Backtracking

BACKTRACKING

Principal

- Problems searching for a set of solutions or which require an optimal solution can be solved using the backtracking method .
- To apply the backtrack method, the solution must be expressible as an n-tuple($x_1,\ldots,x_n$), where the x_i are chosen from some finite set s_i
- The solution vector must satisfy the criterion function $P(x_1 , \ldots , x_n)$.

For a 0/1 knapsack problem (x1,x2,x3,x4)=(1,0,0,1)

Criterion function P is $\sum x_i w_i \leq m$

BACKTRACKING (Contd..)

- Suppose there are m n-tuples which are possible candidates for satisfying the function P.
- Then m= m_1, m_2…..m_n where m_i is size of set s_i $1<=i<=n$.
- The brute force approach would be to form all of these n-tuples and evaluate each one with P, saving the optimum.

BACKTRACKING (Contd..)

- The backtracking algorithm has the ability to yield the same answer with far fewer than m-trials.
- In backtracking, the solution is built one component at a time.
- Modified criterion functions P_i $(x_1...x_n)$ called bounding functions are used to test whether the partial vector $(x_1,x_2,......,x_i)$ can lead to an optimal solution.
- If $(x_1,...x_i)$ is not leading to a solution, $m_{i+1},....,m_n$ possible test vectors may be ignored.

BACKTRACKING – Constraints

EXPLICIT CONSTRAINTS are rules which restrict the values of x_i.

Examples $x_i \geq 0$ or $x_i = 0$ or 1 or $l_i \leq x_i \leq u_i$.

IMPLICIT CONSTRAINTS describe the way in which the x_i must relate to each other .

Example : 8 queens problem.

BACKTRACKING: Solution Space

- Tuples that satisfy the explicit constraints define a **solution space**.

- The solution space can be organized into a tree. Each node in the tree defines a **problem state**.

- All paths from the root to other nodes define the **state-space** of the problem.

- **Solution states** are those states leading to a tuple in the solution space.

- **Answer nodes** are those solution states leading to an answer-tuple (i.e. tuples which satisfy implicit constraints).

BACKTRACKING: Solution Space

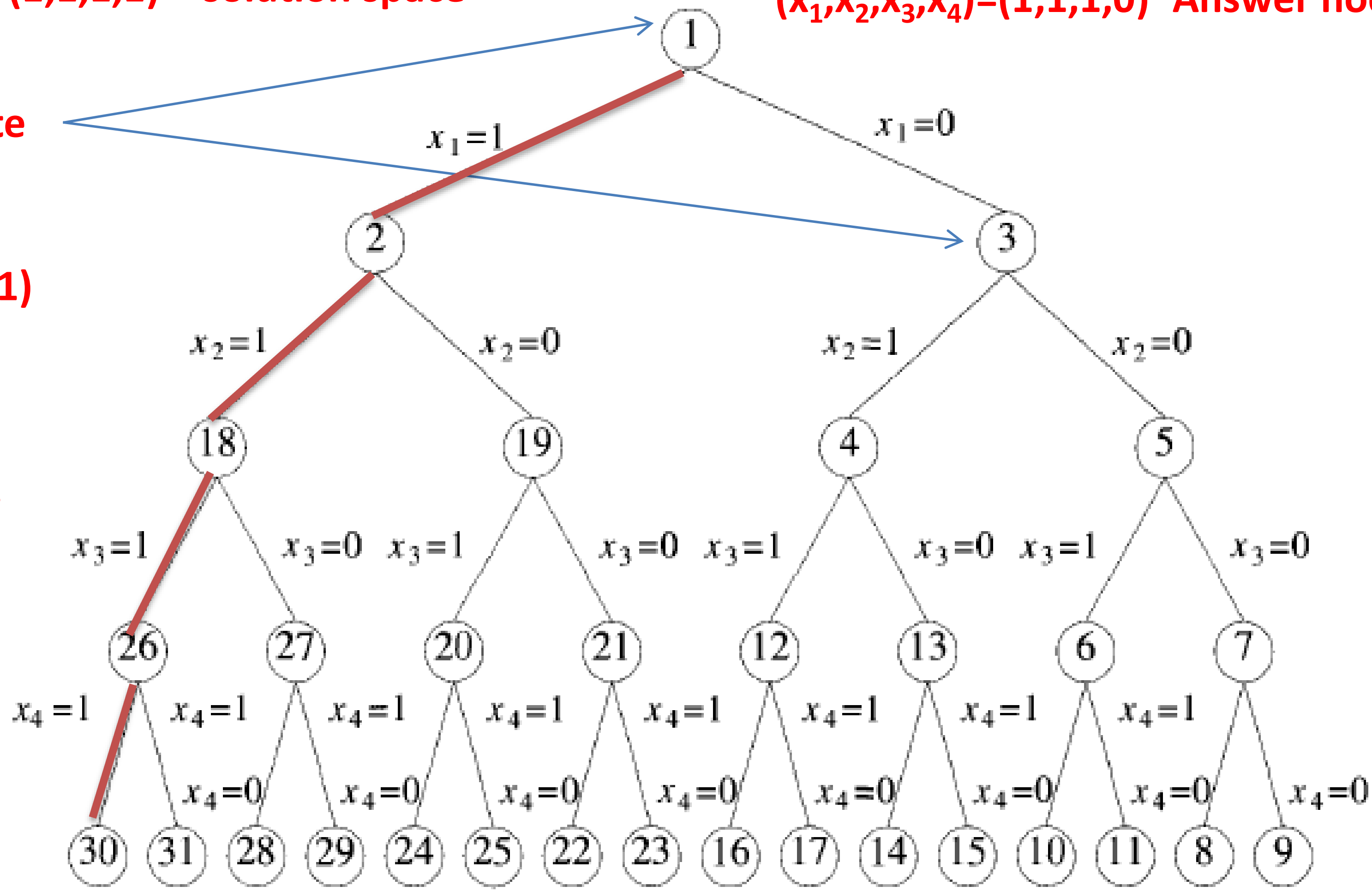
If weights of 4 items are 10,20,30,50 and knapsack capacity is 70
$(x_1,x_2,x_3,x_4)=(1,1,1,0)$ Answer node

$(x_1,x_2,x_3,x_4)=(1,1,1,1)$ solution space

Problem state

State space
$(x_1,x_2,x_3)=(1,1,1)$

Solution States
$(x_1,x_2,x_3,x_4)=$
$(1,1,1,1)$



BACKTRACKING -Terminology

LIVE NODE A node which has been generated and all of whose children are not yet been generated .

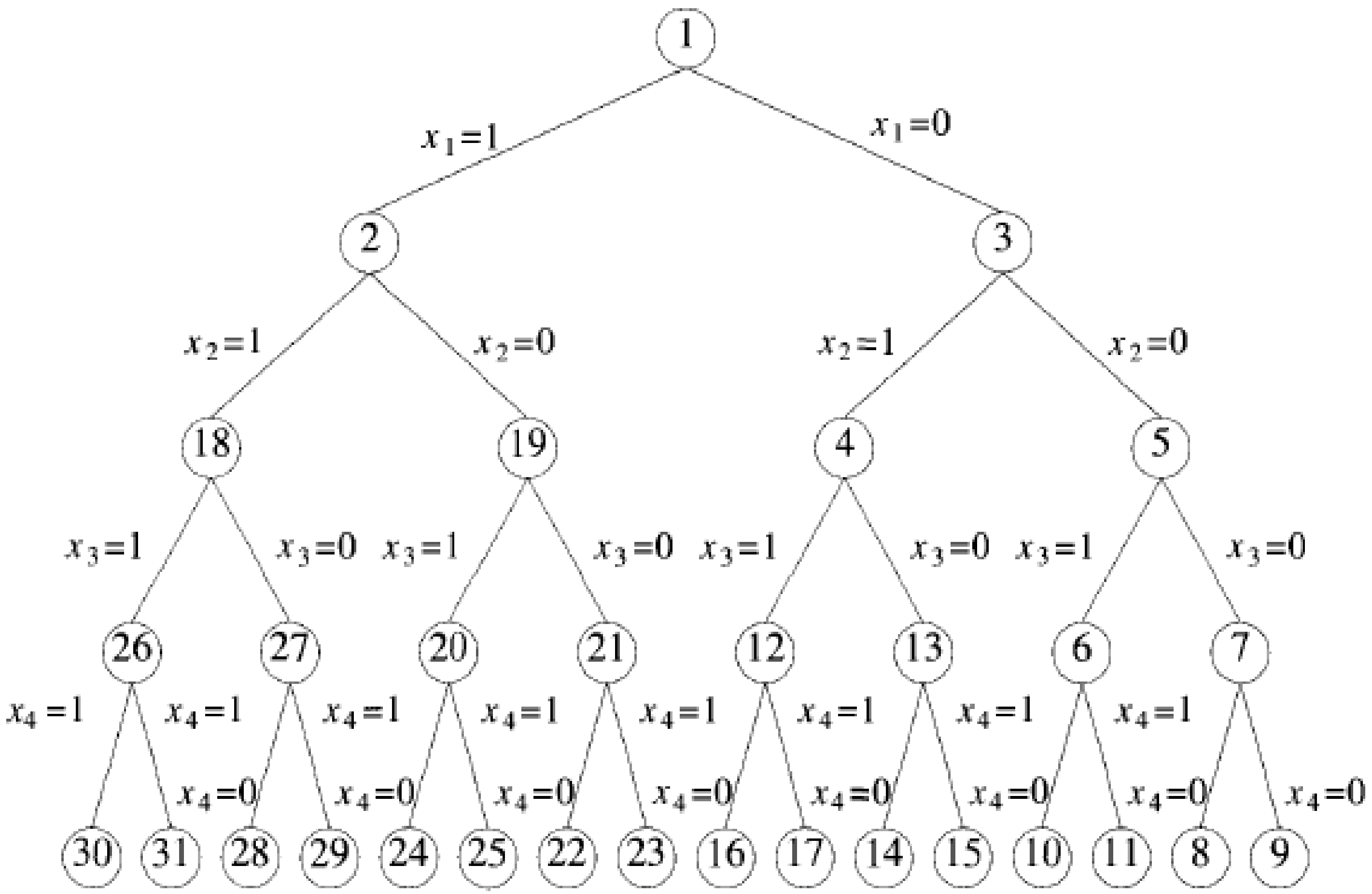
E-NODE (Node being expanded) - The live node whose children are currently being generated .

DEAD NODE - A node that is either not to be expanded further, or for which all of its children have been generated

DEPTH FIRST NODE GENERATION- In this, as soon as a new child C of the current E-node R is generated, C will become the new E-node.

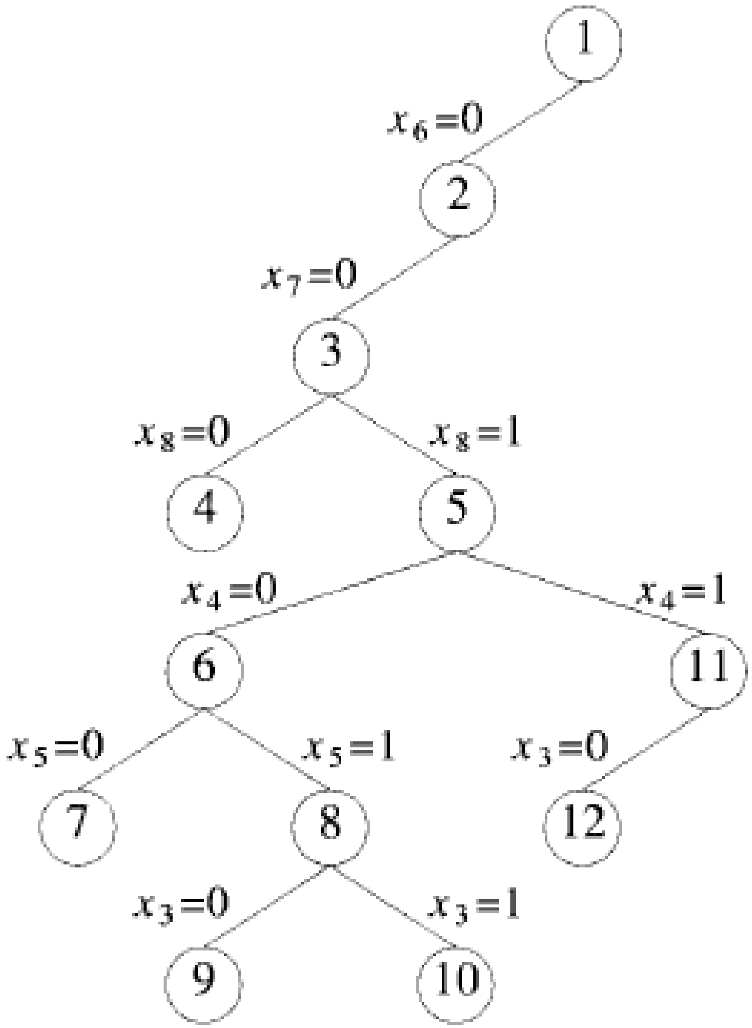
BACKTRACKING -Terminology

Static trees: the tree organizations are independent of the problem instance being solved



BACKTRACKING -Terminology

Dynamic trees: Tree organizations that are problem instance dependent



BACKTRACKING -Terminology

BOUNDING FUNCTION - will be used to kill live nodes without generating all their children.

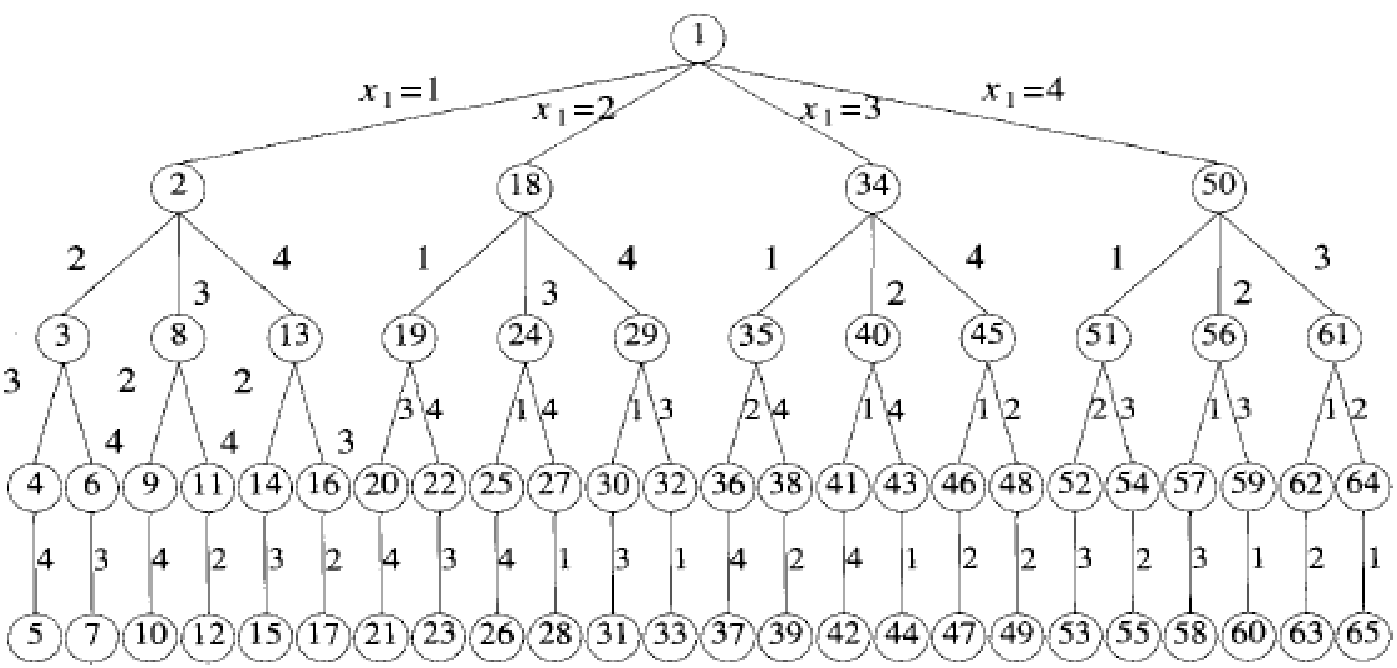
BACTRACKING-is depth – first node generation with bounding functions.

BRANCH-and-BOUND- is a method in which E-node remains E-node until it is dead.

BACKTRACKING -Terminology

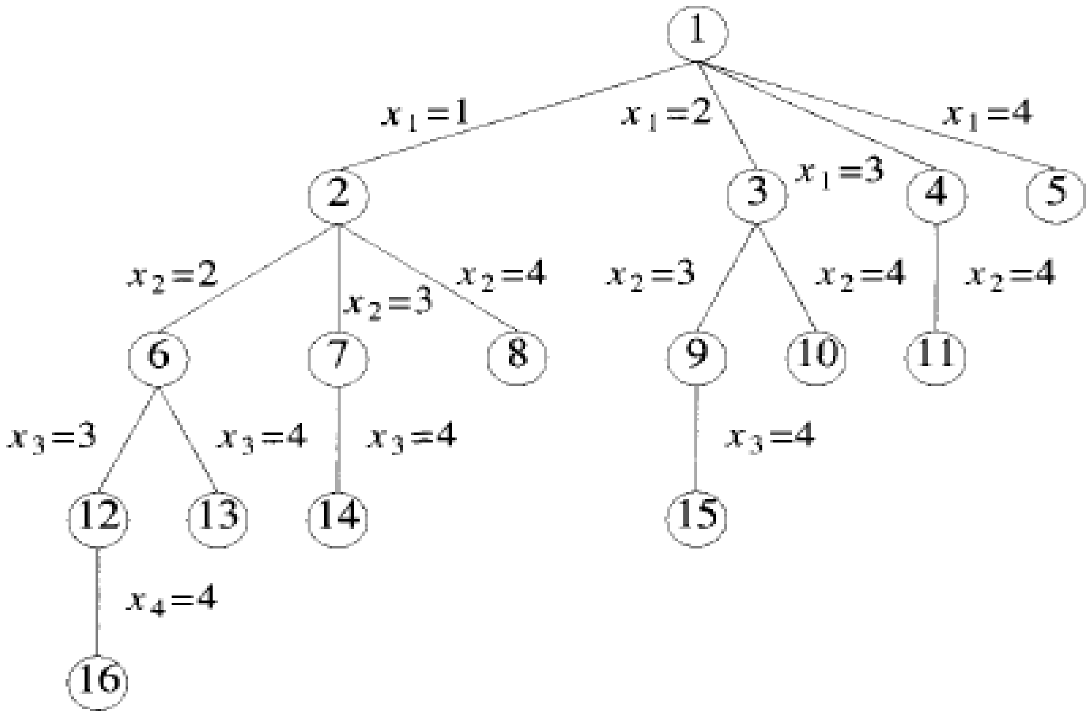
BACKTRACKING-is depth – first node generation with bounding functions.

DEPTH FIRST NODE GENERATION- In this, as soon as a new child C of the current E-node R is generated, C will become the new E-node.



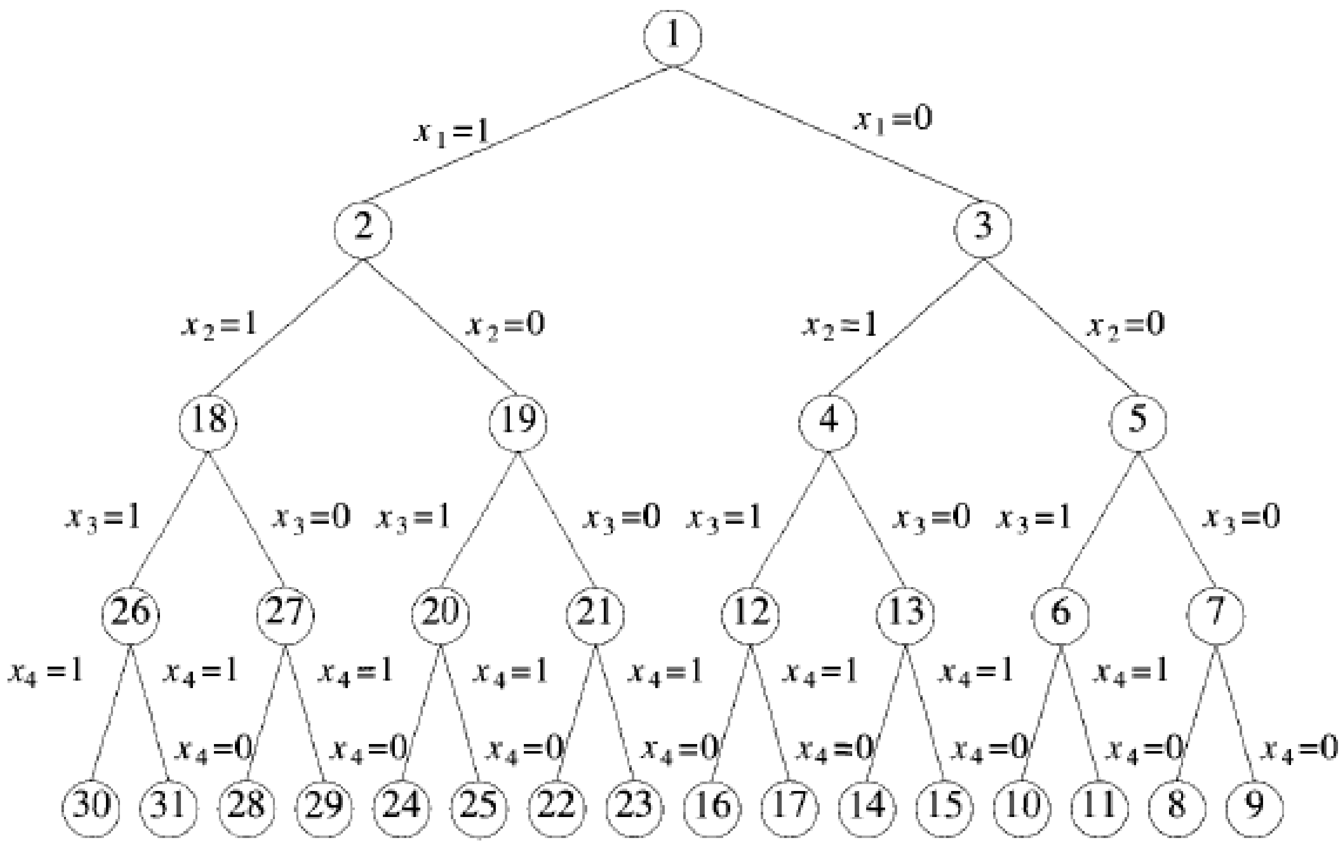
Branch and Bound -Terminology

BREADTH-FIRST-SEARCH : Branch-and Bound with each new node placed in a queue .The front of the queen becomes the new E-node.



Branch and Bound -Terminology

DEPTH-SEARCH (D-Search) : New nodes are placed in to a stack.The last node added is the first to be explored.



Iterative Control Abstraction (General Backtracking Method)

```
Algorithm IBacktrack(n)
// This schema describes the backtracking process.
// All solutions are generated in x[1 : n] and printed
// as soon as they are determined.
{
    k := 1;
    while (k ≠ 0) do
    {
        if (there remains an untried x[k] ∈ T(x[1], x[2], ...,
            x[k − 1]) and B_k(x[1], ..., x[k]) is true) then
        {
            if (x[1], ..., x[k] is a path to an answer node)
                then write (x[1 : k]);
            k := k + 1; // Consider the next set.
        }
        else k := k − 1; // Backtrack to the previous set.
    }
}
```

Recursive Control Abstraction
(General Backtracking Method)

```
Algorithm Backtrack(k)
// This schema describes the backtracking process using
// recursion. On entering, the first k - 1 values
// x[1], x[2], ..., x[k - 1] of the solution vector
// x[1 : n] have been assigned. x[ ] and n are global.
{
    for (each x[k] ∈ T(x[1], ..., x[k - 1]) do
    {
        if (B_k(x[1], x[2], ..., x[k]) ≠ 0) then
        {
            if (x[1], x[2], ..., x[k] is a path to an answer node)
                then write (x[1 : k]);
            if (k < n) then Backtrack(k + 1);
        }
    }
}
```

EFFICIENCY OF BACKTRACKING ALGORITHM Depend on 4 Factors

- The time to generate the next X(k)
- The no. of X(k) satisfying the explicit constraints
- The time for bounding functions B_i
- The no. of X(k) satisfying the B_i for all i