because it caused the greatest increase in the lower bound of the right subtree. This heuristic is useful because it is faster to find the solution by following the left edges, which reduce the dimension as opposed to the right edges which merely add a new ∞ and probably more zeros. However, we did not use this heuristic when splitting at the node containing matrix C. It is left as an exercise to find the optimal solution with fewer node splittings.

From the above example, it seems that the heap is an ideal data structure to use in order to expand the node with the least cost (or maximum cost in case of maximization). Although branch-and-bound algorithms are generally complicated and hard to program, they proved to be efficient in practice.

13.6 Exercises

13.1. Let \(k\)-COLORING be a generalization of the 3-COLORING problem presented in Sec. 13.2. How many nodes are generated by its corresponding backtracking algorithm in the worst case?

13.2. Consider the algorithm for 3-COLORING presented in Sec. 13.2. Give an efficient algorithm to test whether a vector corresponding to a 3-coloring of a graph is legal.

13.3. Consider the algorithm for 3-COLORING presented in Sec. 13.2. Explain how to efficiently test whether the current vector is partial throughout the execution of the algorithm.

13.4. Let Algorithm \(n\)-queens be a generalization of Algorithm 4-QUEENS presented in Sec. 13.3 for the case of an \(n \times n\) chessboard. How many nodes are generated by Algorithm \(n\)-queens in the worst case?

13.5. Show that two queens placed at positions \(x_i\) and \(x_j\) are in the same diagonal if and only if

\[x_i - x_j = i - j \text{ or } x_i - x_j = j - i.\]


13.7. Does the \(n\)-queen problem have a solution for every value of \(n \geq 4\)? Prove your answer.

13.8. Modify Algorithm 4-QUEENS so that it reduces the search space from \(4^4\) to \(4!\) as described in Sec. 13.3.

13.9. Design a backtracking algorithm to generate all permutations of the numbers 1, 2, \ldots, \(n\).
13.10. Design a backtracking algorithm to generate all $2^n$ subsets of the numbers $1, 2, \ldots, n$.

13.11. Write a backtracking algorithm to solve the knight tour problem: Given an $8 \times 8$ chessboard, decide if it is possible for a knight placed at a certain position of the board to visit every square of the board exactly once and return to its start position.

13.12. Write a backtracking algorithm to solve the following variant of the partition problem (see Example 13.3): Given $n$ positive integers $X = \{x_1, x_2, \ldots, x_n\}$ and a positive integer $y$, does there exist a subset $Y \subseteq X$ whose elements sum up to $y$?

13.13. Give a backtracking algorithm to solve the Hamiltonian cycle problem: Given an undirected graph $G = (V, E)$, determine whether it contains a simple cycle that visits each vertex exactly once.

13.14. Consider the knapsack problem defined in Sec. 7.6. It was shown that using dynamic programming, the problem can be solved in time $\Theta(nC)$, where $n$ is the number of items and $C$ is the knapsack capacity.

(a) Give a backtracking algorithm to solve the knapsack problem.

(b) Which technique is more efficient to solve the knapsack problem: backtracking or dynamic programming? Explain.

13.15. Give a backtracking algorithm to solve the money change problem defined in Exercise 7.30.


13.17. Give a backtracking algorithm to solve the assignment problem defined as follows. Given $n$ employees to be assigned to $n$ jobs such that the cost of assigning the $i$th person to the $j$th job is $c_{i,j}$, find an assignment that minimizes the total cost. Assume that the cost is nonnegative, that is, $c_{i,j} \geq 0$ for $1 \leq i, j \leq n$.

13.18. Modify the solution of the instance of the traveling salesman problem given in Sec. 13.5 so that it results in fewer node splittings.

13.19. Apply the branch-and-bound algorithm for the traveling salesman problem discussed in Sec. 13.5 on the instance

\[
\begin{bmatrix}
\infty & 5 & 2 & 10 \\
2 & \infty & 5 & 12 \\
3 & 7 & \infty & 5 \\
8 & 2 & 4 & \infty
\end{bmatrix}
\]

13.20. Consider again the knapsack problem defined in Sec. 7.6. Use branch and bound and a suitable lower bound to solve the instance of this problem in Example 7.6.
13.21. Carry out a branch-and-bound procedure to solve the following instance of the assignment problem defined in Exercise 13.17. There are four employees and four jobs. The cost function is represented by the matrix below. In this matrix, row \( i \) corresponds to the \( i \)th employee, and column \( j \) corresponds to the \( j \)th job.

\[
\begin{bmatrix}
3 & 5 & 2 & 4 \\
6 & 7 & 5 & 3 \\
3 & 7 & 4 & 5 \\
8 & 5 & 4 & 6
\end{bmatrix}.
\]

13.7 Bibliographic notes

There are several books that cover backtracking in some detail. These include Brassard and Bratley (1988), Horowitz and Sahni (1978), Reingold, Nievergelt and Deo (1977). It is also described in Golomb and Brumert (1965). Techniques for analyzing its efficiency are given in Knuth (1975). The recursive form of backtracking was used by Tarjan (1972) in various graph algorithms. Branch-and-bound techniques have been successfully used in optimization problems since the late 1950s. Many of the diverse applications are outlined in the survey paper by Lawler and Wood (1966). The approach to solve the TRAVELING SALESMAN problem in this chapter is due to Little, Murty, Sweeney and Karel (1963). Another technique to solve the TRAVELING SALESMAN problem is described in the survey paper by Bellmore and Nemhauser (1968).