Branch and Bound

Objective:

- To **Apply efficient algorithmic design paradigms**

Syllabus:

Branch and Bound : General method, Applications , Travelling sales person problem , 0/ 1 knapsack problem - LC BB, FIFO BB solutions.

Learning Outcomes:

At the end of the course, Students will be able to

- Apply Branch and Bound method to Solve the 0/ 1 knapsack problem and Traveling Sales person problem

Learning Material

General Method:

- In branch and bound method, a state space tree is built and all the children of E -nodes are generated before any other node can become a live node.
- For exploring new nodes either a BFS (B -Search) or DFS (D -Search) technique can be used.
- In branch and bound technique, BFS like state space search will be called as FIFO search. On other hand the D-Search like state space search will be called as LIFO search or LC Search.
- In this method a space tree of possible solutions is generated. Then partitioning (called as branching) is done at each node of tree. We compute lower bound and upper bound at ea ch node. This computation leads to selection of answer node.
- Bounding functions are used to avoid the generation of sub trees that do not contain an answer node.

General Algorithm for Branch and Bound:

The algorithm for branch and bound is as given bel ow

Algorithm Branch_Bound()

```
{
        / / E is a node pointer;
        E←new(node);//This is the root node
        / / H is heap for all the live nodes.
        While(true)
        {
                If(E is a final leaf) then
                {
                        / / E is an optional 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);
}
}
```

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;

}

Least Cost Search:

- 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 me thod 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 f unction 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 DFS 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 ve 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) $\leq$ g^(y) whenever y is a child of x then the search is a D -search(DFS).
- An LC search with bounding functions is known as LC Branch and Bound search.
- In LC Search, the cost function 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 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.

Control abstraction for LC Search:

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

```
Algorithm LCSearch(t)
// Search t for an answer node.
{
    if *t is an answer node then output *t and return;
    E := t; // E-node.
    Initialize the list of live nodes to be empty;
    repeat
    {
        for each child x of E do
        {
            if x is an answer node then output the path
                from x to t and return;
            Add(x); // x is a new live node.
            (x → parent) := E; // Pointer for path to root.
        }
        if there are no more live nodes then
        {
            write ("No answer node"); return;
        }
        E := Least();
    } until (false);
}
```

In above algorithm if x is in t then c(x) be the minimum cost answer node in t. The algorithm uses two functions Least_cost and ADD() 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 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.

Bounding:

- 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 a nswer node. Each time a new answer node can be obtained.

0/ 1 Knapsack Problem:

Problem Statement:

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

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

$$\text{subject to} \quad \sum_{i=1}^{n} w_i x_i \le m$$

$$x_i = 0 \text{ or } 1, \quad 1 \le i \le n$$

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 u sed 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 $c\hat{}(x) \leq c(x) \leq u(x)$ for every node.

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

```
Algorithm C_Bound(total_profit, total_wt,k)
{
        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)
{
        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;
}
```

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

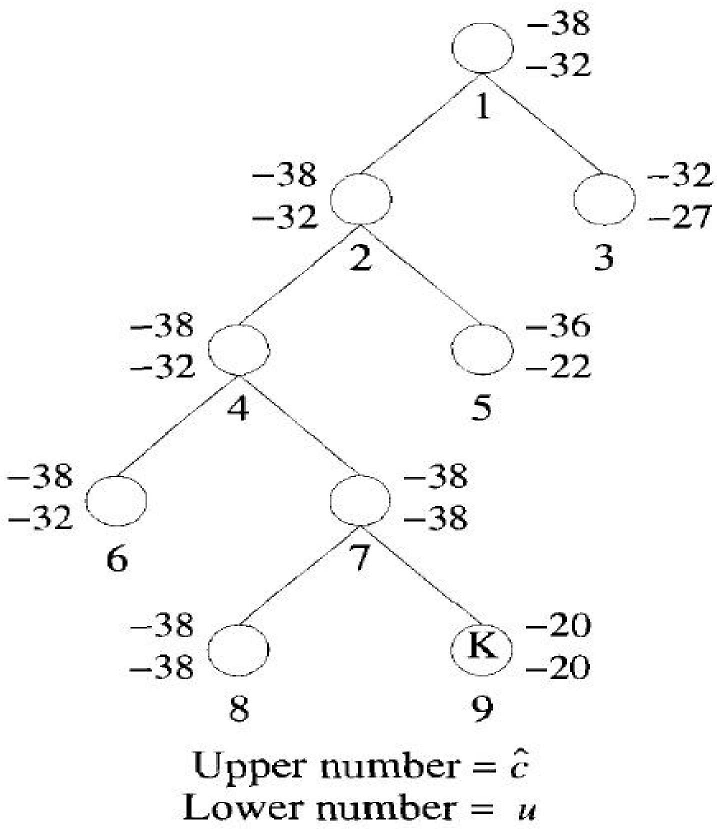
1. draw state space tree
2. compute c^(.) and u(.) for each node
3. if c^(x) > upper kill node x.
4. Otherwise the minimum cost c^(x) becomes E -node. Generate children for E-node.
5. Repeat step 3 and 4 until all the nodes get covered.
6. the minimum cost c^(x) becomes the answer node. Trace the path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

Example Problem:

Consider Knapsack instance n=4 with capacity m=15 such that, (p1, p2, p3, p4)=(10, 10, 12, 18) and (w1, w2, w3, w4)=(2, 4, 6, 9).

Solution:

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



LCBB solution state space tree

<u>Travelling Salesperson Problem(TSP):</u>

Problem Statement:

If there are n cities and cost of traveling 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 traveling salesperson problem is represented by weighted graph.

Row Reduction:

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

Redu_Row(M)= [M_{ij} – min{ M_{ij} | $1 \leq j \leq n$ }] where $M_{ij} < \infty$

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

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

We will find minimum of each row.

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

21 Total reduced cost

	∞	10	20	0	1
Redu_Row(M)=	13	∞	14	2	0
	1	3	∞	0	2
	16	3	15	∞	0
	12	0	3	12	∞

Column Reduction:

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

Redu_Col(M)= [M $_{ji}$ – min{ M $_{ji}$ | $1 \leq j \leq n$ }]   where M $_{ji} < \infty$

Redu_Col(M) can be obtained as,

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

Total = 1+3=4

Note: if row or column contains at least one zero, ignore corresponding row or column.

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

Thus total reduced cost will be= cost(redu_row(M)) + cost(redu_Col(M))

= 21 + 4

= 25

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 dynami c 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[j][1]= ∞
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.

Example problem:

The edge length of a directed graph are given by the below matrix. Using the travelling salesperson algorithm, calculate the optimal tour.

$$\begin{matrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \end{matrix}$$

M=					
	3	5	∞	2	4
	19	6	18	∞	3
	16	4	7	16	∞

Solution:

Fully reduced matrix is,

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

Optimum cost = 21 + 4 = 25

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

(a) Path 1,2; node 2

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 4 & 3 & \infty & \infty & 0 \\ 0 & 0 & \infty & 12 & \infty \end{bmatrix}$$

(b) Path 1,3; node 3

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

(c) Path 1,4; node 4

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 10 & \infty & 9 & 0 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 12 & 0 & 9 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{bmatrix}$$

(d) Path 1,5; node 5

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

(e) Path 1,4,2; node 6

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

(f) Path 1,4,3; node 7

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

(g) Path 1,4,5; node 8

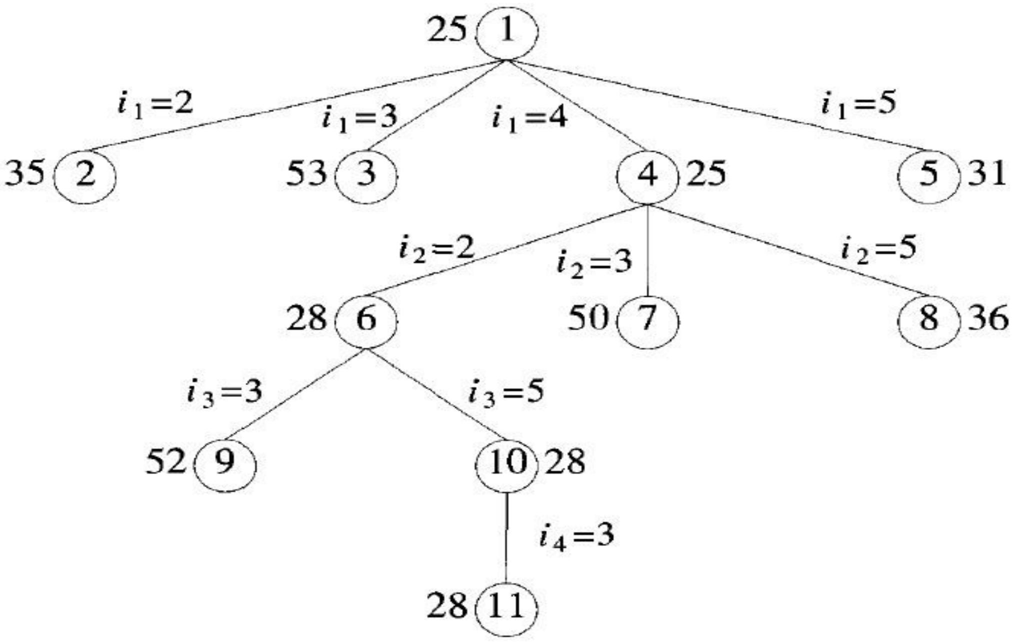
$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \end{bmatrix}$$

(h) Path 1,4,2,3; node 9

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

(i) Path 1,4,2,5; node 10

Reduced cost matrices corresponding to nodes



Numbers outside the node are $\hat{c}$ values

Figure : <u>State space tree generated by procedure LCBB</u>

UNIT -VI

Assignment -Cum -Tutorial Questions

SECTION -A

Objective Questions

1. Bounding functions are used to avoid the expansion of__________that do not contain an answer node.
2. BFS like state space search will be called ________ Branch and Bound technique.
3. **D-search like state space search will be called** ________ Branch and Bound technique.
4. Which data structure is used in BFS like state space search [ ]
   A) Array B) Stack C) Queue D) Linked list
5. Which data structure is used in D -Search [ ]
   A) Array B) Stack C) Queue D) Linked list
6. An airport limousine service w hich parks all its limos at the airport can minimize its cost by using a proper order to pick up passengers from their houses and ret urn to the airport using______ [ ]
   A) set covering problem B) traveling salesman problem
   C) knapsack problem D) fixed charge problem
7. An Avon lady carrying her tote containing makeup materials can maximize her profit from one trip to the rural Mississippi hinterland if she models the process of loading her bag (with the "right" materials having maximum profitability per un it volume) by using_____ [ ]
   A) set covering problem B) traveling salesman problem
   C) knapsack problem D) fixed charge problem
8. Consider Knapsack instance n=4 with capacity m=15. such that,

Object i:	1	2	3	4
profits :	10	10	12	18
Weights :	2	4	6	9

What is its LCBB solution vector? [ ]

A) (1,1,0,1) B) (1,0,0,1) C) (1,1,1,0) D) (0,1,0,1)

9. What is the cost of reducing ROW 1 in solving the TSP for the following cost matrix? [ ]

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

A) 20 B) 10 C) 30 D) 30

10. Determine the cost of reducing coloumn 4 in solving the TSP for the following cost matrix? [ ]

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

A) 6 B) 5 C) 4 D) 8

11. Find the total cost of reducin g the matrix in solving the TSP for the following cost matrix? [ ]

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

A) 30 B) 15 C) 25 D) 10

12. What is the optimal tour of a TSP for the following cost matrix? [ ]

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

A) 1□4□5□2□3□1 B) 1□4□2□5□3□1

C) 1□5□4□2□3□1 D) 1□3□5□2□4□1

13. Consider Knapsack instance n=4 with capacity m=15. such that,

Object i:	1	2	3	4
profits :	10	10	12	18
Weights :	2	4	6	9

What are the initial **upper** and **lower** bound values for the above instance? [ ]

A) -30, -35 B) -38, -32 C) -32, -38 D) -32, -32

SECTION -B

SUBJECTIVE QUESTIONS

1. Explain the Gen eral method of Branch and Bound and differentiate between backtracking and branch and bound
2. Write the control abstraction for least cost search.
3. Explain process of solving the 0/ 1 Knapsack problem with FIFIBB.
4. Draw the portion of state space tree generated by LCBB for the knapsack instance n=4, (p1,p2,p3,p4)=(10,10 ,12,18),(w1,w2,w3,w4)=(2,4,6,9) and m=15.
5. Draw the portion of state space tree generated by FIFOBB for the knapsack instance n=5, (p1,p2,p3,p4,p5)=(10,15,6,8,4),(w1,w2,w3,w4,w5)=(4,6,3,4,2) and m=12.
6. Solve the knapsack instance n=5, (p1,p2,p3,p4,p5)=(w1,w2, w3,w4,w5)=(4,4,5,8,9) and m=15 using LCBB.

7. Solve the knapsack instance n=5, (p1,p2,p3,p4,p5)= (w1,w2,w3,w4,w5)=(4,4,5,8,9) and m=15 using FIFOBB.
8. Consider an instance for TSP given by cost matrix Gas,

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

   a) obtain the reduced cost matrix.
   b) Draw a state space tree generated by LCBB.
   c) Find cost of the optimal TSP tour.
9. Apply the least cost branch and bound method to solve the TSP for the following cost matrix . Draw a state space tree and find the optimum cos t of the tour?

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

10. Solve TSP problem having the following cost matrix using LCBB.

∞	5	2	3
4	∞	1	5
4	2	∞	3
7	6	8	∞