Backtracking and B&B Design Technique

Adnan YAZICI
Dept. of Computer Engineering
Middle East Technical Univ.
Ankara - TURKEY

Backtracking Design Technique

- Backtracking design technique is applicable to problems whose solutions can be expressed as sequences of decisions.
- It is based on a search of an associated state space tree modeling all possible sequences of decisions.
- In its basic form, backtracking resembles a depth-first search in a directed graph.
- Backtrack strategy finds all solutions to a given problem by searching for all goal states in a state space tree associated with the problem.
- When a node is accessed during a backtracking search, it becomes the current node being expanded (e-node), but immediately its first child not yet visited becomes the new e-node.

Backtracking Design Technique

- Backtracking is a recursive method of building up feasible solutions one at a time.
- In an unmodified form, backtracking is exhaustive: all possible feasible solutions are considered.
- Methods involving pruning often reduce the total number of feasible solutions generated, which makes the resulting algorithm much more efficient.
- Backtracking approach also invokes a bounding function to prune more problem states.

Backtracking Design Technique

• We assume for decision-making process that the decision x_k at stage k must be drawn from a finite set S_k of choices.

• For each k >1, the choices available for decision x_k may be limited by choices that have already been made for $x_1,\ldots,x_{k-1}$. In other words, decision x_k may be restricted to a strict subset of S_k.

• For a given problem instance, suppose n is the maximum number of decision stages that can occur. For $k \leq n$, we let P_k denote the set of all possible sequences of k decisions, represented by k-tuples $(x_1, x_2,\ldots, x_k)$.

•Elements of P_k are called problem states, and problem states that correspond to solutions to the problem are called goal states.

Backtracking Design Technique

- Given a problem state $(x_1,x_2,\ldots,x_{k-1}) \in P_{k-1}$, we let $D_k(x_1,x_2,\ldots,x_{k-1})$ denote the decision set consisting of the set of all possible choices for decision x_k.
- More precisely, for $(x_1,x_2,\ldots,x_{k-1}) \in P_{k-1}$,

$$D_k(x_1,x_2,\ldots,x_{k-1}) = \{ x_k \in S_k \mid (x_1,x_2,\ldots,x_k) \in P_k \}$$

- Letting $\varnothing$ denote the null tuple (), note that $D_1(\varnothing)$ is the set of choices for x_1; that is, $D_1(\varnothing) = S_1$.
- The decision sets $D_k(x_1,x_2,\ldots,x_{k-1})$, $k = 1,\ldots,n$, determine a decision tree T of depth n, called the state space tree.
- The nodes of T at level k, $0 \le k \le n$, are the problems states $(x_1,x_2,\ldots,x_k) \in P_k$ (P_0 consists of a null tuple).
- For $0 \le k < n$, the children of $(x_1,x_2,\ldots,x_{k-1})$ are the problem states $\{(x_1,x_2,\ldots,x_{k-1}) \mid x_k \in D_k(x_1,x_2,\ldots,x_k)\}$.

Backtracking Design Technique

The General Backtracking Paradigm:

Procedure **BacktrackRec**(k)

Input: - k (a nonnegative integer, 0 on the initial call)

- implicit state space tree T associated with the given problem and decision set D_k, where $D_k = \emptyset$ for $k \geq n$.

- We assume that an implicit ordering exists for the elements of $D_k(x_1, \ldots, x_{k-1})$.
- Explicit global array X[1:n] maintaining problem states of T, where X[1],…,X[k] are assumed already defined
- bounding function *Bound*

Output: all goals that are descendants of (X[1],…,X[k])

k = k +1

For each $x_k \in D_k$ (X[1],…,X[k-1]) *do*

X[k] = x_k

If (X[1],…,X[k]) is a goal state *then*Print (X[1],…,X[k])

If not.Bounded(X[1],…,X[k]) *then Call* backtrackRec(k)

Backtracking Design Technique

Example: *0-1 knapsack problem*.

• The total number of n-tuples is 2^n, but not all of these are feasible.
• A naïve backtracking algorithm is shown below, which performs no pruning: all 2^n possibilities are generated.
• Since it takes $\Theta(n)$ time to check an n-tuple for feasibility, the entire algorithm is $\Theta(n.2^n)$.

• The procedure would be invoked by:

```
optprofit = 0;
knap (1,&optprofit, optx, w,p,n,kcap);
```

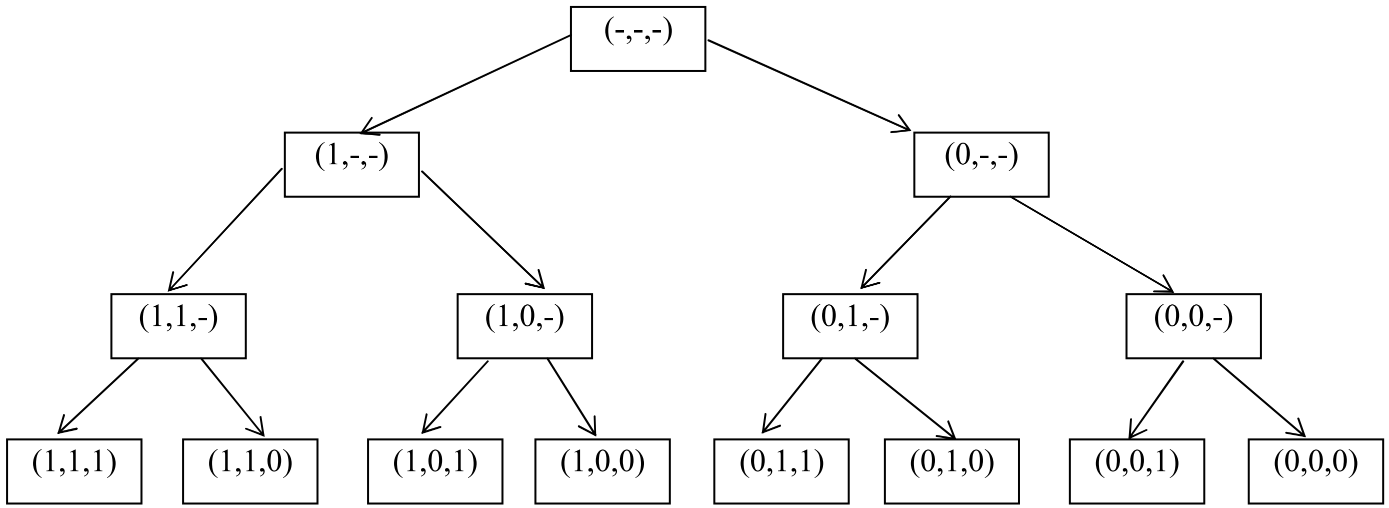
Backtracking Design Technique

```
void function knap (int lev, int *optprofit, int optx [LEN], int
w[LEN], int p[LEN], int n, int knap)
{
  if lev = = (n+1) then
    if ∑w_i x_i ≤ knap then                 // check feasibility
        if  ∑p_i x_i > *optprofit then       // compare profit to opt
         {
            *optprofit = ∑p_i x_i
            optx= x   //whole array assignment, x is a global variable
        }
        else                           // lev ≤ n; try both choices for x_lev
        {       x_lev  = 1;
                knap(lev+1, &optprofit, optx, w, p, n, knap);
                x_lev  = 0;
                knap(lev+1, &optprofit, optx, w, p, n, knap);}}
```

Backtracking Design Technique

• The recursive calls generated by this procedure (knap) can be represented by a binary tree, called the state space tree of the problem instance.

• When n=3, the tree is as follows, where at each node we record the current value of x ('-' indicates an unspecified coordinate).



Backtracking Design Technique

- The revised algorithm for the knapsack problem given below uses pruning whenever $\sum w_i x_i >$ knap (or M).
- In this new algorithm, the parameter curwt records the weight $\sum_{1 \le i \le lev-1} w_i x_i$ of the partial solution $(x_1, x_2, \ldots, x_{lev-1}, -, \ldots, -)$.
- The procedure would be invoked by:

```
optprofit = 0;
knap_1 (1,0,&optprofit, optx, w,p,n,kcap);
```

Backtracking Design Technique

```
void function knap_1 (int lev, int curwt, int *optprofit, int optx
[LEN], int w[LEN], int p[LEN], int n, int knap)
{        ...
         ...
         else     //lev ≤ n; before setting xlev = 1, check feasibility)
         {
          if (curwt + w_lev) ≤ knap the
          {
             x_lev  = 1;
             knap_1(lev+1, curwt+wlev, &optprofit, optx, w, p, n, knap);
          }
             xlev  = 0;  // no check is needed here
             knap_1(lev+1, curwt, &optprofit, optx, w, p, n, knap);
         }}
```

Backtracking Design Technique

Bounding functions:

- Can we do a better pruning job than just checking for feasibility of subsolutions? The answer is yes, in many cases.
- A more sophisticated pruning technique involves the use of a bounding function. A bounding function generates an upper limit (in the case of maximization) of the profit that a partial solution can possibly generate.
- A bounding function is any function B defined on the set of nodes of the state space tree that satisfies the following properties:

1) If X is a feasible solution (lev=n) then B(X) = C(X) = the profit incurred by X

2) For any feasible partial solution X, B(X) ≥ C(X).

- Therefore, B(X) provides an upper bound on the profit of any feasible solution that is a descendant of X.
- We can use a bounding function B(X) to prune the state space tree as follows:
- Suppose, at some stage of the backtrack, that B(X)≤OptP. Then, we can ignore all descendants of X since none of them can yield a profit higher than OptP.
- Effective bounding functions must have the following properties:

  1) Be easy to compute

  2) Be close to C(X), maximum profit of any feasible solution.

Backtracking Design Technique

- By continuing the 0-1 knapsack problem example, let us now apply a bounding function to the problem.
- An obvious bounding function is to assign to node X the profit produced by the rational knapsack problem for the remaining capacity and the remaining objects.
- That is, given a feasible partial solution $X=(x_1,x_2,\ldots,x_{lev},-,\ldots,-)$; $0 \le lev \le n$, define B(X) as;

  $B(X) = \sum_{1\le i\le lev} p_i x_i + ratknap(lev, kcap - \sum_{1\le i\le lev} w_i x_i)$

  $\quad\quad = \sum_{1\le i\le lev} p_i x_i + ratknap(lev, kcap - curwt)$
- Thus, B is the actual profit obtainable from objects 1,2,…,lev plus the optimal profit from the remaining objects and remaining capacity, but allowing rational x's.
- With this as a precondition, the following algorithm applies the bounding function.

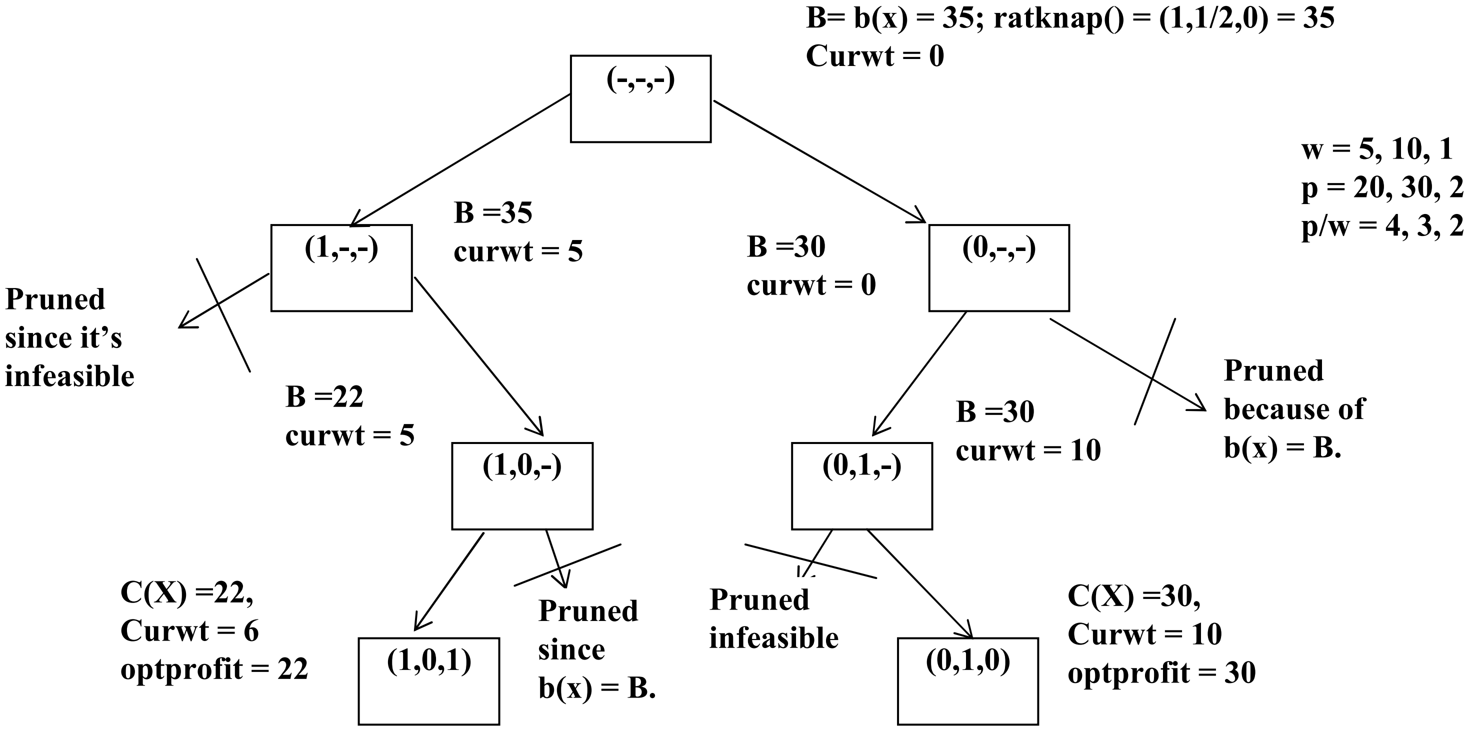
Backtracking Design Technique

void function *knap_2* (int lev, int curwt, int *optprofit, int optx [LEN], int w[LEN], int p[LEN], int n, int knap)

```
{       ...
        else                    //lev ≤ n; before setting x_lev = 1, check feasibility)
        {
                b(x) = ∑_{1≤i≤lev} p_i x_i + ratknap(lev, kcap - curwt)
                if b(x) > optprofit then          // if B ≤ optprofit, do nothing
                {
                   if (curwt + w_lev) ≤ knap then
                   {
                    x_lev  = 1;
                    knap_2(lev+1, curwt + w_lev, &optprofit, optx, w, p, n, knap);
                   }
                if b(x) > optprofit then          // note that optprofit might have
increased since it was last tested
                {
                x_lev  = 0;                       // no check is needed here
                knap_1(lev+1, curwt, &optprofit, optx, w, p, n, knap);
                }}}
```

Backtracking Design Technique

Example: Suppose we three objects (n=3) and w = {5, 10, 1}, p = {20, 30, 2}, and kcap=10.



Backtracking for TSP

- Assume that the problem is to find the minimum cost Hamiltonian circuit starting and ending at vertex 1.
- We can represent a Hamiltonian circuit as a permutation of the integers {2,…,n}, where n is the number of vertices in the graph, which is an (n-1)-tuple.
- Let's denote an (n-1)-tuple by $X = (x_1,\ldots,x_{n-1})$, where we require that $\{x_1,\ldots,x_{n-1}\} = \{2, \ldots, n\}$.
- Then the following algorithm is a basic backtracking method for the TSP.

Backtracking Design Technique

Backtracking for TSP

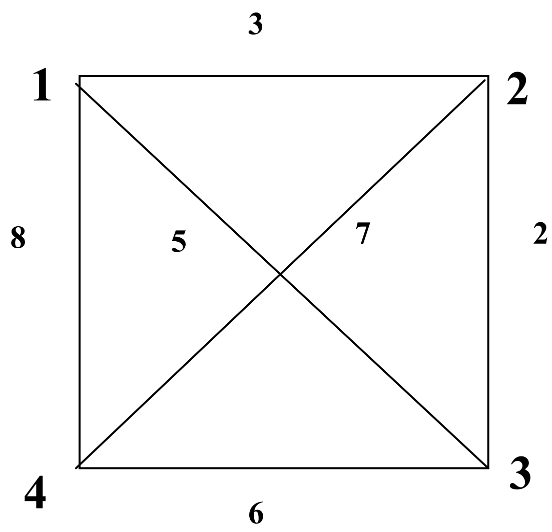
```
Procedure TSP ( lev: integer; var optcost: integer; var optx : arraytype; n:integer)
Begin
        If lev = = n then {        /* we have a Hamiltonian circuit  */
          C = Cost of X;
          If C < optcost then {
                Optcost = C;
                Optx = x;
        }}
        else
          for x_lev = 2 to n do
                if x_lev is distinct from x_1 ,…, x_lev-1 then
                                TSP(lev+1, optcost, optx, n)
end;
```

Backtracking Design Technique

- This algorithm generates the entire state space tree for the problem instance.
- To speed it up, we need a bounding function $B(X) \le C(X)$ (note: we use $\le$ since TSP is a minimization problem, $\ge$ was used for maximization).
- There is no bounding function for TSP that is so obvious as RKP is for BKP.
- A bounding function can be developed by defining a cost matrix for the problem instance as a modified adjacency matrix and then applying the mathematical operation of matrix reduction.
- We now explain the process and an intuitive reason for why it works.
- First let's consider the operation of reduction. A matrix M is said to be reduced if the following properties hold:
  1. all entries in M are non-negative
  2. every row and column of M contains at least one zero element
- M is the cost matrix for a TSP instance defined as follows:
  1. $M(ij)$ = cost of edge ij
  2. $M(ij) = \infty$ if there is no edge or if $i = j$
- Then the sum of the reduction, called $V(M)$, represents a lower bound on the minimum cost Hamiltonian circuit of the original problem.

Backtracking Design Technique

Example TSP:



$$M = \begin{pmatrix} \infty & 3 & 5 & 8 \\ 3 & \infty & 2 & 7 \\ 5 & 2 & \infty & 6 \\ 8 & 7 & 6 & \infty \end{pmatrix}$$

Backtracking Design Technique

The reduction is accomplished by the following steps:

subtract 3 (min of the row) from the first row
subtract 2 from the second row
subtract 2 from the third row
subtract 6 from the fourth row
subtract 1 (min of the first column) from the first column
subtract 0 from the second column
subtract 0 from the third column
subtract 4 from the fourth column

Therefore, V(M) = 18.

For this simple graph, there are only three possible Hamiltonian circuits: the costs are as follows:

1 2 3 4: cost = 3 + 2 + 6 + 8 = 19
1 2 4 3: cost = 3 + 7 + 6 + 5 = 21
1 3 2 4: cost = 5 + 2 + 7 + 8 = 22

Note that V(M) = 18 is less than any of these. It can be proved that in general, V(M) $\leq$ minimum cost Hamiltonian circuit.

Backtracking Design Technique

- For a bounding function, we really want a function that can be computed for partial solutions and that yields a lower bound on the minimum cost of completing the partial solution.
- Suppose we have a partial solution

  $X = (x_1,x_2,\ldots,x_{lev}, -,\ldots,-);\ \ 0 \le lev \le n-1$

which presents a path

  $1\ \ x_1\ \ x_2\ \ldots\ x_{lev};$ all x's distinct

- A completion of X to a Hamiltonian circuit is a path from x_{lev} to node 1 having intermediate vertices in the set $\{2\ldots n\} - \{x_1,x_2,\ldots,x_{lev}\}$.

Define an (n-lev) × (n-lev) matrix M', which is derived from M as follows:

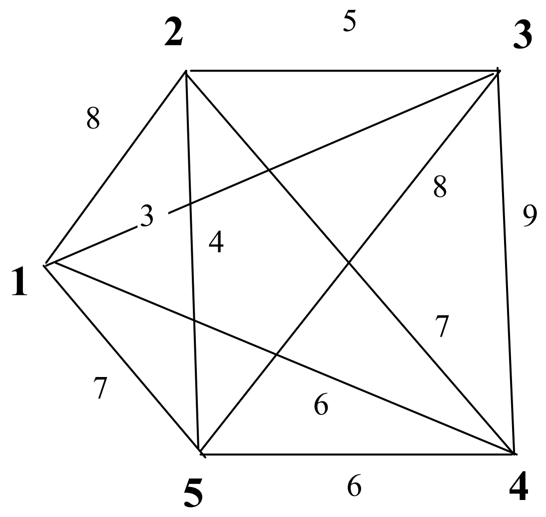
- copy M to M'
- if lev < n-1 then $M'[x_{lev},1] = \infty$
- delete rows 1, x_1, $x_2,\ldots,x_{lev-1}$ from M'
- delete columns x_1, $x_2,\ldots,x_{lev}$ from M'
- Using M', we can define a bounding function for X as follows:

Backtracking Design Technique

```
Procedure TSP ( lev: integer; var optcost: integer; var optx : arraytype;
      n:integer)
begin
      If lev = n then begin  /* we have a Hamiltonian circuit  */
        C = Cost of X;
        If C < optcost then begin
            Optcost = C;
            Optx = x;
        end; end
      else begin
            Compute B = B(X)
            x_lev = 2;
            while (B < optcost) and (x_lev ≤ n) do  begin
                        if x_lev is distinct from x_1, …, x_lev-1 then
                                    TSP(lev+1, optcost, optx,n)
                     x_lev = x_lev + 1
                end; end {else}
end;
```

Backtracking Design Technique

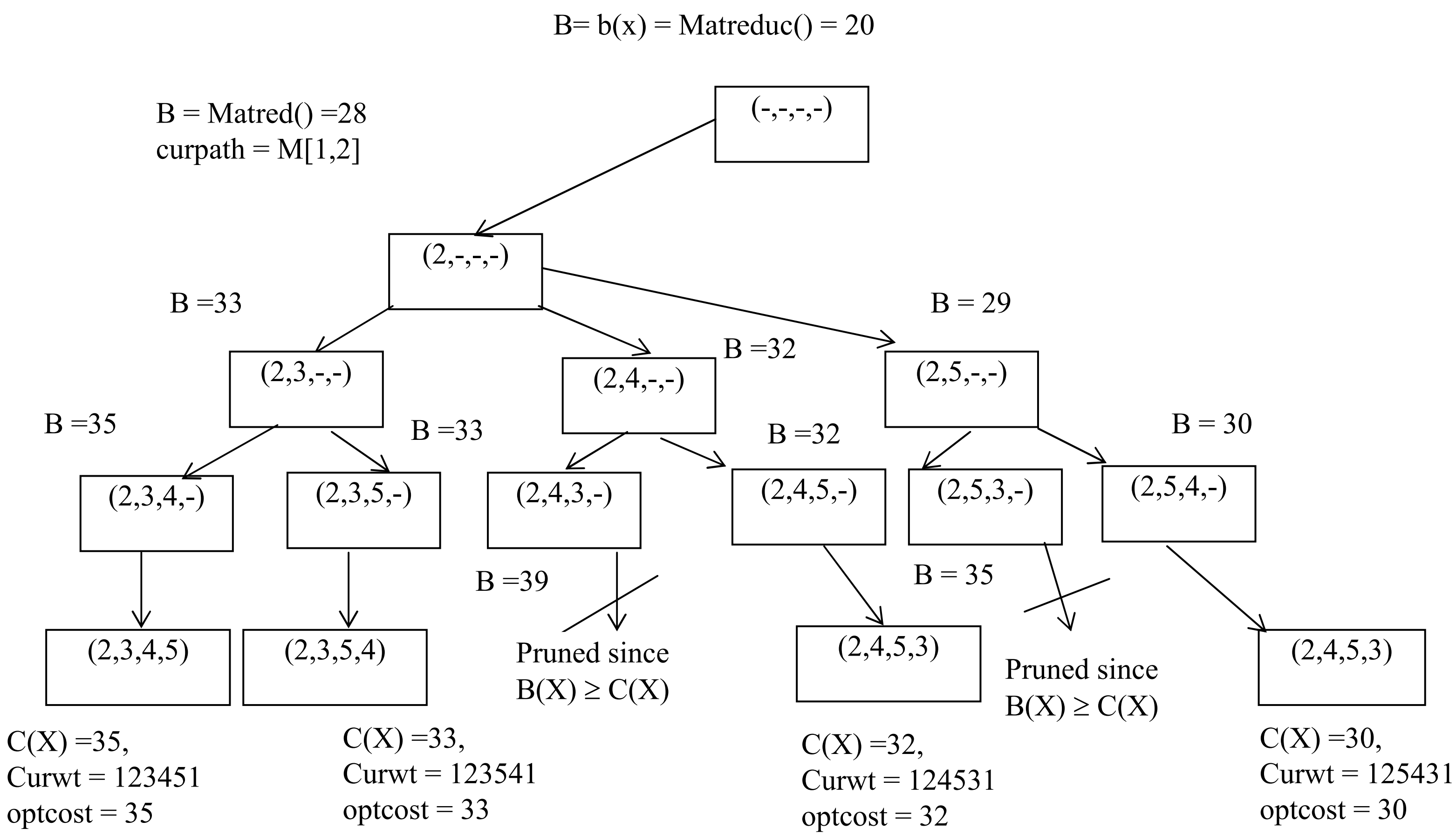
An Example for TSP:



$$M = \begin{pmatrix} \infty & 8 & 3 & 6 & 7 \\ 8 & \infty & 5 & 7 & 4 \\ 3 & 5 & \infty & 9 & 8 \\ 6 & 7 & 9 & \infty & 6 \\ 7 & 4 & 8 & 6 & \infty \end{pmatrix}$$

$$B(X) = V(M'(X)) + M[1,x_1] + M[x_1,x_2] + \ldots + M[x_{lev-1},x_{lev}]$$

Backtracking Design Technique



Backtracking Design Technique

$x_{lev} = 2$

$$\begin{matrix} \infty & 5 & 7 & 4 \\ 3 & \infty & 9 & 8 \\ 6 & 9 & \infty & 6 \\ 7 & 8 & 6 & \infty \end{matrix}$$

$V(M'(X)) = 4 + 3 + 6 + 6 + 1 = 20 \rightarrow$
$B(X) = V(M'(X)) + M[1(x_{lev-1}), 2(x_{lev})]$
$= 20 + 8 = 28$

$$\begin{array}{cc} & \begin{matrix} \quad & x_{lev} & & & \end{matrix} \\ \begin{matrix} 1 \\ \\ \\ \\ \\ \end{matrix} & \begin{pmatrix} \infty & 8 & 3 & 6 & 7 \\ 8 & \infty & 5 & 7 & 4 \\ 3 & 5 & \infty & 9 & 8 \\ 6 & 7 & 9 & \infty & 6 \\ 7 & 4 & 8 & 6 & \infty \end{pmatrix} \end{array}$$

$x_{lev} = 3$

$$\begin{matrix} \infty & 9 & 8 \\ 6 & \infty & 6 \\ 7 & 6 & \infty \end{matrix}$$

$V(M'(X)) = 8 + 6 + 6 = 20 \rightarrow$
$B(X) = V(M'(X)) + M[1, 2(x_1)] +$
$M[2(x_{lev-1}), 3(x_{lev})] = 20+8+5 = 33$

$$\begin{array}{cc} & \begin{matrix} \quad & x_{lev-1} & x_{lev} & & \end{matrix} \\ \begin{matrix} 1 \\ x_{lev-1} \\ \\ \\ \\ \end{matrix} & \begin{pmatrix} \infty & 8 & 3 & 6 & 7 \\ 8 & \infty & 5 & 7 & 4 \\ 3 & 5 & \infty & 9 & 8 \\ 6 & 7 & 9 & \infty & 6 \\ 7 & 4 & 8 & 6 & \infty \end{pmatrix} \end{array}$$

Backtracking Design Technique

$$
\begin{array}{cc}
 & \begin{array}{ccccc} & & & x_{lev} & \end{array} \\
\begin{array}{c} 1 \\ \\ \\ x_{lev-1} \\ \\ \end{array} & \begin{pmatrix} \infty & 8 & 3 & 6 & 7 \\ 8 & \infty & 5 & 7 & 4 \\ 3 & 5 & \infty & 9 & 8 \\ 6 & 7 & 9 & \infty & 6 \\ 7 & 4 & 8 & 6 & \infty \end{pmatrix}
\end{array}
$$

$x_{lev} = 4$

$$\begin{array}{cc} \infty & 6 \\ 7 & \infty \end{array}$$

$V(M'(X)) = 6+7 = 13 \rightarrow B(X) = V(M'(X))$
$+ M[1, 2(x_1)] + M[2(x_2), 3(x_3)] +$
$M[3(x_{lev-1}), 4(x_{lev})] = 13+8+5+9 = 35$

Optcost = M[1, 2] + M[2, 3] + M[3, 4] + M[4, 5] M[5, 1] = 8+5+9+6+7 = 35

$$
\begin{array}{cc}
 & \begin{array}{ccccc} & & & & x_{lev} \end{array} \\
\begin{array}{c} 1 \\ \\ \\ x_{lev-1} \\ \\ \end{array} & \begin{pmatrix} \infty & 8 & 3 & 6 & 7 \\ 8 & \infty & 5 & 7 & 4 \\ 3 & 5 & \infty & 9 & 8 \\ 6 & 7 & 9 & \infty & 6 \\ 7 & 4 & 8 & 6 & \infty \end{pmatrix}
\end{array}
$$

$x_{lev} = 5$

$$\begin{array}{cc} 6 & \infty \\ \infty & 6 \end{array}$$

$V(M'(X)) = 6+6 = 12 \rightarrow B(X) =$
$V(M'(X)) + M[1, 2(x_1)] + M[2(x_2), 3(x_3)]$
$+ M[3(x_{lev-1}), 5(x_{lev})] = 12+8+5+8 = 33$

C(X)= Optcost = M[1, 2] + M[2, 3] + M[3, 5] + M[5, 4] M[4, 1] = 8+5+8+6+6 = 33

Backtracking Design Technique

$$\begin{array}{c} \\ \begin{matrix} 3 & \infty & 8 \\ \infty & 9 & 6 \\ 7 & 8 & \infty \end{matrix} \end{array}$$

V(M'(X)) = 3+6+7+1= 17→
B(X) = V(M'(X)) + M[1, 2 (x_1)] +
M[2(x_{lev-1}), 4(x_{lev})] = 17+8+7 = 32

$$\begin{array}{cc} & \begin{matrix} & & & x_{lev} & \end{matrix} \\ \begin{matrix} 1 \\ x_{lev-1} \\ \\ \\ \\ \end{matrix} & \begin{pmatrix} \infty & 8 & 3 & 6 & 7 \\ 8 & \infty & 5 & 7 & 4 \\ 3 & 5 & \infty & 9 & 8 \\ 6 & 7 & 9 & \infty & 6 \\ 7 & 4 & 8 & 6 & \infty \end{pmatrix} \end{array}$$

$x_{lev} = 3$

$$\begin{matrix} \infty & 8 \\ 7 & \infty \end{matrix}$$

V(M'(X)) = 8+7 = 15 → B(X) =
V(M'(X)) + M[1, 2(x_1)] + M[2(x_2), 4(x_3)] + M[4(x_{lev-1}), 3(x_{lev})]
= 15+8+7+9 = 39

C(X)= Optcost = M[1, 2] + M[2, 4] + M[4, 3] + M[3, 5] + M[5, 1] = 8+7+9+8+7 = 39

Backtracking Design Technique

$x_{lev} = 5$

$$\begin{matrix} 3 & \infty \\ \infty & 8 \end{matrix}$$

V(M'(X)) = 3+8= 11 → B(X) =
V(M'(X)) + M[1, 2(x_1)] + M[2(x_2), 4(x_3)] + M[4(x_{lev-1}), 5(x_{lev})]
= 11+8+7+6 = 32

C(X)= Optcost = M[1, 2] + M[2, 4] + M[4, 5] + M[5, 3] + M[3, 1] = 8+7+6+8+3 = 32

$x_{lev} = 5$

$$\begin{matrix} 3 & \infty & 9 \\ 6 & 9 & \infty \\ \infty & 8 & 6 \end{matrix}$$

V(M'(X)) = 3+6+6+2= 17 →
B(X) = V(M'(X)) + M[1, 2 (x_1)] +
M[2(x_{lev-1}), 5(x_{lev})] = 17+8+4 = 29

$$\begin{array}{c|c} & \begin{matrix} & & & & x_{lev} \end{matrix} \\ \begin{matrix} 1 \\ x_{lev-1} \\ \\ \\ \\ \end{matrix} & \begin{pmatrix} \infty & 8 & 3 & 6 & 7 \\ 8 & \infty & 5 & 7 & 4 \\ 3 & 5 & \infty & 9 & 8 \\ 6 & 7 & 9 & \infty & 6 \\ 7 & 4 & 8 & 6 & \infty \end{pmatrix} \end{array}$$

Backtracking Design Technique

$x_{lev} = 3$

$$\begin{matrix} \infty & 9 \\ 6 & \infty \end{matrix}$$

$V(M'(X)) = 9+6 = 15 \rightarrow$

$B(X) = V(M'(X))+M[1, 2(x_1)]+M[2(x_2),5(x_3)]+M[5(x_{lev-1}),3(x_{lev})]$

$= 15+8+4+8 = 35$

$x_{lev} = 4$

$$\begin{matrix} 3 & \infty \\ \infty & 9 \end{matrix}$$

$V(M'(X)) = 3+9 = 12 \rightarrow$

$B(X) = V(M'(X))+M[1,2(x_1)]+M[2(x_2),5(x_3)]+M[5(x_{lev-1}),4(x_{lev})] = 12+8+4+6 = 30$

$x_{lev} = 3$

$$\begin{matrix} 8 & \infty & 7 & 4 \\ \infty & 5 & 9 & 8 \\ 6 & 7 & \infty & 6 \\ 7 & 4 & 6 & \infty \end{matrix}$$

$V(M'(X)) = 4+5+6+4+2 = 21 \rightarrow$

$B(X) = V(M'(X)) + M[1(x_{lev-1}), 3(x_{lev})]$

$= 21 + 3 = 24$

$$\begin{array}{cc} & \begin{array}{ccccc} & & x_{lev} & & \end{array} \\ \begin{array}{c} 1 \\ \\ \\ \\ \\ \end{array} & \left(\begin{array}{ccc|cc} \cancel{\infty} & \cancel{8} & \cancel{3} & \cancel{6} & \cancel{7} \\ 8 & \infty & 5 & 7 & 4 \\ 3 & 5 & \infty & 9 & 8 \\ 6 & 7 & 9 & \infty & 6 \\ 7 & 4 & 8 & 6 & \infty \end{array}\right) \end{array}$$

Backtracking Design Technique

$x_{lev} = 2$

$$\begin{matrix} \infty & 7 & 4 \\ 6 & \infty & 8 \\ 7 & 6 & \infty \end{matrix}$$

$V(M'(X)) = 4+6+6 = 16 \rightarrow$
$B(X) = V(M'(X)) + M[1, 3(x_1)] + M[3(x_{lev-1}), 2(x_{lev})] = 16+3+5 = 24$

$$\begin{array}{r|c} & \begin{matrix} & x_{lev} & & & \end{matrix} \\ \begin{matrix} 1 \\ \\ x_{lev-1} \\ \\ \\ \end{matrix} & \begin{pmatrix} \infty & 8 & 3 & 6 & 7 \\ 8 & \infty & 5 & 7 & 4 \\ 3 & 5 & \infty & 9 & 8 \\ 6 & 7 & 9 & \infty & 6 \\ 7 & 4 & 8 & 6 & \infty \end{pmatrix} \end{array}$$

$x_{lev} = 4$

$$\begin{matrix} \infty & 6 \\ 7 & \infty \end{matrix}$$

$V(M'(X)) = 6+7 = 13 \rightarrow$
$B(X) = V(M'(X))+M[1,3(x_1)]+M[3(x_2),2(x_3)]+M[2(x_{lev-1}),4(x_{lev})]$
$= 13+3+5+7 = 28$

C(X)= Optcost = M[1, 3] + M[3, 2] + M[2, 4] + M[4, 5] + M[5, 1] = 3+5+7+6+7 = 28.

$x_{lev} = 5$

$$\begin{matrix} 6 & \infty \\ \infty & 6 \end{matrix}$$

$V(M'(X)) = 6+6 = 12 \rightarrow B(X) =$
$V(M'(X)) + M[1, 3(x_1)] + M[3(x_2), 2(x_3)] + M[2(x_{lev-1}), 5(x_{lev})]$
$= 12+3+5+4 = 24$

C(X)= Optcost = M[1, 3] + M[3, 2] + M[2, 5] + M[5, 4] + M[4, 1] = 3+5+4+6+6 = 24

Backtracking Design Technique

- on the minimum cost Hamiltonian circuit of the original problem.

$x_{lev} = 4$

$$\begin{matrix} 8 & \infty & 4 \\ \infty & 7 & 6 \\ 7 & 4 & \infty \end{matrix}$$

$V(M'(X)) = 4+6+4+3 = 17 \rightarrow$
$B(X) = V(M'(X)) + M[1, 3(x_1)] + M[3(x_{lev-1}), 4(x_{lev})] = 17+3+9 = 29$

$$\begin{array}{r c} & \begin{matrix} & & & x_{lev} & \end{matrix} \\ \begin{matrix} 1 \\ \\ x_{lev-1} \\ \\ \\ \end{matrix} & \begin{pmatrix} \infty & 8 & 3 & 6 & 7 \\ 8 & \infty & 5 & 7 & 4 \\ 3 & 5 & \infty & 9 & 8 \\ 6 & 7 & 9 & \infty & 6 \\ 7 & 4 & 8 & 6 & \infty \end{pmatrix} \end{array}$$

$x_{lev} = 5$

$$\begin{matrix} 8 & \infty & 7 \\ 6 & 7 & \infty \\ \infty & 4 & 7 \end{matrix}$$

$V(M'(X)) = 7+6+4 = 17 \rightarrow$
$B(X) = V(M'(X)) + M[1, 3(x_1)] + M[3(x_{lev-1}), 5(x_{lev})] = 17+3+8 = 28$

$$\begin{array}{r c} & \begin{matrix} & & & & x_{lev} \end{matrix} \\ \begin{matrix} 1 \\ \\ x_{lev-1} \\ \\ \\ \end{matrix} & \begin{pmatrix} \infty & 8 & 3 & 6 & 7 \\ 8 & \infty & 5 & 7 & 4 \\ 3 & 5 & \infty & 9 & 8 \\ 6 & 7 & 9 & \infty & 6 \\ 7 & 4 & 8 & 6 & \infty \end{pmatrix} \end{array}$$

Backtracking Design Technique

$x_{lev} = 4$

$$\begin{matrix} 8 & \infty & 5 & 4 \\ 3 & 5 & \infty & 8 \\ \infty & 7 & 9 & 6 \\ 7 & 4 & 8 & \infty \end{matrix}$$

$V(M'(X)) = 4+3+6+4+1 = 18 \rightarrow$
$B(X) = V(M'(X)) + M[1(x_{lev-1}), 4(x_{lev})]$
$= 18 + 6 = 24$

$$\begin{array}{cc} & \begin{matrix} & & & x_{lev} & \end{matrix} \\ \begin{matrix} 1 \\ \\ \\ \\ \\ \end{matrix} & \begin{pmatrix} \infty & 8 & 3 & 6 & 7 \\ 8 & \infty & 5 & 7 & 4 \\ 3 & 5 & \infty & 9 & 8 \\ 6 & 7 & 9 & \infty & 6 \\ 7 & 4 & 8 & 6 & \infty \end{pmatrix} \end{array}$$

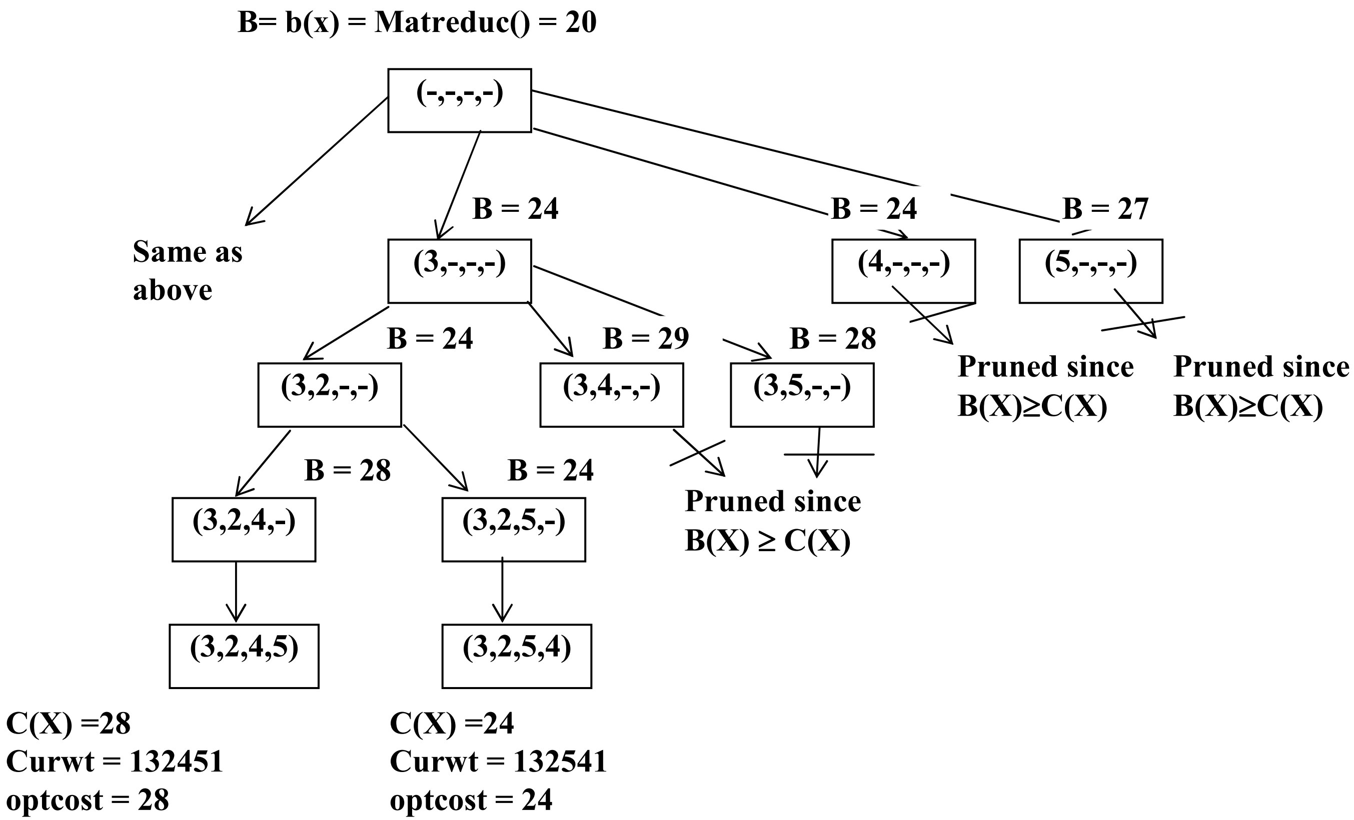
$x_{lev} = 5$

$$\begin{matrix} 8 & \infty & 5 & 7 \\ 3 & 5 & \infty & 9 \\ 6 & 7 & 9 & \infty \\ \infty & 4 & 8 & 6 \end{matrix}$$

$V(M'(X)) = 5+3+6+4+2 = 20 \rightarrow$
$B(X) = V(M'(X)) + M[1(x_{lev-1}), 5(x_{lev})]$
$= 20 + 7 = 27$

$$\begin{array}{cc} & \begin{matrix} & & & & x_{lev} \end{matrix} \\ \begin{matrix} 1 \\ \\ \\ \\ \\ \end{matrix} & \begin{pmatrix} \infty & 8 & 3 & 6 & 7 \\ 8 & \infty & 5 & 7 & 4 \\ 3 & 5 & \infty & 9 & 8 \\ 6 & 7 & 9 & \infty & 6 \\ 7 & 4 & 8 & 6 & \infty \end{pmatrix} \end{array}$$

Backtracking Design Technique



Branch and Bound Design Technique

Input: -function $D_k(x_1, \ldots, x_{k-1})$ determining state space tree T associated with the given problem and decision set D_k,
- bounding function *Bounded*

Output: all goals to the given problem

LiveNodes is initialized to be empty
Call AllocateTreeNodes(Root)
Root → parent := nil
Call Add(LiveNodes, Root) // add root to list of live nodes
While LiveNodes is not empty do
    Call Select(liveNodes,E-node,k) // select E-node from live nodes
    For each $X[k] \in D_k$ (E-node) *do* // for each child of the E-node do
        Call AllocateTreeNode(child)
        Child → info := X[k]
        Child → parent := E-node
        If answer (child) then // if child is a goal node then
            Call Path (child) // output path from child to root
    If .not. Bounded (Child) then
        Call Add(LiveNodes Child) // add child to list of

Branch and Bound Design Technique

• Immediately upon expanding the current E-node, this E-node becomes a dead node and a new E-node is selected from LiveNodes.

• Thus B&B is quite different from backtracking, where we might backtrack to a given node many times, making it the E-node each time all its children have finally been generated or algorithm terminates.

• The nodes of the state space tree at any given point in a B&B algorithm are therefore in one of the following four states: *E-node, live* node*, dead* node*,* or *yet generated*.

• As with backtracking, the efficiency of B&B depends on the utilization of good bounding functions. Such functions are used in the attempt to determine solutions by restricting attention to small portions of the entire state space tree.

• When expanding a given E-node, a child can be bounded if it can be shown that it cannot lead to a goal node.

• We illustrate B&B by revisiting Travel Salesman Problem (*TSP*), where the data structure *LiveNodes* is a queue. Such a B&B, called FIFO B&B, involves performing a breadth-first search of the state space tree. Initially the queue of live nodes is empty.

• The algorithm begins by generating the root node of the state space tree and enqueueing it in the queue *LiveNodes*. At each stage of the algorithm a node is dequeued from *LiveNodes* to become the new E-node. All the children of the *E-node* are then generated.

• The children that are not bounded are enqueued (as they are generated

Branch and Bound Design Technique

```
Procedure TSP ( lev: integer; var optcost: integer; var optx : arraytype; n:integer)
Var B,C,Count: integer; NextCoord: array [1..n] of 2..n; NextB: array [1..n] of integer
begin
      If lev = n then  begin  /* we have a Hamiltonian circuit  */
         C = Cost of X;
      If C < optcost then begin
            Optcost = C; Optx = x;
        end; end
      else begin
            Count = 0;
            For x_lev = 2 to n do
            If x_lev-1 is distinct from x_1, x_2, …, x_lev-1 then Begin
                        Count = Count + 1; NextCoord[Count] = x_lev ;
                        NextB[Count] = B(x);              /* you compute   */
            End;   /* NextCoord and nextB are n-lev arrays  */
            Sort NextCoord and NextB ascending order
            Count = 1;
            while (Count ≤  n-lev) and (NextB[Count] ≤ optcost) do begin
                        if xlev is distinct from x_1 ,…, x_lev-1 then
                         x_lev = NextCoord [Count]
                        TSP(lev+1, optcost, optx,n)
                    Count = count + 1
               end; end {else}; end;
```

Branch and Bound Design Technique

Procedure TSP

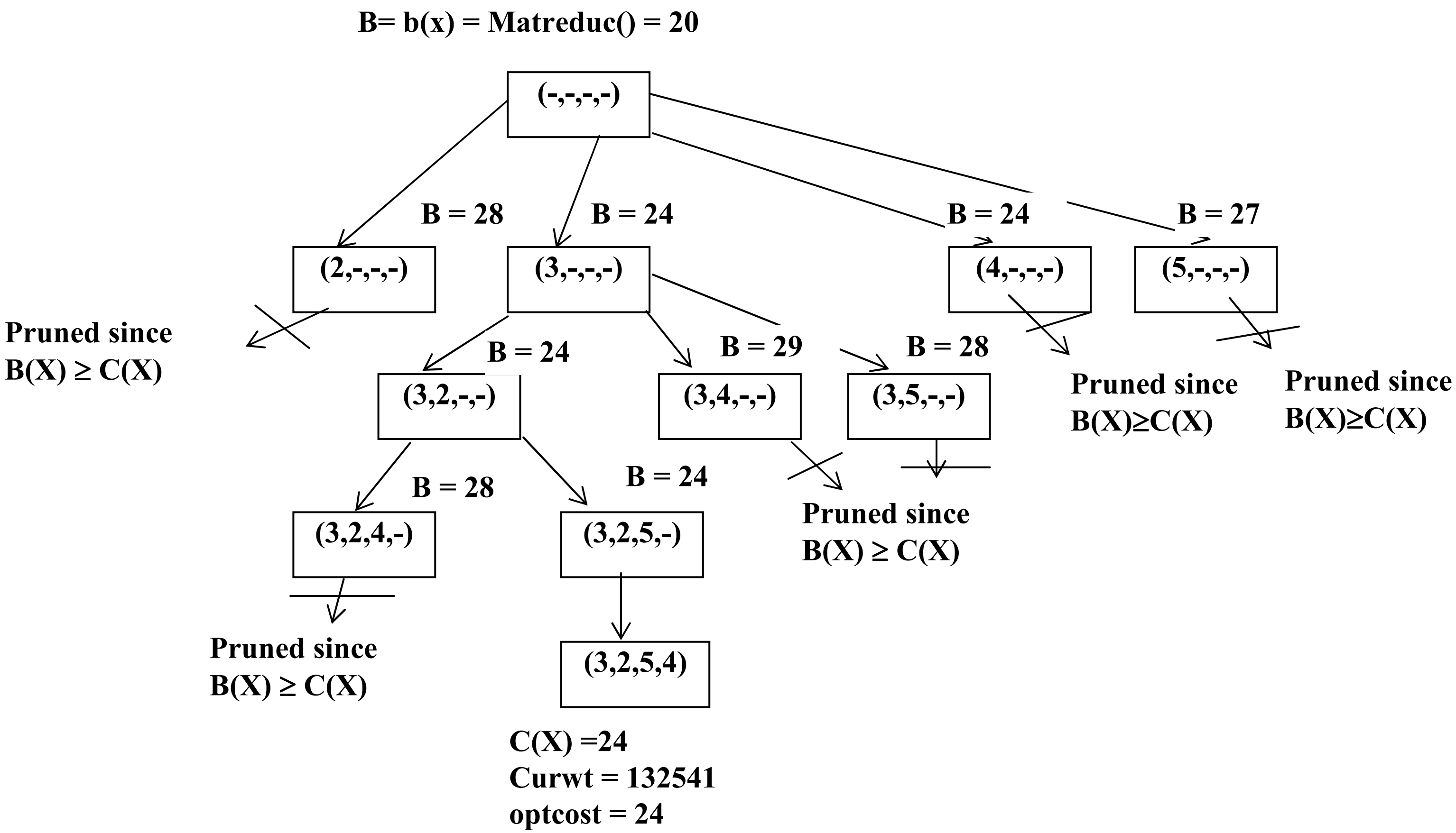


Figure: The State Space Tree for the Branch and Bound Approach

Branch and Bound Design Technique

$x_{lev} = 2$

$$\begin{matrix} \infty & 5 & 7 & 4 \\ 3 & \infty & 9 & 8 \\ 6 & 9 & \infty & 6 \\ 7 & 8 & 6 & \infty \end{matrix}$$

$x_{lev} = 3$

$$\begin{matrix} 8 & \infty & 7 & 4 \\ 3 & 5 & 9 & 8 \\ 6 & 7 & \infty & 6 \\ 7 & 4 & 6 & \infty \end{matrix}$$

$x_{lev} = 4$

$$\begin{matrix} 8 & \infty & 5 & 4 \\ 3 & 5 & \infty & 8 \\ 6 & 7 & 9 & 6 \\ 7 & 4 & 8 & \infty \end{matrix}$$

$x_{lev} = 5$

$$\begin{matrix} 8 & \infty & 5 & 7 \\ 3 & 5 & \infty & 9 \\ 6 & 7 & 9 & \infty \\ 7 & 4 & 8 & 6 \end{matrix}$$

V(M’(X)) = 4+3+6+6+1 = 20 → B(X) = V(M’(X)) + M[1(xlev-1), 2(xlev)]= 20 + 8 = 28
V(M’(X)) = 4+5+6+4+2 = 21 → B(X) = V(M’(X)) + M[1(xlev-1), 3(xlev)] = 21 + 3 = 24
V(M’(X)) = 4+3+6+4+1 = 18 → B(X) = V(M’(X)) + M[1(xlev-1), 4(xlev)] = 18 +3 = 24
V(M’(X)) = 7+5+3+6+4+2 = 20 → B(X) = V(M’(X))+M[1(xlev-1),5(xlev)] = 20+7 = 27

Branch and Bound Design Technique

$x_{lev} = 2$

$$\begin{matrix} \infty & 7 & 4 \\ 6 & \infty & 6 \\ 7 & 6 & \infty \end{matrix}$$

$V(M'(X)) = 4+6+6 = 16 \rightarrow$
$B(X) = V(M'(X)) + M[1, 3(x_1)] + M[3(x_{lev-1}), 2(x_{lev})] = 16+3+5 = 24$

$$\begin{array}{c|c} & \begin{matrix} \quad & x_{lev} & \quad & \quad & \quad \end{matrix} \\ \begin{matrix} 1 \\ x_{lev-1} \\ \\ \\ \\ \end{matrix} & \begin{pmatrix} \infty & 8 & 3 & 6 & 7 \\ 8 & \infty & 5 & 7 & 4 \\ 3 & 5 & \infty & 9 & 8 \\ 6 & 7 & 9 & \infty & 6 \\ 7 & 4 & 8 & 6 & \infty \end{pmatrix} \end{array}$$

$x_{lev} = 4$

$$\begin{matrix} 8 & \infty & 4 \\ \infty & 7 & 6 \\ 7 & 4 & \infty \end{matrix}$$

$V(M'(X)) = 4+6+4+3 = 17 \rightarrow$
$B(X) = V(M'(X)) + M[1, 3(x_1)] + M[3(x_{lev-1}), 4(x_{lev})] = 17+3+9 = 29$

$$\begin{array}{c|c} & \begin{matrix} \quad & \quad & \quad & x_{lev} & \quad \end{matrix} \\ \begin{matrix} 1 \\ \\ x_{lev-1} \\ \\ \\ \end{matrix} & \begin{pmatrix} \infty & 8 & 3 & 6 & 7 \\ 8 & \infty & 5 & 7 & 4 \\ 3 & 5 & \infty & 9 & 8 \\ 6 & 7 & 9 & \infty & 6 \\ 7 & 4 & 8 & 6 & \infty \end{pmatrix} \end{array}$$

$x_{lev} = 5$

$$\begin{matrix} 8 & \infty & 7 \\ 6 & 7 & \infty \\ \infty & 4 & 7 \end{matrix}$$

$V(M'(X)) = 7+6+4 = 17 \rightarrow$
$B(X) = V(M'(X)) + M[1, 3(x_1)] + M[3(x_{lev-1}), 5(x_{lev})] = 17+3+8 = 28$

$$\begin{array}{c|c} & \begin{matrix} \quad & \quad & \quad & \quad & x_{lev} \end{matrix} \\ \begin{matrix} 1 \\ \\ x_{lev-1} \\ \\ \\ \end{matrix} & \begin{pmatrix} \infty & 8 & 3 & 6 & 7 \\ 8 & \infty & 5 & 7 & 4 \\ 3 & 5 & \infty & 9 & 8 \\ 6 & 7 & 9 & \infty & 6 \\ 7 & 4 & 8 & 6 & \infty \end{pmatrix} \end{array}$$

Branch and Bound Design Technique

$$\begin{array}{c|ccccc} & & 2 & 3 & & x_{lev} \\ \hline 1 & \infty & 8 & 3 & 6 & 7 \\ 2 & 8 & \infty & 5 & 7 & 4 \\ x_{lev-1} & 3 & 5 & \infty & 9 & 8 \\ & 6 & 7 & 9 & \infty & 6 \\ & 7 & 4 & 8 & 6 & \infty \end{array} \qquad \mathbf{x_{lev} = 5} \quad \begin{matrix} 6 & \infty \\ \infty & 6 \end{matrix}$$

V(M'(X)) = 6+6 = 12 →
$B(X) = V(M'(X)) + M[1, 3(x_1)] + M[3(x_2), 2(x_3)] + M[2(x_{lev-1}), 5(x_{lev})]$
= 12+3+5+4 = 24

$$\begin{array}{c|ccccc} & & 2 & 3 & x_{lev} & \\ \hline 1 & \infty & 8 & 3 & 6 & 7 \\ 2 & 8 & \infty & 5 & 7 & 4 \\ x_{lev-1} & 3 & 5 & \infty & 9 & 8 \\ & 6 & 7 & 9 & \infty & 6 \\ & 7 & 4 & 8 & 6 & \infty \end{array} \qquad \mathbf{x_{lev} = 4} \quad \begin{matrix} \infty & 6 \\ 7 & \infty \end{matrix}$$

V(M'(X)) = 6+7 = 13 →
$B(X) = V(M'(X))+M[1,3(x_1)]+M[3(x_2),2(x_3)]+M[2(x_{lev-1}),4(x_{lev})]$
= 13+3+5+7 = 28

C(X)= Optcost = M[1, 3] + M[3, 2] + M[2, 5] + M[5, 4] + M[4, 1] = 3+5+4+6+6 = 24
In general, branch&bound is the method of choice for TSP, although it does not always work this well.

Brach and Bound Design Technique

0-1 Knapsack Example:

Suppose we three objects (n=3) and w = {5, 10, 1}, p = {20,30, 2}, and kcap=10.