An agent that learns utility functions must also have a model of the environment in order to make decisions, because it must know the states to which its actions will lead. For example, in order to make use of a backgammon evaluation function, a backgammon program must know what its legal moves are *and how they affect the board position*. Only in this way can it apply the utility function to the outcome states. An agent that learns an action-value function, on the other hand, need not have such a model. As long as it knows its legal moves, it can compare their values directly without having to consider their outcomes. Action-value learners therefore can be slightly simpler in design than utility learners. On the other hand, because they do not know where their actions lead, they cannot look ahead; this can seriously restrict their ability to learn, as we shall see.

We first address the problem of learning utility functions, which has been studied in AI since the earliest days of the field. (See the discussion of Samuel's checker player in Chapter 5.) We examine increasingly complex versions of the problem, while keeping initially to simple state-based representations. Section 20.6 discusses the learning of action-value functions, and Section 20.7 discusses how the learner can generalize across states. Throughout this chapter, we will assume that the environment is nondeterministic. At this point the reader may wish to review the basics of decision making in complex, nondeterministic environments, as covered in Chapter 17.

20.2 PASSIVE LEARNING IN A KNOWN ENVIRONMENT

To keep things simple, we start with the case of a passive learning agent using a state-based representation in a known, accessible environment. In passive learning, the environment generates state transitions and the agent perceives them.[1] Consider an agent trying to learn the utilities of the states shown in Figure 20.1(a). We assume, for now, that it is provided with a model M_{ij} giving the probability of a transition from state i to state j, as in Figure 20.1(b). In each **training sequence**, the agent starts in state (1,1) and experiences a sequence of state transitions until it reaches one of the terminal states (4,2) or (4,3), where it receives a reward.[2] A typical set of training sequences might look like this:

TRAINING SEQUENCE

$$(1,1)\to(1,2)\to(1,3)\to(1,2)\to(1,3)\to(1,2)\to(1,1)\to(2,1)\to(3,1)\to(4,1)\to(4,2)\ \underline{-1}$$
$$(1,1)\to(1,2)\to(1,3)\to(2,3)\to(3,3)\to(4,3)\ \underline{+1}$$
$$(1,1)\to(1,2)\to(1,1)\to(1,2)\to(1,1)\to(2,1)\to(3,1)\to(3,2)\to(4,2)\ \underline{-1}$$
$$(1,1)\to(1,2)\to(1,1)\to(1,2)\to(1,3)\to(2,3)\to(1,3)\to(2,3)\to(3,3)\to(4,3)\ \underline{+1}$$
$$(1,1)\to(2,1)\to(3,1)\to(2,1)\to(1,1)\to(1,2)\to(1,3)\to(2,3)\to(3,3)\to(4,3)\ \underline{+1}$$
$$(1,1)\to(2,1)\to(1,1)\to(1,2)\to(1,3)\to(2,3)\to(3,3)\to(3,2)\to(4,2)\ \underline{-1}$$

The object is to use the information about rewards to learn the expected utility $U(i)$ associated with each nonterminal state i. We will make one big simplifying assumption: the utility of a sequence is the sum of the rewards accumulated in the states of the sequence. That is, the utility

1 Another way to think of a passive learner is as an agent with a fixed policy trying to determine its benefits.

2 The period from initial state to terminal state is often called an **epoch**.