

### 3.5 INFORMED (HEURISTIC) SEARCH STRATEGIES

**INFORMED SEARCH** This section shows how an **informed search** strategy—one that uses problem-specific knowledge beyond the definition of the problem itself—can find solutions more efficiently than can an uninformed strategy.

**BEST-FIRST SEARCH** The general approach we consider is called **best-first search**. Best-first search is an instance of the general TREE-SEARCH or GRAPH-SEARCH algorithm in which a node is selected for expansion based on an **evaluation function**,  $f(n)$ . The evaluation function is construed as a cost estimate, so the node with the *lowest* evaluation is expanded first. The implementation of best-first graph search is identical to that for uniform-cost search (Figure 3.14), except for the use of  $f$  instead of  $g$  to order the priority queue.

**EVALUATION FUNCTION**

The choice of  $f$  determines the search strategy. (For example, as Exercise 3.21 shows, best-first tree search includes depth-first search as a special case.) Most best-first algorithms include as a component of  $f$  a **heuristic function**, denoted  $h(n)$ :

**HEURISTIC FUNCTION**

$$h(n) = \text{estimated cost of the cheapest path from the state at node } n \text{ to a goal state.}$$

(Notice that  $h(n)$  takes a *node* as input, but, unlike  $g(n)$ , it depends only on the *state* at that node.) For example, in Romania, one might estimate the cost of the cheapest path from Arad to Bucharest via the straight-line distance from Arad to Bucharest.

Heuristic functions are the most common form in which additional knowledge of the problem is imparted to the search algorithm. We study heuristics in more depth in Section 3.6. For now, we consider them to be arbitrary, nonnegative, problem-specific functions, with one constraint: if  $n$  is a goal node, then  $h(n) = 0$ . The remainder of this section covers two ways to use heuristic information to guide search.

#### 3.5.1 Greedy best-first search

**GREEDY BEST-FIRST SEARCH**

**Greedy best-first search**<sup>8</sup> tries to expand the node that is closest to the goal, on the grounds that this is likely to lead to a solution quickly. Thus, it evaluates nodes by using just the heuristic function; that is,  $f(n) = h(n)$ .

**STRAIGHT-LINE DISTANCE**

Let us see how this works for route-finding problems in Romania; we use the **straight-line distance** heuristic, which we will call  $h_{SLD}$ . If the goal is Bucharest, we need to know the straight-line distances to Bucharest, which are shown in Figure 3.22. For example,  $h_{SLD}(In(Arad)) = 366$ . Notice that the values of  $h_{SLD}$  cannot be computed from the problem description itself. Moreover, it takes a certain amount of experience to know that  $h_{SLD}$  is correlated with actual road distances and is, therefore, a useful heuristic.

Figure 3.23 shows the progress of a greedy best-first search using  $h_{SLD}$  to find a path from Arad to Bucharest. The first node to be expanded from Arad will be Sibiu because it is closer to Bucharest than either Zerind or Timisoara. The next node to be expanded will be Fagaras because it is closest. Fagaras in turn generates Bucharest, which is the goal. For this particular problem, greedy best-first search using  $h_{SLD}$  finds a solution without ever

<sup>8</sup> Our first edition called this **greedy search**; other authors have called it **best-first search**. Our more general usage of the latter term follows Pearl (1984).

<b>Arad</b>	366	<b>Mehadia</b>	241
<b>Bucharest</b>	0	<b>Neamt</b>	234
<b>Craiova</b>	160	<b>Oradea</b>	380
<b>Drobeta</b>	242	<b>Pitesti</b>	100
<b>Eforie</b>	161	<b>Rimnicu Vilcea</b>	193
<b>Fagaras</b>	176	<b>Sibiu</b>	253
<b>Giurgiu</b>	77	<b>Timisoara</b>	329
<b>Hirsova</b>	151	<b>Urziceni</b>	80
<b>Iasi</b>	226	<b>Vaslui</b>	199
<b>Lugoj</b>	244	<b>Zerind</b>	374

**Figure 3.22** Values of  $h_{SLD}$ —straight-line distances to Bucharest.

expanding a node that is not on the solution path; hence, its search cost is minimal. It is not optimal, however: the path via Sibiu and Fagaras to Bucharest is 32 kilometers longer than the path through Rimnicu Vilcea and Pitesti. This shows why the algorithm is called “greedy”—at each step it tries to get as close to the goal as it can.

Greedy best-first tree search is also incomplete even in a finite state space, much like depth-first search. Consider the problem of getting from Iasi to Fagaras. The heuristic suggests that Neamt be expanded first because it is closest to Fagaras, but it is a dead end. The solution is to go first to Vaslui—a step that is actually farther from the goal according to the heuristic—and then to continue to Urziceni, Bucharest, and Fagaras. The algorithm will never find this solution, however, because expanding Neamt puts Iasi back into the frontier, Iasi is closer to Fagaras than Vaslui is, and so Iasi will be expanded again, leading to an infinite loop. (The graph search version *is* complete in finite spaces, but not in infinite ones.) The worst-case time and space complexity for the tree version is  $O(b^m)$ , where  $m$  is the maximum depth of the search space. With a good heuristic function, however, the complexity can be reduced substantially. The amount of the reduction depends on the particular problem and on the quality of the heuristic.

### 3.5.2 A\* search: Minimizing the total estimated solution cost

A\* SEARCH

The most widely known form of best-first search is called **A\* search** (pronounced “A-star search”). It evaluates nodes by combining  $g(n)$ , the cost to reach the node, and  $h(n)$ , the cost to get from the node to the goal:

$$f(n) = g(n) + h(n) .$$

Since  $g(n)$  gives the path cost from the start node to node  $n$ , and  $h(n)$  is the estimated cost of the cheapest path from  $n$  to the goal, we have

$$f(n) = \text{estimated cost of the cheapest solution through } n .$$

Thus, if we are trying to find the cheapest solution, a reasonable thing to try first is the node with the lowest value of  $g(n) + h(n)$ . It turns out that this strategy is more than just reasonable: provided that the heuristic function  $h(n)$  satisfies certain conditions, A\* search is both complete and optimal. The algorithm is identical to UNIFORM-COST-SEARCH except that A\* uses  $g + h$  instead of  $g$ .