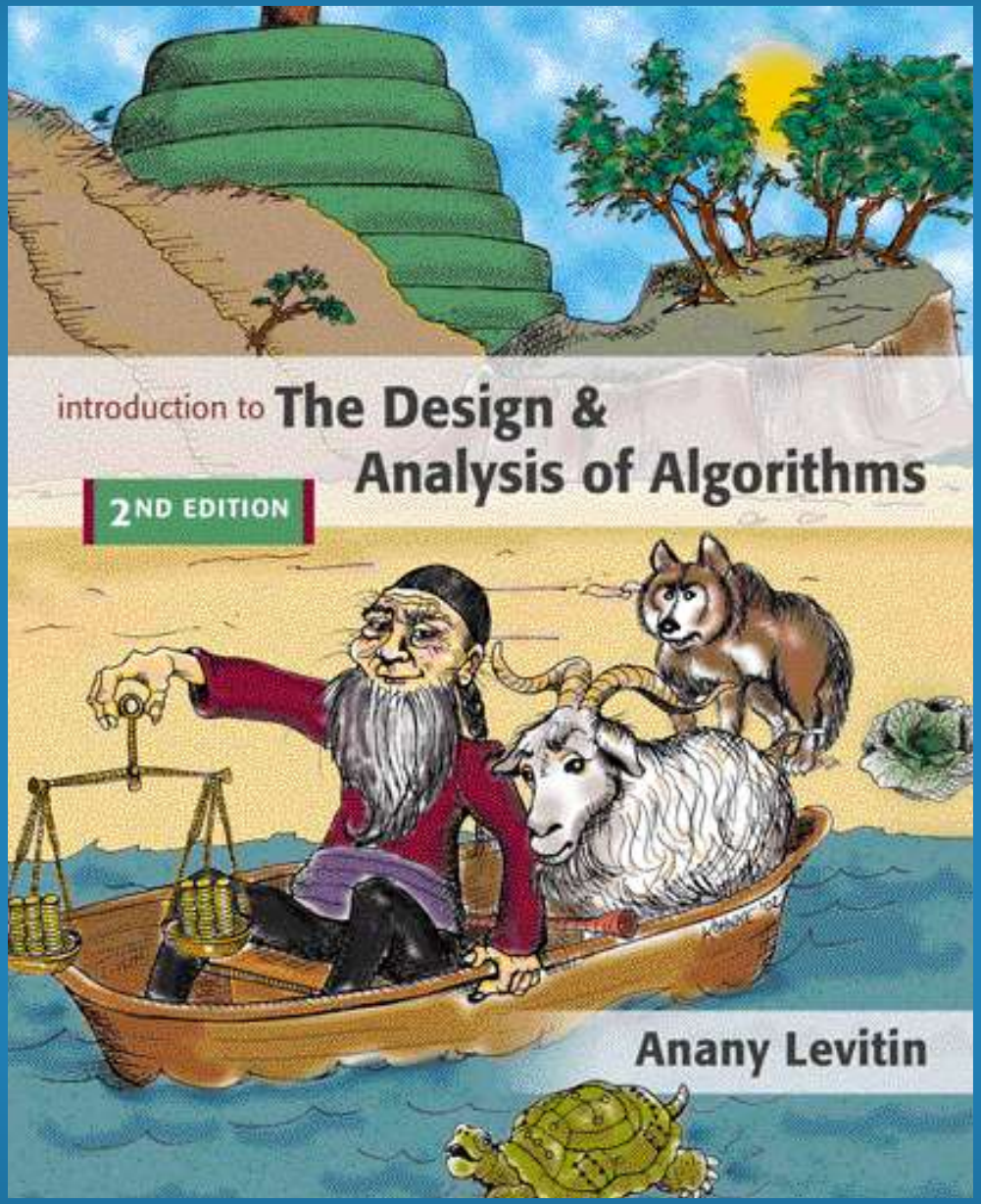
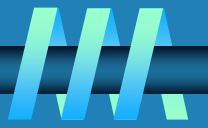
# Chapter 12

## Coping with the Limitations of Algorithm Power



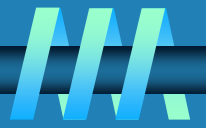
# Tackling Difficult Combinatorial Problems



There are two principal approaches to tackling difficult combinatorial problems (NP-hard problems):

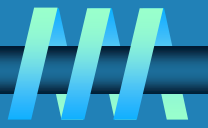
- Use a strategy that guarantees solving the problem exactly but doesn't guarantee to find a solution in polynomial time
- Use an approximation algorithm that can find an approximate (sub-optimal) solution in polynomial time

# Exact Solution Strategies



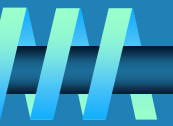
- *exhaustive search* (brute force)
  - useful only for small instances
- *dynamic programming*
  - applicable to some problems (e.g., the knapsack problem)
- *backtracking*
  - eliminates some unnecessary cases from consideration
  - yields solutions in reasonable time for many instances but worst case is still exponential
- *branch-and-bound*
  - further refines the backtracking idea for optimization problems

# Backtracking

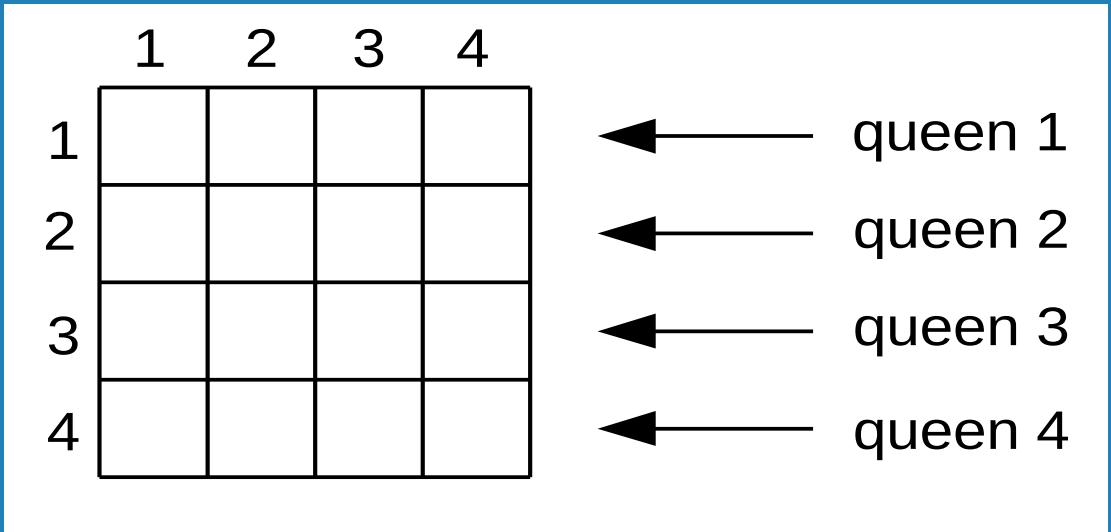


- Construct the *state-space tree*
  - nodes: partial solutions
  - edges: choices in extending partial solutions
- Explore the state space tree using depth-first search
- "Prune" *nonpromising nodes*
  - dfs stops exploring subtrees rooted at nodes that cannot lead to a solution and backtracks to such a node's parent to continue the search

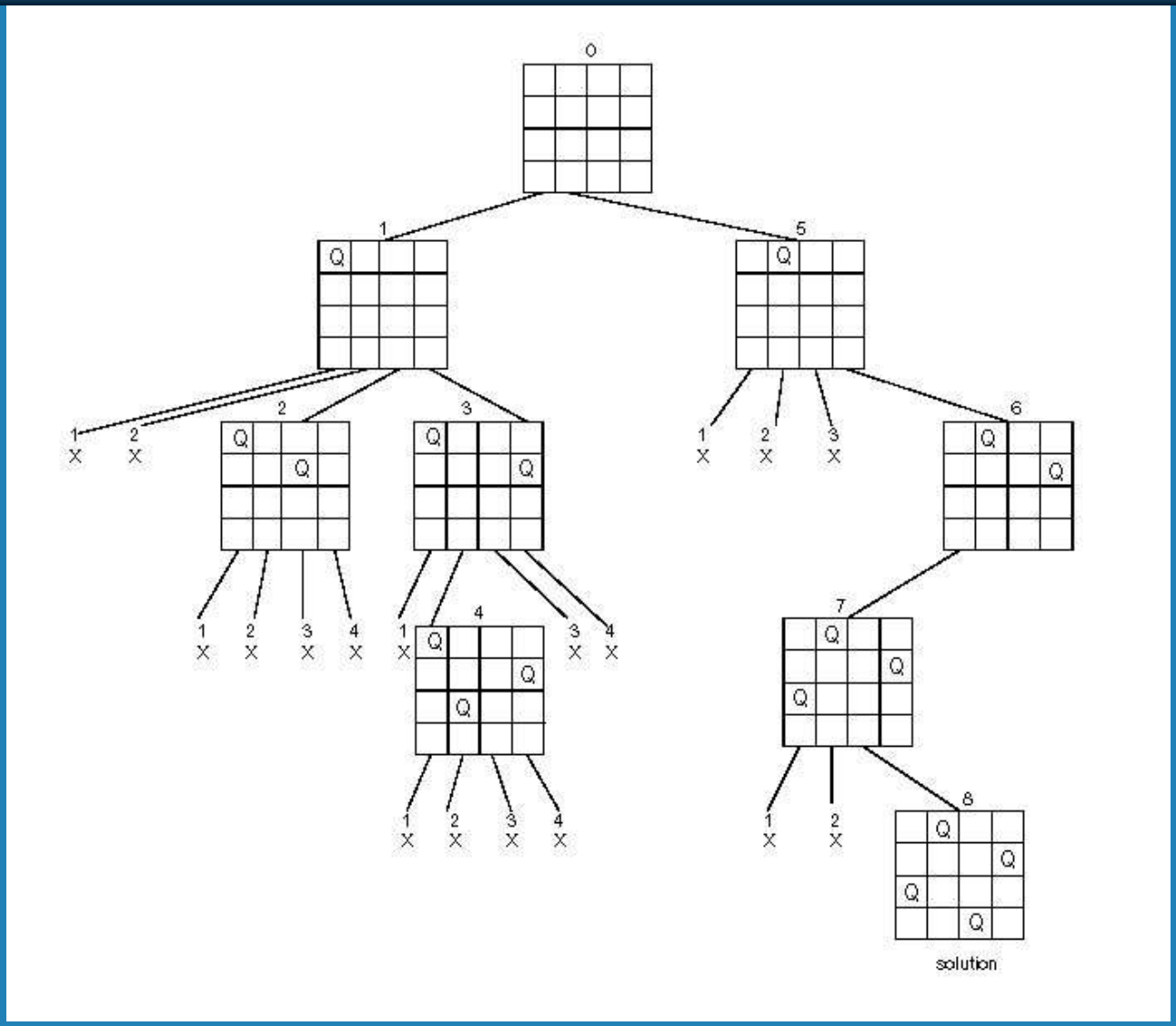
# Example: *n*-Queens Problem



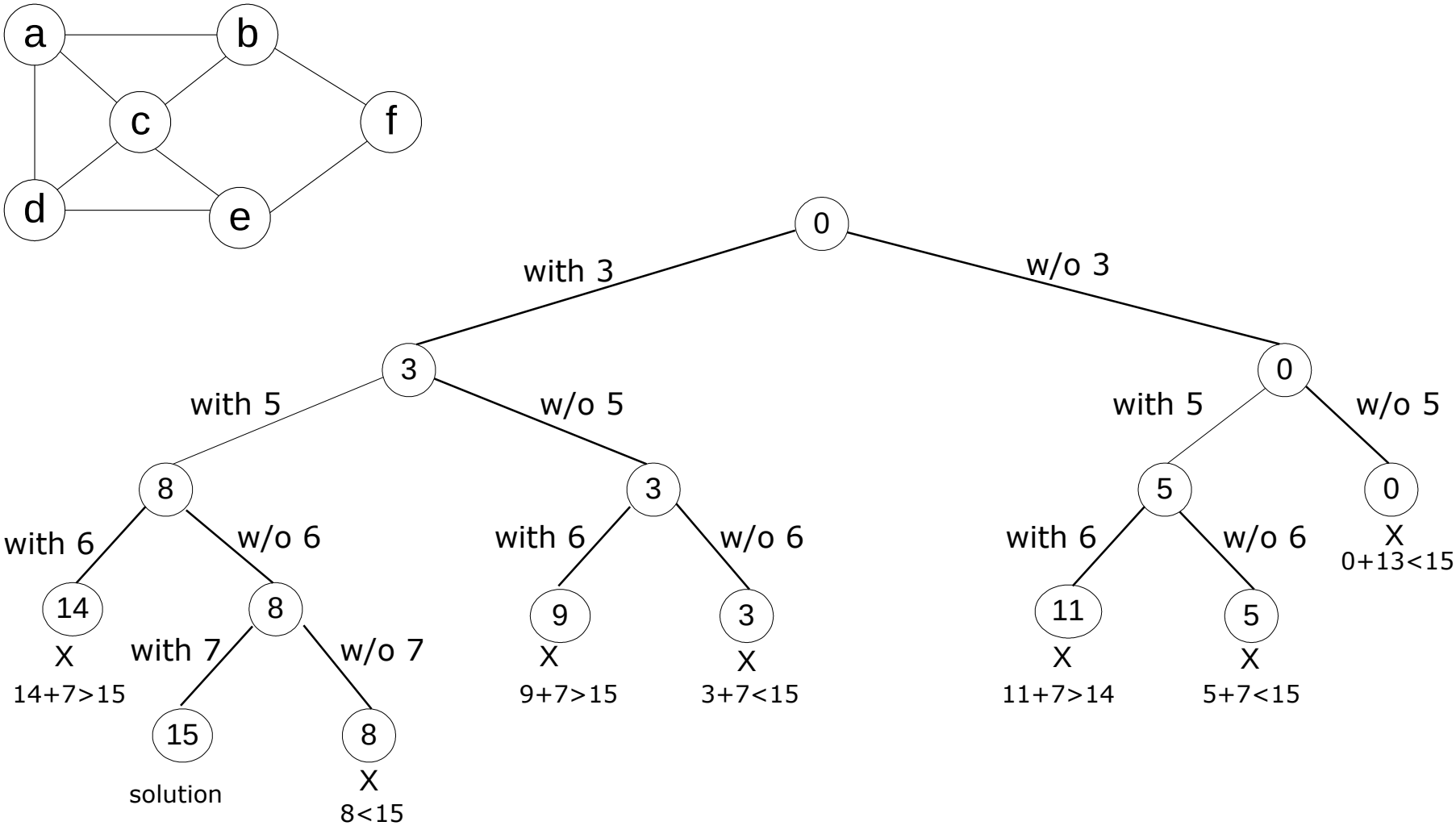
**Place *n* queens on an *n*-by-*n* chess board so that no two of them are in the same row, column, or diagonal**



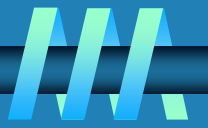
# State-Space Tree of the 4-Queens Problem



# Example: Hamiltonian Circuit Problem

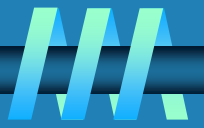


# Branch-and-Bound



- An enhancement of backtracking
- Applicable to optimization problems
- For each node (partial solution) of a state-space tree, computes a bound on the value of the objective function for all descendants of the node (extensions of the partial solution)
- Uses the bound for:
  - ruling out certain nodes as "nonpromising" to prune the tree – if a node's bound is not better than the best solution seen so far
  - guiding the search through state-space

# Example: Assignment Problem



**Select one element in each row of the cost matrix *C* so that:**
- **no two selected elements are in the same column**
- **the sum is minimized**

**Example**

| | Job 1 | Job 2 | Job 3 | Job 4 |
|---|---|---|---|---|
| Person *a* | 9 | 2 | 7 | 8 |
| Person *b* | 6 | 4 | 3 | 7 |
| Person *c* | 5 | 8 | 1 | 8 |
| Person *d* | 7 | 6 | 9 | 4 |

**<u>Lower bound</u>: Any solution to this problem will have total cost at least: 2 + 3 + 1 + 4 (or 5 + 2 + 1 + 4)**

# Example: First two levels of the state-space tree

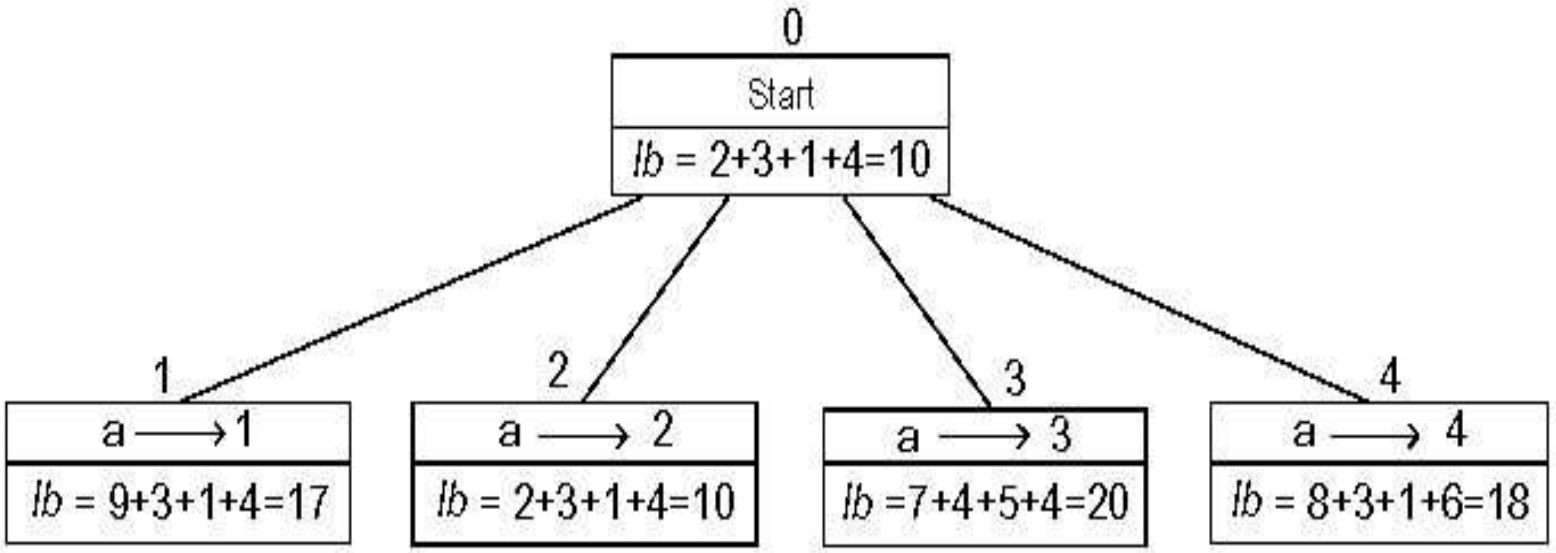



**Figure 11.5** Levels 0 and 1 of the state-space tree for the instance of the assignment problem being solved with the best-first branch-and-bound algorithm. The number above a node shows the order in which the node was generated. A node's fields indicate the job number assigned to person $a$ and the lower bound value, $lb$, for this node.

# Example (cont.)

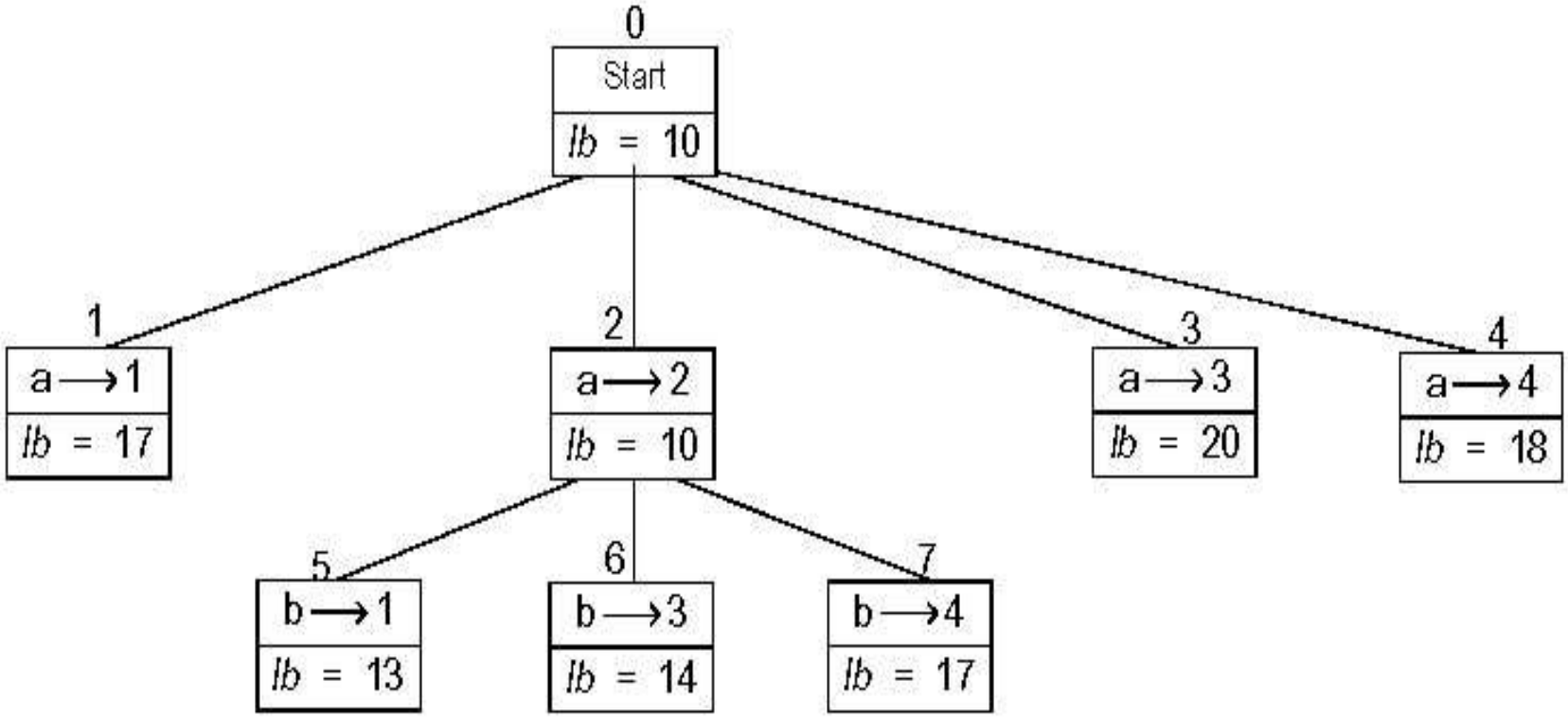



**Figure 11.6** Levels 0, 1, and 2 of the state-space tree for the instance of the assignment problem being solved with the best-first branch-and-bound algorithm

# Example: Complete state-space tree

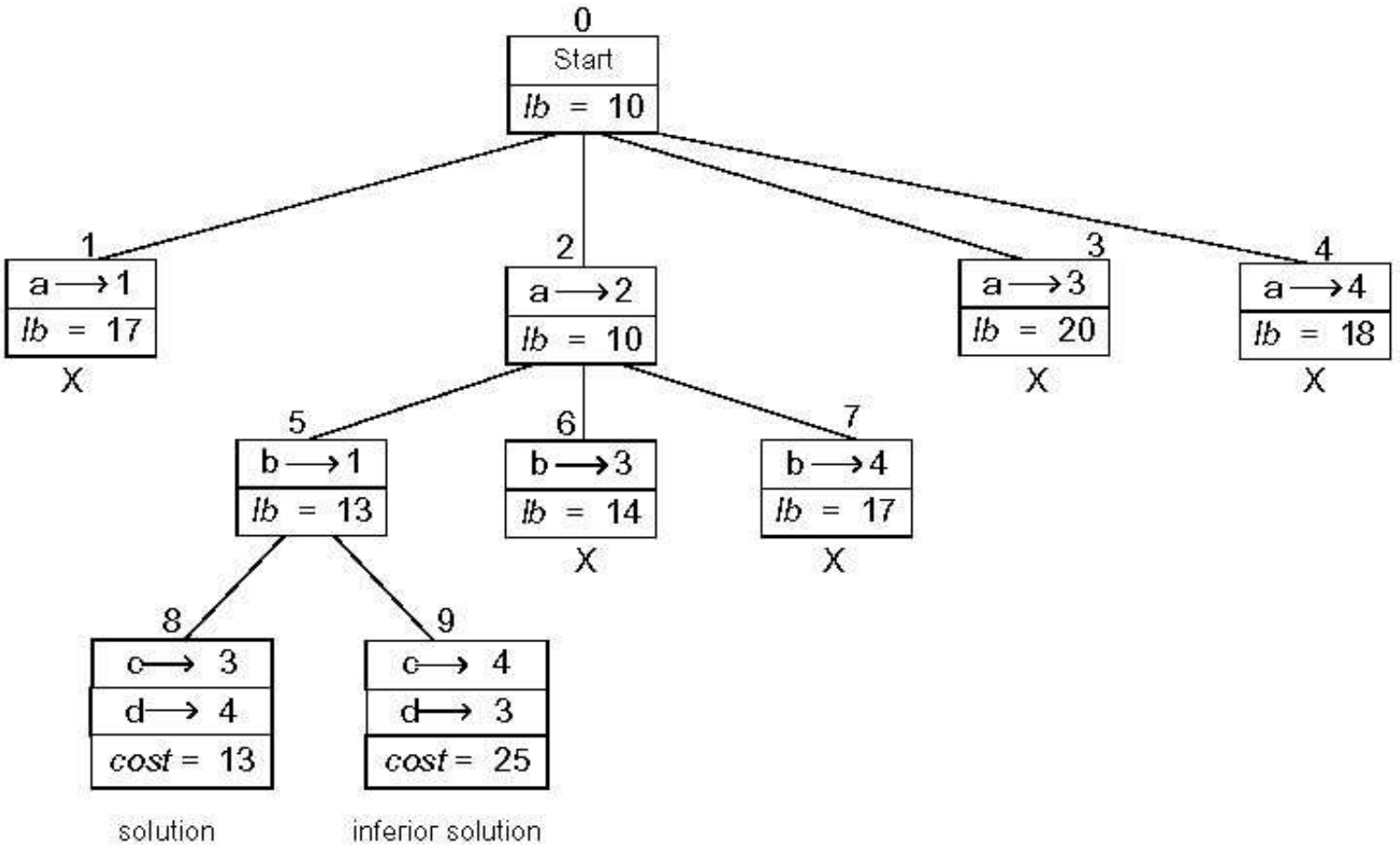



**Figure 11.7** Complete state-space tree for the instance of the assignment problem solved with the best-first branch-and-bound algorithm

# Example: Traveling Salesman Problem