```
function LRTA*-AGENT(problem, s', h) returns an action
  persistent: s, a, the previous state and action, initially null
              result, a table mapping (s, a) to s', initially empty
              H, a table mapping s to a cost estimate, initially empty

  if IS-GOAL(s') then return stop
  if s' is a new state (not in H) then H[s'] ← h(s')
  if s is not null then
      result[s, a] ← s'
      H[s] ← min_{b ∈ ACTIONS(s)} LRTA*-COST(s, b, result[s, b], H)
  a ← argmin_{b ∈ ACTIONS(s)} LRTA*-COST(problem, s', b, result[s', b], H)
  s ← s'
  return a

function LRTA*-COST(problem, s, a, s', H) returns a cost estimate
  if s' is undefined then return h(s)
  else return problem.ACTION-COST(s, a, s') + H[s']
```

Figure 4.24 LRTA*-AGENT selects an action according to the values of neighboring states, which are updated as the agent moves about the state space.

An LRTA* agent is guaranteed to find a goal in any finite, safely explorable environment. Unlike A*, however, it is not complete for infinite state spaces—there are cases where it can be led infinitely astray. It can explore an environment of n states in $O(n^2)$ steps in the worst case, but often does much better. The LRTA* agent is just one of a large family of online agents that one can define by specifying the action selection rule and the update rule in different ways. We discuss this family, developed originally for stochastic environments, in Chapter 22.

4.5.4 Learning in online search

The initial ignorance of online search agents provides several opportunities for learning. First, the agents learn a "map" of the environment—more precisely, the outcome of each action in each state—simply by recording each of their experiences. Second, the local search agents acquire more accurate estimates of the cost of each state by using local updating rules, as in LRTA*. In Chapter 22, we show that these updates eventually converge to *exact* values for every state, provided that the agent explores the state space in the right way. Once exact values are known, optimal decisions can be taken simply by moving to the lowest-cost successor—that is, pure hill climbing is then an optimal strategy.

If you followed our suggestion to trace the behavior of ONLINE-DFS-AGENT in the environment of Figure 4.19, you will have noticed that the agent is not very bright. For example, after it has seen that the *Up* action goes from (1,1) to (1,2), the agent still has no idea that the *Down* action goes back to (1,1) or that the *Up* action also goes from (2,1) to (2,2), from (2,2) to (2,3), and so on. In general, we would like the agent to learn that *Up* increases the y-coordinate unless there is a wall in the way, that *Down* reduces it, and so on.

For this to happen, we need two things. First, we need a formal and explicitly manipulable representation for these kinds of general rules; so far, we have hidden the information inside