Chapter-3

**Bread-first-search:** Breadth-first search is a simple strategy in which the root node is expanded first, then all the BREADTH-FIRST SEARCH successors of the root node are expanded next, then their successors, and so on. In general, all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded. It use FIFO queue. Thus, breadth-first search always has the shallowest path to every node on the frontier. How does breadth-first search rate according to the four criteria from the previous section? We can easily see that it is complete—if the shallowest goal node is at some finite depth d, breadth-first search will eventually find it after generating all shallower nodes (provided the branching factor b is finite). Note that as soon as a goal node is generated, we know it is the shallowest goal node because all shallower nodes must have been generated already and failed the goal test. Now, the shallowest goal node is not necessarily the optimal one. technically, breadth-first search is optimal if the path cost is a nondecreasing function of the depth of the node. The most common such scenario is that all actions have the same cost. So far, the news about breadth-first search has been good. The news about time and space is not so good. Imagine searching a uniform tree where every state has b successors. The root of the search tree generates b nodes at the first level, each of which generates b more nodes, for a total of b2 at the second level. Each of these generates b more nodes, yielding b3 nodes at the third level, and so on. Now suppose that the solution is at depth d. In the worst case, it is the last node generated at that level. Then the total number of nodes generated is b + b2 + b3 + ··· + bd = O(bd) . (If the algorithm were to apply the goal test to nodes when selected for expansion, rather than when generated, the whole layer of nodes at depth d would be expanded before the goal was detected and the time complexity would be O(bd+1).) As for space complexity: for any kind of graph search, which stores every expanded node in the explored set, the space complexity is always within a factor of b of the time complexity. For breadth-first graph search in particular, every node generated remains in memory. There will be O(bd−1) nodes in the explored set and O(bd) nodes in the frontier, so the space complexity is O(bd), i.e., it is dominated by the size of the frontier. Switching to a tree search would not save much space, and in a state space with many redundant paths, switching could cost a great deal of time. An exponential complexity bound such as O(bd) is scary.

Bfs is optimal if every step cost is equal. But it time and space complexity is too scary because it time and space complexity increase by O(bd). The memory requirements are a bigger problem for breadth-first search than is the execution time

**Uniform-cost search:** When all step costs are equal, breadth-first search is optimal because it always expands the shallowest unexpanded node. By a simple extension, we can find an algorithm that is optimal with any step-cost function. Instead of expanding the shallowest node, UNIFORM-COST SEARCH expands the node n with the lowest path cost g(n). This is done by storing the frontier as a priority queue ordered by g.

Both of these modifications come into play in the example shown in Figure 3.15, where the problem is to get from Sibiu to Bucharest. The successors of Sibiu are Rimnicu Vilcea and Fagaras, with costs 80 and 99, respectively. The least-cost node, Rimnicu Vilcea, is expanded next, adding Pitesti with cost 80 + 97 = 177. The least-cost node is now Fagaras, so it is expanded, adding Bucharest with cost 99 + 211 = 310. Now a goal node has been generated, but uniform-cost search keeps going, choosing Pitesti for expansion and adding a second path Section 3.4. Uninformed Search Strategies 85 to Bucharest with cost 80+ 97+ 101 = 278. Now the algorithm checks to see if this new path is better than the old one; it is, so the old one is discarded. Bucharest, now with g-cost 278, is selected for expansion and the solution is returned.

Uniform-cost search is guided by path costs rather than depths, so its complexity is not easily characterized in terms of b and d. Instead, let C∗ be the cost of the optimal solution,7 and assume that every action costs at least . Then the algorithm’s worst-case time and space complexity is O(b1+C∗/), which can be much greater than bd. This is because uniformcost search can explore large trees of small steps before exploring paths involving large and perhaps useful steps. When all step costs are equal, b1+C∗/is just bd+1. When all step costs are the same, uniform-cost search is similar to breadth-first search, except that the latter stops as soon as it generates a goal, whereas uniform-cost search examines all the nodes at the goal’s depth to see if one has a lower cost; thus uniform-cost search does strictly more work by expanding nodes at depth d unnecessarily.

\*\* when c\*/e =e\*d/e=d ar shoman hobe tokhon Uniform-cost search bfs ar moto kaj korbe.

Depth-first search: Depth-first search always expands the deepest node in the current frontier of the search tree. DEPTH-FIRST SEARCH The progress of the search is illustrated in Figure 3.16. The search proceeds immediately to the deepest level of the search tree, where the nodes have no successors. depth-first search uses a LIFO queue

The properties of depth-first search depend strongly on whether the graph-search or tree-search version is used. The graph-search version, which avoids repeated states and redundant paths, is complete in finite state spaces because it will eventually expand every node. The tree-search version, on the other hand, is not complete—for example, in Figure 3.6 the algorithm will follow the Arad–Sibiu–Arad–Sibiu loop forever. Depth-first tree search can be modified at no extra memory cost so that it checks new states against those on the path from the root to the current node; this avoids infinite loops in finite state spaces but does not avoid the proliferation of redundant paths. In infinite state spaces, both versions fail if an infinite non-goal path is encountered. For example, in Knuth’s 4 problem, depth-first search would keep applying the factorial operator forever.

For a state space with branching factor b and maximum depth m, depth-first search requires storage of only O(bm) nodes.