UNIT-5

Branch and Bound

General Method, applications - Travelling sales person problem, 0/1 Knapsack problem - LC Branch and Bound Solution, FIFO Branch and Bound Solution.

NP-Hard and NP-Complete problems: Basic concepts, non-deterministic algorithms, NP-Hard and NP-Complete classes, Cook's theorem.

Branch and Bound Algorithm - General Method

The Branch and Bound Algorithm is a method used in Combinatorial optimization problems to Systematically search for the best solution. Branch and Bound is Commonly used in problems like the travelling Sales man and job scheduling.

* It involves breaking the problem into smaller Sub problems (branching) and using bounds to eliminate sub problems that cannot Contain the optimal solution (bounding). This approach explores a state-space tree where nodes represent sub-problems, and pruning is done to avoid exhaustive search, thus improving efficiency.

General Method:

1. Branching: Split the problem into smaller Subproblems that represent subsets of the solution Space
2. Bounding: Calculate an upper or lower bound on the objective value for each subproblem.
3. Pruning: Discard subproblems whose bound indicate they cannot Contain a better solution that already found.
4. Selection: Explore the remaining subproblems Systematically until all possibilities are evaluated or pruned.

Applications of branch and bound:

The branch and bound method is applied mainly to solve Combinatorial optimization problems optimally and efficiently by reducing the search space. Key applications in DAA include

- Travelling Salesman problem / Traveling sales person problem (TSP):
  optimally finding the shortest route visiting all cities using systematic State-space tree search and pruning.
- 0/1 Knapsack problem: Selecting items to maximize value without exceeding capacity by branching on inclusion/exclusion and bounding via fractional Knapsack relaxations.
- Job Scheduling: Assigning jobs to resources in ways to optimize makespan or minimize cost using branching on job assignments and bounding partial schedules.
- Integer programming: solving discrete optimization problems by branching on integer variable values and using bounds from linear relaxations for pruning.
- Constraint Satisfaction problems:
  Efficient exploration of search space by pruning infeasible partial solutions.
- Resource Allocation problems:
  optimal distributing limited resources among Competing tasks or agents with bounding functions improving search efficiency.

(*) Traveling sales person problem:

The Traveling salesman problem (TSP) using the Branch and Bound Technique is a combinatorial optimization problem where the goal is to find the shortest possible route that visits a set of cities exactly once and returns to the starting city.

The Branch and Bound algorithm solves this by systematically exploring potential routes in a search tree, while pruning suboptimal paths using a lower bound estimate on the route cost to avoid unnecessary computations.

Key points:-

(*) The algorithm starts at the root node representing the initial city.

(*) At each node in the search tree, it calculates a lower bound on the minimum possible total cost to complete the route from that node.

(*) If the calculated bound exceeds the cost of the best known solution, that branch is pruned (ignored).

(*) otherwise, the algorithm recursively explores deeper nodes (routes).

(*) The cost bound is computed by considering minimum costs of leaving and entering each city to estimate a least possible completion cost.

(*) The best solution found through the search is the shortest route for the salesman

This approach Significantly reduces the search Space Compared to brute-force enumeration, making it more efficient though still computationally expensive for very large number of cities.

Travelling sales person problem

Sales man-rules

→ visit each city atleast once
→ do not visit any city more than once
→ Starting city and ending city same.

4-Cities – A, B, C, D

① Row reduction matrix
② Column reduction matrix

$$\begin{array}{cc|cccc|c} & & A & B & C & D & \\ R_1 & A & \infty & 10 & 5 & 3 & (3) \\ R_2 & B & 8 & \infty & 9 & 7 & (7) \\ R_3 & C & 1 & 6 & \infty & 9 & (1) \\ R_4 & D & 2 & 3 & 8 & \infty & (2) \\ & & & & & & 13 \end{array}$$

$$\begin{array}{cccc|l} C_1 & C_2 & C_3 & C_4 & \\ \infty & 7 & 2 & 0 & \\ 1 & \infty & 2 & 0 & \\ 0 & 5 & \infty & 8 & \\ 0 & 1 & 6 & \infty & \\ 0 & 1 & 2 & 0 & - 3 \end{array}$$

$$\begin{bmatrix} \infty & 6 & 0 & 0 \\ 1 & \infty & 0 & 0 \\ 0 & 4 & \infty & 8 \\ 0 & 0 & 4 & 0 \end{bmatrix} \Rightarrow M_1$$

Total reduction = 13+3 = 16

A→B A row → ∞, B col → ∞, B→A as ∞,
Remaining values from M_1

$$\begin{array}{c|cccc|} & A & B & C & D \\ A & \infty & \infty & \infty & \infty \\ B & \infty & \infty & 0 & 0 \\ C & 0 & \infty & \infty & 8 \\ D & 0 & \infty & 4 & 0 \end{array}$$

All rows / cols → ∞
or
atleast one row/col → 0

Reduction = 0

Total cost → Parent matrix Cost + Reduction + $(A \to B)$ from M_1
= 16 + 0 + 6 = 22

$A \to C$ A row $\to \infty$, C col $\to \infty$, $C \to A - \infty$, Remaining all from M_1

$$\begin{array}{c|cccc} & A & B & C & D \\ \hline A & \infty & \infty & \infty & \infty \\ B & 1 & \infty & \infty & 0 \\ C & \infty & 4 & \infty & 8 \\ D & 0 & 0 & \infty & 0 \end{array} (4) \to \begin{bmatrix} \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & 0 \\ \infty & 0 & \infty & 4 \\ 0 & 0 & \infty & 0 \end{bmatrix}$$

Reduction $= 4$

Total cost $= 16 + 4 + 0 = 20$

$A \to D$ A row $\to \infty$, D col $- \infty$, $D \to A - \infty$ Remaining all from M_1

$$\begin{bmatrix} \infty \\ \infty \\ \infty \\ \\ \end{bmatrix} \begin{array}{c|cccc} & A & B & C & D \\ \hline A & \infty & \infty & \infty & \infty \\ B & 1 & \infty & 0 & \infty \\ C & 0 & 4 & \infty & \infty \\ D & \infty & 0 & 4 & \infty \end{array} \Rightarrow M_2$$

Red $= 0$

parent matrix $= 16$

Total cost $= 16 + 0 + 0 = 16$

$(A \to D) \to B$ A row, D row $\to \infty$, B col $\to \infty$, $B - D \to \infty$, $B - A \to \infty$

$$\begin{array}{c|cccc} & A & B & C & D \\ \hline A & \infty & \infty & \infty & \infty \\ B & \infty & \infty & 0 & \infty \\ C & 0 & \infty & \infty & \infty \\ D & \infty & \infty & \infty & \infty \end{array} \Rightarrow M_3 \text{ (parent matrix)}$$

Remaining all from M2

Reduction $= 0$

Total cost $= 16 + 0 + \boxed{0} = 16$ $D \to B$.

$(A \to D) \to C$

$$\begin{array}{c|cccc} & A & B & C & D \\ \hline A & \infty & \infty & \infty & \infty \\ B & 1 & \infty & \infty & \infty \\ C & \infty & 4 & \infty & \infty \\ D & \infty & \infty & \infty & \infty \end{array} \begin{matrix} \\ (1) \\ (4) \\ \\ \end{matrix} \Rightarrow \begin{array}{c|cccc} & A & B & C & D \\ \hline A & \infty & \infty & \infty & \infty \\ B & 0 & \infty & \infty & \infty \\ C & \infty & 0 & \infty & \infty \\ D & \infty & \infty & \infty & \infty \end{array}$$

Reduction $= 5$

$T.C = 16 + 5 + 4 = 25$

$(A-D-B) \rightarrow C$

A, B, D row as ∞, C col as ∞, $(C-B, C-A, C-D) \rightarrow \infty$,

Remaining values from M3

$$\begin{array}{c|cccc} & A & B & C & D \\ \hline A & \infty & \infty & \infty & \infty \\ B & \infty & \infty & \infty & \infty \\ C & \infty & \infty & \infty & \infty \\ D & \infty & \infty & \infty & \infty \end{array}$$

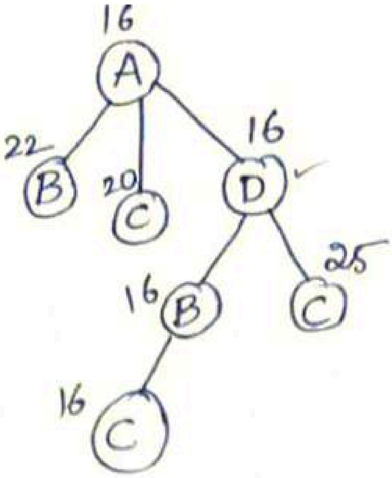
Reduction = 0

$T.C = 16 + 0 + 0 = 16$

path:

$$A \xrightarrow{3} D \xrightarrow{3} B \xrightarrow{9} C \xrightarrow{1} A$$

Total cost $= 3 + 3 + 9 + 1 = 16$



(*) 0/1 Knapsack problem - Branch and Bound :-

The 0/1 Knapsack problem can be effeciently solved using the branch and bound technique, which systematically explores possible item combinations while pruning branches that cannot lead to better solutions based on calculated bounds.

Formulation :-

Given a set of n items, each with a weight w_i and value v_i, and a Knapsack with a maximum capacity W, the goal is to maximize the total value in the Knapsack without exceeding its weight limit. Each item can be either included (1) or excluded (0), that is why it is Knocon as the 0/1 Knapsack.

Method for solving the problem :-

1. Sort items by value/weight ratio in descending order to prioritize more valuable items per unit weight.
2. Start with a root node representing the empty Knapsack.

③ At each node, branch into two: one where the current items is included (if it fits), and one where it is excluded.

④ Calculate the upper bound for each node - often using a greedy approach, sometimes taking fractions of remaining items for calculation purposes only (not as part of the solution).

⑤ use the priority queue to repeatedly process the node with the best bound.

⑥ whenever a node's bound is worse than the best found feasible solution, it's pruned.

⑦ The process Continues until all promising nodes are explored, and the best feasible solution found so far is the answer.

Advantages:

* Efficient Compared to brute-force or basic backtracking since it eliminates large subtrees that cannot yield better solutions.

* Especially useful when weights and values are not integers or the state space is too large for dynamic programing.

0/1 Knapsack - LEAST COST BRANCH AND BOUND

Knapsack $\rightarrow$ bag | Container.

Objective is to place the objects into Knapsack so that the profit is maximum.

Consider n objects, each object is with profit p_i and weight w_i

Knapsack capacity $= m$.

Example:- $n=4$ and $m=15$

$p_i = (10, 10, 12, 18)$ and $w_i = (2, 4, 6, 9)$

First, Convert all profits into negatives

$p_i = (\underset{p_1}{-10}, \underset{p_2}{-10}, \underset{p_3}{-12}, \underset{p_4}{-18})$

For Node ①, calculate upper bound and lower bound

upper bound - fraction values are not allowed

lower bound - fraction values are allowed

Node 1:

Lower bound:

-18	3/9	
-12	6	(3)
-10	4	(9)
-10	2	(13)

$\Rightarrow -10+10-12-\overset{2}{\cancel{18}}\times\frac{3}{\cancel{9}}$

$\Rightarrow -32-6$

$= -38$

upper bound.

		(3)
-12	6	(9)
-10	4	(13)
-10	2	

profit $= -10-10-12$

$= -32$

Node 2:

LB $\rightarrow x_1 = 1$

$= -38$

3/9
6
4
2

$U \cdot P = -32$

Node 3:- $x_1 = 0$

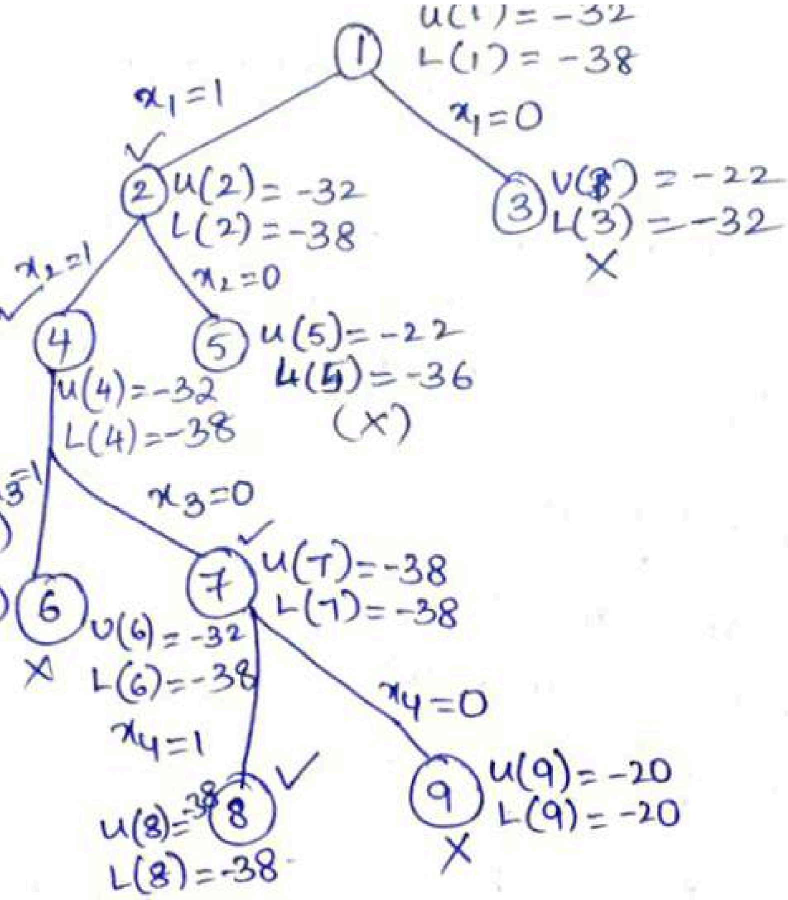
Lower bound

-18	5/9	
-12	6	(5)
-10	4	(11)

$\Rightarrow -10-12-\overset{2}{\cancel{18}}\times\frac{5}{\cancel{9}}$

$= -22-10$

$= -32$



Lowest lower bound

upper bound

-12 | 6 | (5)
-10 | 4 | (11)

= -10 - 12
= -22

Lowest lower bounds Same Compare upper bound.

Node 4 : $x_2=1$, $x_1=1$

Lower bound

-18 | 3/9 | (3)
-12 | 6 | 9
-10 | 4 | 13
-10 | 2 |

= -10 - 10 - 12 - 6
= -38

upperbound

-12 | 6 | (3)
-10 | 4 | (9)
-10 | 2 | (13)

= -32

Node 5 : $x_1=1$, $x_2=0$

L.B

-18 | 7/9 | (7)
-12 | 6 | (13)
-10 | 2 |

$-10-12-\cancel{18}^{2}\times\frac{7}{\cancel{9}}$
$\Rightarrow -22-14$
$= -36$

U.B

-12 | 6 | (13)
-10 | 2 |

-10 - 12
= -22

Node 6 : $x_1=1$, $x_2=1$, $x_3=1$

L.B:

| 3/9 | (3)
-12 | 6 | (9)
-10 | 4 | (13)
-10 | 2 |

$-10-10-12-\frac{3}{\cancel{9}}\times\cancel{18}^{2} \Rightarrow -38$

Upper Bound.

-12 | 6 |
-10 | 4 |
-10 | 2 |

-10 - 10 - 12 = -32

Node 7: $x_1=1, x_2=1, x_3=0$.

L.B

-18	9	(9)
-10	4	(13)
-10	2	

$-10-10-18=-38$

upper bound

-18	9
-10	4
-10	2

$= -10-10-18 = -38$.

Node 8 : $x_1=1, x_2=1, x_3=0, x_4=1$

Lower bound.

-18	9
-10	4
-10	2

$\Rightarrow -10-10-18=-38$

upper bound

-18	9
-10	4
-10	2

$\Rightarrow -38$.

Node 9: $x_1=1, x_2=1, x_3=0, x_4=0$.

Lower bound

-10	4
-10	2

$\Rightarrow -20$

upper bound

-10	4
-10	2

$\Rightarrow -20$.

Final path: $x_1=1, x_2=1, x_3=0, x_4=1$

1, 2, 4 $\rightarrow$ elements placed in Knapsack

Total profit $= -10-10-18$

$= -38$

$\Rightarrow +38$.

⊛ FIFO Branch and Bound :-

The FIFO (First-In-First-Out) method in
Branch and Bound is a strategy used in the
Design and Analysis of Algorithms to solve
optimization and Combinatorial problems efficiently.
It uses Breadth First Search (BFS) technique.
→ In this technique, a queue data structure is
used to manage the list of live nodes (nodes
that have been generated but not yet explored).
→ When a node is expanded (called the E-node),
all its children are generated and added to
the end of the queue.
→ The oldest node – that is, the one inserted first –
is then selected next for exploration, maintaining
the FIFO order.
→ This ensures a level-wise (breadth-first) search
of the state space tree.

Key Terms :-

⊛ E-node (Expanding node): The node currently
being explored.
⊛ Live node : A generated node whose children
are yet to be explored.
⊛ Dead node : A node that is fully explored
or pruned using bounding functions.

The process continues until the optimal or
goal node is found or until no live node remain.

Example for FIFO-branch & bound solution :-

→ It is similar to LC branch and bound. but has global upper bound.

$n = 4$ and $m = 15$

$P_i = (10, 10, 12, 18)$ and $w_i = (2, 4, 6, 9)$

negative profits $= (-10, -10, -12, -18)$

Node 1 :-

Lower bound

-18	3/9	3
-12	6	9
-10	4	13
-10	2	

$\Rightarrow -10-10-12-\cancel{18}^{2}\times\frac{3}{\cancel{9}}$

$L(1) = -38$

upper bound

-12	6
-10	4
-10	2

$u(1) \Rightarrow -32$

Node 2 ; $x_1 = 1$

L.B

-18	3/9
-12	6
-10	4
-10	2

$\Rightarrow -38$

$u(2) = -32$

Node 3 ; $x_1 = 0$

Lower

-18	5/9
-12	6
-10	4

$-10-12-18\times\frac{5}{9} = -32$

upper

-12	6
-10	4

$u(3) \Rightarrow -22$

Node 4 : $x_1 = 1, x_2 = 1$

L.B

-18	3/9	(3)
-12	6	(9)
-10	4	(13)
-10	2	

$\Rightarrow -38$

U.B

-12	6	(9)
-10	4	(13)
-10	2	

$\Rightarrow -32$

Node 5 ; $x_1 = 1, x_2 = 0$

L.B

-18	7/9	7
-12	6	(13)
-10	2	

$\Rightarrow -10-12-\cancel{18}^{2}\times\frac{7}{\cancel{9}}$

$\Rightarrow -36$

U.B

-12	6
-10	2

$\Rightarrow -22$

Node 6: $x_1=0, x_2=1$

L.B

-18	$5/9$
-12	6
-10	4

$\Rightarrow L(6) = -32$

U.B

-12	6
-10	4

$= -22$

Node 7: $x_1=0, x_2=0$

-18	9
-12	6

$L(7) = -30$

U.B

-18	9
-12	6

$U.B(7) = -30$

Node 8: $x_1=1, x_2=1, x_3=1$

$l(8) = -38, u(8) = -32$

Node 9: $x_1=1, x_2=1, x_3=0$

-18	9	9
-10	4	13
-10	2	

$l(9) = -38$

-18	9
-10	4
-10	2

$u(9) = -38$

Node 10: $x_1=1, x_2=1, x_3=1, x_4=1$

×

Node 11: $x_1=1, x_2=1, x_3=1,$
$x_4=0$

Lower

-12	6
-10	4
-10	2

$L(11) = -32$

upper

6
4
2

$U(11) = -32$

Node 12: $x_1=1, x_2=1, x_3=0,$
$x_4=\phi$.

-18	9	9
-10	4	
-10	2	

$l(12) = -38$

-18	9
-10	4
-10	2

$U(12) = -38$

Node 13: $x_1=1, x_2=1, x_3=0, x_4=0$

-10	4
-10	2

$l(13) = -20$

4
2

$U(13) = -20$

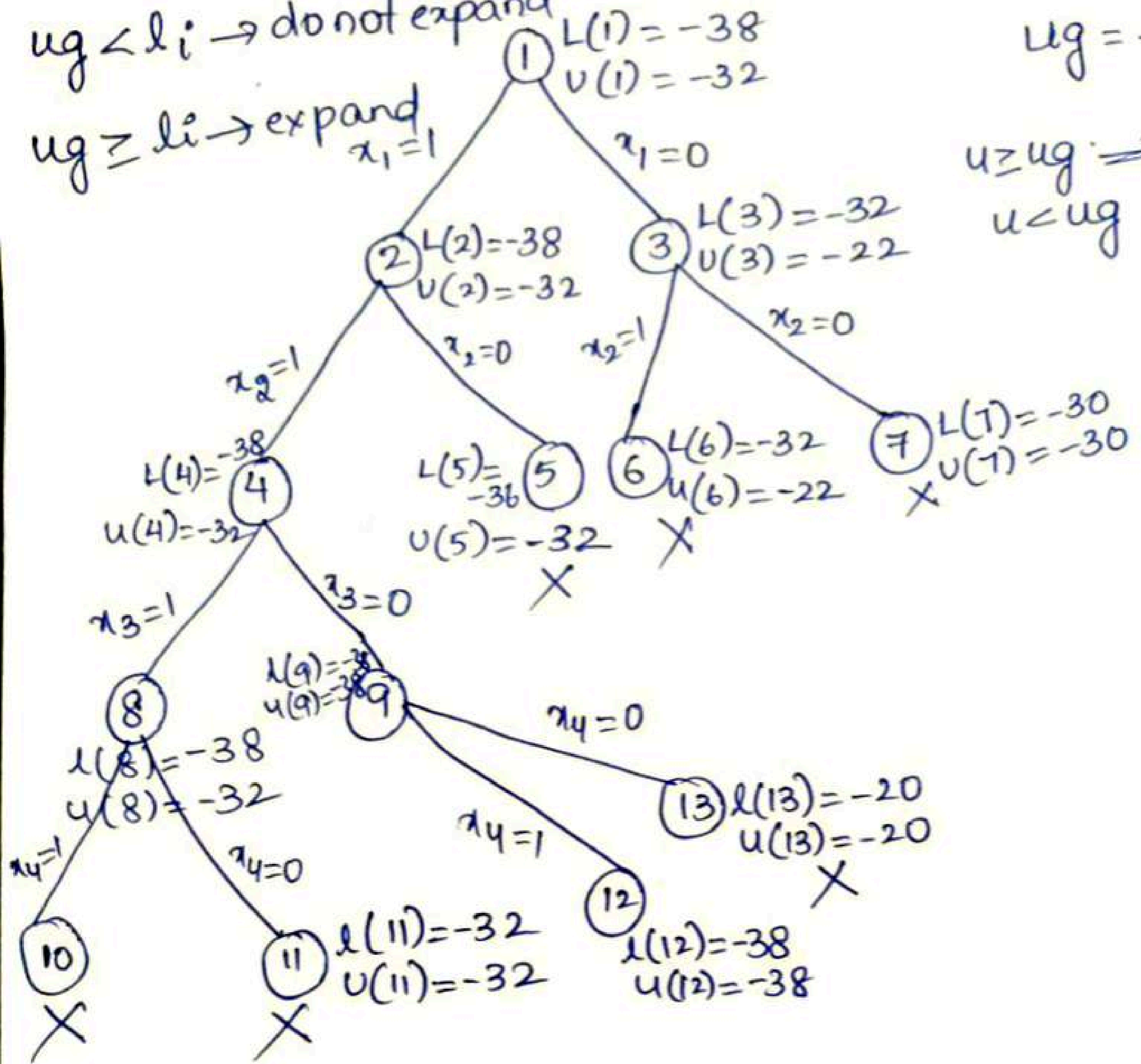
$ug < l_i \rightarrow$ do not expand

$ug \geq l_i \rightarrow$ expand

$ug = -32$

$u \geq ug \Rightarrow$ same

$u < ug \Rightarrow$ update



$ug = -38$

Final path : $x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 1$

Total profit $= +10+10+18 = 38$ //.