Coping with the limitations of Algorithm Power

Backtracking and branch-and-bound are used to solve at least some large instances of difficult combinatorial problems. Both strategies can be considered an improvement over exhaustive search. Unlike exhaustive search, they construct candidate solutions one component at a time and evaluate the partially constructed solutions: if no potential values of the remaining components can lead to a solution, the remaining components are not generated at all.

Both Backtracking and branch-and-bound are based on the construction of a state-space tree whose nodes reflect specific choices made for a solution's components.

Branch-and-Bound	Backtracking
1. applicable only to optimization problems.	1. It applies only to nonoptimization problems.
2. uses BFS to generate state-space tree	2. uses DFS to generate state-space tree.

Backtracking

The principal idea is to construct solutions one component at a time and evaluate such partially constructed candidates as follows. If a partially constructed solution can be developed further without violating the problem constraints, it is done by taking the first remaining legitimate option for the next component. If there is no legitimate option for the next component, no alternatives for any remaining component need to be considered. In this case, the algorithm backtracks to replace the last component of the partially constructed solution with its next option.

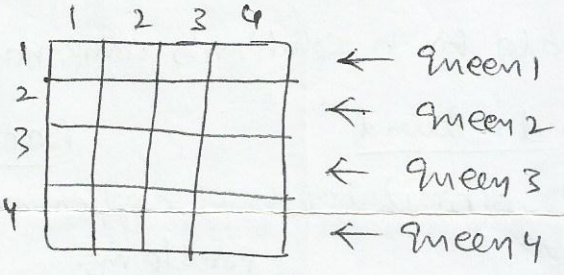
State-space tree

Promising node: A node in a state-space tree is said to be promising if it corresponds to a partially constructed solution that may still lead to a complete solution.

Non promising node: Partially constructed solution may not lead to a complete solution

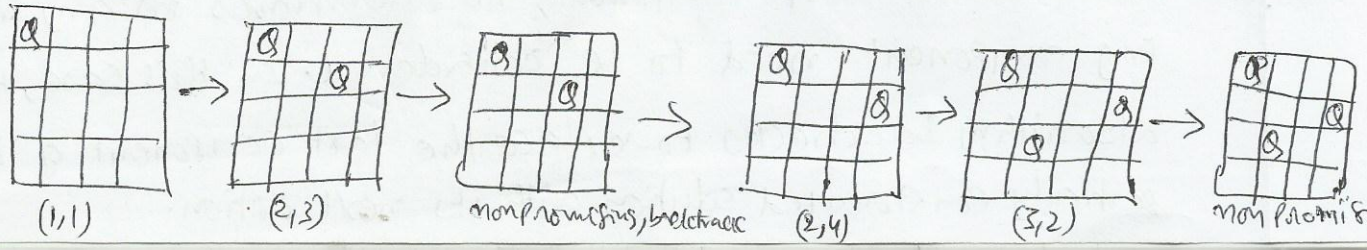
n- Queens Problem

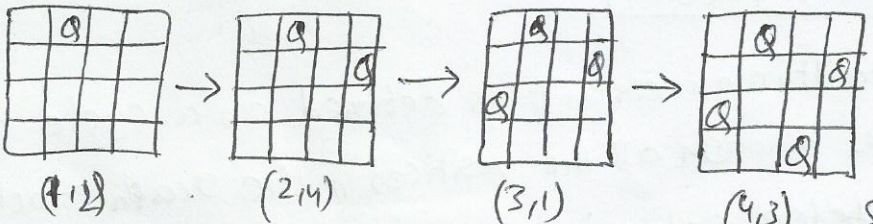
The problem is to place n queens on an n-by-n chessboard so that no two queens attack each other by being in the same row or in the same column or on the same diagonal.



Board for the four-queens problem

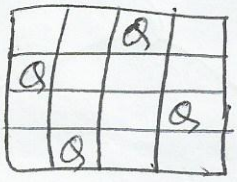
A state-space tree for a backtracking algorithm is constructed in the manner of depth-first search. If the current node is promising, its child is generated by adding the first remaining legitimate option for the next component of a solution, and the processing moves to this child. If the current node turns out to be nonpromising, the algorithm backtracks to the node's parent to consider the next possible option for its last component; if there is no such option, it backtracks one more level up the tree, and so on. Finally, if the algorithm reaches a complete solution to the problem, it either stops (if just one solution is required) or continues searching for other possible solutions.



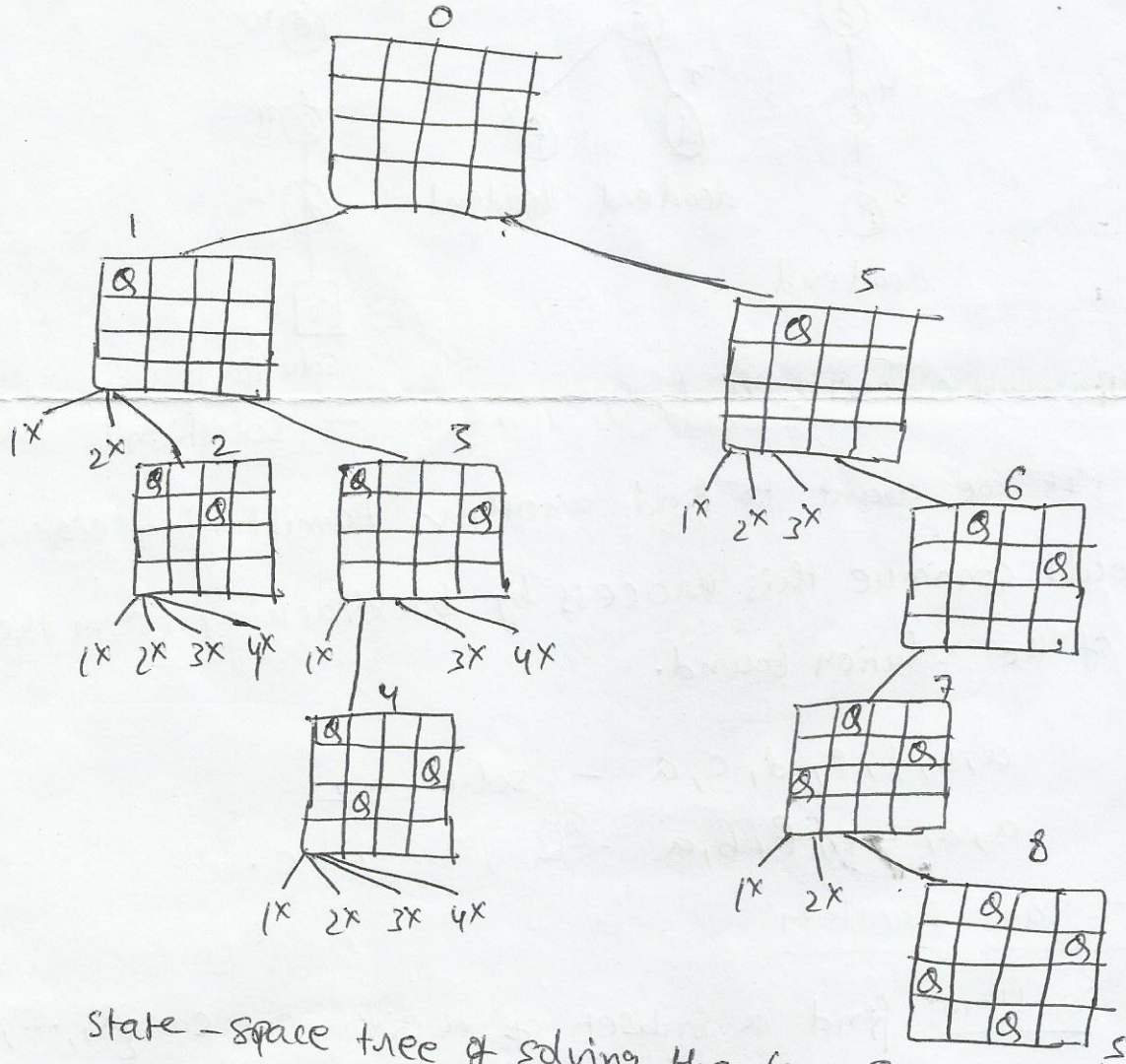


Solution to the Proble

hoco many other solutions are there for the four-queens problem? One



other solution to the Proble

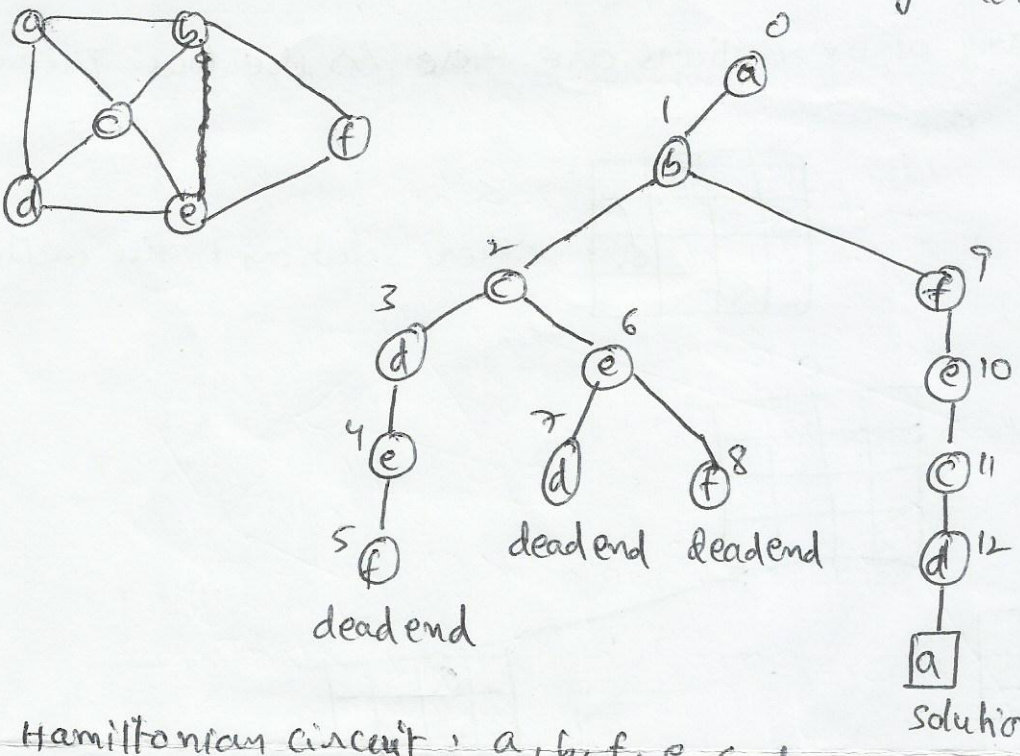


State-space tree of solving the four-queens problem by backtracking. x denotes an unsuccessful attempt to place a queen in the indicated column. The numbers above the nodes indicate the order in which the nodes are generated.

solution

Hamiltonian circuit Problem

Definition: A Hamiltonian circuit is defined as a cycle that passes through all the vertices of the graph exactly once before returning to the city where it started.



Hamiltonian circuit: a, b, f, e, c, d, a — solution 1

If we want to find another Hamiltonian circuit, we could continue this process by backtracking from the leaf of the solution found.

a, b, f, e, d, c, a — solution 2
a, d, c, e, f, b, a — solution 3, ... etc.

Subset-Sum Problem

Problem: find a subset of a given set $S = \{s_1, \cdots, s_n\}$ of n positive integers whose sum is equal to a given positive integer d.

Example: $S = \{1, 2, 5, 6, 8\}$ and $d = 9$, there are two solutions:

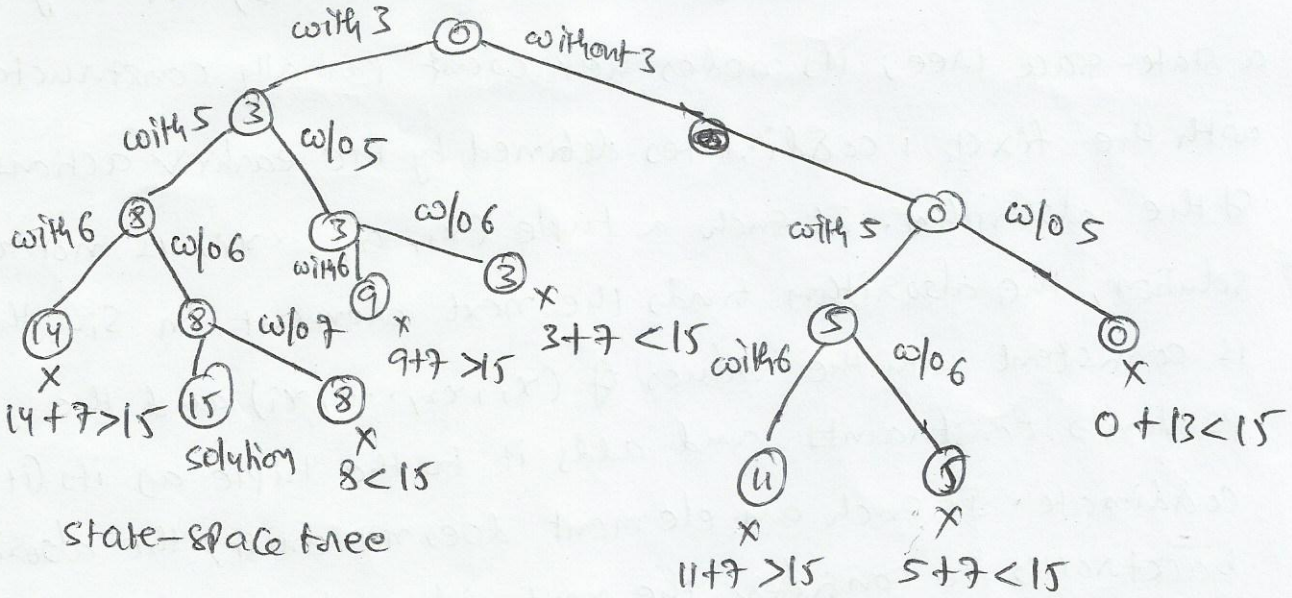
$\{1, 2, 6\}$ and $\{1, 8\}$

let $S = \{s_1, s_2, s_3, s_4, s_5, \ldots\}$ ③

sort the set's elements in increasing order

$$s_1 \le s_2 \le \cdots \le s_n.$$

The state-space tree can be constructed as a binary tree for the instance $S = \{3, 5, 6, 7\}$ and $d = 15$.



State-space tree

→ s' — sum of numbers in the node.

→ If s' is equal to d, we have a solution to the problem.

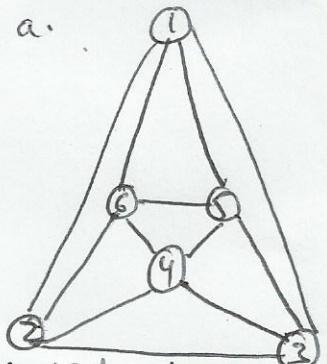
→ If s' is not equal to d, we can terminate the node as non promising if either of the following two inequalities holds:

$$s' + s_{i+1} > d \text{ (the sum } s' \text{ is too large)} \quad \text{Ex: } \overset{s'}{14} + \overset{s_{i+1}}{7} > \overset{d}{15}$$

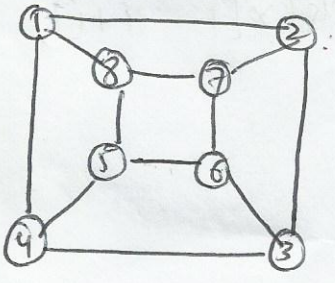
$$s' + \sum_{j=i+1}^{n} s_j < d \text{ (the sum } s' \text{ is too small)} \quad \text{Ex: } \overset{s'}{5} + \overset{\sum_{j=i+1}^{n} s_j}{7} < \overset{d}{15}$$

→ Apply backtracking to the problem of finding a Hamiltonian circuit in the following graphs:

a.



b.



→ Apply backtracking to solve the following instance of the subset-sum problem: $S = \{1, 3, 4, 5\}$ and $d = 11$.

→ Apply backtracking to solve the following instance of the Subset-Sum problem: $S = \{5, 7, 10, 12, 15, 18, 20\}, d = 35$.

→ $S = \{5, 10, 12, 13, 15, 18\}$, $d = 30$.

General Remarks Backtracking Algorithm.

A backtracking algorithm generates, explicitly & implicitly, a state-space tree; its nodes represent partially constructed tuples with the first i coordinates defined by the earlier actions of the algorithm. If such a tuple $(x_1, x_2, \dots x_i)$ is not a solution, the algorithm finds the next element in S_{i+1} that is consistent with the values of $(x_1, x_2, \dots, x_i)$ and the problem's constraints and adds it to the tuple as its $(i+1)^{th}$ coordinate. If such an element does not exist, the algorithm backtracks to consider the next value of x_i, and so on.

```
ALGORITHM Backtrack(x[1...i])
// Gives a template of a generic backtracking algorithm.
// Input: x[1...i] specifies first i promising components of
a solution.
// output: All the tuples representing the problem's solutions
   if X[1...i] is a solution write x[1...i]
   else
      for each element x ∈ S_{i+1} consistent with X[1...i] and
                                   the constraints do
         X[i+1] ← x
         Backtrack(X[1... i+1])
```

Branch-and-Bound

An optimization problem is one that minimizes or maximizes an objective function subject to some constraints.

Ex: a tour's length, the value of items selected, the cost of an assignment, and the like.

feasible solution – It is a point in the problem's search space that satisfies all the problem's constraints.

Ex: A subset of items whose total weight does not exceed the knapsack's capacity in the knapsack problem.

Optimal solution – It is a feasible solution with the best value of the objective function

Ex: the most valuable subset of items that fit the knapsack.

In general, we terminate a search path at the current node in a state-space tree of a branch-and-bound algorithm for any one of the following three reasons:

- The value of the node's bound is not better than the value of the best solution seen so far.
- the node represents no feasible solutions because the constraints of the problem are already violated.
- update the best solution seen so far, if the new solution is better.

Assignment Problem Apply Branch & Bound Technique to solve AP.

Problem: Assigning n people to n jobs so that the total cost of the assignment is as small as possible.

(Or)

select one element in each row of the matrix so that no two selected elements are in the same column and their sum is the smallest possible.

$$C = \begin{array}{c} \begin{matrix} \text{job1} & \text{job2} & \text{job3} & \text{job4} \end{matrix} \\ \begin{bmatrix} 9 & 2 & 7 & 8 \\ 6 & 4 & 3 & 7 \\ 5 & 8 & 1 & 8 \\ 7 & 6 & 9 & 4 \end{bmatrix} \end{array} \begin{matrix} \\ \text{Person a} \\ \text{Person b} \\ \text{Person c} \\ \text{Person d} \end{matrix}$$

How can we find a lower bound on the cost of an optimal selection without actually solving the problem?

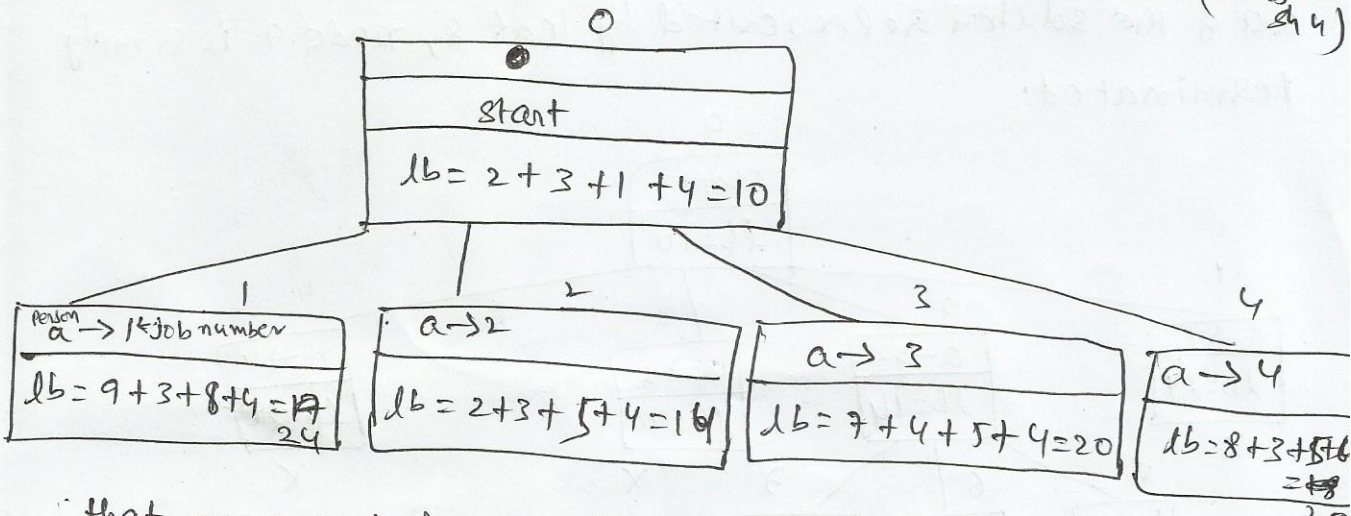
We can do this by several methods. For ex, it is clear that the cost of any solution, including an optimal one, cannot be smaller than the sum of the smallest elements in each of the matrix's rows. For the instance here, this sum is $2+3+1+4=10$. This is not the cost of any legitimate selection (3 & 1 came from the same column of the matrix); it is just a lower bound on the cost of any legitimate selection. For ex, for any legitimate selection that selects 9 from the first row, the lower bound will be $9+3+1+4=17$.

live nodes: nonterminated, i.e., still promising leaves of a current tree are called live nodes.

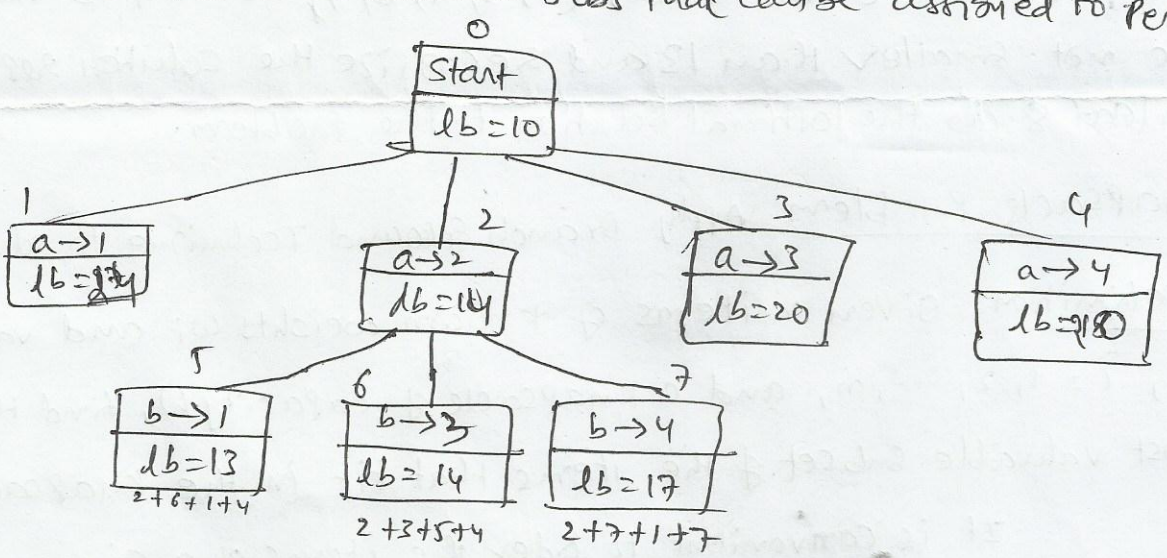
How can we tell which of the nodes is most promising? We can do this by comparing the lower bounds of the live nodes. It is sensible to consider a node with the best bound as most promising, although this does not, of course, preclude the possibility that an optimal solution will ultimately belong to a different branch of the state-space. This variation of the strategy is called the best-first branch-and-bound.

The lower-bound value for the root, denoted lb, is 10. The nodes on the first level of the tree correspond to selections of an element in the first row of the matrix,

i.e., a job for person a. So we have four live nodes (nodes 1 thru 4)

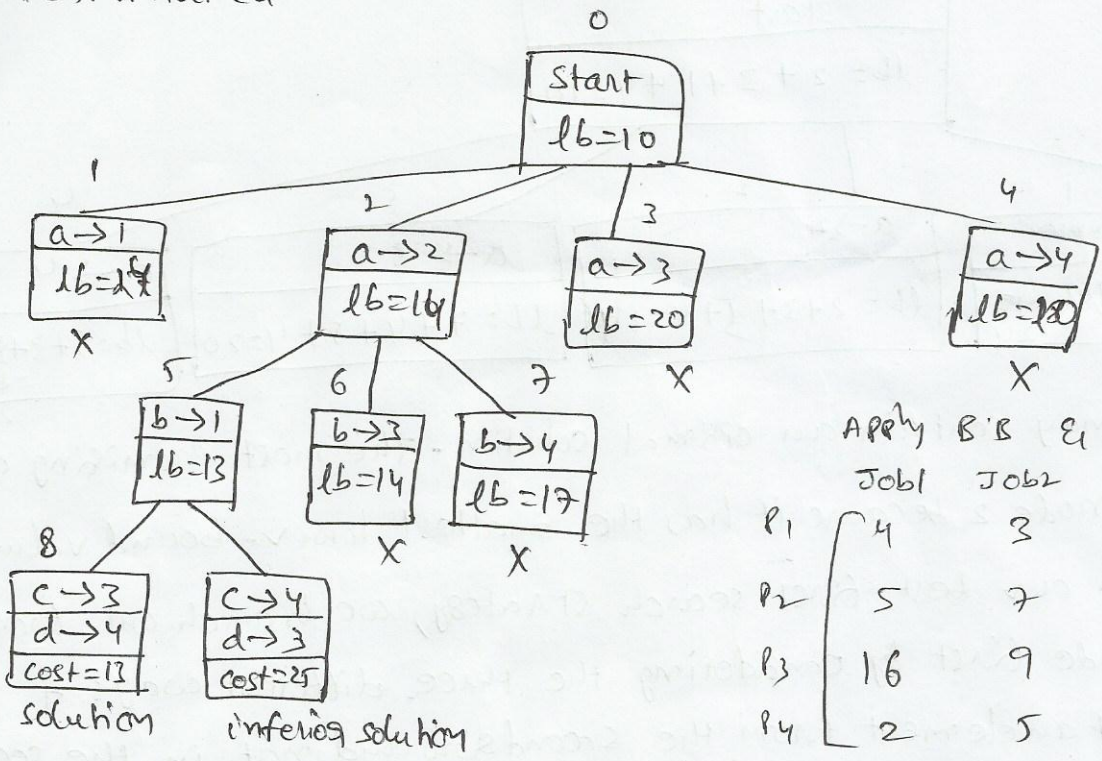


that may contain an optimal solution. The most promising of them is node 2 because it has the smallest lower-bound value. Following our best-first search strategy, we branch out from that node first by considering the three different ways of selecting an element from the second row and not in the second column - the three different jobs that can be assigned to person b



Of the six live nodes (nodes 1, 3, 4, 5, 6 & 7) that may contain an optimal solution, we again choose the one with the smallest lower bound, node 5. First, we consider selecting the third column's element from c's row (i.e., assigning person c to job 3). This leaves us with no choice but to select the element from the fourth column of d's row (assigning person d to job 4). This yields leaf 8. which corresponds to the feasible solution {a → 2, b → 1, c → 3, d → 4} with the total cost of 13. Its sibling, node 9, corresponds to the feasible solution {a → 2, b → 1, c → 4, d → 3}

with the total cost of 25. Since its cost is larger than the cost of the solution represented by leaf 8, node 9 is simply terminated.



Apply B&B & find out solution

	Job1	Job2	Job3	Job4
P_1	4	3	8	6
P_2	5	7	2	4
P_3	16	9	3	1
P_4	2	5	3	7

Terminate all nodes (1, 3, 4, 6, 7) as their lb values are not smaller than 13 and recognize the solution represented by leaf 8 as the optimal solution to the problem.

Knapsack Problem Apply Branch & Bound Technique to solve KP.

Definition: given n items of known weights w_i and values v_i, $i = 1, 2, \cdots, n$, and a knapsack of capacity W, find the most valuable subset of the items that fit in the knapsack.

It is convenient to order the items of a given instance in descending order by their value-to-weight ratios. Then the first item gives the best payoff per weight unit and the last one gives the worst payoff per weight unit,

$$v_1/w_1 \geq v_2/w_2 \geq \cdots \geq v_n/w_n.$$

Problem
$(w_1, w_2, w_3, w_4) = \{7, 3, 4, 5\}$
$(v_1, v_2, v_3, v_4) = \{42, 12, 40, 25\}$
capacity $W = 10$

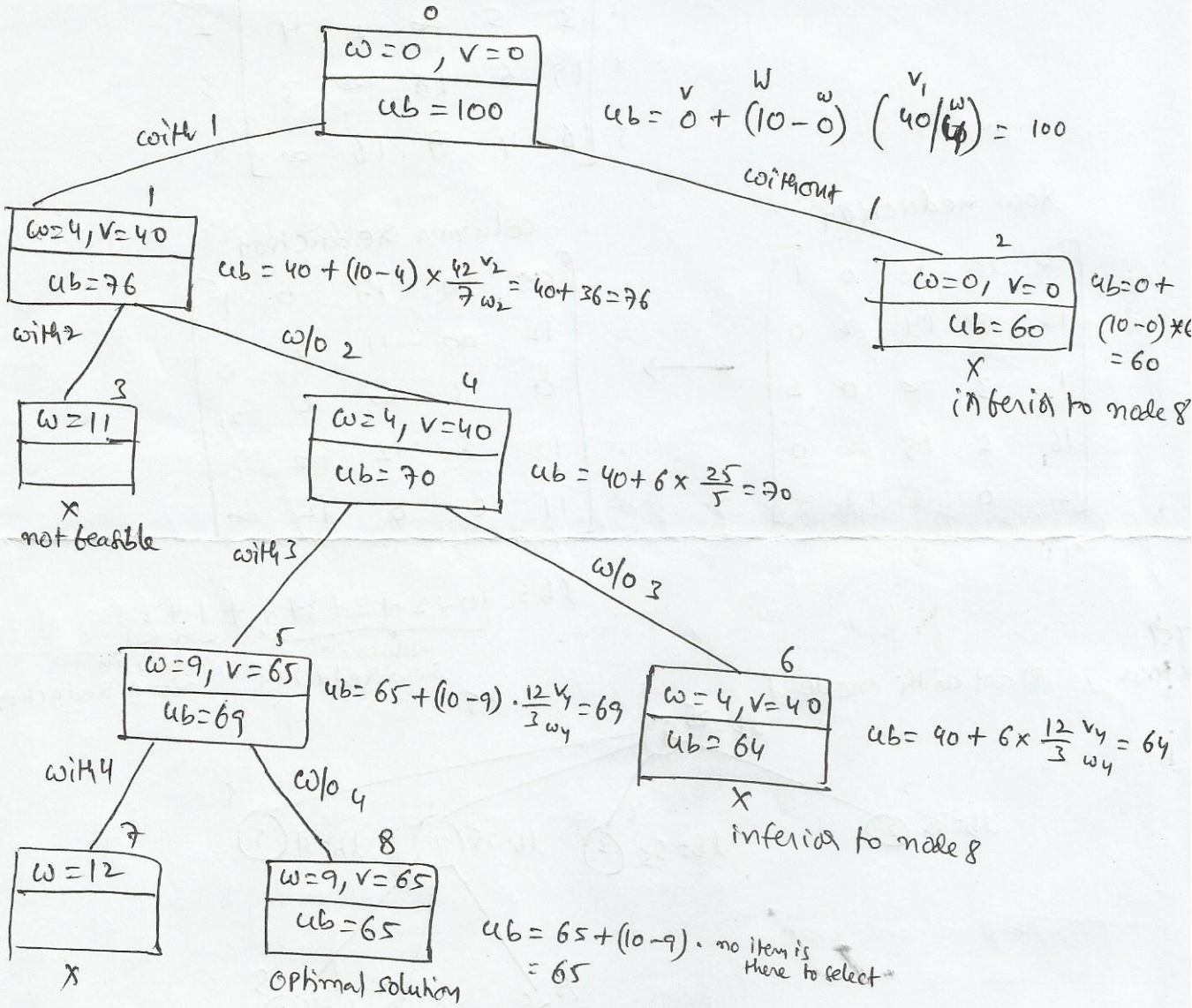
let us apply BB to the same instance of the KP in exhaustive search.

item	weight	value	value/weight
1	4	$40	10
2	7	$42	6
3	5	$25	5
4	3	$12	4

the knapsack's capacity W is 10.

A simple way to compute the upper bound ub is to add to v, the total value of the items already selected, the product of the remaining capacity of the knapsack W−w and the best per unit payoff among the remaining items, which is v_{i+1}/w_{i+1}.

$$ub = v + (W-w)\left(v_{i+1}/w_{i+1}\right) \longrightarrow (1)$$



node 8 represents just a single subset {1, 3}. The remaining live nodes 2 & 6 have smaller upper-bound values than the value of the solution represented by node 8. Hence, both can be terminated making the subset {1, 3} of node 8 the optimal solution to the problem.

→ solve the following instance of the knapsack problem by the branch-and-bound algorithm $(w_1, w_2, \dots, w_6) = \{5, 7, 2, 4, 5, 1\}$ $(v_1, v_2, \dots, v_6) = \{40, 35, 18, 4, 10, 2\}$ & $w = 15$

(TSP)

Travelling salesman problem As it is a minimization problem, minimize tour length.

→ consider the following instance of the TSP, apply branch-and-bound technique to solve it.

given cost matrix A =

	1	2	3	4	5	
1	∞	20	30	10	11	10
2	15	∞	16	4	2	2
3	3	5	∞	2	4	2
4	19	6	18	∞	3	3
5	16	4	7	16	∞	4

Row reduction

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

→

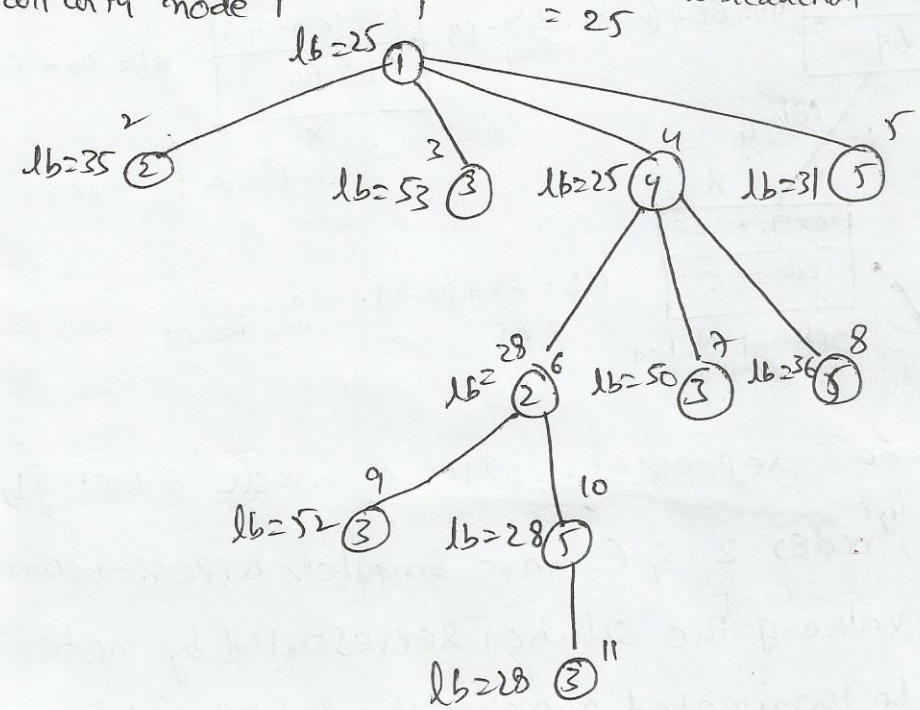
column reduction

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

lb = 10+2+2+3+4 + 1+3 = 25

(10+2+2+3+4: values in row reduction; 1+3: values in column reduction)

TSP tour, start with node 1



Path 1,2

1. make 1st row, 2nd column elements as ∞
2. A(j,1) = ∞

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

values of row & column reduction

$lb = root's\ lb + A(i,j) + \gamma$

$= 25 + 10 + 0$

$= 35$

Path 1,3

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \underline{12}_{1} & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ \underline{15}_{4} & 3 & \infty & \infty & 0 \\ (11)_{0} & 0 & \infty & 12 & \infty \end{bmatrix}$$

11

$lb = root's\ lb + A(1,3) + \gamma$

$= 25 + 17 + 11$

$= 53$

Path 1,4

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

$lb = root's\ lb + A(1,4) + \gamma$

$= 25 + 0 + 0$

$= 25$

Path 1,5

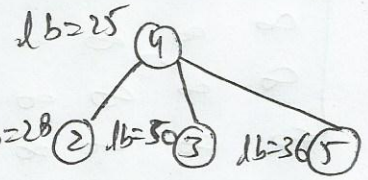
$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \underline{12}^{10} & \infty & \underline{11}^{9} & (2)^{0} & \infty \\ 0 & 3 & \infty & 0 & \infty \\ \underline{15}^{12} & (3)^{0} & \underline{12}^{9} & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{bmatrix} \begin{matrix} 2 \\ \\ \\ 3 \\ \\ \end{matrix}$$

$lb = root's\ lb + A(1,5) + \gamma$

$= 25 + 1 + 5$

$= 31$

Of these four lengths, path 1-4 is having lowest lb. Hence choose this path & matrix. find the path lengths from 4 to 2, 3, & 5.



Path 4-2 4th row, 2nd column element to ∞

$A(4,1) = \infty$

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

$lb = root's\ lb + A(4,2) + \gamma$

$= 25 + 3 + 0 = 28$

Path 4,3 — 4th row, 3rd column elements ∞; A(j,1) = ∞

∞	∞	∞	∞	∞
12	∞	∞	∞	0
∞	3	∞	∞	2
∞	∞	∞	∞	∞
11	0	∞	∞	∞

2 (row reduction), 11 (column reduction)

lb = root's lb + A(i,j) + r (i = 4, j = 3)

= 25 + 12 + 13

= 50

Path 4,5 — 4th row, 5th column elements ∞; A(j,1) = ∞

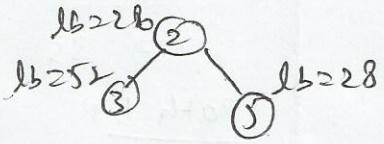
∞	∞	∞	∞	∞
12	∞	11	∞	∞
0	3	∞	∞	∞
∞	∞	∞	∞	∞
∞	0	0	∞	∞

11 (column reduction)

lb = root's lb + A(i,j) + r (i = 4, j = 5)

= 25 + 0 + 11 = 36

6th node is having lowest lb. Hence, it can be chosen as the next exploring node. the nodes to be visited from node 2 are 3 & 5.



Path 2,3 — 1) 2nd row, 3rd column elements ∞ 2) A(j,1) = ∞

∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
∞	∞	∞	∞	2
∞	∞	∞	∞	∞
11	∞	∞	∞	∞

2 (row reduction), 11 (column reduction)

lb = root's lb + A(i,j) + r (i = 2, j = 3)

= 28 + 11 + 13

= 52

Path 2,5

∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
0	∞	∞	∞	∞
∞	∞	∞	∞	∞
∞	∞	0	∞	∞

lb = root's lb + A(i,j) + r (i = 2, j = 5)

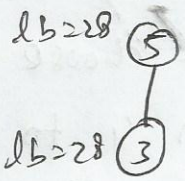
= 28 + 0 + 0

= 28

10th node is the next node to be explored. from node 5 to node 3

Path 5,3 — A(j,1) = ∞

∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
∞	∞	∞	∞	∞



lb = root's lb + A(i,j) + r (i = 5, j = 3)

= 28 + 0 + 0

= 28 optimal solution is 1 - 4 - 2 - 5 - 3 - 1 with lb = 25

optimal path

→ consider the traveling salesperson instance defined by the cost matrix

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

solve it using Branch-and-Bound strategy and construct portion of the state space tree.

→ solve the following TSP instance using Branch-and-Bound & construct state space tree

$$\begin{bmatrix} \infty & 11 & 10 & 9 & 6 \\ 8 & \infty & 7 & 3 & 4 \\ 8 & 4 & \infty & 4 & 8 \\ 11 & 10 & 5 & \infty & 5 \\ 6 & 9 & 5 & 5 & \infty \end{bmatrix}$$

→ Draw the portion of the state space tree generated by the following knapsack instances:

a) $n=5$, $(P_1, P_2, \cdots, P_5) = (10, 15, 6, 8, 4)$, $(\omega_1, \omega_2, \cdots \omega_n) = (4, 6, 3, 4, 2)$, and $W = 12$.

b) $n=5$, $(P_1, P_2, \cdots P_5) = (\omega_1, \omega_2, \omega_3, \omega_4, \omega_5) = (4, 4, 5, 8, 9)$ and $W = 15$.