CHAPTER 16

Planning Based on Markov Decision Processes

16.1 Introduction

Planning based on Markov Decision Processes (MDPs) is designed to deal with nondeterminism, probabilities, partial observability, and extended goals. Its key idea is to represent the planning problem as an optimization problem. It is based on the following conventions.

- A planning domain is modeled as a *stochastic system*, i.e., a nondeterministic state-transition system that assigns probabilities to state transitions. Uncertainty about action outcomes is thus modeled with a probability distribution function.
- Goals are represented by means of *utility functions*, numeric functions that give preferences to states to be traversed and/or actions to be performed. Utility functions can express preferences on the entire execution path of a plan, rather than just desired final states.
- Plans are represented as *policies* that specify the action to execute in each state. The execution of a policy results in conditional and iterative behaviors.
- The planning problem is seen as an *optimization problem*, in which planning algorithms search for a plan that maximizes the utility function.
- Partial observability is modeled by observations that return a probability distribution over the state space, called *belief states*. The problem of planning under partial observability is reformulated as the problem of planning under full observability in the space of belief states, and the generated plans are policies that map belief states to actions.

This chapter describes MDP planning under the assumption of full observability (Section 16.2) and partial observability (Section 16.3). We also discuss planning for

reachability and extended goals (Section 16.4). The chapter ends with discussion and exercises.

16.2 Planning in Fully Observable Domains

Under the hypothesis of full observability, we first introduce stochastic systems, policies, and utility functions and formalize planning as an optimization problem (Section 16.2.1). We then present and discuss some basic MDP planning algorithms: policy, value, and real-time iteration (Section 16.2.2).

16.2.1 Domains, Plans, and Planning Problems

Domains as Stochastic Systems. A stochastic system is a nondeterministic state-transition system with a probability distribution on each state transition. It is a tuple $\Sigma = (S, A, P)$, where:

- S is a finite set of states.
- A is a finite set of actions.
- $P_a(s'|s)$, where $a \in A$, s and $s' \in S$, and P is a probability distribution. That is, for each $s \in S$, if there exists $a \in A$ and $s' \in S$ such that $P_a(s'|s) \neq 0$, we have $\sum_{s' \in S} P(s, a, s') = 1$.

$P_a(s'|s)$ is the probability that if we execute an action a in a state s, then a will lead to state s'. We call $A(s) = \{a \in A \mid \exists s' \in S \,.\, P_a(s'|s) \neq 0\}$ the set of *executable actions* in s, i.e., the set of actions that have probability different than 0 to have a state transition. We do not restrict to the case in which we have at least one action that is executable in each state, i.e., we do not necessarily have that $\forall s \in S\ A(s) \neq \emptyset$. In a state s where a's preconditions are not satisfied, we have that $\forall s' \in S\ P_a(s, s') = 0$.

Example 16.1 Figure 16.1 shows a stochastic system that represents a DWR domain in which there is a single robot r1 that can move among five different locations (l1, l2, l3, l4, and l5). We have two sources of nondeterminism. First, when moving from location l2 to l3, there is a probability of 0.8 that the robot does it correctly and a probability of 0.2 that it gets confused and ends up in location l5. Second, when the robot tries to move from location l1 to l4, there is a probability of 0.5 that the way will be closed, in which case the robot will remain in l1. In the stochastic system, s1, s2, s3, s4, and s5 are the states in which the robot is at location l1, l2, l3, l4, and l5, respectively. We assume there is an action wait (not represented in the figure) that leaves each state of the domain unchanged. The set of states of the stochastic system is $S = \{s1, s2, s3, s4, s5\}$. The actions are move(r1,li,lj) with $i, j = 1, 2, 3, 4, 5$ and $i \neq j$, and wait. We have probabilities $P_{\text{move(r1,l1,l2)}}(\text{s2}|\text{s1}) = 1$, $P_{\text{move(r1,l2,l3)}}(\text{s3}|\text{s2}) = 0.8$, and so on. ■

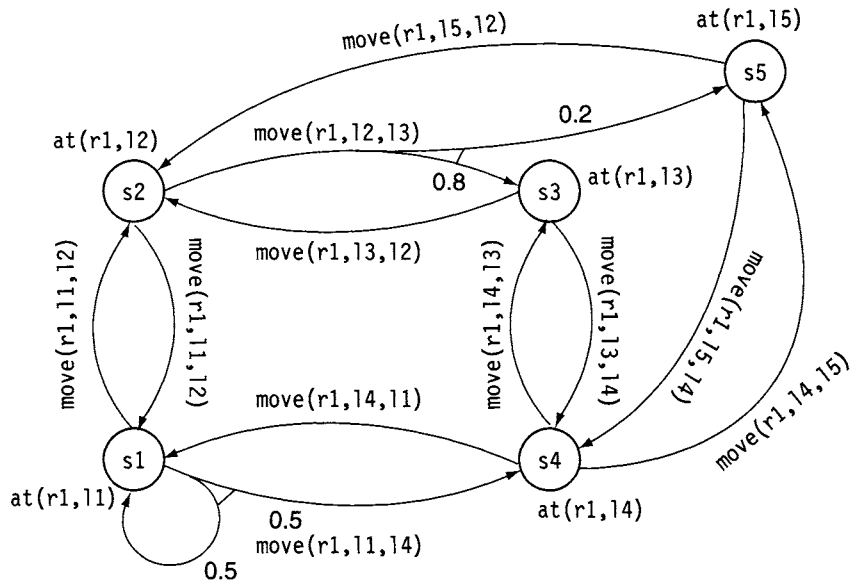

Figure 16.1 A stochastic system. There are five states (s1, s2, s3, s4, and s5), one for each location (l1, l2, l3, l4, and l5, respectively). In the figure, we label only arcs that have positive probability and the action wait such that, for each s, $P_{wait}(s|s) = 1$ is not represented. There are two nondeterministic actions: move(r1,l2,l3) and move(r1,l1,l4).

Plans as Policies. A plan specifies the actions that a controller should execute in a given state. A plan can thus be represented as a *policy* π, i.e., a total function mapping states into actions:

$$\pi : S \rightarrow A$$

This policy is generated by the planner and is given in input to a controller that executes actions one by one and observes into which state each action puts the system (see Figure 16.2).

Example 16.2 In the domain in Figure 16.1, consider the following policies.

$$\begin{aligned}\pi_1 = \{&(\text{s1}, \text{move(r1,l1,l2)}),\\ &(\text{s2}, \text{move(r1,l2,l3)}),\\ &(\text{s3}, \text{move(r1,l3,l4)}),\\ &(\text{s4}, \text{wait}),\\ &(\text{s5}, \text{wait})\}\end{aligned}$$

```
Execute-Policy(π)
    Loop
        sense the current state s
        a ← π(s)
        execute action a
end
```

Figure 16.2 Policy execution.

$$\pi_2 = \{(\mathsf{s1}, \mathsf{move(r1,l1,l2)}),$$
$$(\mathsf{s2}, \mathsf{move(r1,l2,l3)}),$$
$$(\mathsf{s3}, \mathsf{move(r1,l3,l4)}),$$
$$(\mathsf{s4}, \mathsf{wait}),$$
$$(\mathsf{s5}, \mathsf{move(r1,l5,l4)})\}$$

$$\pi_3 = \{(\mathsf{s1}, \mathsf{move(r1,l1,l4)}),$$
$$(\mathsf{s2}, \mathsf{move(r1,l2,l1)}),$$
$$(\mathsf{s3}, \mathsf{move(r1,l3,l4)}),$$
$$(\mathsf{s4}, \mathsf{wait}),$$
$$(\mathsf{s5}, \mathsf{move(r1,l5,l4)})\}$$

All three policies try to move the robot toward state s4. π_1 waits forever in s5 if execution gets to that point. π_2 is "safer": in the unlikely case that execution leads to state s5, it moves the robot to s4. π_3 tries to move the robot to s4 through state s1. π_2 takes a longer but "safer" route, and π_3 takes the shorter route that is "riskier" in the sense that the execution of π_3 may get stuck in s1 in the unlikely case that all the infinite attempts to move to l4 do not succeed. ■

Note that a policy does not allow sequencing, e.g., there is no policy equivalent to the sequential plan that executes move(r1,l1,l4) followed by move(r1,l1,l2). On the other hand, there are no sequential plans equivalent to the policy that executes move(r1,l1,l4) until state s4 is reached. Policy executions correspond to infinite sequences of states, called *histories*, which are Markov Chains, i.e., sequences of random values (in our case states) whose probabilities at a time interval (at each step) depend on the value of the number at the previous time (the previous step in the sequence).

Example 16.3 Some examples of histories for the stochastic system considered so far are the following.

$$h_0 = \langle \mathsf{s1}, \mathsf{s3}, \mathsf{s1}, \mathsf{s3}, \mathsf{s1}, \ldots \rangle$$

$$h_1 = \langle \mathsf{s1}, \mathsf{s2}, \mathsf{s3}, \mathsf{s4}, \mathsf{s4}, \ldots \rangle$$
$$h_2 = \langle \mathsf{s1}, \mathsf{s2}, \mathsf{s5}, \mathsf{s5}, \ldots \rangle$$
$$h_3 = \langle \mathsf{s1}, \mathsf{s2}, \mathsf{s5}, \mathsf{s4}, \mathsf{s4}, \ldots \rangle$$
$$h_4 = \langle \mathsf{s1}, \mathsf{s4}, \mathsf{s4}, \ldots \rangle$$
$$h_5 = \langle \mathsf{s1}, \mathsf{s1}, \mathsf{s4}, \mathsf{s4}, \ldots \rangle$$
$$h_6 = \langle \mathsf{s1}, \mathsf{s1}, \mathsf{s1}, \mathsf{s4}, \mathsf{s4}, \ldots \rangle$$
$$h_7 = \langle \mathsf{s1}, \mathsf{s1}, \ldots \rangle$$

■

Given a policy, some histories are more likely to happen than others, i.e., they have greater probabilities than others. For instance, h_0 has zero probability for any policy, and it is rather intuitive that h_1 has a greater probability than h_3 if we execute policy π_2. Given a policy, we can compute the probability of a history. Let π be a policy and $h = \langle s_0, s_1, s_2, \ldots \rangle$ be a history. The probability of h induced by π is the product of all transition probabilities induced by the policy:

$$P(h|\pi) = \prod_{i \geq 0} P_{\pi(s_i)}(s_{i+1}|s_i) \tag{16.1}$$

Example 16.4 Some examples of computing probability with Formula 16.1 follow.

$$P(h_1|\pi_1) = 0.8$$
$$P(h_1|\pi_2) = 0.8$$
$$P(h_1|\pi_3) = 0$$
$$P(h_2|\pi_1) = 0.2$$
$$P(h_2|\pi_2) = 0.2$$
$$P(h_2|\pi_3) = 0$$
$$P(h_3|\pi_2) = 0.2$$
$$P(h_4|\pi_3) = 0.5$$
$$P(h_5|\pi_3) = 0.5 \times 0.5 = 0.25$$
$$P(h_6|\pi_3) = 0.5 \times 0.5 \times 0.5 = 0.125$$
$$P(h_7|\pi_3) = 0.5 \times 0.5 \times 0.5 \ldots = 0$$

■

Goals as Utility Functions. In planning based on MDPs, goals are utility functions. This is a major departure from classical planning, where goals are sets of states to be reached. Let us explore this choice with the following example.

Example 16.5 Using the scenario from Example 16.1 (see page 380), suppose that l1 is 100 miles from l2, that l3 and l5 are 100 miles from l4, that l1 is 1 mile from l4, and that l2 is 1 mile from l3 and l5. Suppose we want the robot to move to location l4 in as short a distance as possible. We can represent this by assigning "costs" to actions that are proportional to the distances, e.g., a cost of 100 to the action of moving the

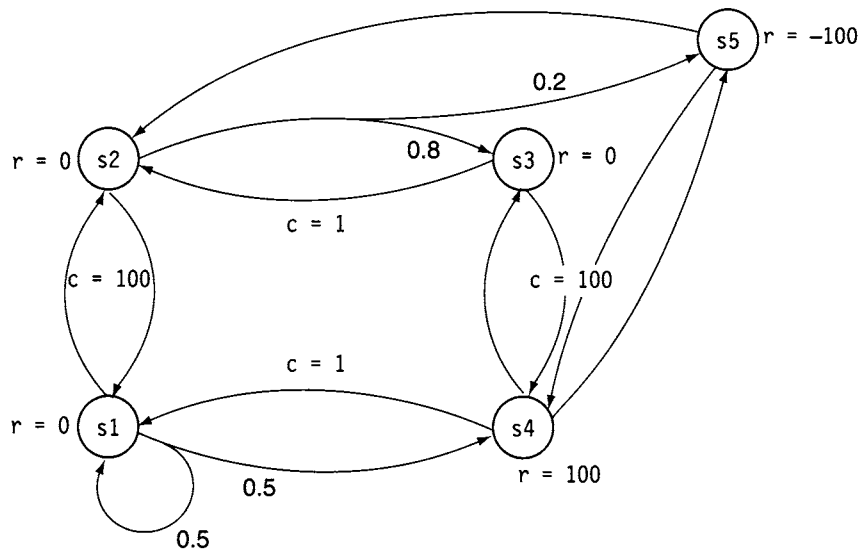

Figure 16.3 Costs and rewards.

robot from l1 to l2, and so on (see Figure 16.3). We assign a cost of 0 to the action of waiting in any state.

Suppose also that we want the robot to move to some states and to avoid other ones, e.g., l5 should be avoided as much as possible because it is a dangerous location. We can represent this by assigning different "rewards" to different states, e.g., a reward of −100 to state s5; a reward of 0 to states s1, s2, and s3; and a reward of 100 to state s4, as shown in the figure. ■

The costs and rewards introduced in Example 16.5 can represent more general conditions than goals as sets of desired final states because they provide a "quantitative" way to represent preferences along all the execution paths of a plan. For instance, in Figure 16.3, the numbers are a way to state that we want the robot to move to state s4 and avoid state s5. Note also that costs and rewards provide an easy way to express competing criteria. Consider, for instance, the criteria of shortest path versus location avoidance. They may be competing because the shortest path may go through locations that should be preferably avoided. We can assign a high cost to transitions that represent long distances for the robot and assign low rewards to states that represent locations that should be avoided by the robot.

The costs and rewards are criteria that can be used to define a *utility function*, i.e., a function that tells us how desirable the histories of a policy are. Let $C : S \times A \rightarrow \Re$ be a *cost function* and $R : S \rightarrow \Re$ be a *reward function* for a stochastic system Σ.

We can define the utility in a state s with an action a as $V(s,a) = R(s) - C(s,a)$, and the utility of a policy in a state as $V(s|\pi) = R(s) - C(s,\pi(s))$. This generalizes to histories. Let $h = \langle s_0, s_1, \ldots \rangle$ be a history. The *utility* of history h induced by a policy π is defined as

$$V(h|\pi) = \sum_{i \geq 0} (R(s_i) - C(s_i, \pi(s_i)))$$

One problem with this definition is that it usually will not converge to a finite value. In fact, it is important that the total accumulated utility is finite; otherwise, there is no way to compare histories. A common way to ensure a bounded measure of utilities for infinite histories is to introduce a *discount factor* γ, with $0 < \gamma < 1$, that makes rewards and costs accumulated at later stages count less than those accumulated at early stages. Let h be a history $\langle s_0, s_1, s_2, \ldots \rangle$. We then define the utility of h induced by a policy π as follows:

$$V(h|\pi) = \sum_{i \geq 0} \gamma^i (R(s_i) - C(s_i, \pi(s_i))) \tag{16.2}$$

The use of a discounting factor is often justified by another argument: to reduce the contribution of distant rewards and costs to the current state.[1] Given a utility function, we can compute the *expected utility of a policy* by taking into account the probability of histories induced by the policy. Let Σ be a stochastic system, H be the set of all the possible histories of Σ, and π be a policy for Σ. Then the expected utility of π is

$$E(\pi) = \sum_{h \in H} P(h|\pi) V(h|\pi) \tag{16.3}$$

Planning Problems as Optimization Problems. A policy π^* is an *optimal policy* for a stochastic system Σ if $E(\pi^*) \geq E(\pi)$, for any policy π for Σ, i.e., if π^* has maximal expected utility. We can now define a planning problem as an optimization problem: given a stochastic system Σ and a utility function, a *solution* to a planning problem is an *optimal policy*.

As a consequence, a policy that is not optimal is *not* a solution. Note that this is a significant departure from the notion of solution in other planning approaches, e.g., classical planning or even different approaches to planning under uncertainty (see Chapter 17), where the notion of optimal solution is distinct from that of solution (which, in general, may be nonoptimal).

1. In economics models (where a great deal of the basic MDP work originates), the rewards may be money earned following a transaction, that in MDP is represented by an action. A fixed amount of money that will be earned in two years has less value than the same amount of money that will be earned in two weeks because of inflation or because of the uncertainty in the future.

Example 16.6 In Figure 16.3, the expected utilities of the policies π_1, π_2, and π_3 are given by:

$$E(\pi_1) = V(h_1|\pi_1)P(h_1|\pi_1) + V(h_2|\pi_1)P(h_2|\pi_1)$$
$$E(\pi_2) = V(h_1|\pi_2)P(h_1|\pi_2) + V(h_3|\pi_2)P(h_3|\pi_2)$$
$$E(\pi_3) = V(h_4|\pi_3)P(h_4|\pi_3) + V(h_5|\pi_3)P(h_5|\pi_3) + \cdots$$

■

In the particular case when there are no rewards associated with the states, the optimal policy π^* minimizes the expected costs.[2]

$$E(\pi) = \sum_{h \in H} P(h|\pi)C(h|\pi) \tag{16.4}$$

where

$$C(h|\pi) = \sum_{i \geq 0} \gamma^i C(s_i, a_i) \tag{16.5}$$

Formula 16.5 can be obtained from Formula 16.2 by assuming $R(s_i) = 0$ for all $s_i \in h$ and by inverting the sign. Similarly, Formula 16.4 can be obtained from Formula 16.3 by replacing $V(h|\pi)$ with $C(h|\pi)$.

16.2.2 Planning Algorithms

In this section we focus on some basic planning algorithms for solving planning problems based on MDPs. For simplicity, we assume that utility functions are determined by cost functions. The discussion generalizes easily to the case of utility functions that include rewards.

Under this assumption, the planning problem is the problem of finding an optimal policy π^*, i.e., such that $E(\pi^*)$ given by Formula 16.4 is minimal. $E(\pi^*)$ is also called the *optimal cost.* Let $E(s)$ be the expected cost in a state s. We define $Q(s, a)$, the expected cost in a state s when we execute action a:

$$Q(s, a) = C(s, a) + \gamma \sum_{s' \in S} P_a(s'|s)\, E(s') \tag{16.6}$$

It can be shown that the optimal cost $E(\pi^*)$ satisfies the fixed-point equation

$$E(s) = \min_{a \in A} Q(s, a) \tag{16.7}$$

2. In the following text, we use the same symbol C for $C(s, a)$ and $C(h|\pi)$.

for all $s \in S$.[3] Formula 16.7 is called the *Bellman Equation.* We call $E_{\pi^*}(s)$ the optimal cost in state s. From the Bellman Equation we have that:

$$E_{\pi^*}(s) = \min_a \{C(s,a) + \gamma \sum_{s' \in S} P_a(s'|s)\, E_{\pi^*}(s')\} \tag{16.8}$$

Given the formulas above, two possible ways to compute π^* are the following two algorithms. The first algorithm, called *policy iteration,* given an initial arbitrary policy π, solves the system of equations:

$$E(s) = C(s,a) + \gamma \sum_{s' \in S} P_a(s'|s)\, E(s') \tag{16.9}$$

This is a system of $|S|$ equations in $|S|$ unknown variables $E(s)$, one for each $s \in S$. Note that because the policy is given, we know that the action a in Formula 16.9 is $\pi(s)$:

$$E_\pi(s) = C(s,\pi(s)) + \gamma \sum_{s' \in S} P_{\pi(s)}(s'|s)\, E_\pi(s') \tag{16.10}$$

Then policy iteration finds actions that decrease the value of $E_\pi(s)$ in state s and updates the policy with such actions until it converges to the optimal policy.

The second algorithm, called *value iteration,* given an initial arbitrary value for each $E(s)$, say $E_0(s)$, computes iteratively in a dynamic programming style the value

$$E_k(s) \leftarrow \min_{a \in A} \{(s,a) + \gamma \sum P_a(s'|s) E_{k-1}(s')\}$$

by increasing k at each step until $E_k(s)$ converges to the optimal cost and the corresponding policy to the optimal policy. In the following discussion we describe the two algorithms in more detail.

Policy Iteration. The Policy-Iteration algorithm is presented in Figure 16.4. Given a stochastic system Σ, a cost function C, and a discount factor γ, it returns an optimal policy π. The basic idea is to start with a randomly selected initial policy and to refine it repeatedly. The algorithm alternates between two main phases: (1) a *value determination phase,* in which the expected cost of the current policy is computed by solving the system of equations shown in Formula 16.9, and (2) a *policy improvement phase,* in which the current policy is refined to a new policy that has a smaller expected cost. For any state s, if an action a exists such that $C(s,a) + \gamma \sum_{s' \in S} P_a(s'|s)$, with $E_\pi(s')$ smaller than the current estimated cost $E_\pi(s)$, then $\pi(s)$ is replaced with a in the policy.

The algorithm terminates when no alternative actions exist that can improve the policy (condition $\pi = \pi'$ in the main loop).

3. This equation depends on π^* in the sense that $a = \pi^*(s)$.

```
Policy-Iteration(Σ, C, γ)
    π ← ∅
    select any π' ≠ ∅
    While π' ≠ π do
        π ← π'
        for each s ∈ S,
            E_π(s) ← the solution of the system of equations
                        E_π(s) = C(s, π(s)) + γ Σ_{s'∈S} P_{π(s)}(s'|s) E_π(s')
        for each s ∈ S do
            if ∃a ∈ A s.t. E_π(s) > C(s, a) + γ Σ_{s'∈S} P_a(s'|s) E_π(s')
                then π'(s) ← a
                else π'(s) ← π(s)
    return(π)
end
```

Figure 16.4 Policy iteration.

Example 16.7 Consider the situation shown in Figure 16.5. Suppose Policy-Iteration selects π_1 as the initial policy π'. We solve the system of equations and compute $E_{\pi_1}(s)$ for each $s \in \{\text{s1}, \text{s2}, \text{s3}, \text{s4}, \text{s5}\}$.

$$
\begin{aligned}
E_{\pi_1}(\text{s1}) &= C(\text{s1}, \text{move}(\text{r1}, \text{l1}, \text{l2})) + \gamma E_{\pi_1}(\text{s2}) \\
E_{\pi_1}(\text{s2}) &= C(\text{s2}, \text{move}(\text{r1}, \text{l2}, \text{l3})) + \gamma(0.8\, E_{\pi_1}(\text{s3}) + 0.2\, E_{\pi_1}(\text{s5})) \\
E_{\pi_1}(\text{s3}) &= C(\text{s4}, \text{move}(\text{r1}, \text{l3}, \text{l4})) + \gamma E_{\pi_1}(\text{s4}) \\
E_{\pi_1}(\text{s4}) &= C(\text{s4}, \text{wait}) + \gamma E_{\pi_1}(\text{s4}) \\
E_{\pi_1}(\text{s5}) &= C(\text{s5}, \text{wait}) + \gamma E_{\pi_1}(\text{s5})
\end{aligned}
$$

Suppose $\gamma = 0.9$. Then:

$$
\begin{aligned}
E_{\pi_1}(\text{s1}) &= 100 + (0.9)\, E_{\pi_1}(\text{s2}) \\
E_{\pi_1}(\text{s2}) &= 1 + (0.9)(0.8\, E_{\pi_1}(\text{s3}) + 0.2\, E_{\pi_1}(\text{s5})) \\
E_{\pi_1}(\text{s3}) &= 100 + (0.9)\, E_{\pi_1}(\text{s4}) \\
E_{\pi_1}(\text{s4}) &= 0 + (0.9)\, E_{\pi_1}(\text{s4}) \\
E_{\pi_1}(\text{s5}) &= 100 + (0.9)\, E_{\pi_1}(\text{s5})
\end{aligned}
$$

and

$$
\begin{aligned}
E_{\pi_1}(\text{s1}) &= 181.9 \\
E_{\pi_1}(\text{s2}) &= 91 \\
E_{\pi_1}(\text{s3}) &= 100 \\
E_{\pi_1}(\text{s4}) &= 0 \\
E_{\pi_1}(\text{s5}) &= 1000
\end{aligned}
$$

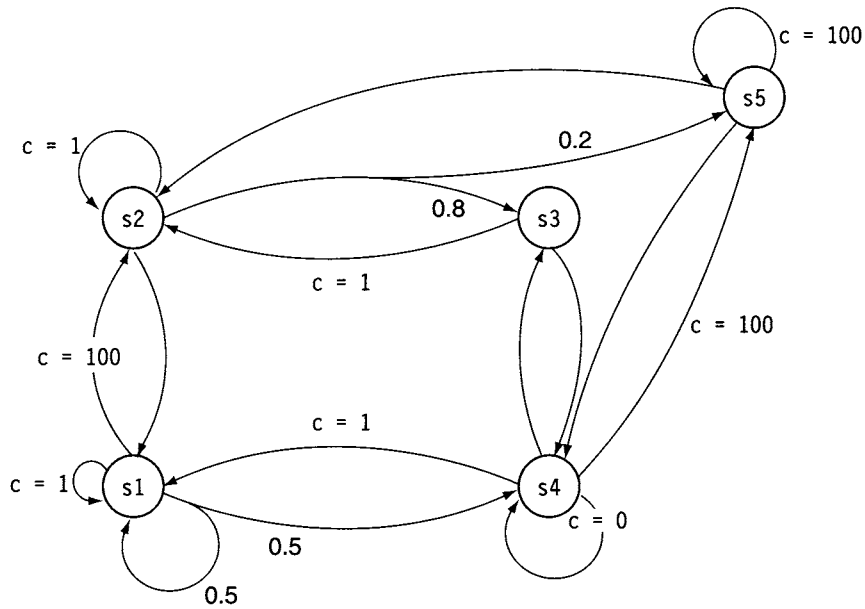

Figure 16.5 Utility function determined by costs.

We can improve the policy by choosing move(r1,l1,l4) in s1, move(r1,l2,l1) in s2, move(r1,l3,l4) in s3, and move(r1,l5,l4) in s5. Indeed, an optimal policy is:

$$\begin{aligned}\pi_4 \quad = \quad & \{\,(\mathsf{s1}, \mathsf{move(r1, l1, l4)}) \\ & \quad (\mathsf{s2}, \mathsf{move(r1, l2, l1)}) \\ & \quad (\mathsf{s3}, \mathsf{move(r1, l3, l4)}) \\ & \quad (\mathsf{s4}, \mathsf{wait}) \\ & \quad (\mathsf{s5}, \mathsf{move(r1, l5, l4)})\,\}\end{aligned}$$

Policy π_4 tries to move the robot to location l4, where the cost of waiting is null, and avoids s5 as much as possible. ■

Value Iteration. The Value-Iteration algorithm is shown in Figure 16.6. It starts with a randomly selected estimated cost $E_0(s)$ for each $s \in S$. Then, it iteratively refines the value for each state by selecting an action that minimizes its expected cost. At each step k, the value of the expected cost E_k is computed for each state from the value E_{k-1} that was computed at the previous step. The algorithm finds an action a such that $E_k(s)$ is minimal and stores it in the policy. It can be shown that there exists a maximum number of iterations needed to guarantee that Value-Iteration returns an optimal policy. However, in practice, the condition used to stop iteration

```
Value-Iteration (Σ,C,γ)
    for each s ∈ S, select any initial value E_0(s)
    k ← 1
    while k < maximum number of iterations do
        for each s ∈ S do
            for each a ∈ A do
                Q(s,a) ← C(s,a) + γ Σ_{s'∈S} P_a(s'|s) E_{k-1}(s')
            E_k(s) ← min_{a∈A} Q(s,a)
            π(s) ← arg min_{a∈A} Q(s,a)
        k ← k + 1
    return(π)
end
```

Figure 16.6 Value iteration.

is the following:

$$\max_{s \in S} |E^n(s) - E^{n-1}(s)| < \epsilon \tag{16.11}$$

This stopping criterion guarantees that the returned policy is an ϵ-optimal policy, i.e., it has an expected cost that does not differ from the optimum by more than an arbitrarily small number ϵ.

Example 16.8 Consider again the situation shown in Figure 16.5. Suppose Value-Iteration initializes the expected cost value to zero: $E_0(s) = 0$ for any $s \in \{s1, s2, s3, s4, s5\}$. Suppose we require an ϵ-optimal policy, and we stop iteration according to Formula 16.11 with $\epsilon = 1$. We discuss the possible iterations for state s1. At the first step (with $n = 1$), in state s1:

1. If $a = \text{move}(r1, l1, l4)$, we have $E_1(s1) = 1$.
2. If $a = \text{move}(r1, l1, l2)$, we have $E_1(s1) = 100$.
3. If $a = \text{wait}$, we have $E_1(s1) = 1$.

Then the algorithm can choose action $a = \text{wait}$ or $a = \text{move}(r1, l1, l4)$. ■

Policy versus Value Iteration. The essential difference between the policy and value iteration algorithms is the following. In policy iteration, we compute a sequence of policies $\pi_1, \pi_2, \pi_3, \ldots, \pi_i, \ldots$ and a sequence of sets of values $E_i(s)$ for each $s \in S$. For each state s:

$$E_i(s) = C(s, a_i) + \gamma \sum_{s' \in S} P_{a_i}(s'|s)\, E_i(s')$$

In value iteration, we also compute a sequence of policies $\pi_1, \pi_2, \pi_3, \ldots, \pi_i, \ldots$ and a sequence of sets of values $E_i(s)$ for each $s \in S$. For each state s:

$$E_i(s) = C(s, a_i) + \gamma \sum_{s' \in S} P_{a_i}(s'|s)\, E_{i-1}(s')$$

The essential difference is that, inside the summation, in the case of policy iteration we use E_i (and thus we need to solve n equations with n unknowns, with $n = |S|$), and in the case of value iteration we use E_{i-1}, i.e., we use the values computed during the previous iteration. Intuitively, each step of Policy-Iteration is computationally more expensive than each step of Value-Iteration. On the other hand, Policy-Iteration needs fewer iterations to converge than Value-Iteration. Both Value-Iteration and Policy-Iteration are polynomial in the number of states $|S|$, the number of actions $|A|$, and the maximum number of bits B required to represent any component of the probability distribution P or the cost function C. B represents the numerical precision of P and C (i.e., the number of significant digits). However, notice that the algorithms are polynomial *in the size of the state space.* In classical planning, the size of the state space is exponential in the size of the planning problem statement. Therefore, this does contradict the fact that the planning problem is NP-hard! In other words, the order of the polynomial is rather large. For realistic domains, the number of states $|S|$ is huge, and both Value-Iteration and Policy-Iteration can hardly be applied.

Real-Time Value Iteration. The simple algorithms we have presented in this chapter suffer clearly from the state explosion problem. Littman *et al.* [365, 366] have analyzed the complexity of MDP planning under the assumption that the states are produced by a set of nondeterministic actions. In the worst case, the complexity is EXPTIME-complete, although it can be made lower by imposing various restrictions. Recall that Value-Iteration updates the value $Q(s, a)$ in parallel for all the states $s \in S$. Since $|S|$ is usually huge, this is the major cause of inefficiency of the algorithm.

Several approaches exist to reduce the state explosion problem (see Section 16.5). One that has been applied successfully is *real-time value iteration* [82]. The name comes in part from the name used for "real-time search" algorithms, i.e., algorithms that search a state space by selecting at each step an action (or state transition) such that a heuristic function is maximized. Here the heuristic function consists of choosing the action such that $Q(s, a)$ is minimal. The idea underlying real-time value iteration is to perform a forward search from an initial set of states to a set of goal states and to update the value of $Q(s, a)$ only in the states visited by the search. The algorithm updates the value of $Q(s, a)$ in the states visited by a search guided by the value of the expected cost computed at each step.

In general, real-time value iteration is not guaranteed to return an optimal solution or even to not terminate. It can, however, be shown that if (1) the initial expected cost $E_0(s)$ is nonoverestimating, i.e., if $\forall s\ E_0(s) \leq E_{\pi^*}(s)$, and (2) there is a path with positive probability from every state to a state in the goal states, then the

algorithm will eventually terminate and return a policy. If we add the condition that the search space is strongly connected, i.e., there is a path with positive probability from every state to every other state, then the algorithm will eventually return an optimal policy.

It has been shown experimentally that real-time value iteration can solve much larger problems than standard value and policy iteration [84]. The trade-off is that the solution is not optimal and the algorithm is not complete; in fact, it may even not terminate.

16.3 Planning under Partial Observability

Planning under Partial Observability in MDP (POMDP) relaxes the assumption that the controller has complete knowledge about the state of Σ. We introduce for the first time in the book *observations* that represent the part of Σ that is visible. In general, observations correspond to more than one state, and this is the cause of uncertainty. More precisely, in MDP, observations correspond to probability distributions over states of Σ. We formalize partial observable domains in Section 16.3.1 and briefly discuss possible algorithms in Section 16.3.2.

16.3.1 Domains, Plans, and Planning Problems

Partially Observable Domains. A partially observable stochastic system is:

- a stochastic system $\Sigma = (S, A, P)$, where S, A, and P are the same as defined in Section 16.2.1, and
- a finite set O of *observations* with probabilities $P_a(o|s)$, for any $a \in A$, $s \in S$, and $o \in O$. $P_a(o|s)$ represents the probability of observing o in state s after executing action a. We require that the probabilities are defined for each state $s \in S$ and action $a \in A$ and that, given a state s and an action a, their sum is 1, i.e., $\sum_{o \in O} P_a(o|s) = 1$.

The set of observations O is introduced to model partial observability. There is no way to get any information about the state other than through observation. Therefore, the controller may not distinguish among different states from the observations that are available. Indeed, different states of the system Σ may result from the same observation. Let s and s' be two distinguished states of Σ, i.e., $s, s' \in S$ and $s \neq s'$. We say that s and s' are *indistinguishable* if $\forall o \in O \; \forall a \in A \;\; P_a(o|s) = P_a(o|s')$.

The same state may correspond to different observations depending on the action that has been executed, i.e., we can have $P_a(o|s) \neq P_{a'}(o|s)$. The reason is that observations depend not only on the state in which the system is but also on the action that leads to that state. For instance, we can have some "sensing action" a that does not change the state, i.e., $P_a(s|s) = 1$, but that makes some observation

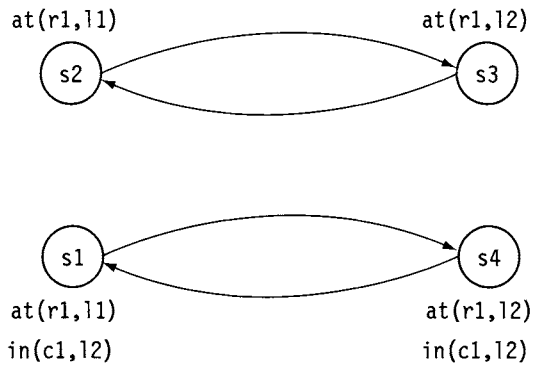

Figure 16.7 A partially observable stochastic system.

available, e.g., for some $o \in O$, $P_a(o|s) = 1$, while $P_{a'}(o|s) \neq 1$ for any $a' \neq a$. If an observation o in a state s does not depend on the action that has been executed, i.e., $\forall a \in A \, \forall a' \in A \;\; P_a(o|s) = P'_a(o|s)$, we write $P(o|s)$.

Example 16.9 Consider a simplified version of the DWR domain, in which there is one robot r1, which can move between two locations l1 and l2, and one container c1. Suppose that the container can be in location l2, written in(c1,l2), and that the robot can observe whether c1 is in l2 only if the robot is in l2. The stochastic system Σ is shown in Figure 16.7.

In states s1 and s2, the robot is in location l1, while in states s3 and s4, it is in location l2. In states s1 and s4 the container is in location l2, while there is no container at location l2 in the states s2 and s3 ($\neg$in(c1,l2) holds in states s2 and s3). Suppose we have two observations, $O = \{\mathsf{f}, \mathsf{e}\}$, with f for full, i.e., the container is in location l2, and e for empty, i.e., the container is not in location l2. We model the robot's ability to observe the container in location l2 by assigning the following probability distributions $P(o|s)$, for any $o \in \{\mathsf{f}, \mathsf{e}\}$ and any $s \in \{\mathsf{s1}, \mathsf{s2}, \mathsf{s3}, \mathsf{s4}\}$ (we suppose that observations do not depend on actions):

$$P(\mathsf{f}|\mathsf{s1}) = P(\mathsf{e}|\mathsf{s1}) = P(\mathsf{f}|\mathsf{s2}) = P(\mathsf{e}|\mathsf{s2}) = 0.5$$
$$P(\mathsf{f}|\mathsf{s4}) = P(\mathsf{e}|\mathsf{s3}) = 1$$
$$P(\mathsf{f}|\mathsf{s3}) = P(\mathsf{e}|\mathsf{s4}) = 0$$

Notice that states s1 and s2 are indistinguishable. ■

Belief States. In POMDPs, the controller can observe a probability distribution over states of the system, rather than exactly the state of the system. Probability distributions over states are called *belief states*. Let b be a belief state and B the set of belief states. Let $b(s)$ denote the probability assigned to state s by the

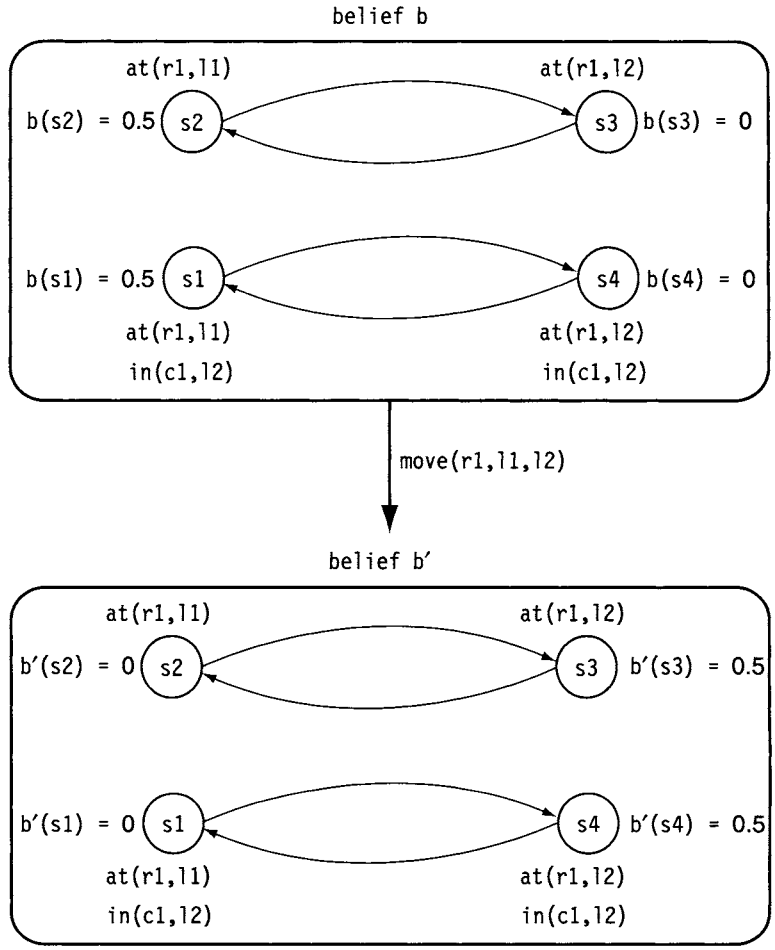

Figure 16.8 Belief states and transitions on belief states.

belief state b. Because $b(s)$ is a probability, we require $0 \leq b(s) \leq 1$ for all $s \in S$ and $\sum_{s \in S} b(s) = 1$.

Example 16.10 Consider the situation described in Example 16.9. Figure 16.8 (a) shows a belief state b such that $b(\mathsf{s1}) = b(\mathsf{s2}) = 0.5$ and $b(\mathsf{s3}) = b(\mathsf{s4}) = 0$. Belief state b models the fact that the controller knows that the system is either in state s1 or s2 with the same probability, 0.5. ■

Given a belief state b, the execution of an action a results in a new belief state b'. We call b_a the belief state that results from performing action a in belief state b.

For each $s \in S$, the probability $b_a(s)$ can be computed as the sum of the probability distribution determined by b weighted by the probability that action a leads from s' to s:

$$b_a(s) = \sum_{s' \in S} P_a(s|s')b(s') \tag{16.12}$$

Example 16.11 Figure 16.8 (b) shows a belief state $b' = b_a$ with $a = \mathsf{move}(\mathsf{r1}, \mathsf{l1}, \mathsf{l2})$. We have $b'(\mathsf{s1}) = b'(\mathsf{s2}) = 0$ and $b'(\mathsf{s3}) = b'(\mathsf{s4}) = 0.5$. The belief state b' models the ability of the controller to observe whether the system is in either state s2 or s3 with the same probability, 0.5. Indeed, moving the robot from location l1 to location l2, leads to a belief state where the robot is at l2, but the robot still cannot observe whether the location is full or empty. ■

We can compute the probability of observing $o \in O$ after executing action $a \in A$ as follows:

$$b_a(o) = \sum_{s \in S} P_a(o|s)b(s) \tag{16.13}$$

Example 16.12 Consider the situation shown in Figure 16.8. Let $a = \mathsf{move}(\mathsf{r1}, \mathsf{l1}, \mathsf{l2})$. Then $b'(e) = b_a(e) = b'(f) = b_a(f) = 0.5$. Because the system can be in either state s3 or s4 with the same probability, we can observe either e or f with the same probability. States s3 and s4 are indistinguishable. ■

We can now compute $b_a^o(s)$, i.e., the probability that the state is s after executing action a in belief state b and observing o:

$$b_a^o(s) = \frac{P_a(o|s)b_a(s)}{b_a(o)} \tag{16.14}$$

Example 16.13 In the same situation described previously, we have $b_a^f(\mathsf{s4}) = 1$ while $b_a^f(\mathsf{s1}) = b_a^f(\mathsf{s2}) = b_a^f(\mathsf{s3}) = 0$, and $b_a^e(\mathsf{s3}) = 1$ while $b_a^e(\mathsf{s1}) = b_a^e(\mathsf{s2}) = b_a^e(\mathsf{s4}) = 0$. This models the fact that we observe that the location l2 is full or empty in state s4 or in state s3. We can now distinguish between state s4 and s3 by observing that the former is full while the latter is empty. ■

Plans as Policies on Belief States. In POMDPs, a policy is a function that maps belief states to actions. Let B be the set of belief states. A policy is a function $\pi : B \rightarrow A$.

Notice that while the set of states S is finite, the set B of probability distributions over states is infinite and continuous because a belief is a probability distribution.

Example 16.14 Consider the POMPD domain used in Example 16.13. The policy that executes move(r1,l1,l2) in belief state $b = \{b(\mathsf{s1}) = 0.5, b(\mathsf{s2}) = 0.5, b(\mathsf{s3}) = 0, b(\mathsf{s3}) = 0\}$ determines whether there is a container in location l2. Suppose now that we have a further action available in the domain, observe-container, such that:

- $\forall o \, \forall s \, \forall a \neq \mathsf{observe} - \mathsf{container} P_a(o|s) = 0.5$
- $\forall o \forall s \in \{\mathsf{s1}, \mathsf{s2}\} P_{\mathsf{observe-container}}(o|s) = 0.5$
- $P_{\mathsf{observe-container}}(\mathsf{f}|\mathsf{s4}) = P_{\mathsf{observe-container}}(\mathsf{e}|\mathsf{s3}) = 1$
- $P_{\mathsf{observe-container}}(\mathsf{f}|\mathsf{s3}) = P_{\mathsf{observe-container}}(\mathsf{e}|\mathsf{s4}) = 0$

The policy $\pi = \{(\mathsf{b0}, \mathsf{move}(\mathsf{r1}, \mathsf{l1}, \mathsf{l2})), (\mathsf{b1}, \mathsf{observe} - \mathsf{container})$, where $\mathsf{b0} = b$ and $\mathsf{b1} = b'$, is the policy that determines whether there is a container in l2. ■

Planning Problems as Optimization Problems. Planning problems in POMDPs can be stated as optimization problems where an optimal policy $\pi : B \rightarrow A$ has to be generated. This can be done by seeing the POMDP planning problem as a fully observable MDP planning problem on the infinite set of belief states. The equation on belief states corresponding to the Bellman Equation (Formula 16.7) is the following:

$$E(b) = \min_{a \in A} C(b, a) + \gamma \sum_{o \in O} b_a(o) \, E(b_a^o) \tag{16.15}$$

where

$$C(b, a) = \sum_{s \in S} C(s, a) b(s) \tag{16.16}$$

16.3.2 Planning Algorithms

A way to solve POMDPs is to use the algorithms for completely observable MDPs over belief states. However, computationally the POMDP problem is very hard to solve because the resulting space of belief states, i.e., B, is infinite and continuous. The known POMDP algorithms that return optimal policies can in practice be applied only to problems of rather small size. Algorithms that return approximations of optimal policies can be more practical. A possible way to search for a plan is to perform a forward search on belief states. (Here we briefly give some hints.) At each step, the search procedure selects the action that minimizes the expected cost in the current belief state, computes the belief state that results from applying the action, and so on. One way to do this is to consider only actions that are "applicable" in the current belief state, i.e., actions such that for any state s of the belief state, there exists a s' such that $P_a(s'|s) \neq 0$. The search can be guided by different heuristics, i.e., by defining a heuristic function $h(b)$ for a belief state b, e.g., in terms of the expected cost. Typically these kinds of algorithms are not guaranteed to return an

optimal policy; however, they can solve problems that cannot be solved in practice by algorithms returning optimal policies.

16.4 Reachability and Extended Goals

Given a stochastic system $\Sigma = (S, A, P)$, we could define goals in the classical way, i.e., as sets of states to be reached $S_g \subseteq S$, choose an initial state $s_o \in S$, and define the planning problem as the triple (Σ, s_0, S_g). However, the definition of a solution policy for the planning problem is not obvious as in classical planning. Indeed, while in classical planning a solution plan is a plan that results in a final state that is a goal state, the same policy may result, as we have seen, in different histories—some of them leading to a goal state, others not achieving the goal.

Example 16.15 Consider the stochastic system shown in Figure 16.1. Let s1 be the initial state and let {s4} be the set of goal states. All the history with positive probability for policy π_2 leads to the goal, while, if we consider policy π_1, history h_1 leads to the goal with probability 0.8, while history h_2 does not lead to the goal with probability 0.2. All the histories with positive probability lead to the goal state according to policy π_3. All the histories with positive probabilities of both π_2 and π_3 lead to the goal state. Notice indeed that h_7 has probability 0 both for π_1 and π_3. ■

The MDP framework can be used to solve planning problems with classical reachability goals, i.e., goals that represent sets of desired final states. Given a stochastic system Σ and a planning problem $P = (\Sigma, s_o, S_g)$, and given the cost function C, we modify the stochastic system and the cost function such that the following hold.

- Actions in goal states produce no changes, i.e., $\forall s \in S_g \; \forall a \in A \; P(s|s) = 1$.
- Actions in goal states have no costs, i.e., $\forall s \in S_g \; \forall a \in A \; C(s, a) = 0$.

Given the stated assumptions, the planning problem can still be defined as the optimization problem defined in Section 16.2.

In Section 16.2 we showed how MDPs can represent goals that express preferences on the entire execution path. In the following examples we further explore how costs and rewards can be used to represent extended goals.

Example 16.16 Consider a DWR domain with two locations l1 and l2. Containers can dynamically and unpredictably be downloaded from a ship to a pile at location l1. We would like to express the condition that the robot should try to keep location l1 empty by continuing to deliver containers to location l2. We can represent this domain with the nondeterministic stochastic system shown in Figure 16.9. The fact that containers may nondeterministically arrive at location l1 is captured by

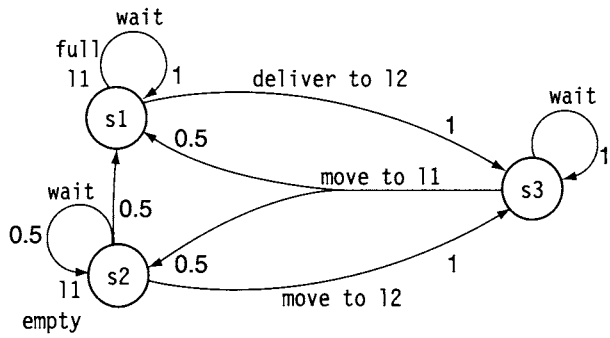

Figure 16.9 A stochastic system for continuous delivery.

the nondeterminism of actions move(r1,l2,l1) and wait that can lead to a state with or without containers (full or empty) with uniform probability distribution. We can express the goal described informally by assigning a high reward to state s2. Given this planning problem, the planning algorithms generate the obvious policy {(s1, deliver to l2), (s2, wait), (s3, move(r1, l2, l1)}. This policy continuously delivers containers to location l2 as soon as they are downloaded at location l1. ■

Example 16.17 Consider the stochastic system shown in Figure 16.1. Suppose the goal for the robot is to keep going around visiting all the locations but avoiding s5 (e.g., a dangerous zone). Assigning high rewards to all states but s5 is not enough because a policy that stays in a state other than s5 forever would have a high utility function. A possible way to express this goal is to give action wait a cost higher than the cost of the other actions, so that the robot is "pushed" to move to adjacent nodes.

It is possible to play with costs and rewards in order to specify different kinds of goals. An interesting problem is to express the condition that the robot has to visit infinitely often states s2 and s4 but never go through state s5. ■

16.5 Discussion and Historical Remarks

MDPs take a very different view from the approaches we have described in the previous chapters. The planning problem is seen as an optimization problem. The idea of maximizing a utility function has intrinsic advantages and disadvantages. A utility function can very easily express preferences on states and actions, and many applications require such preferences. Moreover, costs and rewards can very naturally express competing criteria for optimality. For example, it is usually easy to express two competing criteria such as shortest path and state avoidance by assigning proper costs to actions and penalties to states to be avoided. For instance,

in Example 16.5 (see page 383), it is very easy to express competing criteria. On the other hand, sometimes it is not easy to formulate goals in terms of state rewards (or action costs). For instance, in the same example, one might wonder whether we would have the same optimal policy if we changed the reward for s5 from -100 to -1 or if we changed the reward for s4 from 100 to 0.

The idea of using transition probabilities also has its pros and cons. Probabilities allow a certain level of expressiveness. However, they are in most cases statistical estimates, and how they can affect solutions when those estimates deviate from the accurate values is a critical issue. This issue is even more critical in applications where statistical estimates are not available.

In this chapter, we described a rather basic class of MDP planning problems and rather simple planning algorithms. Some important works initially investigated the MDP approach to planning [108, 144, 294]. Here are some references to different problems and alternative solutions.

- We have considered the case of MDPs over infinite horizons, i.e., we assume histories to be infinite sequences of states and compute expected utilities over infinite histories. A discussion of the problem with a finite horizon can be found in Boutilier *et al.* [88].
- We have assumed the utility function to be a simple combination of rewards and costs. There are other possibilities, such as additive versus nonadditive, average versus discounted, etc. (see [88] for a review).
- We have assumed that policies are functions that map states to actions, i.e., they are memoryless and stationary. There exist more complex policies, e.g., those that take into account the whole history, such as nonmemoryless and/or nonstationary policies, decision trees, etc. (see again [88] for an overview).
- There are several different implementations of policy iteration and value iteration. Their complexity analysis can be found in two sources [108, 262].
- In the satisficing approach to MDPs [340], goals are sets of states, and the planning problem is defined by requiring that histories get to a goal state with probability over a given threshold.
- There exist different approaches to the POMPD planning problem (see, e.g., [80, 87, 446, 447]) and different approaches that address the problem of state explosion, such as factorized MDPs [89, 429], abstraction [147, 350], symbolic approaches to first-order MDPs [90], the use of decision diagrams, e.g., in the SPUDD planner [269], and heuristic search in AND/OR graphs [258].
- The problem of expressing goals in temporal logic in the MDP framework is addressed in several works [30, 31, 506].
- MDP planning can be seen as an extension to heuristic search in the unifying approach proposed by Bonet and Geffner [82, 84], where real-time value iteration algorithms are proposed and implemented in the GPT system [83], available at http://www.cs.ucla.edu/~bonet/.

- Markov games are an extension of MDPs where contingent events are taken into account [364]. In this framework, in addition to a set of controllable actions, we have a set of uncontrollable events, and the probability distribution is defined on the occurrence of an action and an event. Some common and general assumptions are that actions and events occur simultaneously at each step; the action is chosen by the planner; and the event is chosen by the environment, which is seen as an opponent. A utility function can then be defined as usual, and the problem is again to optimize the utility function.

16.6 Exercises

16.1 In Example 16.6 (see page 386), what are the utility values of π_1, π_2, and π_3?

16.2 Consider Example 16.5 (see page 383). Exchange the probabilities for nondeterministic actions (e.g., $P_{\text{move c2 to p2}}(2|1) = 0.2$ and $P_{\text{move c2 to p2}}(4|1) = 0.8$). Costs are uniform and unary. Assign reward 100 to state s3, and a reward 0 to all other states. What is the optimal policy?

16.3 In Example 16.5, if we change the reward for s5 from -100 to -1, do we have the same optimal policy? And what about changing the reward for s4 from 100 to 0? Describe how the optimal policy changes depending on different configurations of rewards and costs.

16.4 In Example 16.5, if we change the probability distribution of the two nondeterministic actions, how do they do impact the optimal policy? For instance, if $P_{\text{move(r1,l1,l4)}}(\text{s4}|\text{s1}) = 0.1$, which is the optimal policy?

16.5 Consider Example 16.5. How would you express a goal like "keep visiting s2 and s4"? Which policy would be returned by the planning algorithms? Do the same in the case where there are no arcs between s4 and s5.

16.6 Modify the policy and the value iteration algorithms so they can solve a classical planning problem with a fully deterministic stochastic system, an initial state, and a set of goal states.

16.7 Modify the policy and the value iteration algorithms so they can solve a planning problem with a nondeterministic stochastic system, a set of initial states, and a set of goal states.

16.8 Consider a simplified version of Exercise 5.7 in which there is no washing machine and no shower, and Dan only wants to wash the dishes. Suppose that start-fill and start-wash each have a 0.5 probability of working correctly and a 0.5 probability of doing nothing. Suppose that the other actions work correctly with probability 1. We can represent this as an MDP with the following states.

s_0 is the initial state.

s_1 is the result of executing start-fill(dw) successfully in s_0.

s_2 is the result of executing end-fill(dw) in s_1.

s_3 is the result of executing start-wash(dw) successfully in s_2.

s_4 is the result of executing end-wash(dw,dishes) in s_3.

Executing an inapplicable action leaves the state unchanged. Each state has a reward of 0, each action has a cost of 1, and the discount factor is $\gamma = 1/2$. If we ever reach s_4, the system terminates and there are no more costs and rewards.

(a) Consider this policy:

$$\pi = \{(s_0, \text{start-fill(dw)}), (s_1, \text{end-fill(dw)}),$$
$$(s_2, \text{start-wash(dw)}), (s_3, \text{end-wash(dw)})\}$$

What is the cost of π?

(b) If we start with $E = 1$ at each node of the MDP and do one round of value iteration, what are the values of E at each node afterward?

(c) Let π' be any policy that produces the E values in part (b). If we start with π' and do one round of policy iteration, then what are the resulting policy and the resulting E values?

(d) Write all policies that have noninfinite cost.

(e) Professor Prune says, "To write the MDP representation of the washing problem takes at least as much work as it takes to solve the original classical planning version of the washing problem." Is he right? Why or why not?

16.9 Redo Exercise 16.8 with $\gamma = 1$.

16.10 Rewrite the Policy-Iteration algorithm (Figure 16.4) to incorporate rewards as shown in Formula 16.2.

16.11 Rewrite the Value-Iteration algorithm (Figure 16.6) to incorporate rewards as shown in Formula 16.2.