	∞	0
∞	∞	∞	∞	∞

Hence total cost for node 6 will be -

= Optimum cost at node 5 + Column-reduced cost + M [5] [2]

= 27 + 3 + 4

= 34

Consider path [1, 5, 3]. Make 1st row, 5th row and 3rd column as ∞.

Set M [5] [1] = M [3] [1] = ∞.

∞	∞	∞	∞	∞	→ ignore
4	∞	∞	∞	∞	→ ignore
∞	0	∞	0	∞	→ ignore
5	5	∞	0	∞	→ 5
∞	4	∞	∞	∞	→ ignore

Subtracting 5 from 4th row

∞	∞	∞	∞	∞
4	∞	∞	0	∞
∞	0	∞	0	∞
1	1	∞	0	∞
∞	4	∞	∞	∞
↑	↑	↑	↑	↑
1	Ignore	Ignore	Ignore	Ignore

Subtract 1 from 1st column

∞	∞	∞	∞	∞
3	∞	∞	0	∞
∞	0	∞	0	∞
0	1	∞	∞	∞
∞	4	∞	0	∞

The total cost for node 7 will be -

= Optimum cost at node 5 + Column reduced cost + M [5] [3]

= 27 + 1 + 0

= 28.

Consider the path [1, 5, 4]. Make first row, fifth row and forth column to be ∞. Set M [5] [1] = M [4] [1] = ∞.

∞	∞	∞	∞	∞	→ ignore
4	∞	4	∞	1	→ 1
3	0	∞	∞	4	→ ignore
∞	5	0	∞	0	→ ignore
∞	∞	∞	∞	∞	→ ignore

Subtracting 1 from 2nd row

∞	∞	∞	∞	∞
3	∞	3	∞	0
3	0	∞	∞	4
∞	5	0	∞	0
∞	∞	∞	∞	∞
↑ 1	↑ Ignore	↑ Ignore	↑ Ignore	↑ Ignore

Subtracting 3 from 1st column ⇒

∞	∞	∞	∞	∞
0	∞	3	∞	0
0	0	∞	∞	4
∞	5	0	∞	0
∞	∞	∞	∞	∞

The total cost for node 8 will be

= Optimum cost at node 5 + Column-reduced + Row-reduced cost + M [5] [4]

= 27 + (1 + 3) + 0

= 31

The partial state space tree will be as shown in Fig. 7.8.10

The node 7 has optimum cost. Hence 7 becomes an E node and it will be expanded further.

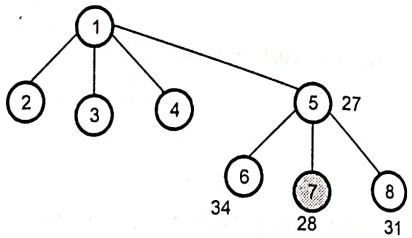


Fig. 7.8.10

Step 4 : Now we will consider paths [1, 5, 3, 2] and [1, 5, 3, 4] for further computations.

Consider path 1, 5, 3, 2 and make 1st, 5th, 3rd row as ∞ and 2nd column as ∞.

Set M [2] [1] = M [3] [1] = M[5][1] = ∞.

The matrix becomes -

∞	∞	∞	∞	∞	→ ignore
∞	∞	4	0	1	→ ignore
∞	∞	∞	∞	∞	→ ignore
5	∞	0	∞	0	→ ignore
∞	∞	∞	∞	∞	→ ignore
↑ 5	↑ Ignore	↑ Ignore	↑ Ignore	↑ Ignore	

Subtract 5 from 1st column.

∞	∞	∞	∞	∞
∞	∞	4	0	1
∞	∞	∞	∞	∞
0	∞	0	∞	0
∞	∞	∞	∞	∞

The reduced cost at node 9 [i.e. path [1, 5, 3, 2]] will be

= Optimum cost at node 7 + Column-reduced cost + M [3] [2]

= 28 + 5 + 0

= 33

Consider the path [1, 5, 3, 4]. Make 1st row, 5th row, 3rd row and 4th column as ∞.

Set M [5] [1] = M [3] [1] = M [4] [1] = ∞.

∞	∞	∞	∞	∞	
4	∞	4	∞	1	→ ignore
∞	∞	∞	∞	∞	→ ignore
∞	5	0	∞	0	→ ignore
∞	∞	∞	∞	∞	→ ignore
					→ ignore

Subtracting 1 from second row,

∞	∞	∞	∞	∞	
3	∞	3	∞	0	
∞	∞	∞	∞	∞	
∞	5	0	∞	0	Subtracting 3 from 1st column and 5 from 2nd column
∞	∞	∞	∞	∞	
↑ 3	↑ 5				

∞	∞	∞	∞	∞
0	∞	3	∞	0
∞	∞	∞	∞	∞
∞	0	0	∞	0
∞	∞	∞	∞	∞

The total reduced cost at node 10 [i.e. path 1, 5, 3, 4] is

= Optimum cost at node 7 + Row-reduced cost + M [3] [4]

= 28 + (3 + 5) + 0

= 36

The partial state space tree will be –

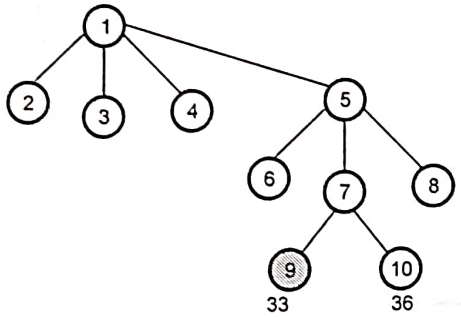


Fig. 7.8.11

Step 5 : Now consider path [1, 5, 3, 2, 4]

∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
∞	5	0	∞	0
∞	∞	∞	∞	∞

↓

5 mminvalue

Now subtract 5 from 2nd column

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

The reduced cost at node 11 will be -

= Optimum cost at node 9 + column reduced cost + M [2] [4]

= 33 + 5 + 0

= 38

The final state space tree will be –

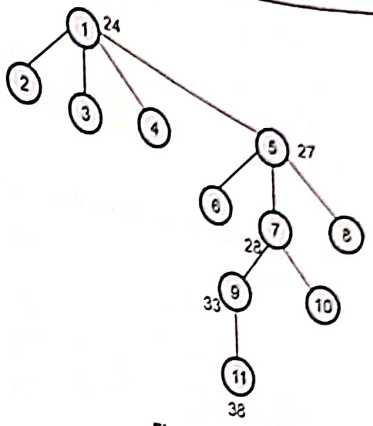
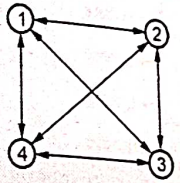


Fig. 7.8.12

The tour with minimum cost is 29. The path will be 1, 5, 3, 2, 4, 1.

Ex. 7.8.2 *What is travelling salesperson problem ? Find the solution of the following travelling salesperson problem using dynamic approach and branch and bound approach.* **SPPU : May-12, Marks 16**



$$\begin{bmatrix} 0 & 10 & 15 & 20 \\ 0 & 0 & 9 & 10 \\ 6 & 13 & 0 & 12 \\ 8 & 8 & 9 & 0 \end{bmatrix}$$

Fig. 7.8.13

Sol. :

Step 1 : We will find the minimum value from each row and subtract it from corresponding row.

∞	10	15	20	10
5	∞	9	10	5
6	13	∞	12	6
8	8	9	∞	8
				29

⇒

∞	0	5	10
0	∞	4	5
0	7	∞	6
0	0	1	∞

Reduced Matrix

Now we will obtain min value from each column. If some column contains 0 value then ignore that column. The fully reduced matrix can be obtained as –

∞	0	5	10
0	∞	4	5
0	7	∞	6
0	0	1	∞
↑ ignore	↑ ignore	1	5

⇒

∞	0	4	5
0	∞	3	0
0	7	∞	1
0	0	0	∞

Total reduced cost = Total reduced row cost + Total reduced column cost

= 29 + 6 = 35

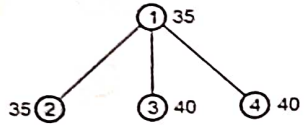


Fig. 7.8.14

The 35 is now set as optimum cost

Step 2 : Now consider path [1, 2]. Mark 1st row and 2nd column as ∞. Also set M[2][1] = ∞.

Then we will find min from each row.

∞	∞	∞	∞	→ ignore
∞	∞	3	0	→ ignore
0	∞	∞	1	→ ignore
0	∞	0	∞	→ ignore

Now we will find min from each column

Hence total reduced cost for node 2 is

= Optimum cost + Old value of M[1][2]

= 35 + 0 = 35

∞	∞	∞	∞	
∞	∞	3	0	There is no min value
0	∞	∞	1	from any column
0	∞	0	∞	
↑ ignore	↑ ignore	↑ ignore	↑ ignore	

Now consider path (1, 3). Make 1st row and 3rd column to be ∞. set M[3][1] = ∞.

∞	∞	∞	∞	→ ignore
0	∞	∞	∞	→ ignore
∞	7	∞	0	→ 1
0	0	∞	1	→ ignore
		↑ ignore	↑ ignore	

⇒

∞	∞	∞	∞
0	∞	∞	0
∞	6	∞	0
0	0	∞	∞

Total cost for node 3 is

= Optimum cost + Old value of M[1][3] + Reduced cost

= 35 + 4 + 1 = 40

Now consider path (1, 4). Marks 1st row and 4th column to be ∞. Set M[4][1] = ∞.

∞	∞	∞	∞	
0	∞	3	∞	→ ignore
0	7	∞	∞	→ ignore
∞	0	0	∞	→ ignore
↑ ignore	↑ ignore	↑ ignore	↑ ignore	→ ignore

Optimum cost for node 3 is =

Optimum cost + Old value of M[1][4]

= 35 + 5 = 40

The partial tree will be

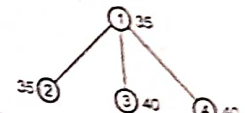


Fig. 7.8.15

Node 2 with minimum cost. Hence it will be expanded further.

Step 3 : Consider paths [1, 2, 3] , [1, 2, 4].

Consider path 1, 2, 3. Make 1st row, 2nd row and 3rd column as ∞. Mark M[2][1] and M[3][1] = ∞.

∞	∞	∞	∞	→ X
∞	∞	∞	∞	→ X
∞	7	∞	1	→ 1
0	0	∞	∞	→ X

⇒

∞	∞	∞	∞
∞	∞	∞	∞
∞	6	∞	0
0	0	∞	∞

Hence total cost for node 5 will be

= Optimum cost at node 2 + Reduced cost + M[2][3]

= 35 + 1 + 3 = 39

Consider path [1, 2, 4]. Make 1st row, 2nd row and 4th column as ∞.

M[2][1] = M[4][1] = ∞

∞	∞	∞	∞	→ X
∞	∞	∞	∞	→ X
∞	7	∞	0	→ X
0	0	∞	∞	
↑ X	↑ X	↑ X	↑ X	

No minimum value

∴ Optimum cost at node 6 will be

= Optimum cost at node 2 + Reduced cost + M[2][4]

= 35 + 0 + 0 = 35

The partial tree will be

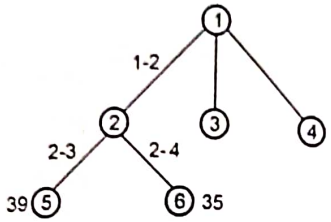


Fig. 7.8.16

Step 4 : Consider path [1, 2, 4, 3].

Mark 1st row = 2nd row = 4th row = 3rd column = ∞

M[3][1] = M[4][1] = M[2][1] = ∞

∞	∞	∞	∞	→ X
∞	∞	∞	∞	→ X
∞	7	∞	1	→ 1
∞	∞	∞	∞	→ X

⇒

∞	∞	∞	∞
∞	∞	∞	∞
∞	6	∞	0
0	0	∞	∞

Subtract 6 from 2nd column.

⇒

∞	∞	∞	∞
∞	∞	∞	∞
∞	0	∞	0
∞	∞	∞	∞

Optimum cost = Optimum cost at node 6 + M[4][3]

= 35 + 0 = 35

Final state space tree will be

The path will be 1 - 2 - 4 - 3 - 1.

The tour with minimum cost is 35.

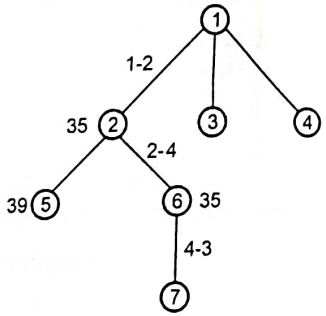


Fig. 7.8.17

Ex. 7.8.3 *Explain branch and bound strategy. Take an example of travelling salesman problem using branch and bound.* SPPU : Dec. - 13, Marks 8

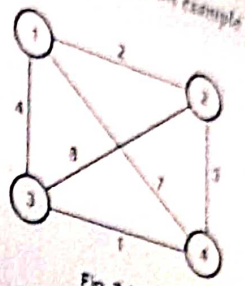


Fig. 7.8.18

Sol. : Branch and bound strategy

Refer section 7.2.1.

We will create adjacency matrix for given graph.

	1	2	3	4
1	∞	2	4	7
2	2	∞	8	3
3	4	8	∞	1
4	7	3	1	∞

Step 1 : We will find the minimum value from each row and subtract it from corresponding row.

∞	2	4	7	2
2	∞	8	3	2
4	8	∞	1	1
7	3	1	∞	1
Total reduced row cost				6

⇒

∞	0	2	5
0	∞	6	1
3	7	∞	0
6	2	0	∞

Now we will obtain minimum value from each column. If some column contains 0 value the ignore that column. The fully reduced matrix can be -

Matrix M [Used in further steps]

∞	0	2	5	← This is reduced matrix
0	∞	6	1	
3	7	∞	0	
6	2	0	∞	
↑ ignore	↑ ignore	↑ ignore	↑ ignore	

∴ Total reduced cost = Total reduced row cost + Total reduced column cost
= 6 + 0 = 6

The 6 is now set as optimum cost.

Step 2 : Consider path [1, 2]. Mark first row and 2nd column as ∞. Also set M[2, 1] = ∞. We will then find minimum from each row.

The reduced matrix will be

∞	∞	∞	∞	→ ignore
∞	∞	6	1	→ 1
3	∞	∞	0	→ ignore
6	∞	0	∞	→ ignore

∞	∞	∞	∞
∞	∞	5	0
3	∞	∞	0
6	∞	0	∞

3 ignore

The reduced matrix will be

∞	∞	∞	∞
∞	∞	5	0
0	∞	∞	0
3	∞	0	∞

∴ Reduced cost for node 2 = Optimum cost + Old value M [1, 2] + Reduced cost
= 6 + 0 + (1 + 3)
= 10

Step 3 : Consider path (1, 3). Mark first row and 3rd column as ∞. Also set M [3, 1] = ∞.

∞	∞	∞	∞	
0	∞	∞	1	→ ignore
∞	7	∞	0	
6	2	∞	∞	→ 2

⇒

∞	∞	∞	∞
0	∞	∞	1
∞	7	∞	0
4	0	∞	∞

ignore

Cost for node 3 = Optimum cost + Old value M [1, 3] + Reduced cost
= 6 + 2 + (2 + 0)
= 10

Step 4 : Consider path . Mark first row and 4th column as ∞. Set M [4, 1] = ∞

∞	∞	∞	∞	
0	∞	6	∞	→ ignore
3	7	∞	∞	→ 3
∞	2	0	∞	→ ignore

⇒

∞	∞	∞	∞
0	∞	6	∞
0	4	∞	∞
∞	2	0	∞

2

⇒

∞	∞	∞	∞
0	∞	6	∞
0	2	∞	∞
∞	0	0	∞

Reduced cost at node 4
= Optimum cost + Old value + Reduced cost = 6 + 5 + (3 + 2) = 16

The partial tree will be

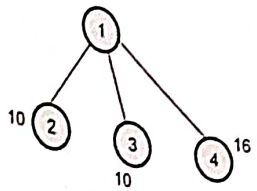


Fig. 7.8.19

We can choose either node 2 or node 3. Let us select node 3. We will expand it further.

Step 5 : Consider path [1, 3, 4] and [1, 3, 2] First consider path [1, 3, 2]. Mark 1st, 3rd row as ∞ and 2nd column as ∞. Also M [2, 1] = M [3, 1] = ∞.

∞	∞	∞	∞	
∞	∞	6	1	1
∞	∞	∞	∞	
6	∞	0	∞	

⇒

∞	∞	∞	∞
∞	∞	5	0
∞	∞	∞	∞
6	∞	0	∞

6

⇒

∞	∞	∞	∞
∞	∞	5	0
∞	∞	∞	∞
0	∞	0	∞

Reduced cost at node 5 will be optimum cost at node 3 + M [3, 2] + Reduced cost
= 10 + 7 + (1 + 6)
= 24

Consider path [1, 3, 4]. Mark row 1 = row 3 = ∞ and column 4 = ∞. M [4, 1] = M [3, 1] = ∞.

From Both row and column. we cannot obtain any minimum value. Hence reduced cost = 0.

∴ Reduced cost at node 6 will be optimum cost at node 3 + m [3, 4] + Reduced cost = 10 + 1 + 0 = 11

∞	∞	∞	∞
0	∞	6	∞
∞	∞	∞	∞
∞	2	0	∞

Partial tree will be

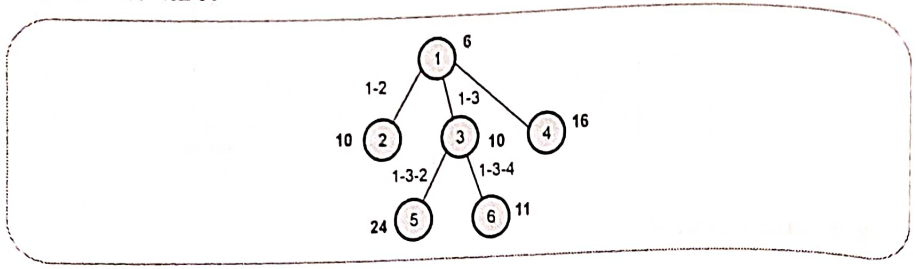


Fig. 7.8.20

Naturally node 6 will be expanded.

Step 6 : Consider path [1, 3, 4, 2]

Mark 1^{st} row = 3^{rd} row = 4^{th} row = ∞

Mark 2^{nd} column = ∞

M [2, 1] = M [4, 1] = M [3, 1] = ∞

∞	∞	∞	∞
∞	∞	6	1
∞	∞	∞	∞
∞	∞	∞	∞

1 ⇒

∞	∞	∞	∞
∞	∞	5	0
∞	∞	∞	∞
∞	∞	∞	∞

5 ⇒

∞	∞	∞	∞
∞	∞	0	0
∞	∞	∞	∞
∞	∞	∞	∞

∴ Reduced cost at node 7

= Optinum cost at node 6 + m [4, 2] + Reduced cost

= 11 + 2 + (1 + 5)

= 19

The tree will be

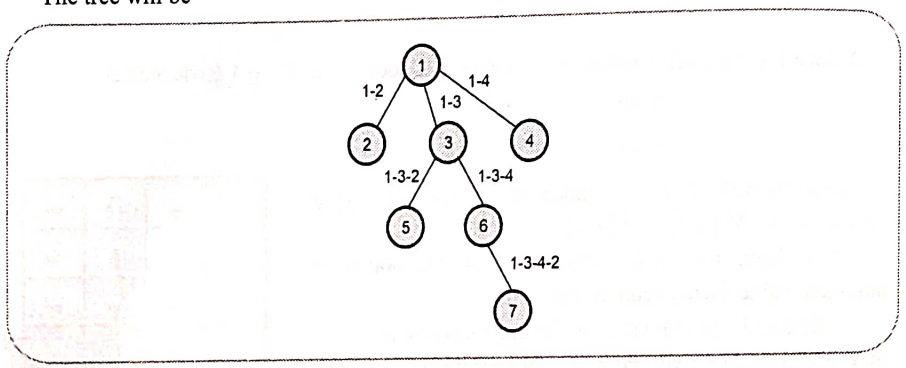


Fig. 7.8.21

Thus path with optinum tour length is 1-3-4-2-1.

The cost of tour = 10.

Ex. 7.8.4 *What is travelling salesman problem ? Find the solution of following travelling salesman problem using branch and bound method.*

SPPU : Dec.-15. (End Sem). Marks 18

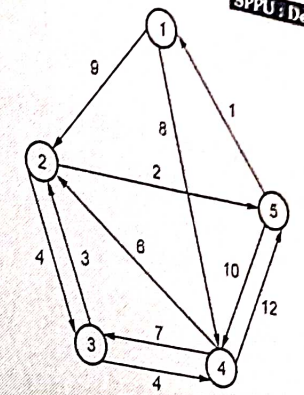


Fig. 7.8.22

Sol. : We will create adjacency matrix for given graph.

	1	2	3	4	5
1	∞	9	∞	8	∞
2	∞	∞	4	∞	2
3	∞	3	∞	4	∞
4	∞	6	7	∞	12
5	1	∞	∞	10	∞

Step 1 : We will find the minimum value from each row and this value is subtracted from corresponding row to get ROW reduced matrix.

	1	2	3	4	5	
1	∞	9	∞	8	∞	8
2	∞	∞	4	∞	2	2
3	∞	3	∞	4	∞	3
4	∞	6	7	∞	12	6
5	1	∞	∞	10	∞	1

Reduce Row →

	1	2	3	4	5
1	∞	1	∞	0	∞
2	∞	∞	2	∞	0
3	∞	0	∞	1	∞
4	∞	0	1	∞	6
5	0	∞	∞	9	∞

Now we will reduce column by each minimum value from corresponding column.

	1	2	3	4	5
1	∞	1	∞	0	∞
2	∞	∞	2	∞	0
3	∞	0	∞	1	∞
4	∞	0	1	∞	6
5	0	∞	∞	9	∞

↓ 1

Reduce → Column

	1	2	3	4	5
1	∞	1	∞	0	∞
2	∞	∞	1	∞	0
3	∞	0	∞	1	∞
4	∞	0	0	∞	6
5	0	∞	∞	9	∞

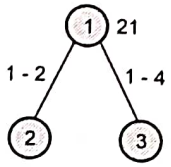
Total reduced cost = Row reduction cost + Column reduction cost

$= (8 + 2 + 3 + 6 + 1) + (1) = 21$

Cost (1) = 21

The reduced matrix is

	1	2	3	4	5
1	∞	1	∞	0	∞
2	∞	∞	1	∞	0
3	∞	0	∞	1	∞
4	∞	0	0	∞	6
5	0	∞	∞	9	∞



Step 2 : Now we will select path 1–2 : **Node 2.**

Cost of edge 1–2 is M[1, 2] = 1. Set row # 1 as ∞, set column # 2 as set M[2, 1] = ∞. The resultant cost matrix will be

	1	2	3	4	5	
1	∞	∞	∞	∞	∞	
2	∞	∞	1	∞	0	
3	∞	∞	∞	1	∞	→ 1
4	∞	∞	0	∞	6	
5	0	∞	∞	9	∞	

Reduce matrix for row reduction for row # 3

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	1	∞	0
3	∞	∞	∞	0	∞
4	∞	∞	0	∞	6
5	0	∞	∞	9	∞

There is no column reduction. Hence

Cost (2) = Cost (1) + M [1, 2] + Reduced cost

= 21 + 1 + 1

Cost (2) = 23

Step 3 : Select path 1–4 : **Node 3**

Cost of edge 1–4 is M[1, 4] = 0 set row # 1 as , set column # 4 as ∞.

Set M [4 ,1] = ∞.

The resultant cost matrix will be

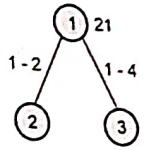
	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	1	∞	0
3	∞	0	∞	∞	∞
4	∞	0	0	∞	6
5	0	∞	∞	∞	∞

There no value for row reduction or column reduction. Hence

Cost (3) = Cost (1) + M [1, 4]

= 21 + 0

Cost = 21



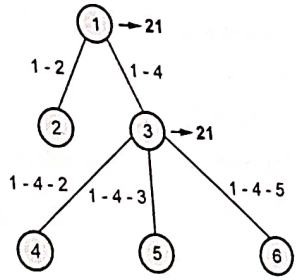
Here

We will choose node 3 for expansion as it has optimum cost.

Step 4 : The vertices possible from vertex 4 are 2, 3 and 5. Now we will start from cost matrix at node 3.

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	1	∞	0
3	∞	0	∞	∞	∞
4	∞	0	0	∞	6
5	0	∞	∞	∞	∞

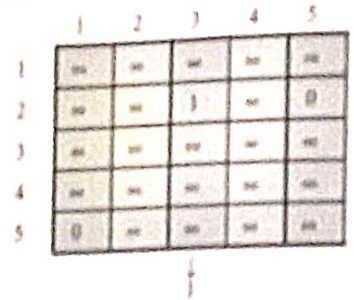
Cost (3)



Now we will choose vertex 2 : **Node 4 (1–4–2)**

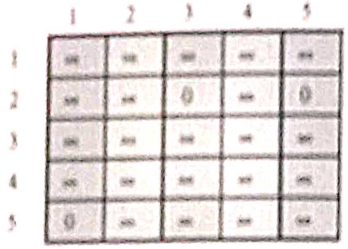
- Cost of edge <4, 2> = 0
- Set row #4 = ∞, Column # 2 = ∞
- Set M [2, 1] = ∞

The resultant matrix will be



Reduced cost = 1 (column 3)

The reduced cost matrix is



The cost at node 4 is

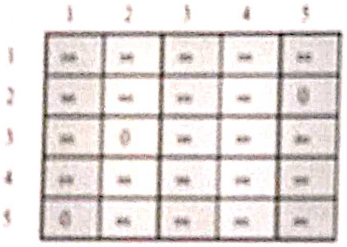
$$\text{Cost (4)} = \text{Cost (3)} + \text{Reduced cost} + M[4, 2]$$

$$= 21 + 1 + 0$$

$$\text{Cost (4)} = 22$$

Step 5 : Choose vertex 3 : **Node 5** (path 1 - 4 - 3)

- Cost of edge <4, 3> is M [4, 3] = 0
- Set row # 4 = column # 3 = ∞
- Set M [3, 1] = ∞
- The resultant cost matrix will be



- There is no reduced column or row value.

The cost at node 5 is

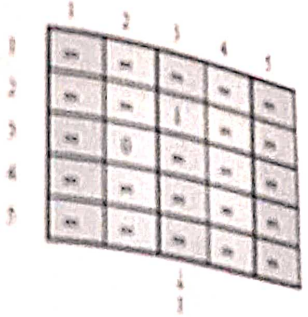
- $\text{Cost (5)} = \text{Cost (3)} + \text{Reduced cost} + M[4, 3]$

$$= 21 + 0 + 0$$

- $\text{Cost (5)} = 21$

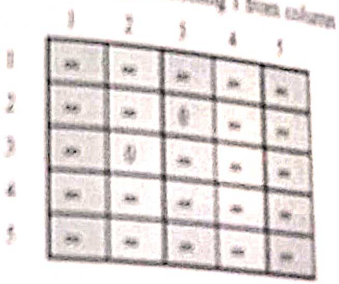
Step 6 : Choose vertex 5 : **Node 6** (Path 1-4-5)

- Cost of edge <4, 5> is M [4, 5] = 6
- Set row # 4 = column # 5 = ∞
- Set M [5, 1] = ∞
- The resultant matrix is



- Reduced cost = Column cost = 1

Hence the reduced matrix is obtained by subtracting 1 from column 1 of above matrix

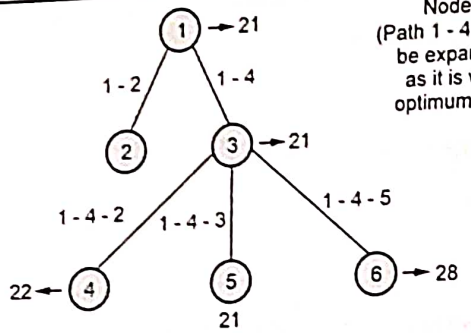


The cost at node 6 is

$$\text{Cost (6)} = \text{Cost (3)} + \text{Reduced cost} + M[4, 5]$$

$$= 21 + 1 + 6$$

$$\textbf{Cost (6)} = \textbf{28}$$



Node 5 (Path 1 - 4 - 3)will be expanded as it is with optimum cost

Step 7 : The vertices possible from vertex 3 is 2. Hence we will start from cost matrix at node 5.

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	∞	∞	0
3	∞	0	∞	∞	∞
4	∞	∞	∞	∞	∞
5	0	∞	∞	∞	∞

Now choose vertex 2 : Node 7 (path : 1–4–3–2)

- Cost of edge <3, 2>, M [3, 2] = 0
- Set row # 3 = column # 2 = ∞
- Set M [2 ,1] = ∞
- The resultant matrix will be

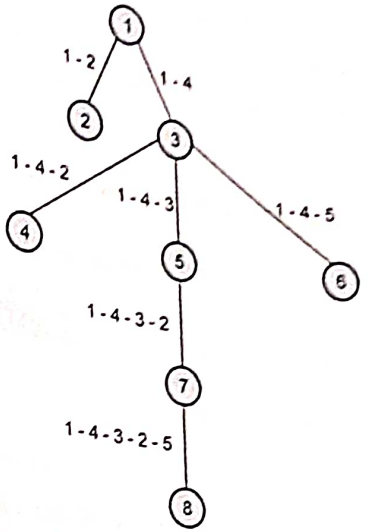
	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	∞	∞	0
3	∞	∞	∞	∞	∞
4	∞	∞	∞	∞	∞
5	0	∞	∞	∞	∞

- There is no reduced row or column value.
- The cost at node 7 will be
- **Cost (7)** = Cost (5) + Reduced cost + M [3, 2]

  = 21 + 0 + 0

  Cost (7)= 21

Step 8 : The vertex possible from 2 is 5



We will start from cost matrix at node 7

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	∞	∞	0
3	∞	∞	∞	∞	∞
4	∞	∞	∞	∞	∞
5	0	∞	∞	∞	∞

Cost (7)

The vertex 5 is chosen for node 8 (path 1 – 4 – 3 – 2 – 5)

- Cost of edge <2, 5>, M [2, 5] = 0
- Set row # 2 = column # 5 = ∞
- Set M [5, 1] = ∞
- The resultant matrix will be

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	∞	∞	∞
3	∞	∞	∞	∞	∞
4	∞	∞	∞	∞	∞
5	∞	∞	∞	∞	∞

- There is no reduced row or column value
- The cost at node 8 will be
- Cost (8) = cost (7) + Reduced cost + M [2, 5]

  = 21 + 0 +0

Cost (8) = 21

Thus now there is no further vertex to visit. Hence in order to complete the tour, Visit vertex 1 from vertex 5.

Thus we get optimum tour as 1 – 4 – 3 – 2 – 5 – 1 with the cost = (8 + 7 + 3 + 2 + 1) = **21**

Ex. 7.8.5 *What is travelling salesman problem? Find the solution of the following travelling salesman problem using branch and bound method.* **SPPU : May-18, End Sem, Marks 10**

$$\text{Cost Matrix} = \begin{bmatrix} \infty & 4 & 2 \\ 3 & \infty & 4 \\ 1 & 8 & \infty \end{bmatrix}$$

Sol. :

Step 1 : We will find minimum value from each row and subtract it from corresponding row

∞	4	2	2
3	∞	4	3
1	8	∞	1

→

∞	2	0
0	∞	1
0	7	∞

Now we will obtain min value from each column. If some column contains 0, then ignore that column.

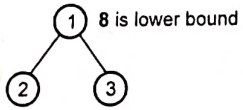
The fully reduced matrix is

∞	0	0
0	∞	1
0	5	∞

Total reduced cost = Reduced row cost + Reduced column cost

= (2 + 3 + 1) + (2)

= 8



The 8 is now set as optimum cost.

Step 2 : i) Now consider path (1, 2)

Mark 1st row and 2nd column as also set M[2, 1] = ∞

Then we will find min from each row

∞	∞	∞
∞	∞	1
0	∞	∞

→1 ⟹

∞	∞	∞
∞	∞	0
0	∞	∞

Now we will find min from each column.

There is no min value from column.

Total cost for node (1, 2) is optimum cost + Old value M[1, 2] + Reduced cost

= 8 + 0 + 1

= 9

ii) Now consider path (1, 3)

Mark 1st row ∞ and 3rd column.

Also set M[3, 1] = ∞.

Then we will find min from each row

∞	∞	∞
0	∞	∞
∞	5	∞

Min val ⟹

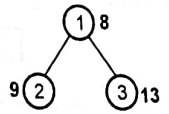
∞	∞	∞
0	∞	∞
∞	0	∞

∴ Total cost for node (1, 3) is optimum cost + Old value M[1, 3] + Reduced cost

= 8 + 0 + 5

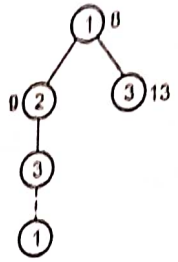
= 13

The partial space tree is



As node 2 is with minimum cost node 2 becomes **E-node.** Hence only one child is generated for this node. So the optimum tour length path is **1 – 2 – 3**

The state space tree is



The cost of tour is 9

Review Questions

Q.1 Explain how branch and bound method can be used to solve travelling salesperson problem. **SPPU : May-08, Dec.-08, Marks 6**

Q.2 Explain the branch and bound algorithmic strategy for solving the problem take an example of traveling salesman problem using branch and bound. **SPPU : Dec.-11, 12, Marks 10**

Q.3 Explain dynamic reduction with all steps with respect to Travelling Salesperson problem. **SPPU : May-14, Marks 6**

7.9 Multiple Choice Questions

Q. 1 The solution to the Knapsack problem, with W = 10 and (p1, p2, p3, p4) = (42, 49, 30,4) and (w1, w2, w3, w4) = (4, 7, 5, 1) is _____.

[a] (1, 3) [b] (1, 3, 4) [c] (2, 4) [d] (2, 3)

Q. 2 Consider the Knapsack problem, with W = 5 and (p1, p2, p3, p4) = (12, 10, 20, 15) and (w1, w2, w3, w4) = (2, 1, 3, 2) The Upper Bound at the root node of a state space tree is _____.

[a] 30 [b] 42 [c] 50 [d] 37

Q. 3 Consider the Knapsack problem, with W = 5 and (p1, p2, p3, p4)=(12, 10, 20, 15) and (w1, w2, w3, w4)=(2, 1, 3, 2)
The maximum profit that can be earned is _____ .

[a] 30 [b] 42 [c] 50 [d] 37

Q. 4 The worst case time complexity of traveling Salesperson problem using branch and bound is _____.

[a] O(n^2) [b] O(2^n)

[c] O() [d] O(nlogn)

Q. 5 The worst case time complexity of traveling salesperson problem using branch and bound is same as that using dynamic programming is _____.

[a] false [c] can be true depending upon the graph

[b] true [d] can be false depending upon the graph

Q. 6 If there are m live nodes then addition and deletion of a node can be carried out in the time of _____.

[a] O(log m) [b] O(1) [c] (1) [d] O(m)

Q. 7 If there are m live nodes then to test whether a list is empty can be carried out in _____ time.

[a] O(log m) [b] O(1) [c] (1) [d] O(m)

Q. 8 _____ function is used to kill live nodes without generating all their children.

[a] Binary [b] Non recursive

[c] Recursive [d] Bounding

Q. 9 In case of branch and bound, we terminate the search path at the current node in state space tree for following reason : _____.

[a] The value of the node's bound is better than the value of the best solution seen so far.

[b] The node represents no feasible solution as constraint of the problem is violated.

[c] The subset of feasible solutions represented by the node consists of a single point

[d] All of the above

Q. 10 Which method can be used to estimate the number of nodes that can be generated by backtracking algorithm working on a certain instance _____.

[a] Branch and Bound [b] Monte Carlo

[c] Brute Force [d] Both A and C

Answer Keys for Multiple Choice Questions :

Q.1	b	Q.2	a	Q.3	d	Q.4	c	Q.5	b
Q.6	a	Q.7	c	Q.8	d	Q.9	d	Q.10	a

□□□