Back–tracking and Branch–and–Bound

Fundamentals

- Make it possible to solve at least some large instances of difficult combinatorial problems
- Both strategies can be considered an improvement over exhaustive search
- In exhaustive search, all candidate solutions are constructed and then evaluvated

Fundamentals

- 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.

Fundamentals

- Makes it possible to solve some large instances of difficult combinatorial problems
- In worst case, still face the same curse of exponential explosion encountered in exhaustive search
- Based on construction of a state–space tree whose nodes reflect specific choices made for a solution's components

Fundamentals

- Both techniques terminate a node as soon as it can be guaranteed that no solution to the problem can be obtained by considering choices that correspond to the node's descendants
- Techniques differ in nature of problems they can be applied to

Backtracking vs Branch–and–bound

- Branch–and–bound – applicable only to optimization problems because it is based on computing a bound on possible values of the problem's objective function
- Backtracking – not constrained by this demand, but more often than not, it applies to nonoptimization problems

Backtracking vs Branch–and–bound

- Other distinction lies in the order in which nodes of the state–space tree are generated
- For backtracking, this tree is usually developed depth–first (i.e., similar to DFS)
- Branch–and–bound can generate nodes according to several rules: most natural one is so–called best–first rule

Backtracking

- Principal idea is to construct solutions one component at a time and evaluate such partially constructed candidates
- If a partially constructed solution can be developed further without violating the problem's constraints, then the first remaining legitimate option for the next component is taken

Backtracking

- 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.

Backtracking

- It is convenient to implement this kind of processing by constructing a tree of choices being made, called the state–space tree
- Its root represents an initial state before the search for a solution begins.

Backtracking

- Nodes of the first level in the tree represent the choices made for the first component of a solution, the nodes of the second level represent the choices for the second component, and so on.
- 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; otherwise, it is called nonpromising

Backtracking

- Leaves represent either nonpromising dead ends or complete solutions found by the algorithm
- In majority cases, 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.

Backtracking

- 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.

n–Queens Problem

- Place n queens on an n × 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
- For n = 1, the problem has a trivial solution, and it is easy to see that there is no solution for n = 2 and n = 3
- Let us consider the four–queens problem and solve it by the backtracking technique

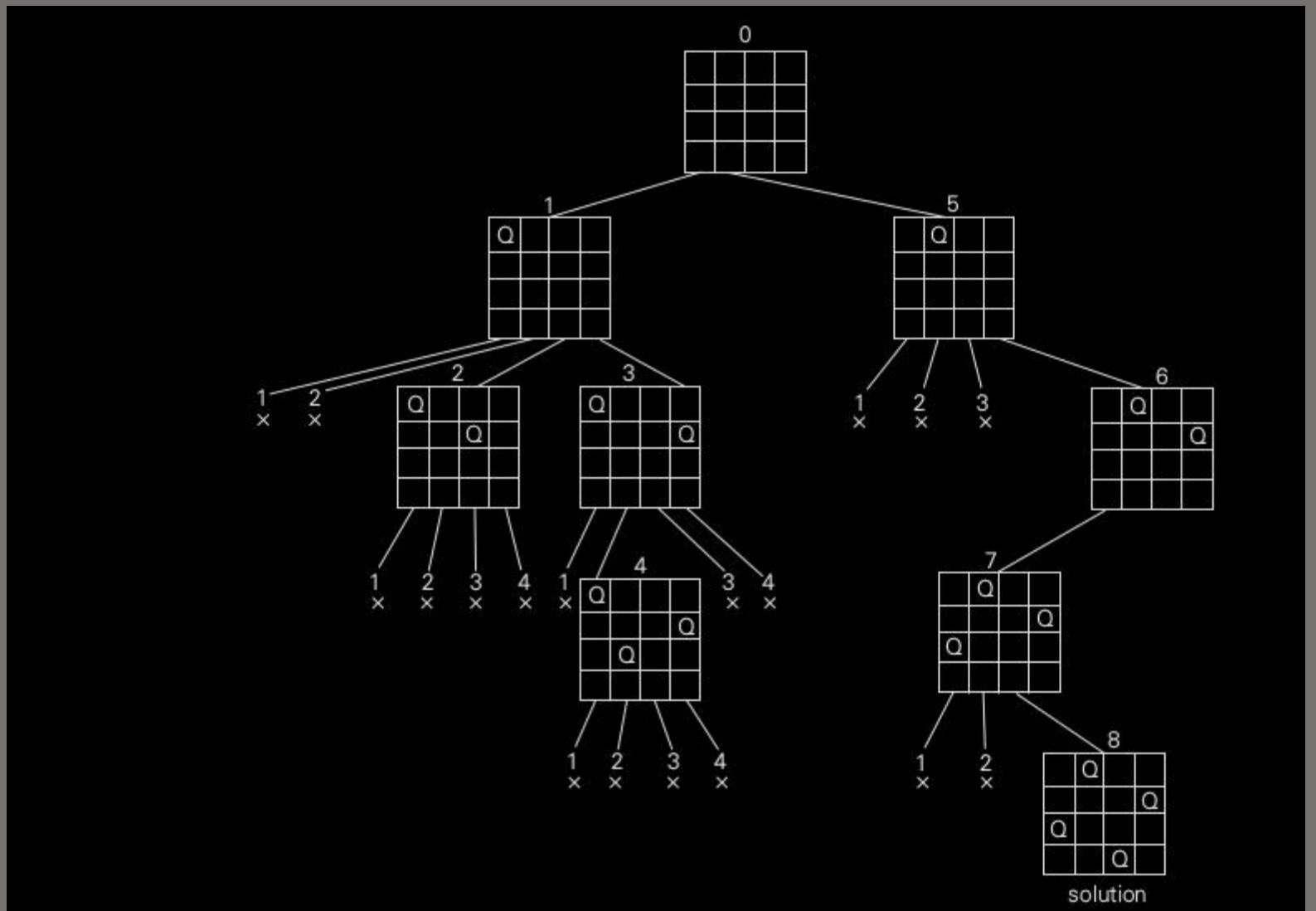

FIGURE 12.2 State-space tree of solving the four-queens problem by backtracking. × denotes an unsuccessful attempt to place a queen in the indicated column. The numbers above the nodes indicate the order in which the nodes are generated.

n–Queens Problem

```
n–queens(board, n, curr_row,  pos)
   if(curr_row>n) return true
   if sizeof(pos)<curr_row then i = 1 and i = pop(pos) +1 otherwise
   for i to n
        if board[curr_row][i] is not attacked
              push i into pos
              n–queens(board, n, curr_row+1,pos)

   n–queens(board, n, curr_row–1,pos)
```

n–Queens Problem

- If other solutions need to be found (how many of them are there for the four–queens problem?)
- algorithm can simply resume its operations at the leaf at which it stopped.
- Alternatively, we can use the board's symmetry for this purpose.

n–Queens Problem

- Finally, it should be pointed out that a single solution to the n–queens problem for any $n \geqslant 4$ can be found in linear time
- In fact, over the last 150 years mathematicians have discovered several alternative formulas for nonattacking positions of n queens
- Such positions can also be found by applying some general algorithm design strategies

Hamiltonian Cycle or Circuit

- Hamiltonian Path in an undirected graph is a path that visits each vertex exactly once.
- A Hamiltonian cycle (or Hamiltonian circuit) is a Hamiltonian Path such that there is an edge (in the graph) from the last vertex to the first vertex of the Hamiltonian Path.

Hamiltonian Cycle or Circuit

- Without loss of generality, we can assume that if a Hamiltonian circuit exists, it starts at vertex 'a'
- Accordingly, we make vertex a the root of the state–space tree
- First component of our future solution, if it exists, is a first intermediate vertex of a Hamiltonian circuit to be constructed

Hamiltonian Cycle or Circuit

- Using the alphabet order to break the three–way tie among the vertices adjacent to a, we select vertex b
- From b, the algorithm proceeds to c, then to d, then to e, and finally to f, which proves to be a dead end
- So the algorithm backtracks from f to e, then to d, and then to c, which provides the first alternative for the algorithm to pursue

Hamiltonian Cycle or Circuit

- Going from c to e eventually proves useless, and the algorithm has to backtrack from e to c and then to b
- Idea is to use Depth–First Search algorithm to traverse the graph until all the vertices have been visited.
- We traverse the graph starting from a vertex (arbitrary vertex chosen as starting vertex) and

Hamiltonian Cycle or Circuit

- At any point during the traversal we get stuck (i.e., all the neighbor vertices have been visited), we backtrack to find other paths (i.e., to visit another unvisited vertex).

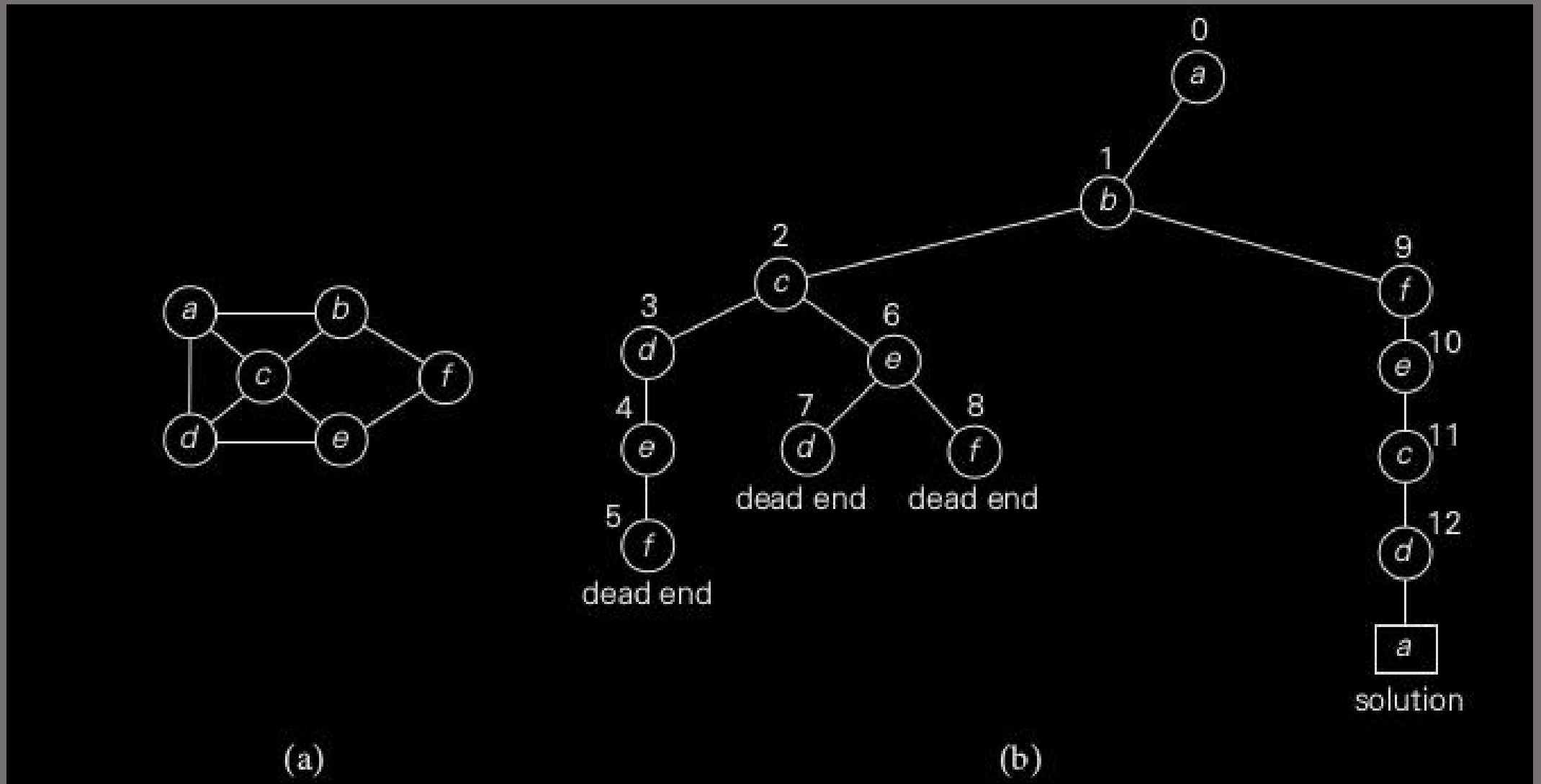

FIGURE 12.3 (a) Graph. (b) State-space tree for finding a Hamiltonian circuit. The numbers above the nodes of the tree indicate the order in which the nodes are generated.

Subset–Sum Problem

- We consider the subset–sum problem: find a subset of a given set $A = \{a_1 ,....,a_n\}$ of n positive integers whose sum is equal to a given positive integer d
- For example, for A = {1, 2, 5, 6, 8} and d = 9, there are two solutions: {1, 2, 6} and {1, 8}
- Some instances of this problem may have no solutions

Subset–Sum Problem

- It is convenient to sort the set's elements in increasing order. So, we will assume that $a_1 < a_2 < \ldots < a_n$
- State–space tree can be constructed as a binary tree like that in Figure 12.4 for the instance $A = \{3, 5, 6, 7\}$ and $d = 15$

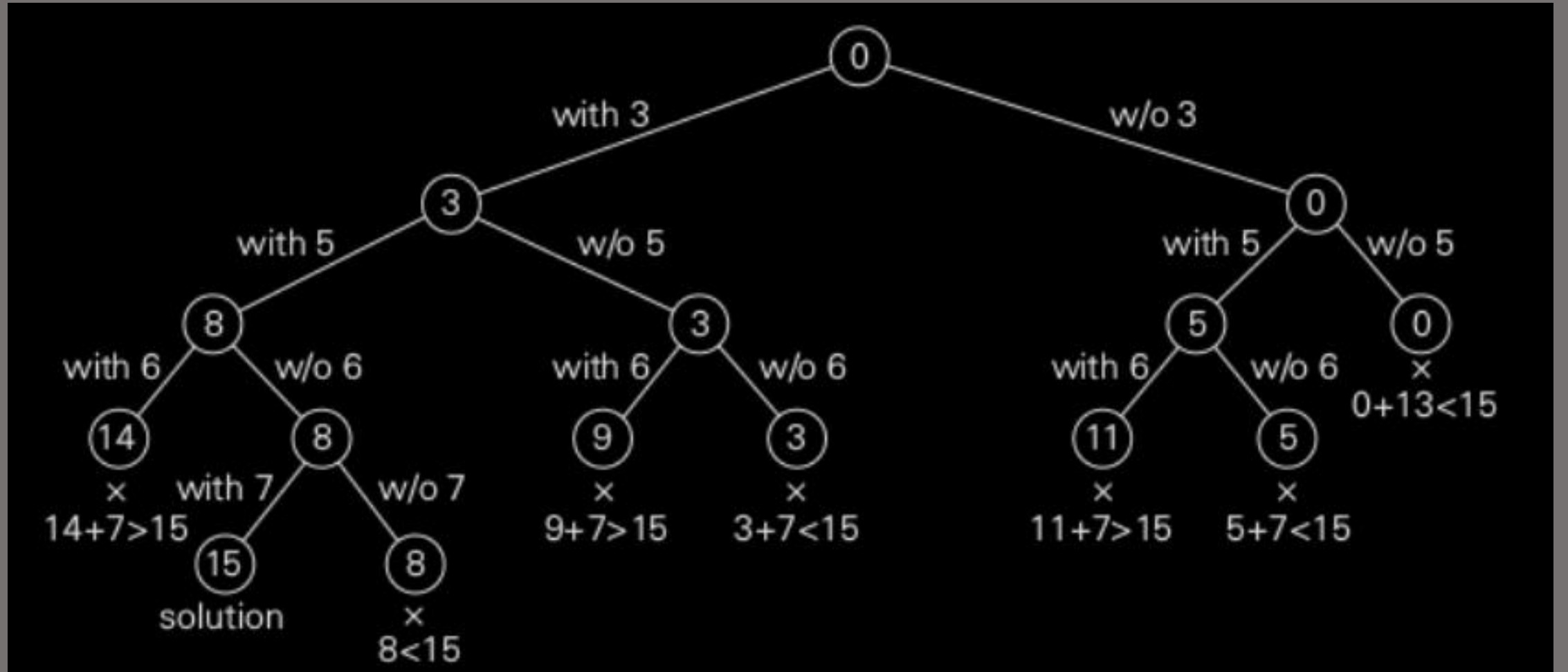

FIGURE 12.4 Complete state-space tree of the backtracking algorithm applied to the instance $A = \{3, 5, 6, 7\}$ and $d = 15$ of the subset-sum problem. The number inside a node is the sum of the elements already included in the subsets represented by the node. The inequality below a leaf indicates the reason for its termination.

Subset–Sum Problem

- The root of the tree represents the starting point, with no decisions about the given elements made as yet
- Its left and right children represent, respectively, inclusion and exclusion of a_1 in a set being sought
- Similarly, going to the left from a node of the first level corresponds to inclusion of a_2 while going to the right corresponds to its exclusion, and so on

Subset–Sum Problem

- Thus, a path from the root to a node on the ith level of the tree indicates which of the first i numbers have been included in the subsets represented by that node
- We record the value of s, the sum of these numbers, in the node. If s is equal to d, we have a solution to the problem.

Subset–Sum Problem

- We can either report this result and stop or, if all the solutions need to be found, continue by backtracking to the node's parent.
- If s is not equal to d, we can terminate the node as nonpromising if either of the following two inequalities holds:

General Remarks

- Output of a backtracking algorithm is a n–tuple (x_1 , x_2 , ... , x_n)
- tuple may need to satisfy some additional constraints (e.g., the nonattacking requirements in the n–queens problem)
- Depending on problem, all solution tuples can be of the same length (the n–queens and the Hamiltonian circuit problem) and of different lengths (the subset–sum problem)

General Remarks

- In the worst case, backtracking may have to generate all possible candidates in an exponentially growing state space
- hope is that a backtracking algorithm will be able to prune enough branches of its state–space tree before running out of time or memory or both

General Remarks

- Success of this strategy is known to vary widely, not only from problem to problem but also from one instance to another of the same problem
- There are several tricks that might help to reduce the size of a state–space tree
- One is to exploit the symmetry often present in combinatorial problems

General Remarks

- Eg: board of n–queens problem has several symmetries so that some solutions can be obtained from others by reflection or rotation
- we need not consider placements of first queen in last $\lfloor n/2 \rfloor$ columns, because any solution with first queen in square (1, i), $\lceil n/2 \rceil \leqslant i \leqslant n$, can be obtained by reflection (which?) from a solution with the first queen in square (1, n − i + 1)

General Remarks

- Another trick is to preassign values to one or more components of a solution, as we did in the Hamiltonian circuit example
- Data presorting in the subset–sum example demonstrates potential benefits of yet another opportunity: rearrange data of an instance given

Size of state–space tree

- Specifically, let c_1 be the number of values of the first component x_1 that are consistent with the problem's constraints.
- We randomly select one of these values (with equal probability $1/c_1$) to move to one of the root's c 1 children
- Repeating this operation for c_2 possible values for x_2 that are consistent with x 1 and the other constraints, we move to one of

Size of state–space tree

- We continue this process until a leaf is reached after randomly selecting values for $x_1, x_2, \ldots, x_n$
- By assuming that the nodes on level i have c_i children on average, we estimate the number of nodes in the tree as
- $1 + c_1 + c_1c_2 + \ldots + c_1c_2 \ldots c_n$

Size of state–space tree

- Generating several such estimates and computing their average yields a useful estimation of the actual size of the tree, although the standard deviation of this random variable can be large.

Backtracking in a Nutshell

- Applied to difficult combinatorial problems for which no efficient algorithms for finding exact solutions possibly exist
- Backtracking at least holds a hope for solving some instances of nontrivial sizes in an acceptable amount of time
- backtracking does not eliminate any elements of a problem's state space and ends up generating all its elements, it provides a specific technique for doing so,