way to locate something in the British Museum is to explore the entire museum (Winston, 1992). The above algorithm simply searches the entire search space.

Our focus in this chapter will be to search as little of the space as possible, while guaranteeing the optimal solution.

5.2 Branch & Bound

The intuition behind *Branch & Bound* (*B&B*) is as follows.

- Organize a search space that does not preclude any solution. This could be the state space, in which a partial sequence of moves is extended in each move. This could also be a refinement space in which an abstract solution is refined.
- Continue looking for a solution by refining the cheapest candidate until
  - A complete solution is found
  - No candidate solution, partial or complete, with an estimated cost smaller than that of the complete solution exists

A high level algorithm is given in Figure 5.2. Here the task is to find the lowest cost path from a start node to a goal node, and the algorithm extends the cheapest cost partial solution at each stage.

```
B&B Procedure()
  1 open ← {(start, NIL, Cost(start))}
  2 closed ← {}
  3  while TRUE
  4   do if Empty(open)
  5         then return FAILURE
  6      Pick cheapest node n from open
  7      if GoalTest(Head(n))
  8         then return ReconstructPath(n)
  9         else  children ← MoveGen(Head(n))
 10               for each m ∈ children
 11                 do Cost(m) ← Cost(n) + K(m, n)
 12                    Add (m, Head(n), Cost(m)) to open
```

FIGURE 5.2 Branch and Bound extends the cheapest solution till the cheapest solution reaches the goal.

The basic idea behind *B&B* is to *ignore* those regions of a search space that are *known* not to contain a better solution. We look at an example in which the cost of a move corresponds to the length of an edge in a graph. Then the least cost solution corresponds to the shortest path in the graph. Let the graph in Figure 5.3 represent a tiny search space to illustrate the algorithm.

B&B begins with the start node *S*. The partial cost of *S* is zero. It expands *S*, generating partial paths *S-A* with cost 3, *S-B* with cost 4 and

S-C with cost 8. These paths are stored in the list *OPEN*. *S* is transferred to list *CLOSED*, shown shaded in Figure 5.4.

B&B continues extending the cheapest partial path. It terminates when the goal node is picked for expansion.

The example in Figure 5.4 shows *Branch & Bound* extending partial solutions. This is an example of state space search. We have seen earlier (Chapter 3) that when the search space is made up of candidate solutions, we search in the solution space.

If the candidate solution is only partially specified then we can think of it as a set of solutions that share the specified part. A refinement operator partitions this set into two sets by specifying another component of the solution. Each (complete) candidate from the set is some complete refinement of the partially specified solution. Search, then involves, decisions amongst the different possible refinements of a given partial solution (Kambhampati, 1997). That is, search involves choosing a refinement of some candidate by specifying more information. It is interesting to note that the state space search algorithms seen earlier are a special case of refinement, in which the partial solution specifies the path from the start node to some node *n*, and the choice is between different extensions of the partial path.

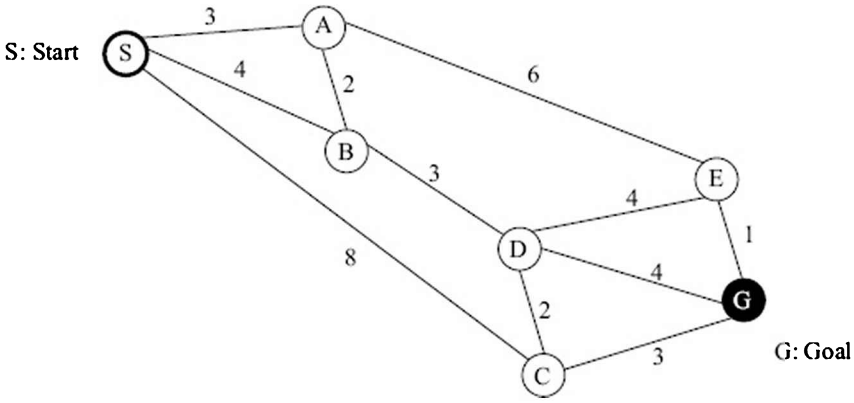
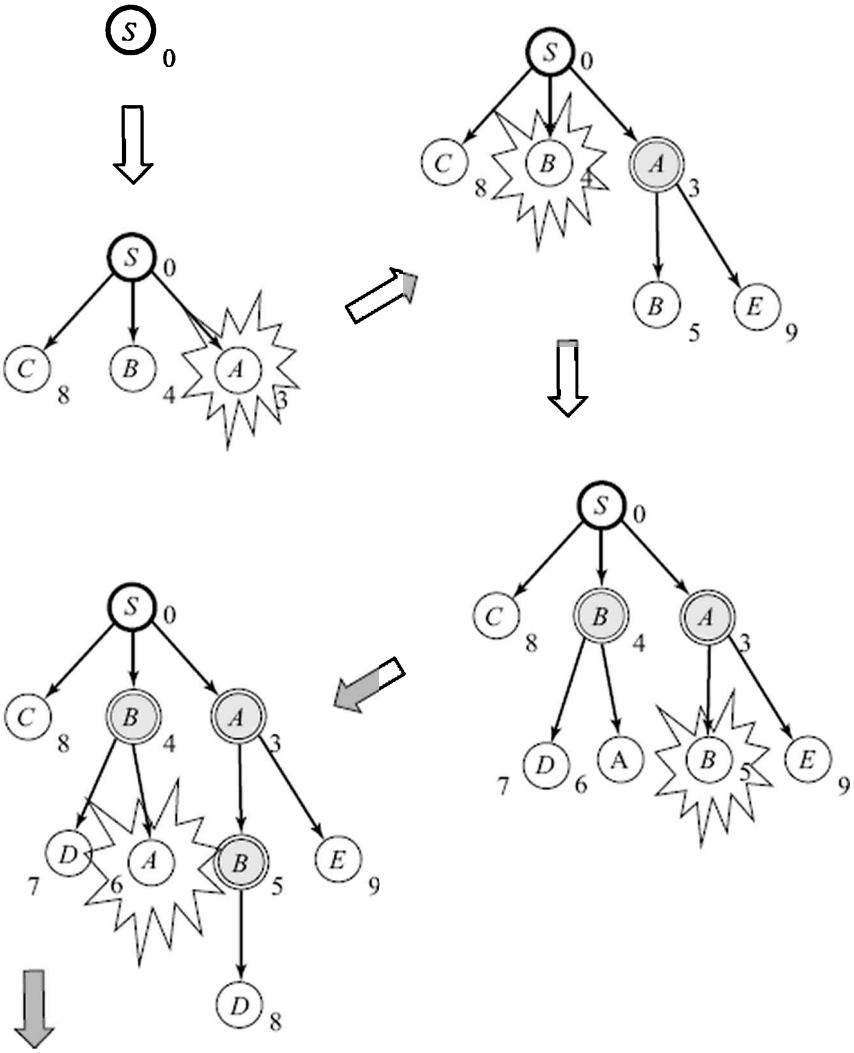
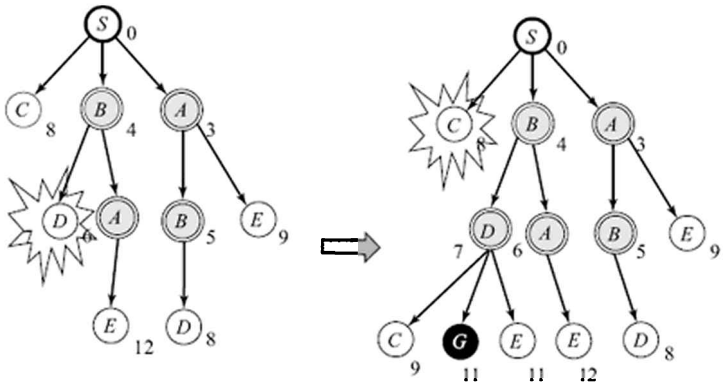
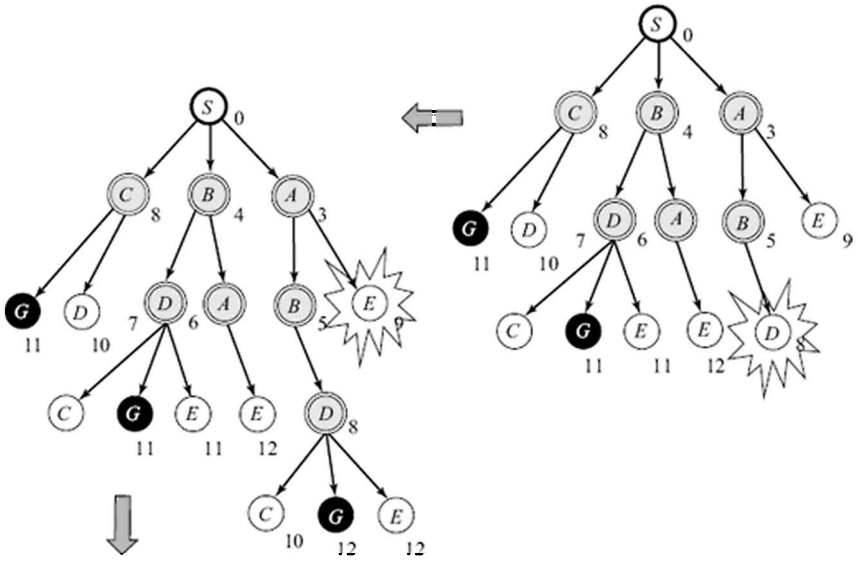


FIGURE 5.3 A tiny search graph.





Observe that even though a path to G has been found $B\&B$ continues by expanding C which is cheaper.



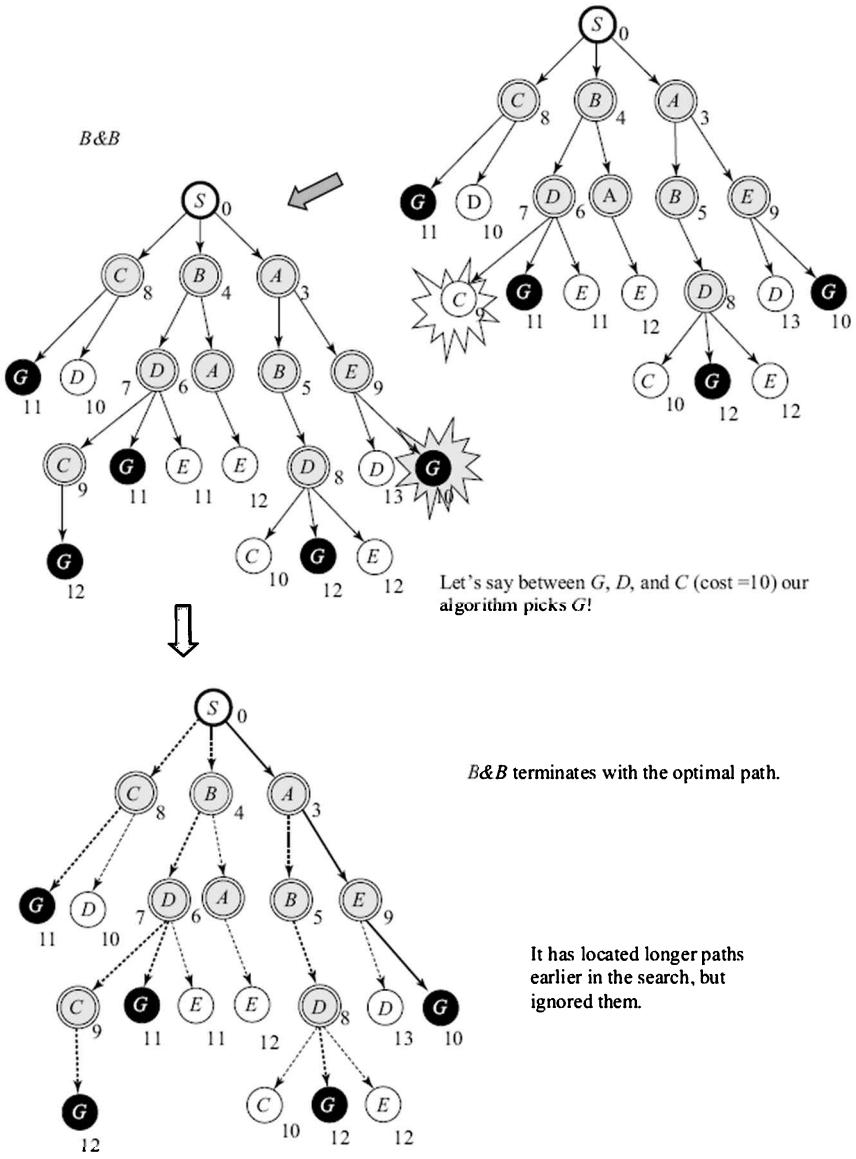


Figure 5.4 Branch and Bound explores the tiny search graph.

We consider the travelling salesman problem (TSP) again to observe the *Branch & Bound* algorithm in a refinement space.

5.3 Refinement Search

Refinement search begins with the *set* of all candidate solutions, represented by the root node. Each new node in the search tree