Branch–and–Bound

Fundamentals

- Compared to backtracking, branch–and–bound requires two additional items:
- For every node of a state–space tree, a bound on the best value of the objective function on any solution that can be obtained by adding further components to the partially constructed solution represented by the node
- the value of the best solution seen so far

Fundamentals

- If this information is available, we can compare a node's bound value with the value of the best solution seen so far.
- If the bound value is not better than the value of the best solution seen so far—i.e., not smaller for a minimization problem and not larger for a maximization problem—the node is nonpromising and can be terminated (branch is "pruned").

Terminating Conditions for Branch and Bound

- Value of node's bound is not better than value of best solution seen so far
- Node represents no feasible solutions because constraints of problem are already violated

Terminating Conditions for Branch and Bound

- Subset of feasible solutions represented by node consists of a single point (and hence no further choices can be made)—in this case, we compare the value of the objective function for this feasible solution with that of the best solution seen so far and update the latter with the former if the new solution is better.

Assignment Problem

- an instance of the assignment problem is specified by an n × n cost matrix C so that we can state the problem as follows:
- 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

$$C = \begin{array}{c} \begin{matrix} \text{job 1} & \text{job 2} & \text{job 3} & \text{job 4} \end{matrix} \\ \begin{bmatrix} 9 & 2 & 7 & 8 \\ 6 & 4 & 3 & 7 \\ 5 & 8 & 1 & 8 \\ 7 & 6 & 9 & 4 \end{bmatrix} \end{array} \begin{matrix} \text{person } a \\ \text{person } b \\ \text{person } c \\ \text{person } d \end{matrix}$$

Find a lower bound on cost

- find a lower bound on the cost of an optimal selection without actually solving the problem
- 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 $2 + 3 + 1 + 4 = 10$

Find a lower bound on cost

- this is not the cost of any legitimate selection (3 and 1 came from the same column of the matrix)
- Just a lower bound on the cost of any legitimate selection
- apply the same thinking to partially constructed solutions
- For any legitimate selection that selects 9 from the first row, the lower bound will be 9 + 3 + 1 + 4 = 17

Find a lower bound on cost

- Rather than generating a single child of the last promising node as we did in backtracking, generate all children of most promising node among nonterminated leaves in the current tree
- to know most promising node, compare lower bounds of the live nodes
- sensible to consider a node with the best bound as most promising

Find a lower bound on cost

- Although this does not, of course, preclude the possibility that an optimal solution will ultimately belong to a different branch of the state–space tree
- Variation of the strategy is called the best–first branch–and–bound

Solution Construction – Assignment Problem

- As we already discussed, 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'

Solution Construction – Assignment Problem

- We have four live leaves—nodes 1 through 4
- Most promising of them is node 2 because it has smallest lower–bound value
- Following our best–first search strategy, we branch out from that node first by considering three different ways of selecting an element from second row and not in second column—three different jobs that can be assigned to person b

State Space Tree for Assignment Problem

$$C = \begin{bmatrix} 9 & 2 & 7 & 8 \\ 6 & 4 & 3 & 7 \\ 5 & 8 & 1 & 8 \\ 7 & 6 & 9 & 4 \end{bmatrix}$$

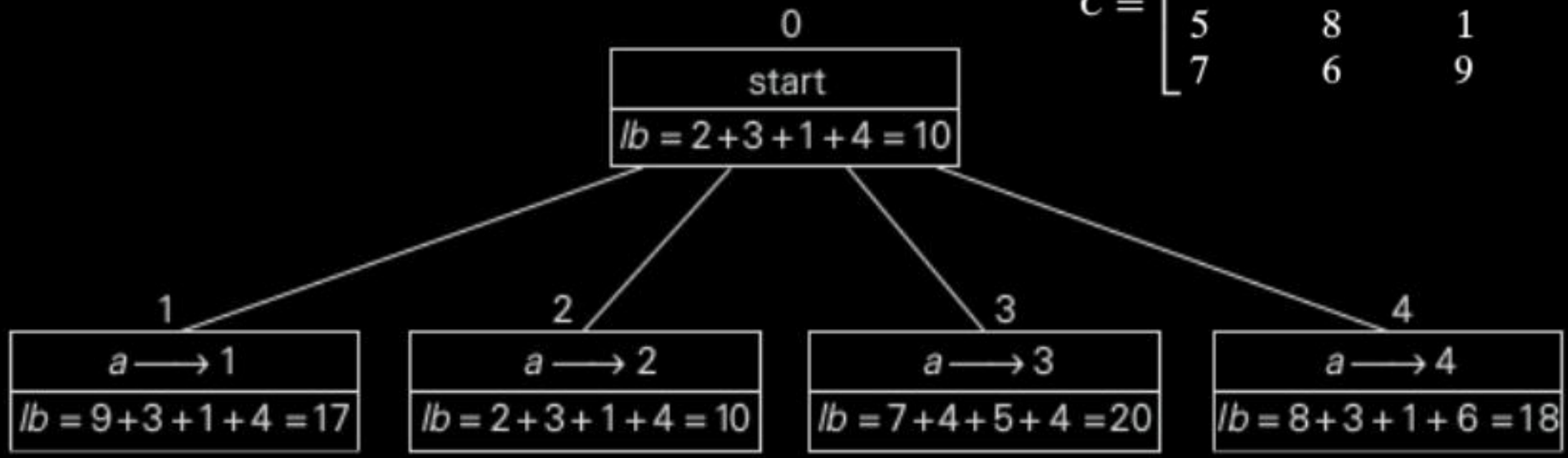


FIGURE 12.5 Levels 0 and 1 of the state-space tree for the instance of the assignment problem being solved with the best-first branch-and-bound algorithm. The number above a node shows the order in which the node was generated. A node's fields indicate the job number assigned to person a and the lower bound value, lb, for this node.

Solution Construction – Assignment Problem

- Following our best–first search strategy, we branch out from that node first by considering three different ways of selecting an element from second row and not in second column—three different jobs that can be assigned to person b

State Space Tree for Assignment Problem

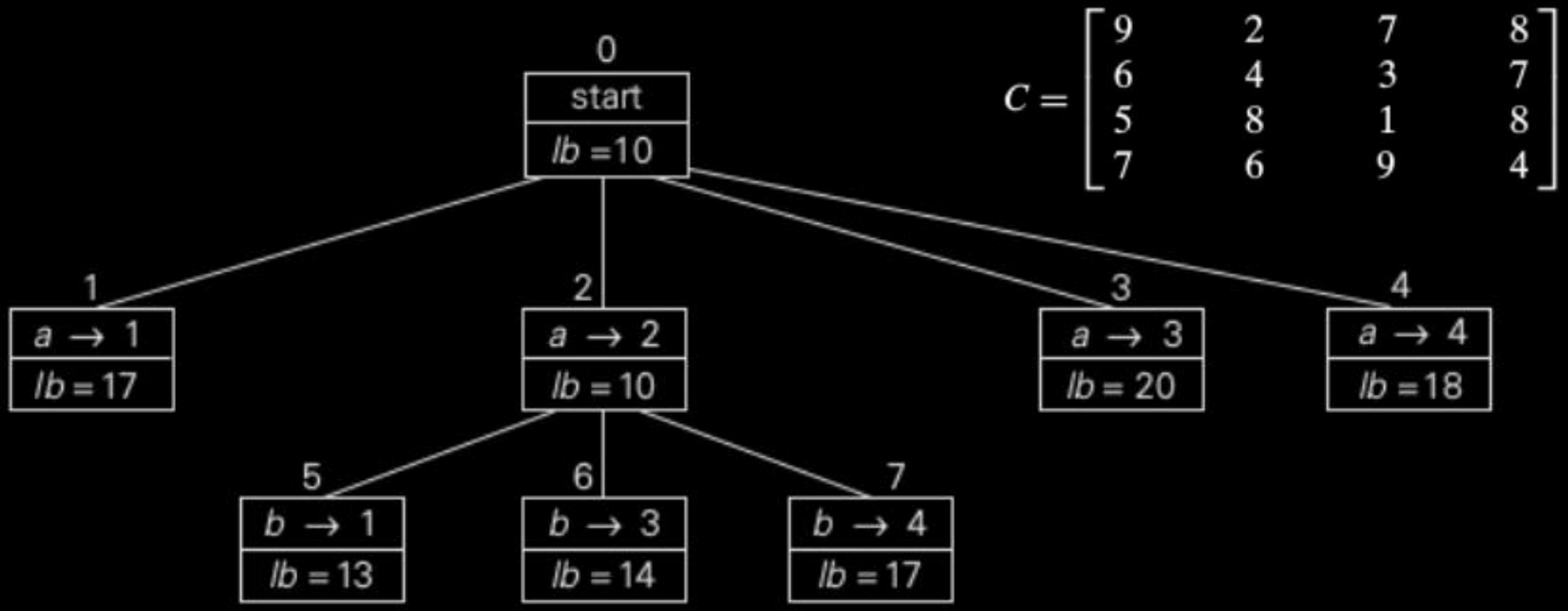


FIGURE 12.6 Levels 0, 1, and 2 of the state-space tree for the instance of the assignment problem being solved with the best-first branch-and-bound algorithm.

Solution Construction – Assignment Problem

- Of the six live leaves—nodes 1, 3, 4, 5, 6, and 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);

Solution Construction – Assignment Problem

- 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 (Figure 12.7), which corresponds to the feasible solution {a → 2, b → 1, c → 3, d → 4} with the total cost of 13.

State Space Tree for Assignment Problem

$$C = \begin{bmatrix} 9 & 2 & 7 & 8 \\ 6 & 4 & 3 & 7 \\ 5 & 8 & 1 & 8 \\ 7 & 6 & 9 & 4 \end{bmatrix}$$

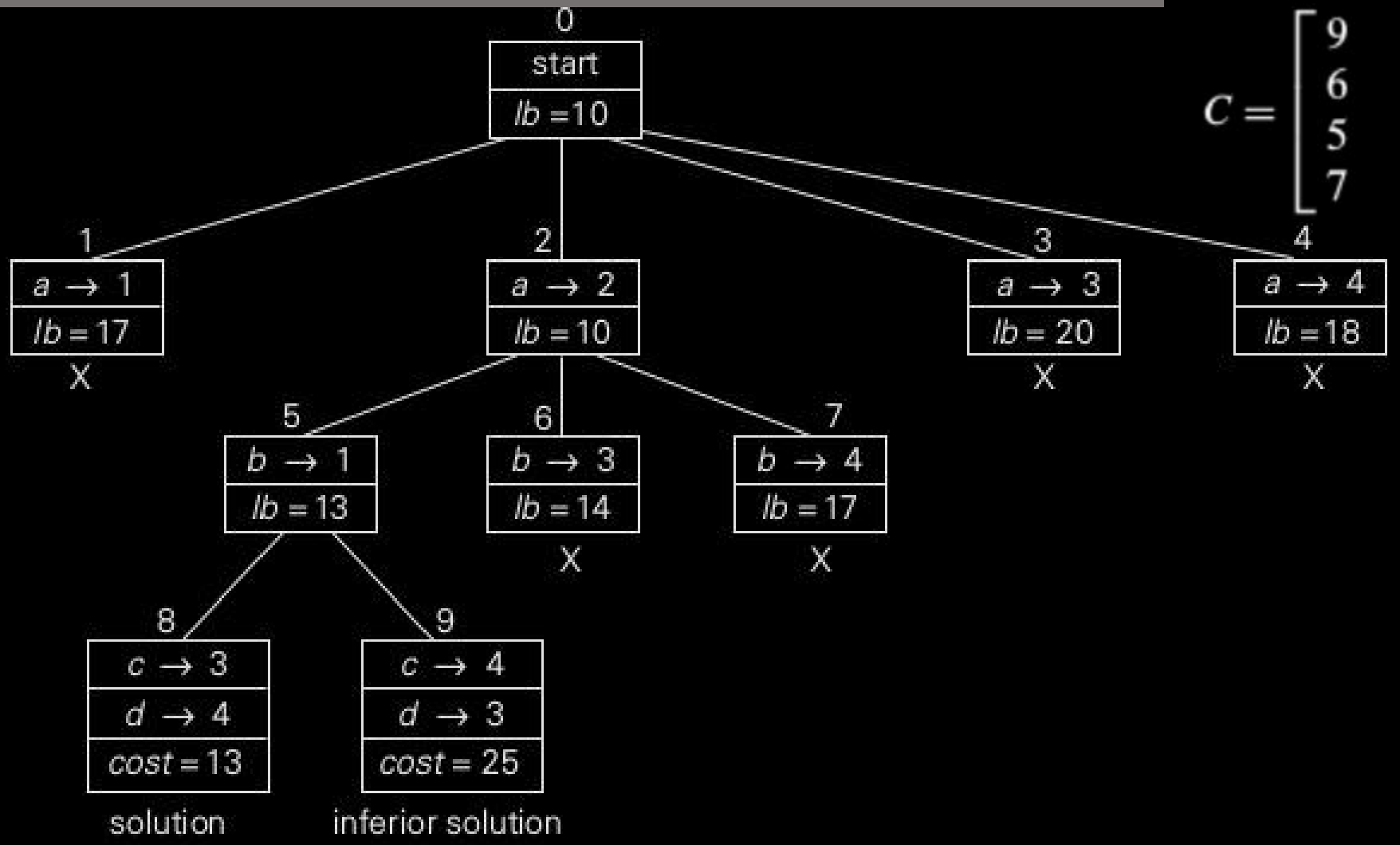


FIGURE 12.7 Complete state-space tree for the instance of the assignment problem solved with the best-first branch-and-bound algorithm.

Solution Construction – Assignment Problem

- the value of the best selection seen so far (leaf 8)
- Hence, we terminate all of them and recognize the solution represented by leaf 8 as the optimal solution to the problem

Knapsack Problem

- first item gives the best payoff per weight unit and the last one gives the worst payoff per weight unit, with ties resolved arbitrarily
- $v_1 / w_1 \geqslant v_2 / w_2 \geqslant \ldots \geqslant v_n / w_n$

Knapsack Problem

- 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} :

$$ub = v + (W - w)(v_{i+1}/w_{i+1}). \qquad \textbf{(12.1)}$$

Knapsack Problem

- Reorder items in descending order of their value–to–weight ratios

item	weight	value	$\frac{\text{value}}{\text{weight}}$
1	4	$40	10
2	7	$42	6
3	5	$25	5
4	3	$12	4

The knapsack's capacity W is 10.

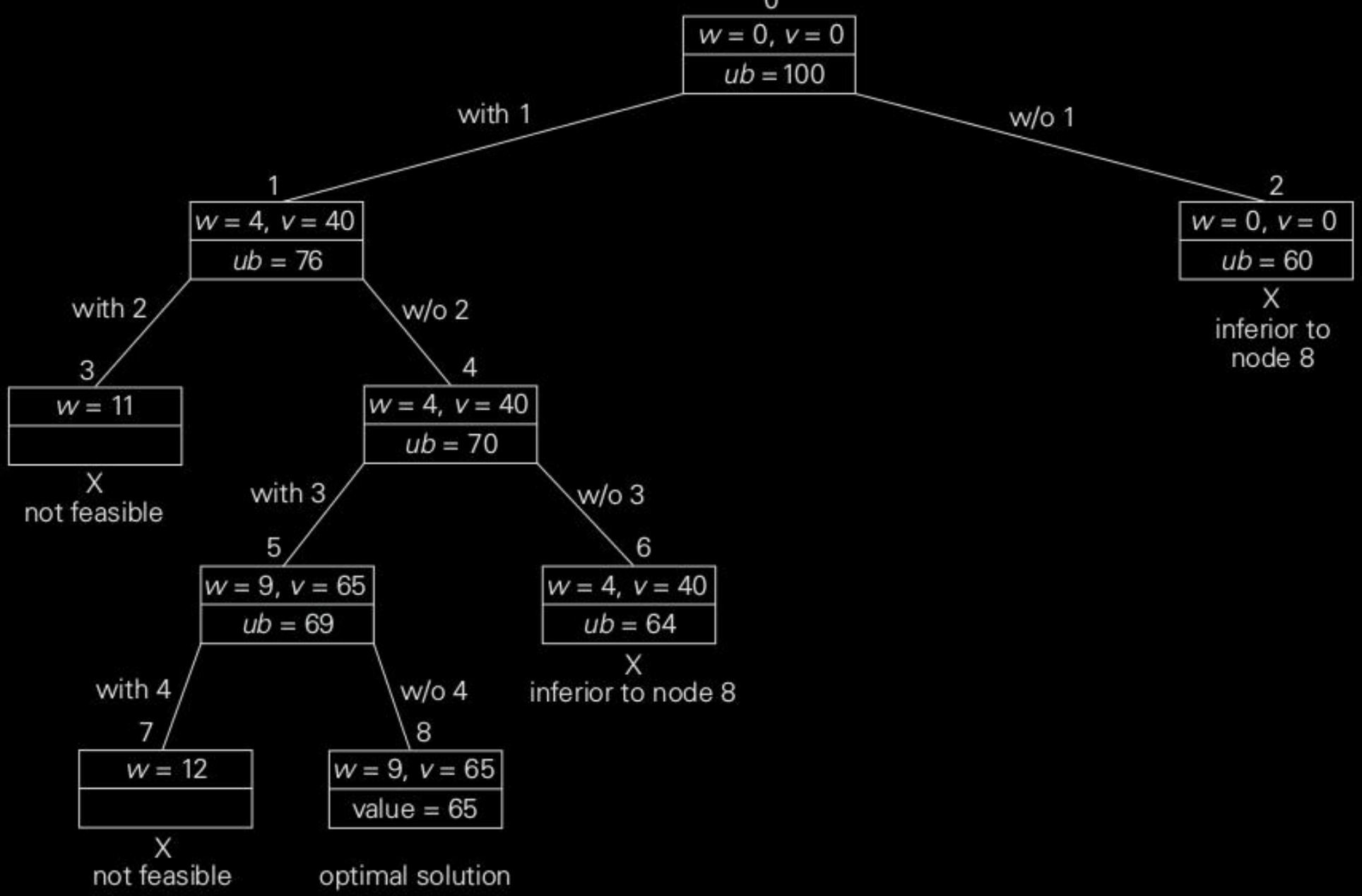


FIGURE 12.8 State-space tree of the best-first branch-and-bound algorithm for the instance of the knapsack problem.

Traveling Salesman Problem

- One very simple lower bound can be obtained by finding the smallest element in the intercity distance matrix D and multiplying it by the number of cities n
- But there is a less obvious and more informative lower bound for instances with symmetric matrix D, which does not require a lot of work to compute

Traveling Salesman Problem

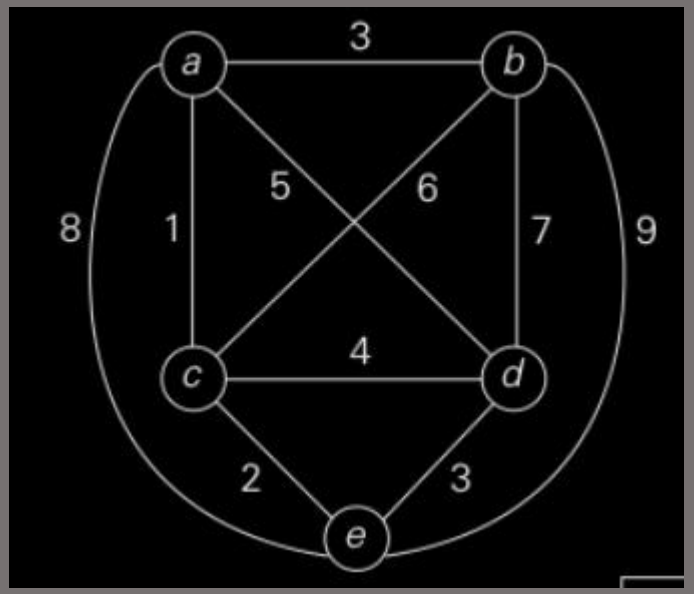
- One very simple lower bound can be obtained by finding the smallest element in the intercity distance matrix D and multiplying it by the number of cities n
- But there is a less obvious and more informative lower bound for instances with symmetric matrix D, which does not require a lot of work to compute

Traveling Salesman Problem

- For each city i, $1 \leqslant i \leqslant n$, find sum s_i of distances from city i to two nearest cities; compute sum s of these n numbers, divide result by 2, and, if all distances are integers, round up result to nearest integer:
- lb = $\lceil s/2 \rceil$

Traveling Salesman Problem

- For example, for instance in Figure 12.9a, formula (12.2) yields
- $lb = \lceil [(1 + 3) + (3 + 6) + (1 + 2) + (3 + 4) + (2 + 3)]/2 \rceil = 14$
- Find shortest Hamiltonian circuit for the graph



Restrictions to Reduce Complexity

- To reduce amount of potential work:

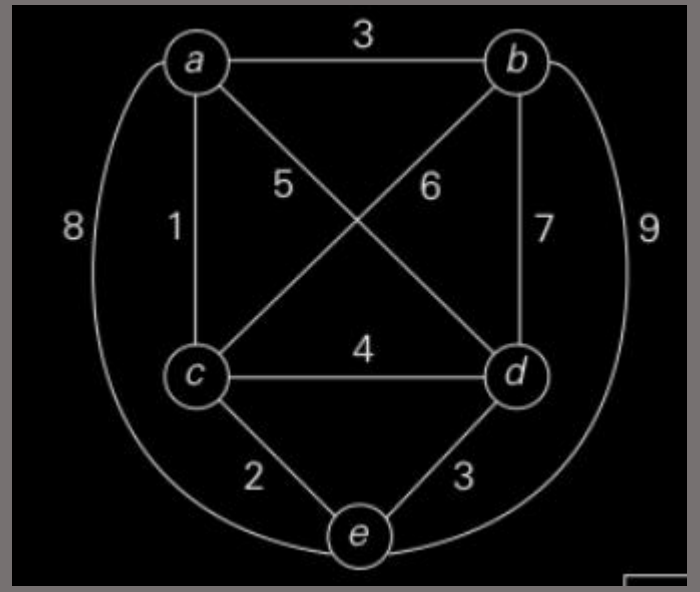
1. Consider only tours that start at vertex 'a'

2. Because our graph is undirected, we can generate only tours in which b is visited before c

3. after visiting $n - 1 = 4$ cities, a tour has no choice but to visit the remaining unvisited city and return to the starting one

Traveling Salesman Problem

- For all the Hamiltonian circuits of the graph that must include edge (a, d), we get the lower bound by summing up lengths of two shortest edges incident with each of the vertices, with required inclusion of edges (a, d) and (d, a)



- lb = ⌈[(1 + 5) + (3 + 6) + (1 + 2) + (3 + 5) + (2 + 3)]/2⌉ = 14

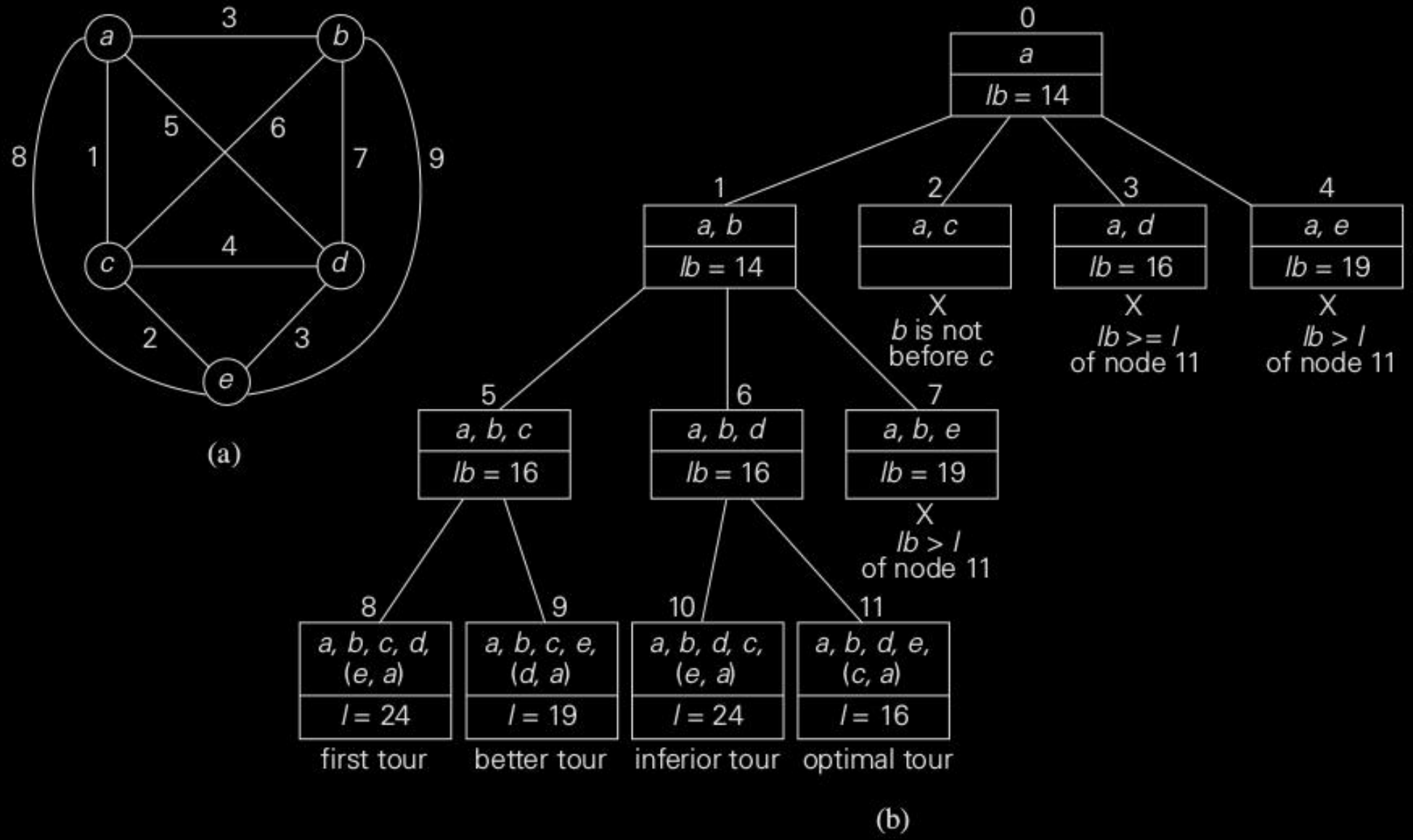


FIGURE 12.9 (a) Weighted graph. (b) State-space tree of the branch-and-bound algorithm to find a shortest Hamiltonian circuit in this graph. The list of vertices in a node specifies a beginning part of the Hamiltonian circuits represented by the node.