UNIT 5
BACKTRACKING

BACKTRACKING (General Method)

Definition

- Depth First node generation with bounding function is called backtracking.
- Suppose mi is the size of set S_i. Then there are m=m1,m2…..m_n n-tuples that are possible candidates for satisfying the function P.
- The Backtrack algorithm has its virtue the ability to yield the answer with far fewer than m trials. It's basic idea is to build up the solution vector one component at a time and to use modified criterion functions Pi(x1,……..x_i) to test whether the vector being formed has any chance of success.
- If it is realized that partial vector (x1,……..x_i) can in no way lead to an optimal solution, then m_i+1,….m_n possible test vectors can be ignored entirely.
- Problems solved through backtracking require that all the solution satisfy a complex set of constraints. i.e, (Implicit, Explicit constraints).

Explicit Constraint:

Are Rules that each x_i to take on values only from a given set. The explicit constraints depends on the particular instance of I of the problem being solved.

E.g

1. $x_i >= 0$ or S_i = { all non negative real numbers }
2. $x_i = 0$ or 1 or S_i = { 0, 1 }

Implicit Constraint :

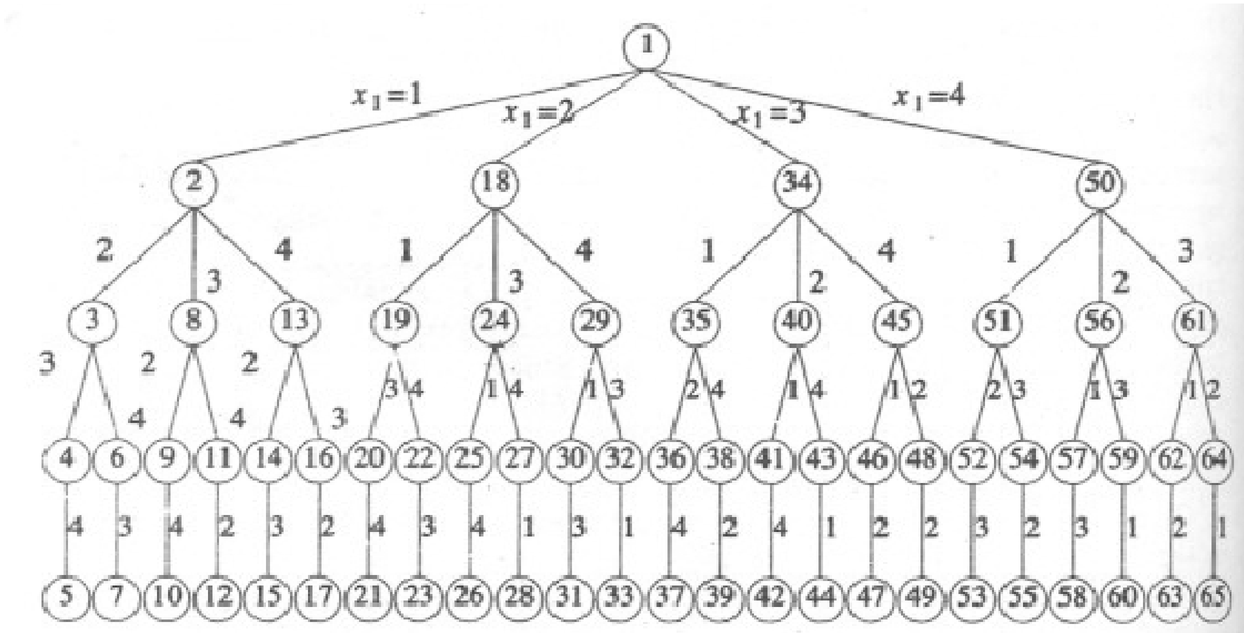
Implicit Constraint are rules that determine which of the tuple in the solution space of I satisfy the criterion function. Thus implicit constraints describe the way in which the X; must relate to each other.

Ex:

Number two queen attack each other are implicit constraints in the 8-queen problem.

Example General method [4 – queens problem]

- **Bounding Function:** If (x_1, x_2 ,…, x_i) is the path to current E-node, then all children nodes with parent – child labelings x_{i+1} are such that (x_1,x_2,..,x_{i+1}) represents a chess board configuration in which no two queens are attacking.
- Start with the root node as the only live node.This become E – node, gererate child .Thus node number 2 is generated now the path is (1). This corresponding to placing queen 1 *o*n column 1.
- Node 2 become the E – node. Node 3 generated & immediately killed . The next node generated is 8 and the Path becomes (1,3). Node 8 – become E - node



Tree Organization of the 4 – queen solution space.
[Nodes are numbered in DFS]

- 8 gets killed as all its children represent bound configuration that cannot lead to an answer node. <u>Backtrack to node 2.</u> and generate another child, node 13.
- The following fig. Shows the backtracking algorithm goes through as it tries to find a solution. The dots indicates placement of a queen, which were tried and rejected.

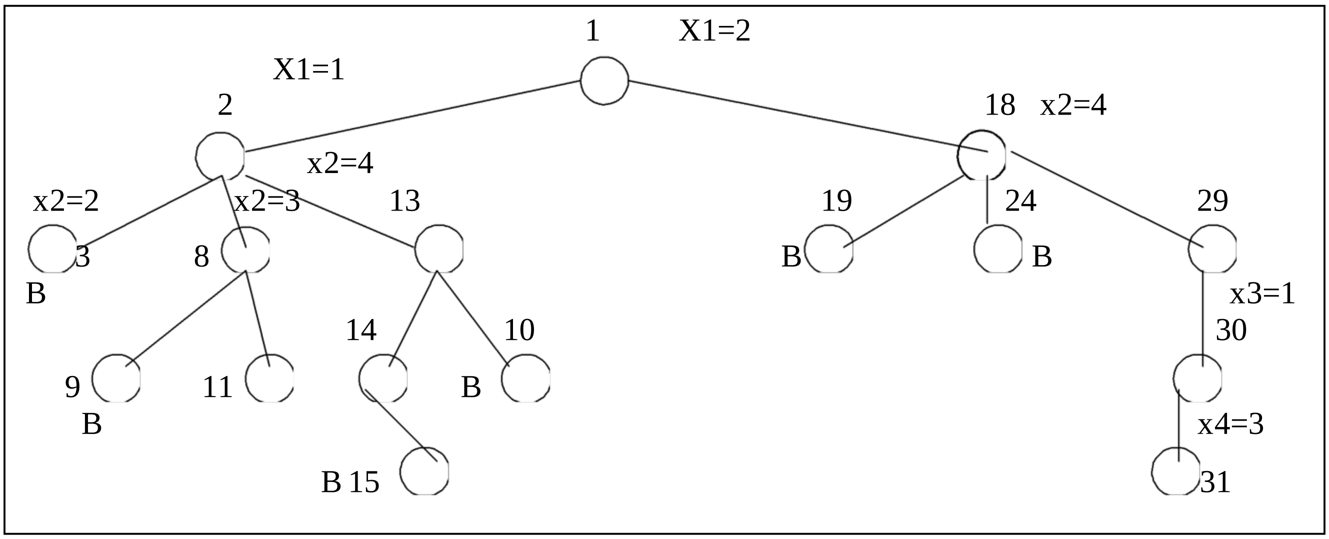
1			
			2
	3		
□	□	□	□

1			

1			
□	□	□	2

1			
			2
3			
□	□	4	

Part of the solution space tree that is generated during backtracking. Nodes are numbered in the order in which they are generated.



B- A node that gets killed as a result of the bounding function.

<u>Back Tracking Process:</u>

- All answer nodes are to be found.
- Let $(x_1,x_2,\ldots x_i)$ be a path from the root to a node in a state space tree.
- Let $T(x_1,\ldots x_i)$ be the set of all possible values for $x_{i+1.}$ S.T. $(x_1,x_2,..x_{i+1})$ is also a path to a problem state.
- $T(x_1,x_2\ldots x_n) = \varphi$ (null).
- Bounding function B_{i+1} $(x_1,x_2,..x_{i+1})$ is false, if path cannot be extended to reach ans node from $x_1,x_2..x_{i+1}$ th place.
- Backtrack start with <u>Backtrack (1).</u>

```
Algorithm Backtrack (k)
{
    for (each x[k] ∈ T(x[1],…,x[k-1]) do
    {
        if(Bk(x[1],x[2],…x[k])≠ 0) then
        {
            if(x[1],x[2],…,x[k]. is a path to an  ans.node)
                then write(x[1:k])
            if(k<n) then Backtrack(k+1);
        }
    }
}
```

Estimating Number of nodes generated by backtracking algorithm
(Using monte carlo)

```
algorithm estimate()
{
k:=1;
m:=1;
r:=1;
repeat
{
Tk:={x[k]|x[k]  E T(a[1],x[2],……x[k-1])
And Bk (x[1],…..x[k] )is true};
If(size (Tk)=0) then
Return m;
R:=r* Size(Tk);
M:= m +r;
X[k] : = choose(Tk);
K: = k +1;
}until(false);
}
```

//Estimating the effiency of backtracking.

- The function <u>size</u> return the size of the set Tk.
- The function <u>Choose</u> makes a random choice of an element in Tk.
- The desired sum is built using the variable m & r.
- A better estimate of the number of unbounded nodes that will be generated by a backtracking algorithm can be obtained by selecting several different random paths and determining the average of these values.

Queen Problem

given a problem to place eight queens on an 8* 8 chess board so that no two "attack" that is , so that no two of then are on the same row, column, or diagonal

- if the imagine the chess board squares indices of the two – dimensional array a[1:n,1:n], then we observe that every element on the same diagonal that rows from the upper left to the right has same row – column value.
- Every element on the same diagonal that goes from the upper right to the lower left has same row+ column value.
- Suppose two queens are placed at position (i,j) and (k,l) then they are on the same diagonal iff
  $I - j = k - l$ *or* $I + j = k + l$
- the first eq *impiles* $j-l = I - k$
- the second eq implies $j-l = k - I$

therefore two queens lies on the same diagonal

iff $|j-l| = |I-k|$ otherwise ($|a|$=abs(a))

Algorithm

```
Algorithm place(k,i)
//return true if a queen can be placed in kth row  I th column. Else it return false. X[] is a
global array. Abs ( ) return absolute value of r//
{
for  I =1 to k –1 do
if ((x[j] =I) or (abs(x[j] –I )= abs (j-k))
then return false;
return true;
}

Algorithm  nqueen(k,n)
//this procedure prints all possible placement of n queue on an n*n chessboard so that
//they are non-attaching
    {
     for  I=1 to n do
       {
         if place(k,I)then
          {
            x[k]=I;
            if (k=n) then write(x[1:n]);
             else nqueen(k+1,n);
           }
        }
      }
```

Analysis/efficiency of 8-queens

- using the e estimate function five 8*8 chessboards were created.
- The placement of each queen on the chessboard was chosen randomly.
- Track of no.of..columns or queen could legitimately be placed is given as a vector beneath called chessboard.
- The average of five trial is 1625.
- Total no.of nodes in 8-queen state space tree is

$$1+\sum_{j=0}^{7}\left[\prod_{I=0}^{j}(8-I)\right] = 69{,}281$$

so the estimated number of unbounded nodes is only about 2.34% of the total no.of nodes in the 8-queen state space tree

- following are the 8*8 chessboards that were created using estimate.

	1						
			2				
3							
		4					
				5			

(8,5,4,3,2)=16 49

			1				
					2		
		3					
				4			
						5	
6							

(8,5,3,1,2,1)= 769

1							
							2
					3		
		4					
						5	
	6						
			7				

(8,6,4,2,1,1,1)=1401

1							
		2					
				3			
	4						
			5				

(8,6,4,3,2)=1977

		1					
					2		
	3						
						4	
5							
			6				
							7
				8			

(8,5,3,2,2,1,1,1)=2329

7.3 <u>SUM OF SUBSETS</u>

Suppose we are given n distinct positive numbers (called weights) and we desire to find all combinations of these numbers whose sum are m. this is called the sum of subset problem.

e.g

if n=4; (w1,w2,w3,w4)=(11,13,24,7); m=31.

Then solution vectors may be

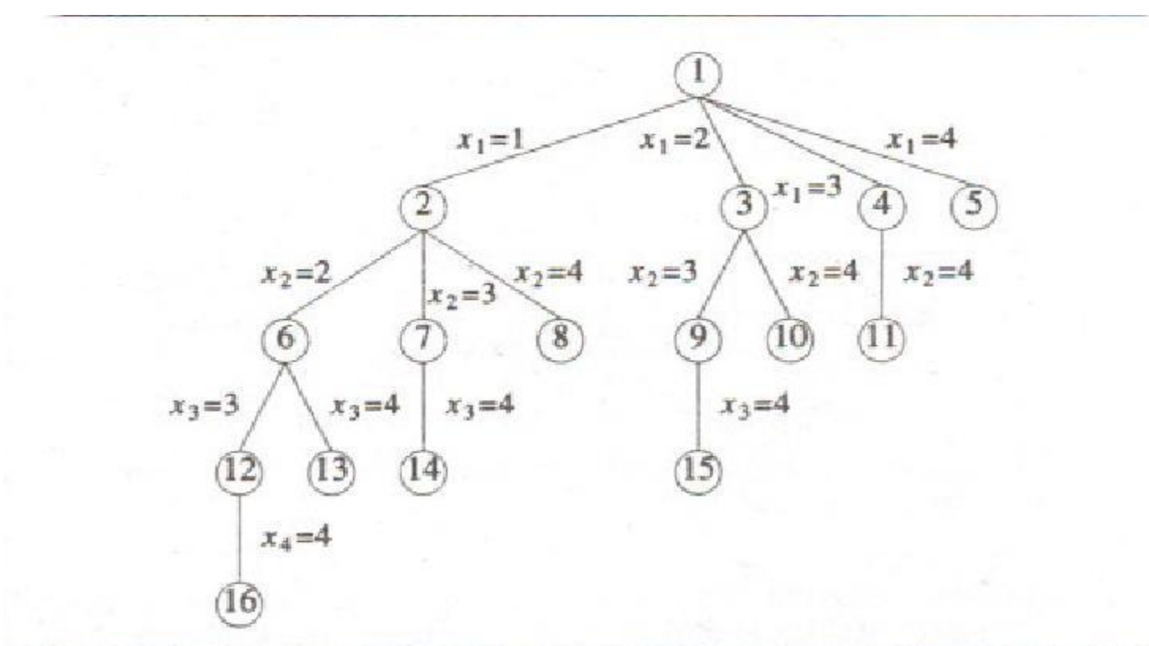
(1,2,4), (3,4) etc…

(elements in the solution vector. Are indices of w)

ie w1+w2+w4+=31

and w3+w4=31 etc..

A possible solution space organization for the sum of subsets problem [nodes numbered in BFS]



Algorithm sum of sub

- Formula if $X_k = 1$, then $\sum_{i=1}^{k} w_i x_i + \sum_{i=k+1}^{n} w_i > m$.

- This algorithm avoids computing $\sum_{i=1}^{k} w_i x_i$ and $\sum_{i=k+1}^{n} w_i$ each time by keeping these

values in variables s and r.

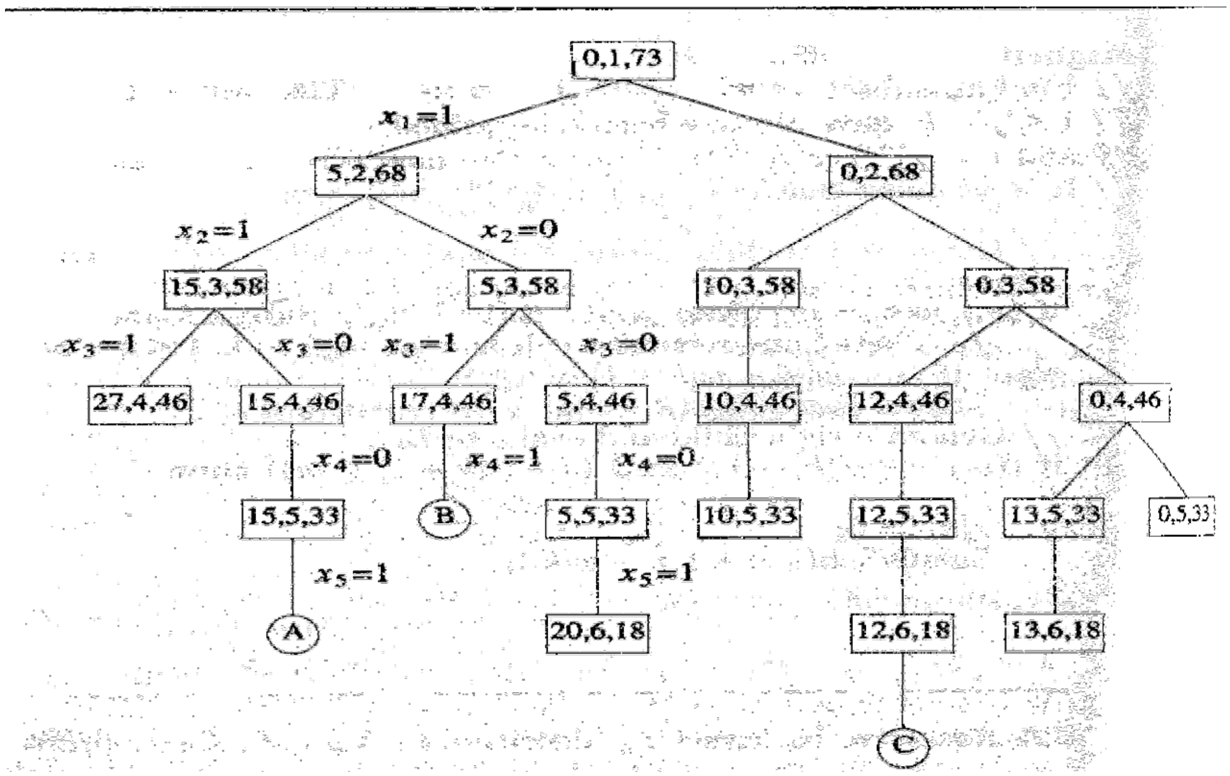
- It assumes $w_1 \leq m$ and $\sum_{i=1}^{n} w_i \geq m$

- The initial call is sum of sub $(0, 1, \sum_{i=1}^{n} w_i)$

```
Algorithm sumofsub(s, k, r)
// finds all subsets of w[1:n] that sum to m.
// The values x[j]; j = 1 to k have already determined
        k-1                  n
// s =  ∑ w[j] * x[j] ; r =  ∑ w[j]
        j-1                 j=k
{
        x[k] = 1;
        if ( s + w[k] = m ) then write (x[1:k]);
        else if ( s + w[k] + w[k + 1] ≤ m )
        then Sumofsub ( s + w[k], k+1, r-w[k]);
        if(( s + r - w[k] ≥ m ) and (s + w[k+1] ≤ m)) then
        {
                x[k] = 0;
                Sumofsub (s, k+1, r-w[k]);
        }
}
```

Example for Sumofsub problem

- Let n = 6; m = 30; w[1:6] = { 5, 10, 12, 13, 15, 18}.
- The rectangular node in fig. Lists values of 3, k, r on each call of above algorithm. Circular node represent points at which result are printed out. At node A, B, C respectively (1,1,0,0,1), (1,0,1,1) and (0,0,1,0,0) are outputs



~~GRAPH COLORING~~

~~m-colorability decision problem:~~

Let G be a graph and m be a given positive integer. Determining whether the nodes of G can be colored in such a way that no two adjacent odes have the same color yet only m colors one used is called M-colorability decision prob.

~~m-colorability optimization prob:~~

m-colorability optimization problem asks the smallest integer m for which the graph G be colored. m is called chromatic number of the graph.

M coloring

To determine all the different ways in which a given graph can be colored using at most m colors .

Suppose we represent a graph by its adjacency matrix G[1:n,1:n];
The colors are represented by the integers 1,2,3,………,m.
The solution are given by the n-tuple (x1,….xn) where xi is the color of node i.
Function m coloring is begun by first assigning the graph to its adjacency matrix, setting the array z[] to zero .and involving statement mcoloring (1);

```
Algorithm m coloring (k)
{
  repeat
     {     // Generate all legal assignment for x [k]
           nextvalue (k);
           if (x[k]=0) then return; //no color possible
           if (k=n) then // at most m color have been used
                write (x[1:n]);
              else
                   mcoloring(k+1);
          } until(false);
   }

Algorithm next value (k)
{
    repeat
    {
      x[k]=(x[k]+1) mod (m+1); //next highest color
      if (x[k]=0) then return; //all colors have been used
     for j:=1 to n do
       {
         if ((G[k,j] ≠0) and (x[k]=x[j]))
         // adjacent vertices have the same color
            then break;
        }
      if (j=n+1) then return; //new color found
   }until (false);  //other try to find another color
}
```

time complexity for m coloring

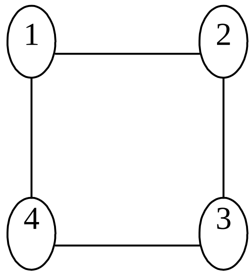
upper bound of complexity time can be calculated by number of internal nodes in the state space tree. ie, $\sum_{i=n-1} m^i$

At each internal node,O(mn) time is spent by NextValue
Hence the total time is bounded by

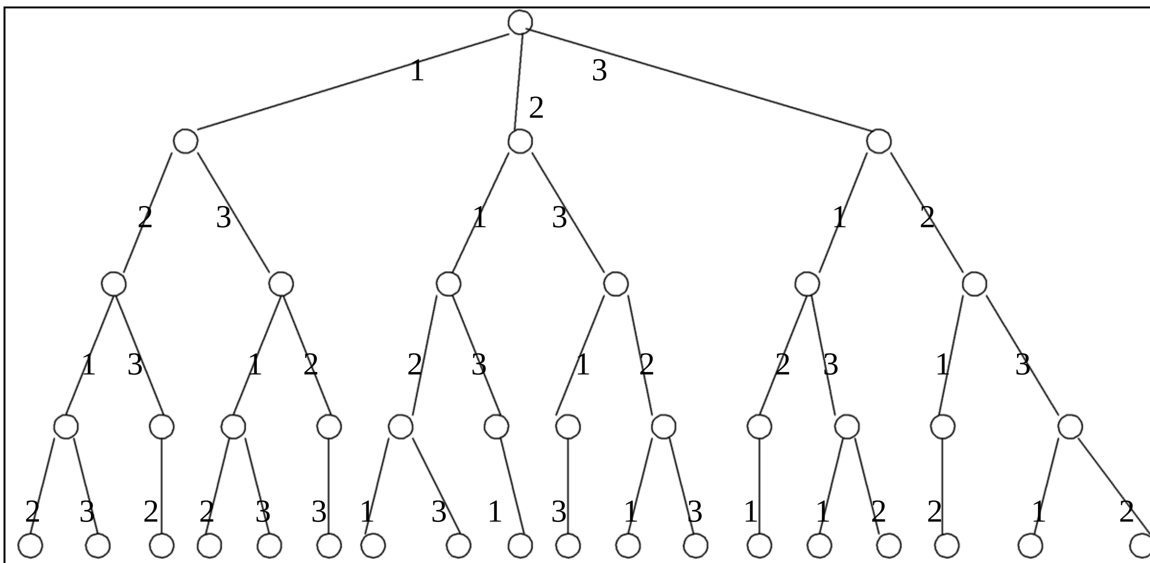
$\sum_{i=n-1} m^{i+1} = \sum_{i=n-1} m^{I} = n(m^{n+1}-2)/(m-1) = O(nm^{n})$

Example for mcoloring problem

Consider the graph of four nodes.



The tree generated by m-coloring for the above graph with m=3 is



A 4-NODE GRAPH AND ALL POSSIBLE 3-COLORING

In this tree after choosing x_1=2 and x_2=1, the possible choices for x_3 are 2 & 3.
After selecting x_1=2 , x_2=1 , x_3=2 possible choice for x_4=1 & 3 and so on.

HAMILTANIAN CYCLES

Definition:(Informal)

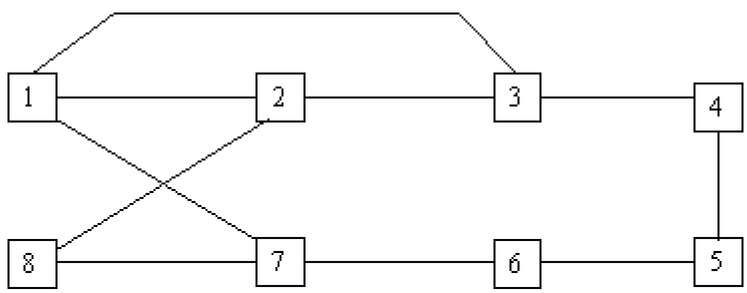
Hamiltonian Cycle is a round –trip path along n-edges of G that visits every vertex once and return to its starting position.

Formal Definition:

Hamilton cycle begins at some vertex $v_1 \in G$ and the vertices of G are visited in the order $V_1, V_2, \ldots V_{n+1}$ then the edges (V_i, V_{i+1}) are in E, $1 \le i \le n$, and the vi are the distinct except for v_1 and v_{n+1}, which are equal.

Example: Graph

G (contain Hamiltonian cycle)



Back tracking solution to Hamiltonian cycles

* Backtracking Alg finds all the Hamiltonian cycles in a graph.
* The solution vector $(x_{1 \ldots Xn)}$ is defined so that x_i represent the ith visited vertex of proposed cycle.
* It begin by $x_1=1$; x_k (for k=2 to n-1) can be any vertex v that is distinct from $x_1, x_2, \ldots x_{k-1}$ and v is connected by an edge to x_{k-1}.
* The vertex x_n can be only be the one remaining vertex after all k_s. and it must be connected to both x_{n-1} and x_1.
* This algorithm is started by initializing matrix g[1:n][1:n] then setting x[2:n] to zero, and x[1] to 1.
* Hamiltonian (2); is called first.

Algorithm Hamiltonian(k)

```
{
repeat
{ // generate values for x[k]
next value(k); if(x[k]=0)
then return; if(k=n) then
write(x[1:n]); else
Hamiltonian(k+1);
} until (false);
}
```

Algorithm nextvalue(k)

```
{
repeat
{
x[k]=(x[k]+1) mod (n+1); // next vertex
if (x[k]=0) then return;
if( g[x[k-1],x[k] <>0) then
{
//is there an edge?
For j=1 to k-1 do
If (x[j]=x[k])then break;
If(j=k)then
If((k<n)or<((k=n) and g[x[n],x[1]]<>0))then return;
}
}until(false);
}
```

UNIT-6

BRANCH AND BOUND

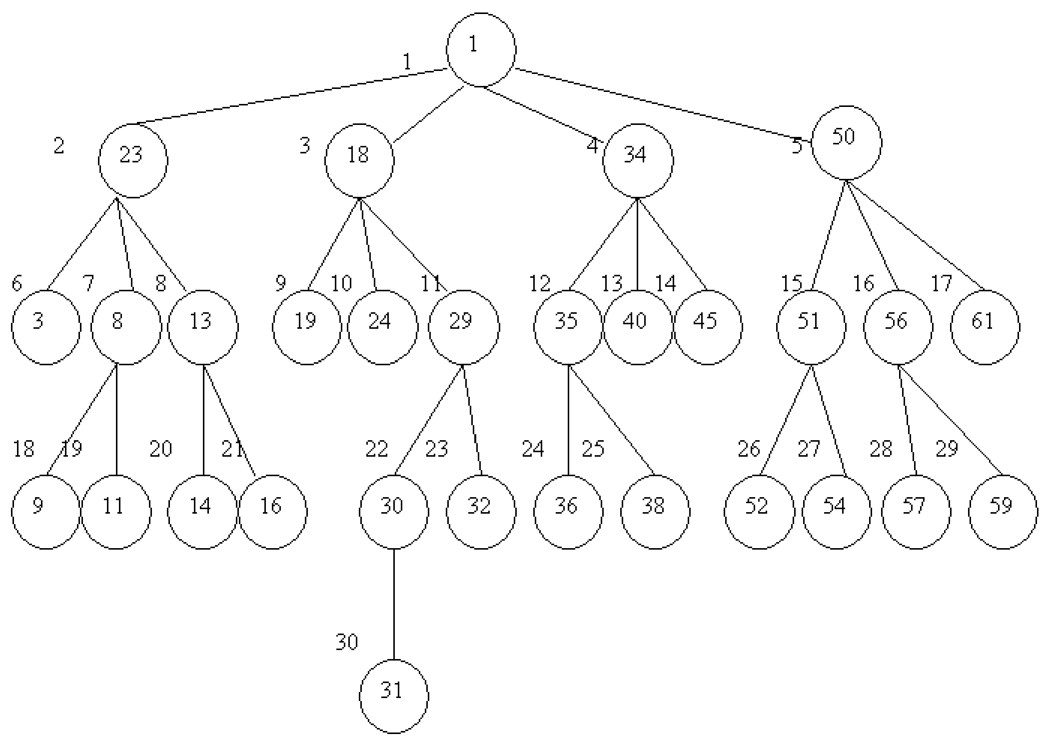
Definition:

The term branch and bound refers to all state search methods in which all children of the E-node are generated before any other live node can become the E-node

- BFS:- like state space search will be called FIFO
- D Search:- like state space search will be called LIFO

Example:

A FIFO branch-and-bound algorithm searches the state space tree for eight queen problem as follows:



In the above LIFO and FIFO branch-and-bound the selection rule for the next E-node does not give any preference to a node that has a very good change of getting the search to an answer node quickly

Solution:

Least cost(LC) search:

The search for an answer node can often speeded by using an "intelligent" ranking function c(.) for lives nodes. The next E-node is selected on the basis of this ranking function

Definition:

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

15-puzzle example:

1)To determine whether the goal state is reachable from the initial sate. Let position(i) be the position number in the initial sate of the tile numbered i. Then position(lb) will denote the position of the empty spot

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

figure (a)

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

figure (b)

figure (c)

For any state let(i) be the number of tiles j such that j<I and position(j)>position(i).

e.g; Less(1)=0, less(4)=1& less(12)=6

let x=1 if in the initial state the empty spot is at one of the shaded positions of figure(c). otherwise x=0.

Theorem:

The goal state of figure (b) is reachable from the initial state iff Σ_i^{16} =less(i)+x is even.

2. Cost Estimation

one possible choice for g(x) is g(x)=number of nonblank tiles not in their goal position.

Example:

An LC search of the figure [part of the state space tree for the puzzle] will begin by using node 1 as the E-node.

a) All children of node 1 are generated and node 1 dies and leaves behind the line nodes 2,3,4 and 5.

b) The next node to become E-node is a live node with least c^(x). Now, c^(2)=1+4, c^(3)=1+4, c^(4)=1+2, c^(5) =1+4. Hence node 4 becomes E-node.

c) All children of node 1 are generated and node 1 dies and leaves behind the line nodes 2,3,4 and 5.

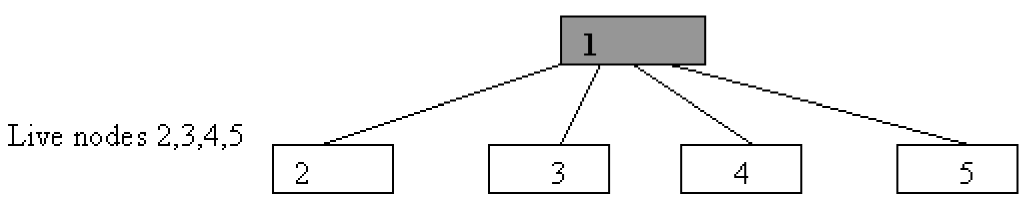
d) The next node to become E-node is a live node with least c^(x). Now, c^(2)=1+4, c^(3)=1+4, c^(4)=1+2, c^(5) =1+4. Hence node 4 becomes E-node.

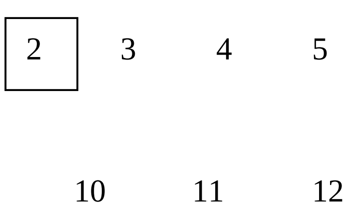
   Fourth node children are generated. The live nodes at this time are 2,3,5,10,11, and 12.

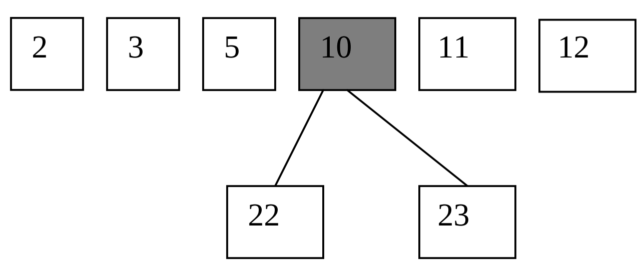
   C^(10)=2+1, c^(11)=2+3, c^(12)2+3 Hence node 10 becomes E-node[since the live node with least c^ is node 10]

e) from node 10, nodes 22, and 23 are generated next. Hence node 23 is the goal node, the search terminates[c^=0]

figure(a)







Node 4 dies

Fig(b) Live nodes 2,3,5,10,11,12 fig(c)

Least select node 10

FIFO BRANCH-AND-BOUND

[BFS+FIFO+Bounding Condition]

eg:

problem:

We are given

- n-jobs,one processor
- job i has associated with a three tuple(pi,di,ti)

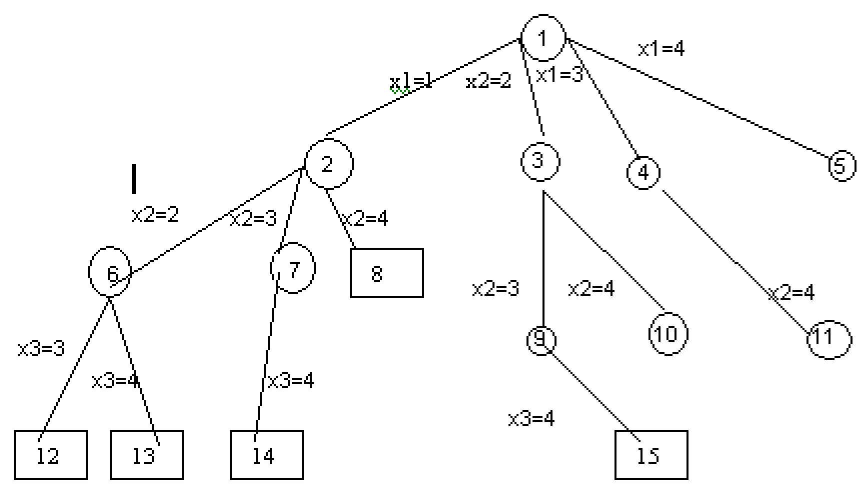
Pi=penalty incurred when the processing not completed by the dead line di

Ti=required units of processing time

Objective is to select a subset J of n jobs such that all jobs in J can be completed by their deadlines & the penalty incurred is minimum among all possible subsets J.(a penalty can be incurred only on those jobs not in J).

Let n=4; (p1,d1,t1)=(5,1,1);(p2,d2,t2);=(10,3,2)
(p3,d3,t3)=(6,2,1);(p4,d4,t4);=(3,1,1)

Solution space tree for the above problem instance is:



○ =Infeasible Subsets.

□ =Answer nodes

Jobs index set
{1,2,3,4}

if x1=1;x2=4 in the tree
J={1,4}

<u>Bounding:</u>

<u>Cost Function: c(x)</u>

For any circular node x, c(x) is the minimum penalty corresponding to any node in the sub tree with root x.

For example, $c(x)=\infty$ for sequence node

C(3)=8; c(2)=9;c(1)=8 etc.

<u>A bound $c^{\wedge}(x)$:</u>

Let Sx be the subset of jobs selected for J at node x. if $m=\max\{i \mid i\in Sx)$, then

$$\boxed{c^{\wedge}(x)= \sum_{\substack{i<m \\ i\notin Sx}} Pi}$$

is an estimate for c(x).

<u>Example</u>:

For mode 7, $S_7=\{1,3\}$ and m=3

Therefore,

$$\sum_{\substack{i<3 \\ I\notin S7}} Pi=P2=10$$

<u>Upper bound u(x):</u> (Cost of a minimum-cost answer node)

$$U(x)=\sum_{i\notin Sx} P_i$$

<u>Example:</u>

$$U(2)= \sum_{i\notin S2} P_i = \sum^{4}_{i=2} P_i = P_2+P_3+P_4 =19$$

$S_2=\{1\}$

LC BRANCH AND BOUND

(LCBB for given Job Schedule Problem Instance)

Procedure:

Step 1: Set upper = ∞ or $\sum_{i=1 \text{ to } n} Pi$

Step 2: Node 1 is Enode. It is expanded. Children 2,3,4,5 are generated.

Step 3: $C^{\wedge}(x)$ is calculated with each child node x of node1.
If u (x) is minimum than upper them upper will set to u(x).
Hence upper becomes u(3) i.e. 14.
$C^{\wedge}(4)$, $C^{\wedge}(5)$ > upper. So 4,5 nodes get deleted.

Step 4: Next E-Node is 2. Since C^(2) < C^(3) (Least cost BB).
Nodes 6,7,8 generated.
C^(7) > upper and 8 is infeasible, both are killed.

Step 5: Next E-Node is 6. Since out of all give nodes i.e. 6,3.
C^(6) < C^(3) (i.e. Least cost BB)
All its children one generated i.e. 12,13 but both are infeasible.

Step 6: Next E-Node is 3. Node 9 (child of 3) is generated. u(9) < upper and
C^(9) < upper. So upper becomes u(9) i.e. 8.
Node 10 is killed. Since C^(10) > upper i.e. 8.

Step 7: Next E-Node is node 9. It s child is infeasible. Here no live node remains.
The Search terminates with node 9 representing minimum-cost as node.

Procedure with Example

X	1	2	3	4	5	6	7	8	9	10	11
C∋()	**0**	**0**	**5**	**15**	**21**	**0**	**10**	**-**	**5**	**11**	**15**
U()	**24**	**19**	**14**	**18**	**21**	**9**	**10**	**16**	**8**	**1**	**15**

- Set upper = ∞ (or upper = $\sum_{i=1 \text{ to } n}$ Pi) // upper bound on the cost of a minimum-cost and node.
- Set node 1 as E-node. Generate child nodes.
- Upper will be set to 19 than 14(when node 3 is generated)
- If c∧(x) for current generated child is > upper than kill the nodes. Hence nodes 4,5 get killed.
- Node 2 becomes next E-node. Generate children nodes 6,7,8. u(6)=9; hence upper=9. c∧(x) for node 7 > upper. So 7 get killed. Node 8 is infeasible so it's killed.
- Node 3 becomes E-node I nod 9,10 generated. U(9)=8; hence upper=8. C∧(10) > upper hence node 10 is killed.
- Node 6 becomes E-node; its children are infeasible.
- Node 9 becomes E-node; its child is infeasible.

Hence minimum cost answer node is 9, it has a cost of 8.

Travelling salesman problem

Let G=(V,E) be a directed graph
Let Cij be the cost of edge<i,j>,Cij=∞ if <i,j>∉E
Let │V│=n.
Every tour starts and ends at vertex 1.

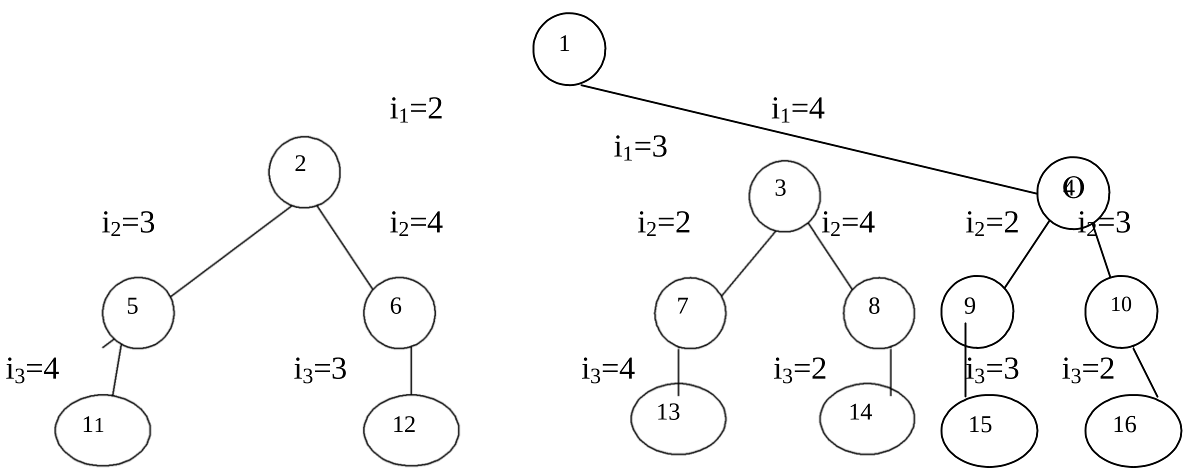
Objective:

To find minimum cost tour.

Solution:

In order to use LC Branch&Bound to search the travelling salesperson tree,we need to define a cost function C()and other two functions C^(),u().

State space tree for the travelling salesperson problem with n=4 & $i_0 = i_1 = 1$.



A cost estimation C^(.)such that C(A)≥C^(A) for all node A is obtained by defining C^(A) to be the length of the path defined at node A.

Procedure LCBB :

Step 1: A matrix is reduced by reducing rows and column of the matrix. A row(column) is said to reduced iff it contains atleast one zero and all remaining entries are non-negative.

Step 2: C (**.**) may be obtained by using the reduced cost matrix corresponding to G.
C(**.**) =[sum of minimum row value]+[sum of minimum column value]

Step 3: If an edge <i,j> in the tour, then change all entries in row i and column j of A to ∞.

Let A be a reduced cost matrix for node R. Let S be child of R.

Step 4: Set A(j,1) to ∞ .

Step 5: Reduce all rows and columns in the resulting matrix except for rows and columns

Containing only ∞. Let the resulting matrix be B.

Step 6: Step 3 and Step 4 are valid as no tour in the sub tree S can contain edges of the type

< i , k> or < k ,j >.

Step 7: If r is the total amount subtracted in Step 5 then C(S)=C(R)+A(i , j) + r.

Step 8: If leaf nodes C(**.**) = c( ) is easily computed as each leaf defines a unique tour.

Step 9: For the upper bound function u, we may use u(R) = ∞ for all nodes R.

Example: <u>Cost Matrix</u>

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

Reduce by Row wise

$$\begin{array}{cc} \begin{bmatrix} \infty & 4 & 0 & 9 & 5 \\ 0 & \infty & 3 & 11 & 6 \\ 0 & 3 & \infty & 1 & 13 \\ 6 & 0 & 2 & \infty & 8 \\ 10 & 6 & 1 & 0 & \infty \end{bmatrix} & \begin{matrix} 3 \\ 3 \\ 5 \\ 3 \\ 8 \end{matrix} \end{array}$$

<u>Reduce by Column wise</u>

$$\begin{array}{c} \begin{bmatrix} \infty & 4 & 0 & 9 & 5 \\ 0 & \infty & 3 & 11 & 6 \\ 0 & 3 & \infty & 1 & 13 \\ 6 & 0 & 2 & \infty & 8 \\ 10 & 6 & 1 & 0 & \infty \end{bmatrix} \\ \begin{matrix} 0 & \;\;0 & \;\;0 & \;\;0 & \;\;5 \end{matrix} \end{array}$$

ĉ(x)=[3+3+5+3+8]+[5}=27

(1,2)- Make all the elements in 1[St] row and 2[nd] column to ∞ and A(2,1) to ∞ .

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ .\infty & \infty & 3 & 11 & 1 \\ 0 & \infty & \infty & 1 & 8 \\ 6 & \infty & 2 & \infty & 3 \\ 10 & \infty & 1 & 0 & \infty \end{bmatrix}$$

<u>Reduce by Row wise:</u>

$$\begin{array}{cc} \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ .\infty & \infty & 2 & 10 & 0 \\ 0 & \infty & \infty & 1 & 8 \\ 4 & \infty & 0 & \infty & 1 \\ 10 & \infty & 1 & 0 & \infty \end{bmatrix} & \begin{array}{c} \\ 1 \\ 0 \\ 2 \\ 0 \end{array} \end{array}$$

<u>Reduce by Column wise</u>

(Same as previous matrix since every Column has an element 0)

C(S)=C(R)+A(i,j)+r.

=27 + 4 + 3 = 34

(1,3) 3

$$\begin{array}{ccccc|c}
\infty & \infty & \infty & \infty & \infty & \\
0 & \infty & \infty & 11 & \cancel{1}0 & 0 \\
\infty & \cancel{3}2 & \infty & \cancel{1}0 & \cancel{8}\cancel{7}6 & 1 \\
6 & 0 & \infty & \infty & \cancel{3}2 & 0 \\
10 & 6 & \infty & 0 & \infty & 0 \\
\hline
0 & 0 & & 0 & 1 &
\end{array}$$

27+0+2=29

(1,4) 4

$$\begin{array}{ccccc|c}
\infty & \infty & \infty & \infty & \infty & \\
0 & \infty & 3 & \infty & \cancel{1}0 & 0 \\
0 & 3 & \infty & \infty & \cancel{8}7 & 0 \\
\infty & 0 & 2 & \infty & \cancel{3}2 & 0 \\
\cancel{10}9 & \cancel{6}5 & \cancel{1}0 & \infty & \infty & 1 \\
\hline
0 & 0 & 0 & & 1 &
\end{array}$$

27+9+2=38

(1,5) 5

$$\begin{array}{ccccc|c}
\infty & \infty & \infty & \infty & \infty & \\
0 & \infty & \cancel{3}2 & 11 & \infty & 0 \\
\infty & 3 & \infty & 1 & \infty & 0 \\
6 & \infty & \cancel{2}1 & \infty & \infty & 0 \\
\infty & 6 & \cancel{1}0 & 0 & \infty & 0 \\
\hline
0 & 0 & 1 & 0 & &
\end{array}$$

27+0+1=28

(1,5) is minimum

(1,5,2) 6

$$\begin{array}{ccccc|c}
\infty & \infty & \infty & \infty & \infty & \\
\infty & \infty & \cancel{2}0 & \cancel{11}\cancel{9}8 & \infty & 2 \\
\infty & \infty & \infty & \cancel{1}0 & \infty & 0 \\
\cancel{\infty} & \infty & \cancel{1}0 & \infty & \infty & 1 \\
\infty & \infty & \infty & \infty & \infty & \\
\hline
0 & & 0 & 1 & &
\end{array}$$

28+6+4=38

(1,513) 7

$$\begin{array}{ccccc|c}
\infty & \infty & \infty & \infty & \infty & \\
0 & \infty & \infty & 11 & \infty & 0 \\
\infty & \cancel{3}2 & \infty & \cancel{1}0 & \infty & 1 \\
6 & 0 & \infty & \infty & \infty & 0 \\
\infty & \infty & \infty & \infty & \infty & \\
\hline
0 & 0 & & 0 & &
\end{array}$$

(1, 5, 4) 8

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \cancel{2}^{1} & \infty & \infty \\ 0 & 3 & \infty & \infty & \infty \\ \infty & 0 & \cancel{1}^{0} & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix} \begin{matrix} \\ \\ 0 \\ 0 \\ 0 \end{matrix}$$

0 0 1
28 + 0 + 1 = 29

Since
(1,5,3)
& = 29
(1,5,4)

LCBB try for
both case

(1, 5, 3,2)

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & {}^{0}\cancel{11} & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 6 & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix} \begin{matrix} \\ 11 \\ \\ 0 \\ \\ \end{matrix}$$

6 0
29 + 2 + 17 = 48

10

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

0 0
29 + 0 + 0 = 29

11

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

29 + 0 + 0 = 29

12

$$\begin{bmatrix} \infty & \infty & {}^{0}\cancel{1} & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix} \begin{matrix} 1 \\ 0 \\ \\ \\ \end{matrix}$$

29 + 0 + 1 = 30

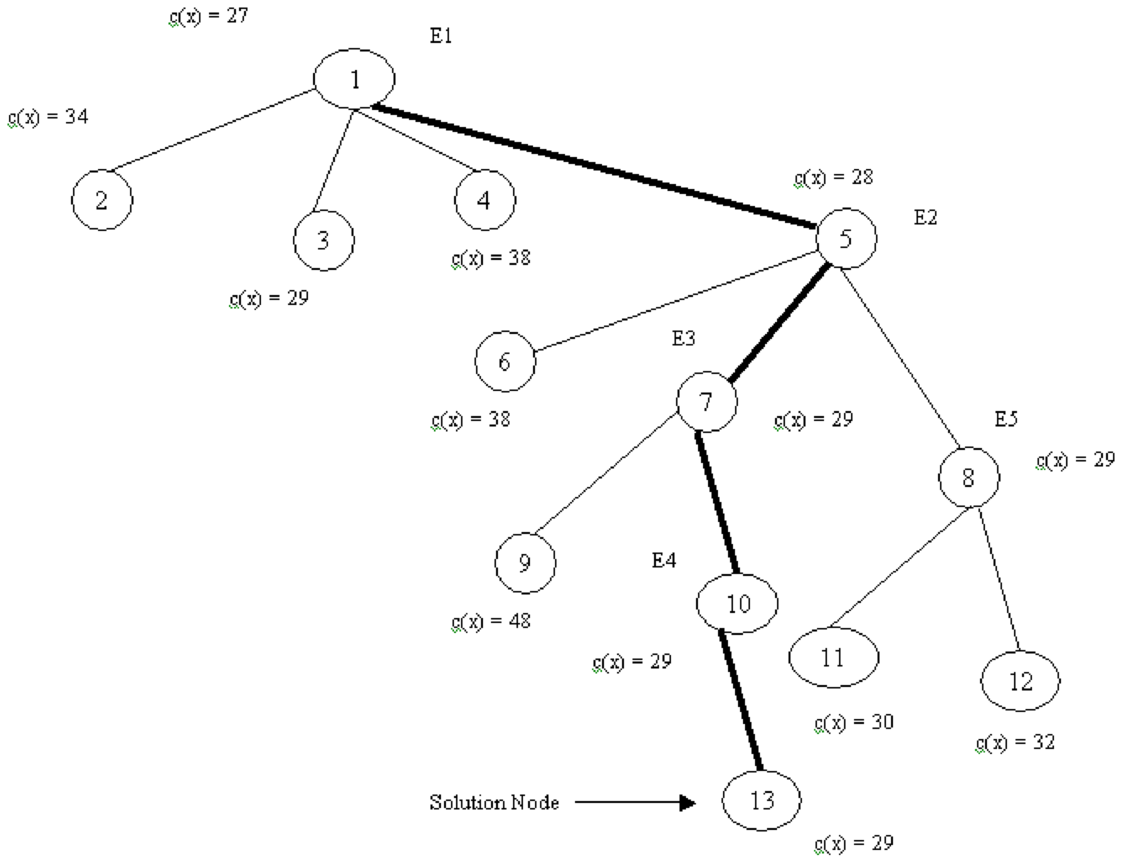
13

$$\begin{bmatrix} 0 & \infty & \infty & \infty & \infty \\ \infty & {}^{0}\cancel{3} & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix} \begin{matrix} 0 \\ 3 \\ \\ \\ \end{matrix}$$

0 0
29 + 0 + 3 = 32

Hence the four has the path nodes 1 -> 5-> 2-> 4 -> 2->1
Cost that is 8+9+6+3+3 = 29

STATE SPACE TREE GRENERATED BY PROCEDURE LCBB



E1, E2, ………. E5 – are E nodes E_{num} indicates order of selecting next E node. (next E-node is selected based on Least Cost ie ĉ (x)).