15 Reinforcement Learning of Optimal Controls

John K. Williams

15.1 Introduction

As humans, we continually interpret sensory input to try to make sense of the world around us, that is, we develop mappings from observations to a useful estimate of the "environmental state". A number of artificial intelligence methods for producing such mappings are described in this book, along with applications showing how they may be used to better understand a physical phenomenon or contribute to a decision support system. However, people don't want simply to understand the world around us. Rather, we interact with it to accomplish certain goals – for instance, to obtain food, water, warmth, shelter, status or wealth. Learning how to accurately estimate the state of our environment is intimately tied to how we then use that knowledge to manipulate it. Our actions change the environmental state and generate positive or negative feedback, which we evaluate and use to inform our future behavior in a continuing cycle of observation, action, environmental change and feedback.

In the field of machine learning, this common human experience is abstracted to that of a "learning agent" whose purpose is to discover through interacting with its environment how to act to achieve its goals. In general, no teacher is available to supply correct actions, nor will feedback always be immediate. Instead, the learner must use the sequence of experiences resulting from its actions to determine which actions to repeat and which to avoid. In doing so, it must be able to assign credit or blame to actions that may be long past, and it must balance the exploitation of knowledge previously gained with the need to explore untried, possibly superior strategies. Reinforcement learning, also called stochastic dynamic programming, is the area of machine learning devoted to solving this general learning problem. Although the term "reinforcement learning" has traditionally been used in a number of contexts, the modern field is the result of a synthesis in the 1980s of ideas from optimal control theory, animal learning, and temporal difference methods from artificial intelligence. Finding a mapping that prescribes actions based on measured environmental states in a way that optimizes some long-term measure of success is the subject of what mathematicians and engineers call "optimal control" problems and psychologists call "planning" problems. There is a deep body of mathematical literature on optimal control theory describing how to analyze a system and develop optimal mappings. However, in many applications the system is poorly understood, complex, difficult to analyze mathematically, or changing in time. In such cases, a machine learning approach that learns a good control strategy from real or simulated experience may be the only practical approach (Si et al. 2004).

This chapter begins with a brief introduction to the origins of reinforcement learning, then leads the reader through the definitions of Markov Decision Processes (MDPs), policies and value functions and the formulation of the Bellman Optimality Equation, which characterizes the solution to an MDP. The notion of Q-values is presented with a description of how they can be used to improve policies and how value functions and Q-values may be estimated from

John K. Williams (✉)
Research Applications Laboratory
National Center for Atmospheric Research
Address: P.O. Box 3000, Boulder, CO 80307, USA
Phone: 303-497-2822;
Fax: 303-497-8401;
Email: jkwillia@ucar.edu

an agent's experience with the environment. Optimal Q-values, which are associated with optimal policies for MDPs, may be learned through Q-learning or several related algorithms, which are described next. Partially observable MDPs, in which only partial state information is available, are discussed. It is then shown how reinforcement learning algorithms may be applied to MDPs for which the state and action spaces are large or continuous through the use of function approximation, including neural networks. Finally, three sample applications are presented: dynamic routing in a wireless sensor array, control of a scanning remote sensor, and optimal aircraft route selection given probabilistic weather forecasts. Some readers may wish to read the first few sections and then jump to the applications, returning as needed to the theoretical sections as needed to understand the notation and techniques illustrated there.

15.2 History and Background

As mentioned above, the field of reinforcement learning synthesizes ideas from the fields of mathematics, engineering, and psychology. A key mathematical concept is dynamic programming, developed in the 1950s by Richard Bellman (Bellman 1957). Bellman built on earlier work by Hamilton and Jacobi, including the formulation of the Hamilton-Jacobi equation for classical mechanics (Hamilton 1835). Dynamic programming addresses how to find optimal controls in dynamical systems subject to a degree of randomness. In a standard formulation of the problem, actions drive transitions from one state of the system, or "environment," to another, with each transition accompanied by a "cost". Both the transitions and the costs may be random variables that are functions of the starting state and the action taken. Bellman showed that the optimal control strategy in such a problem can be determined from the solution to a certain equation, now called the Bellman or Hamilton-Jacobi-Bellman Optimality Equation (see Section 15.5). Dynamic programming techniques solve the optimal control problem by solving this equation, and it is fundamental to the reinforcement learning methods described in this chapter.

On the other hand, reinforcement theories of animal learning are a cornerstone of psychology. It has long been recognized that an animal's behavior when presented with a given situation can be modified by rewarding or punishing certain actions. Trial-and-error learning algorithms for computers based on this sort of feedback or "reinforcement" date back to the work of A. M. Turing in the 1940s (Turing 1948, 1950). The idea of temporal difference learning, described in Section 15.8, may also owe its genesis to psychology via the concept of a *secondary reinforcer*. In animal behavior studies, a secondary reinforcer may be paired with a primary reinforcer (reward or punishment) through training. After establishing the association with the primary reinforcer, the animal may be trained using the secondary reinforcer *in place* of the primary reinforcer. For example, a pigeon may be trained to perform a task to receive a food reward, where the food is accompanied by a musical tone. Later, the pigeon can be taught to learn tasks for which the "reward" is the tone itself, even if a food treat no longer accompanies it. Temporal-difference methods provide a way to propagate feedback information "backward" in a sequence of experiences so that actions leading to a successful outcome are reinforced even when their ultimate payoff is significantly delayed. The successful use of temporal-difference algorithms in artificial intelligence dates back to Arthur Samuel's famous checkers-playing program (Samuel 1959), and has been used successfully for many game playing and other practical applications since.

These ideas – dynamic programming, trial-and-error learning, and temporal differences – were brought together by Chris Watkins in his 1989 Ph.D. dissertation, *Learning from Delayed Rewards*, in which he presented the "Q-learning" algorithm (Watkins 1989; see also Watkins and Dayan 1992). The simplicity and versatility of Q-learning quickly made it one of the most popular and widely used reinforcement learning algorithms, and its invention helped spawn the expansion of reinforcement learning into a broad field of research. Dozens, if not hundreds, of reinforcement learning algorithms have since been proposed and applied to a wide variety of optimal control problems.

In real-world applications, the possible environmental states and actions for an optimal control problem often cannot reasonably be enumerated, so learning a "lookup table" that prescribes an action for each possible state may not be practical. Reinforcement learning algorithms accommodate large or continuous state and action spaces by using function approximators, including linear maps and neural networks, to represent the information needed to map states to actions. Neural networks offer the advantage that well-formulated methods exist for incrementally updating their parameters as new "training" data become available, and they often work well when states and actions are encoded in such a way that similar states and actions reliably lead to similar state transitions and feedback. Because of the popularity of this approach, some researchers have coined the term "neuro-dynamic programming" to describe the fusion of ideas from neural networks and dynamic programming (Bertsekas and Tsitsiklis 1996). However, reinforcement learning with function approximation has been shown to be unstable and to fail to converge for some problems and algorithms. The theory and methods described in this chapter are presented first for finite state and action spaces, thereby illustrating many essential issues in reinforcement learning in this more straightforward domain. The integration of function approximation with these algorithms is discussed in Section 15.12.

Of course, it is only possible to cover a few highlights of reinforcement learning theory and its applications in this single short introductory chapter. For a more comprehensive treatment, the reader is referred to the excellent texts *Neuro-Dynamic Programming* by Bertsekas and Tsitsiklis (1996) and *Reinforcement Learning* by Sutton and Barto (1998).

15.3 Markov Decision Processes

Before reinforcement learning algorithms can be described in detail, it is first necessary to define the problems to which they apply, which are called *Markov decision processes* or *Markov decision problems* (MDPs). In an MDP, the learning agent is able to interact with its environment in a limited fashion: it observes the *state* of the environment, chooses one of several available *actions*, and receives feedback, or *reinforcement*, in the form of some numerical *cost* or *reward* as a result of the action taken and the resulting state transition. The goal of the agent is to devise a *policy* – a rule for choosing actions based on observed states – that minimizes some measure of long-term costs (or, equivalently, maximizes long-term rewards). This policy could be either a deterministic mapping from states to actions or a rule specifying a probability distribution from which an action is to be randomly chosen in each state. Figure 15.1 shows a diagram of

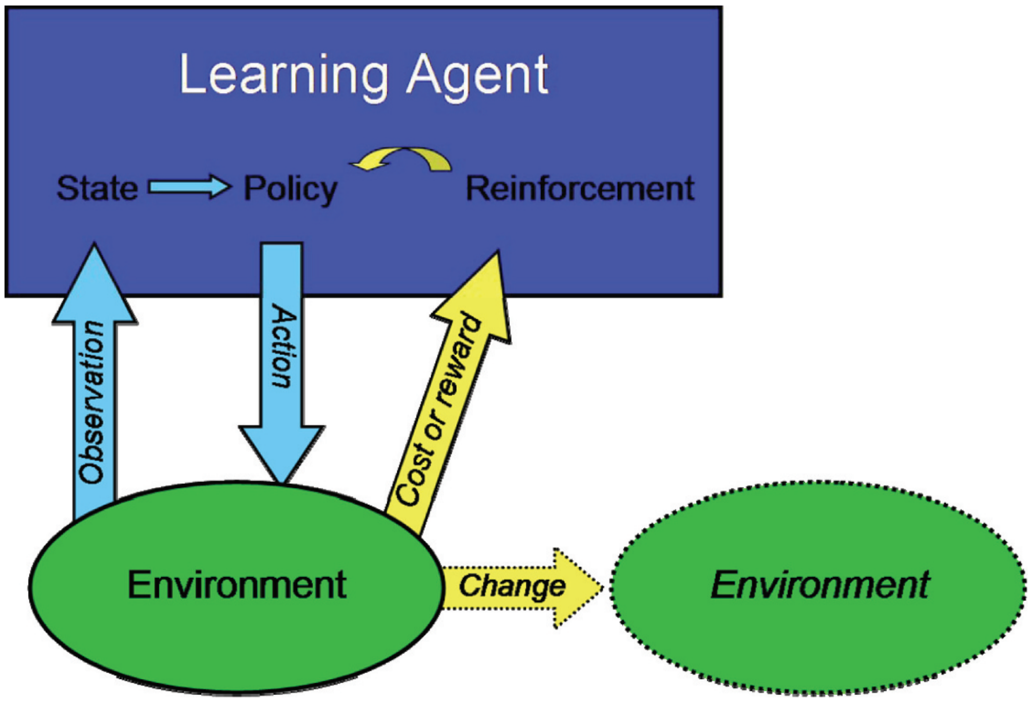

Fig. 15.1 Diagram of a reinforcement learning agent interacting with its environment: observations provide information about the environmental state that in turn may be used to determine an action based on the current policy. The action may result in a change to the environment and a cost or reward reinforcement signal that can be used to modify the policy.

this interaction between learning agent and a real or simulated environment. The key feature of an MDP (the "Markov property") is that the probabilities of state transitions and costs are a function only of the present state and action taken, not the past history of actions or states. In other words, the state of the environment by itself contains all the information needed to determine the result of a course of action, at least in a probabilistic sense.

To be more mathematically precise, the possible states obtained from observations of the environment comprise a set $\boldsymbol{S}$, and from each state $i \in \boldsymbol{S}$ there is a set of available actions that may be denoted $U(i)$. For the sake of simplicity, we shall initially assume that both $\boldsymbol{S}$ and $U(i)$ are finite sets, though infinite and even continuous state and action spaces may be dealt with via function approximation (see Section 15.12). The probability of a transition from state i to state j under action u may be written as $P^u(i, j)$ and is assumed not to change as a function of time. The cost of a state transition from i to j under action u is a random variable $g(i, u, j)$ that we will assume has a finite expected (mean) value $\bar{g}(i, u, j)$ and finite variance. As in the field of economics, where the same approach is used to compute the present value of future assets and liabilities, the MDP is equipped with a discount factor, α, a number between 0 and 1. A cost incurred one time unit in the future must be multiplied by the discount factor to compare it to an immediate cost. Thus if $\alpha = 1$, it means that future costs are valued the same as present costs; if $\alpha < 1$, future costs are less significant than immediate costs of the same amount. For example, given an inflation rate of 5% per timestep, α might be about 0.95, meaning that a future asset or liability of \$1.00 next year is worth only \$0.95 in current value, making it worthwhile to take rewards immediately, while their values are highest, while delaying the payment of fixed costs as long as possible. MDPs may be categorized according to the value of α. If $\alpha < 1$, the MDP is called a *discounted problem* (DCP), since future costs are discounted. If $\alpha = 1$ and there is a "final" state that ends the process (usually denoted as state 0), it is called a *stochastic shortest path problem* (SSPP) since a typical problem of this sort is finding a shortest path through a maze or a network. For a thorough exposition of MDPs, the reader is referred to the text by Puterman (2005). A summary of the symbols used in this chapter for describing MDPs and the concepts and reinforcement learning algorithms for solving them may be found in Table 15.1.

If the transition probability values $P^u(i, j)$ and average costs $\bar{g}(i, u, j)$ are known to the learning

Table 15.1 Descriptions of some common symbols used in this chapter to describe MDPs and reinforcement learning algorithms.

Symbol	Description
$\boldsymbol{S}$	Set of environmental states for an MDP.
$U(i)$	Set of actions available from state i.
$P^u(i, j)$	Probability of transition from state i to state j under action u.
$g(i, u, j)$	Random variable cost of transition from state i to state j under action u.
$\bar{g}(i, u, j)$	Expected (mean) cost of transition from state i to state j under action u.
α	Discount factor between 0 and 1.
i_t, u_t	State and action taken at time t.
μ	A deterministic policy, which maps states to actions.
π	A stochastic policy, which maps state-action pairs to probabilities.
J^μ	The value function for the policy μ; see (15.1).
J^*	The optimal value function.
Q^μ	The action-value function, or Q-value, for the policy μ; see (15.13).
Q^*	The optimal Q-value.
J_t, Q_t	Value function and Q-value iterates for a learning algorithm at time t.
γ	Learning rate or step size parameter.
$R_t^{(N)}$	N-step return beginning at time t; see (15.22).
R_t^λ	λ–return (average of N-step returns) beginning at time t; see (15.23).
δ_t	One-step temporal difference at time t; see (15.25).
e	Eligibility trace for a state or state-action pair.
$\boldsymbol{M}$	Set of messages for a POMDP.

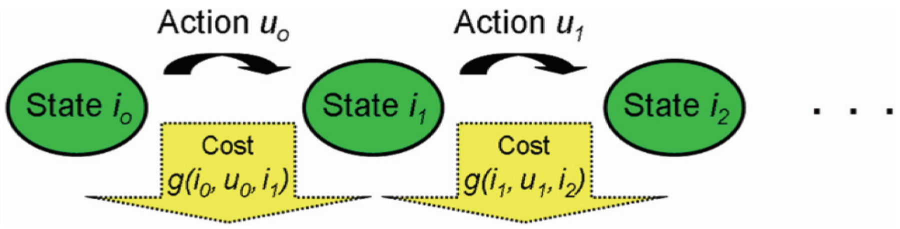

Fig. 15.2 A trajectory of an MDP, with a sequence of actions generating state transitions accompanied by feedback signals ("costs").

agent for all states i, j and actions u, we say that a *model* of the environment is available. When a model is not provided, the learning problem is *model-free*. Some reinforcement learning algorithms use experience (e.g., simulation with various choices of actions) to estimate a model of the environment, then use dynamic programming methods to solve for the optimal value function and policy. Other algorithms, such as Q-learning and temporal difference learning, are model-free, operating directly on the cost and transition experiences as they are obtained. To refer to the sequence of states, actions, and costs experienced by a learning agent in these algorithms, we use subscripts to count the timesteps. So, for instance, i_t refers to the state occupied at the t^{th} timestep, u_t refers to the t^{th} action, and the cost incurred in the transition to the next state is $g(i_t, u_t, i_{t+1})$, which is sampled from a random variable having mean value $\bar{g}(i_t, u_t, i_{t+1})$ and finite variance. The trajectory of the MDP is then written as the sequence of states, actions, and costs: i_0, u_0, $g(i_0, u_0, i_1)$, i_1, u_1, $g(i_1, u_1, i_2), \ldots$ (see Fig. 15.2). The goal of a reinforcement learning algorithm is to use such a sequence of experiences to determine an optimal policy.

15.4 Policies and Value Functions

Solving an MDP means finding an optimal policy: a mapping from states to actions which, when used for action selection, minimizes some measure of long-term costs. For an MDP, it turns out that there is always an optimal policy that minimizes the expected sum of future discounted costs for each state (see below); this policy is deterministic and does not change as a function of time. However, in the process of finding an optimal policy, a learning agent must balance between acting according to the best policy found so far and exploring new, untried actions that may lead to even better results. This is the famous "exploitation vs. exploration" conundrum of reinforcement learning. One way to perform this balance is to use action selection that includes some degree of randomness while learning, that is, a *stochastic* policy that specifies a probability distribution over available actions for each state. Defining these concepts formally, a *deterministic policy* μ is a mapping from states to actions such that $\mu(i) \in U(i)$ for all states $i \in \boldsymbol{S}$. A *stochastic policy* π is a mapping from state-action pairs to probabilities such that (a) $0 \leq \pi(i, u) \leq 1$ for all states $i \in \boldsymbol{S}$ and actions $u \in U(i)$, (b) $\pi(i, u) = 0$ whenever $u \notin U(i)$, and (c) $\sum_{u \in U(i)} \pi(i, u) = 1$ for each state $i \in \boldsymbol{S}$. The sequence of states encountered in the MDP while acting under a fixed policy is called a *Markov chain*. For SSPPs, the policies that create a Markov chain that is guaranteed to terminate (with probability one) are called *proper* policies.

In order to evaluate a policy, we must specify an objective way to measure its success. A common metric, and the main one used in this chapter, is the "expected" (i.e., mean) total long-term discounted cost incurred from executing the policy. This long-term expected cost is a function of the starting state, and is called the policy's *value function*. Formally, the value function for the deterministic policy μ, written J^μ, is defined for each state $i \in S$ by

$$J^\mu(i) = E\left[\sum_{t=0}^{\infty} \alpha^t g(i_t, \mu(i_t), i_{t+1}) \mid i_0 = i\right] \qquad (15.1)$$

The "E" in (15.1) denotes the expected value – the mean or average over all the randomness in the state transitions and costs g resulting from the prescribed actions – and the vertical line "|" means "such that". Thus, this equation defines the value function for state i as the expected value of the infinite sum of future

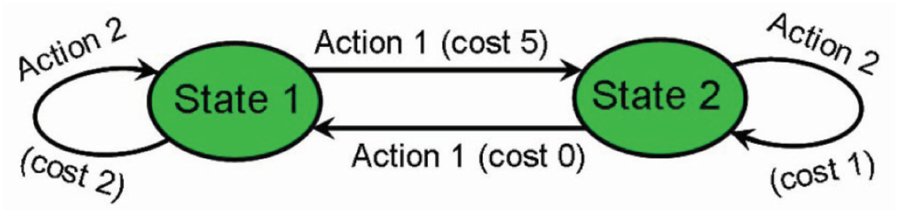

Fig. 15.3 Example MDP with two states, two available actions and deterministic state transitions represented by the arrows. Costs associated with the state transitions are indicated in parentheses, and α = 0.9, making this a discounted problem (DCP).

discounted costs incurred when beginning in state i and choosing actions according to the policy μ. The value function for a stochastic policy π is denoted J^π and is similarly defined by

$$J^\pi(i) = E\Bigg[\sum_{t=0}^{\infty} \alpha^t g(i_t, u_t, i_{t+1}) \mid i_0 = i\,; \quad \text{each } u_t \text{ chosen according to } \pi\Bigg] \tag{15.2}$$

We will make assumptions on the MDPs and policies that ensure that the value functions defined in (15.1) and (15.2) are finite; for instance, this will be true in the common case that the costs g are bounded and the discount factor $\alpha < 1$.

Rewriting the expectation in (15.1) by pulling out the first term of the sum leads to the so-called *consistency* or *Bellman* equation:

$$\begin{aligned} J^\mu(i) &= E_j\left[g(i, \mu(i), j) + \alpha J^\mu(j)\right] \\ &= \sum_{j \in S} P^{\mu(i)}(i, j)\ (\bar{g}(i, \mu(i), j) + \alpha J^\mu(j)) \end{aligned} \tag{15.3}$$

where the expectation E_j is over the (possibly) random cost g and random transition to a next state j. It can be shown that, for any deterministic policy, μ, J^μ is the *unique* solution to the consistency equation, that is (15.3) completely characterizes J^μ (this is a corollary to the Bellman Optimality Equation presented in the next section). The consistency equation for stochastic policies may be written analogously.

An *optimal value function* is one that has the smallest achievable value for each state. That is, if we denote the optimal value function by J^*, then for all states $i \in \boldsymbol{S}$, $J^*(i) \leq J^\pi(i)$ for all policies π and $J^*(i) = J^\pi(i)$ for some policy π. It will be shown in the next section that, under certain reasonable assumptions, there is at least one deterministic policy μ such that $J^*(i) = J^\mu(i)$ for all states $i \in \boldsymbol{S}$. Briefly, this is a result of the fact that, due to (15.3), any change of a policy that reduces the expected discounted sum of future costs from one state can only cause an improvement in other states as well. Naturally, any policy whose value function is equal to the optimal value function is called an *optimal policy*. The next section characterizes such policies. First, though, we consider two examples to make these concepts more concrete.

Figure 15.3 shows a representation of a two-state MDP with discount factor $\alpha = 0.9$, making it a DCP. In State 1, two actions are possible: Action 1 causes a transition to State 2 with a cost of 5, and Action 2 exacts a cost of 2 but doesn't change the state. In State 2, Action 1 causes a transition to State 1 with a cost of 0, and Action 2 loops back to State 2 with a cost of 2. The four possible policies are given in Table 15.2 along with their value functions. For example, the policy here arbitrarily called Policy 2, given by $\{\mu(1) = 1, \mu(2) = 2\}$, has transition probabilities $P^{\mu(1)}(1, 1) = 0$, $P^{\mu(1)}(1, 2) = 1$, $P^{\mu(2)}(2, 1) = 0$ and $P^{\mu(2)}(2, 2) = 1$ with costs $\bar{g}(1, \mu(1), 2) = 5$ and $\bar{g}(2, \mu(2), 2) = 1$. Under this policy, if the initial state is State 1, the first step will transition to State 2 with

Table 15.2 Policies and value functions for the MDP depicted in Fig. 15.3

Policy 1	$\{\mu(1) = 1, \mu(2) = 1\}$	$J(1) = 500/19 \approx 26.3$ $J(2) = 450/19 \approx 23.7$
Policy 2	$\{\mu(1) = 1, \mu(2) = 2\}$	$J(1) = 14$ $J(2) = 10$
Policy 3	$\{\mu(1) = 2, \mu(2) = 1\}$	$J(1) = 20$ $J(2) = 18$
Policy 4	$\{\mu(1) = 2, \mu(2) = 2\}$	$J(1) = 20$ $J(2) = 10$

a cost of 5 and then the Markov chain will remain in State 2 with an additional cost of 1 per timestep. Thus, the value function for this policy may be computed via (15.1) as

$$\begin{aligned} J^{\mu}(1) &= 5 + \alpha \cdot 1 + \alpha^2 \cdot 1 + \alpha^3 \cdot 1 + \ldots \\ &= 5 + \sum_{t=1}^{\infty} \alpha^t = 5 + \frac{\alpha}{1-\alpha} = 5 + 9 = 14 \\ J^{\mu}(2) &= 1 + \alpha \cdot 1 + \alpha^2 \cdot 1 + \alpha^3 \cdot 1 + \ldots \\ &= \sum_{t=0}^{\infty} \alpha^t = \frac{1}{1-\alpha} = 10 \end{aligned} \quad (15.4)$$

or by solving the system of equations given by (15.3):

$$\begin{aligned} J^{\mu}(1) &= 5 + \alpha J^{\mu}(2) \\ J^{\mu}(2) &= 1 + \alpha J^{\mu}(2) \end{aligned} \quad (15.5)$$

which yields the same result. If the transitions had a random element so that, for instance, Action 2 in State 2 would occasionally cause a transition to State 1, then (15.1) or (15.3) would become a bit more complicated, accounting for the cost of all possible trajectories along with their probabilities, but the main idea remains the same. As can be seen from Table 15.2, Policy 2 is the optimal policy since its value function is smallest.

A second problem, a very small "robot maze" SSPP, is depicted in Fig. 15.4. In both State 1 and State 2, the robot may attempt to move north, south, east or west, but the only possible transitions to a new state are from State 1 to State 2 by moving east or from State 2 to State 0 by moving south. Each attempted move incurs a cost of 1, representing the robot's energy usage or the elapsed time, and the trajectory ends when State 0 is reached, representing escape from the maze. The optimal policy is the one that leads most quickly out of the maze, which in this case is $\{\mu(1) = \text{East}, \mu(2) = \text{South}\}$. For this policy, $J^{\mu}(1) = 2$ and $J^{\mu}(2) = 1$; for any other deterministic policy μ, $J^{\mu}(1) = \infty$, and $J^{\mu}(2) = \infty$ unless $\mu(2) = \text{South}$, in which case $J^{\mu}(2) = 1$. Thus $\{\mu(1) = \text{East}, \mu(2) = \text{South}\}$ is the optimal policy, and in fact is the only proper deterministic policy because it is the only one that produces a trajectory guaranteed to terminate regardless of the initial state. Again, these calculations would become more complex if the transitions were not deterministic, e.g., if there were some chance that the robot could knock over a wall or trip when passing through a doorway.

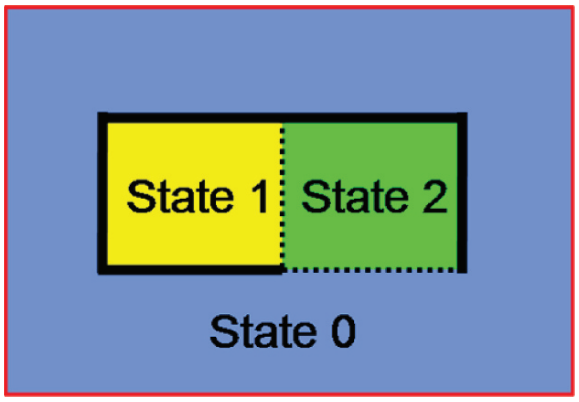

Fig. 15.4 Three state "robot maze" problem. Dotted lines represent open doorways, and solid lines represent walls. The robot may attempt to move north, south, east or west, but will only be successful in moving east from State 1 and west or south from State 2. Each move has a cost of 1, and the discount factor $\alpha = 1$, making this a stochastic shortest path problem (SSPP).

15.5 The Bellman Optimality Equation

In the 1950s, Richard Bellman formulated an equation – or, more precisely, a system of equations – that characterizes the optimal value function for an MDP (Bellman 1957). This equation forms the cornerstone of the classical theory of dynamic programming, and it is essential to reinforcement learning theory as well. The following version is adapted from those stated and proved in Bertsekas (1995) for SSPPs and Bertsekas and Tsitsiklis (1996) for DCPs.

The Bellman Optimality Equation

Suppose that an MDP is either a DCP or an SSPP such that there exists a proper deterministic policy and such that, for every improper deterministic policy, μ, $J^{\mu}(i) = \infty$ for some $i \in \boldsymbol{S}$. Then the optimal value function J^* exists, has finite components, and is the unique solution to the equations

$$\begin{aligned} J^*(i) &= \min\nolimits_{u \in U(i)} E_j \left[g(i,u,j) + \alpha J^*(j) \right] \\ &= \min\nolimits_{u \in U(i)} \sum_{j \in S} P^u(i,j)(\bar{g}(i,u,j) \\ &\quad + \alpha J^*(j)) \end{aligned} \quad (15.6)$$

for all states $i \in \boldsymbol{S}$, where the expectation is over the random cost g and next state j.

All theoretical results presented in this chapter for MDPs will assume that the MDP under discussion satisfies the hypotheses of the Bellman Optimality Equation.

Returning to the examples from the previous section, we note that for the DCP the optimal value function with $J^*(1) = 14$ and $J^*(2) = 10$ does satisfy (15.6) because

$$\begin{aligned}\min_{u\in U(i)} E_j\left[g(1,u,j)+\alpha J^*(j)\right] &= \min\left\{\bar g(1,1,2)+\alpha J^*(2), \bar g(1,2,1)+\alpha J^*(1)\right\}\\ &= \min\{5+(0.9)(10),\ 2+(.9)(14)\}\\ &= \min\{14,\ 14.6\} = 14 = J^*(1) \qquad (15.7)\end{aligned}$$

and

$$\begin{aligned}\min_{u\in U(i)} E_j\left[g(2,u,j)+\alpha J^*(j)\right] &= \min\left\{\bar g(2,1,1)+\alpha J^*(1), \bar g(2,2,2)+\alpha J^*(2)\right\}\\ &= \min\{0+(0.9)(14),\ 1+(.9)(10)\}\\ &= \min\{12.6,\ 10\} = 10 = J^*(2) \qquad (15.8)\end{aligned}$$

Similarly, for the SSPP example,

$$\begin{aligned}\min_{u\in U(i)} E_j\left[g(1,u,j)+\alpha J^*(j)\right] &= \min\left\{\bar g(1,\text{“east”},2)+\alpha J^*(2),\ \bar g(1,\text{“south”},1)\right.\\ &\quad \left.+\alpha J^*(1),\ \bar g(1,\text{“west”},1)+\alpha J^*(1),\ldots\right\}\\ &= \min\{1+(1)(1),\ 1+(1)(\infty), 1+\ (1)(\infty),\\ &\quad 1+(1)(\infty)\} = 2 = J^*(1) \qquad (15.9)\end{aligned}$$

and

$$\begin{aligned}\min_{u\in U(i)} E_j\left[g(2,u,j)+\alpha J^*(j)\right] &= \min\left\{\bar g(2,\text{East},2)+\alpha J^*(2), \bar g(2,\text{South},0)\right.\\ &\quad \left.+\alpha J^*(0), \bar g(2,\text{West},1)+\alpha J^*(1),\ \ldots\right\}\\ &= \min\{1+(1)(1),\ 1+(1)(0), 1+(1)(2),\\ &\quad 1+(1)(1)\} = 1 = J^*(2) \qquad (15.10)\end{aligned}$$

The Bellman Optimality Equation is important for several reasons. First, its characterization of the optimal value function is useful for theoretical purposes. Second, it leads to direct methods for computing the optimal value function. One procedure, called *value iteration*, performs an iteration based on (15.6) to successively approximate J^*. The Bellman Optimality Equation can also be written as a linear program, which can be solved by classical methods for small state and action spaces (Bertsekas and Tsitsiklis 1996, p. 37):

$$\begin{aligned}&\text{maximize} \sum_{i\in S} J(i)\\ &\text{subject to } J(i) \le \sum_{j\in S} P^u(i,j)\ \left(\bar g(i,u,j)+\alpha J(j)\right) \qquad (15.11)\end{aligned}$$

Third, the Bellman Optimality Equation provides a connection between the optimal value function and optimal policies. It can be shown that a deterministic policy μ is optimal *if and only if*

$$\begin{aligned}&E_j\left[g(i,\mu(i),j)+\alpha J^*(j)\right]\\ &\quad = \min_{u\in U(i)} E_j\left[g(i,u,j)+\alpha J^*(j)\right] \qquad (15.12)\end{aligned}$$

for all states $i \in \boldsymbol{S}$ (see Williams 2000). That is, a policy is optimal if and only if the action it prescribes for each state is one that achieves the optimal value function as characterized by the Bellman Optimality Equation.

If a model of the MDP is available, the transition probabilities $P^u(i, j)$ and expected costs $\bar g(i,u,j)$ may be substituted into (15.6) and it can be solved iteratively, or the linear program in (15.11) could be used. Once J^* is found, (15.12) determines an optimal policy. If a model is not available, a learning agent could use exploratory actions and their resulting costs and state transitions to create one by estimating the transition probabilities and associated expected costs, then use one of these approaches. Alternatively, the learning agent could use the same experience to iteratively improve an estimate of the optimal value function, but then an alternative to (15.12) must be found for determining an optimal policy. This approach to learning from experience can be achieved via iterative methods that use an extended concept of value functions that apply to pairs of states and actions, as described in the following section.

15.6 *Q*-Values

The total expected future discounted cost obtained when a certain action is executed in some state and then a fixed policy is followed thereafter is called an *action value function* or, for historical reasons, a *Q*-factor or *Q*-value. *Q*-values are particularly useful in determining how to change a policy to improve

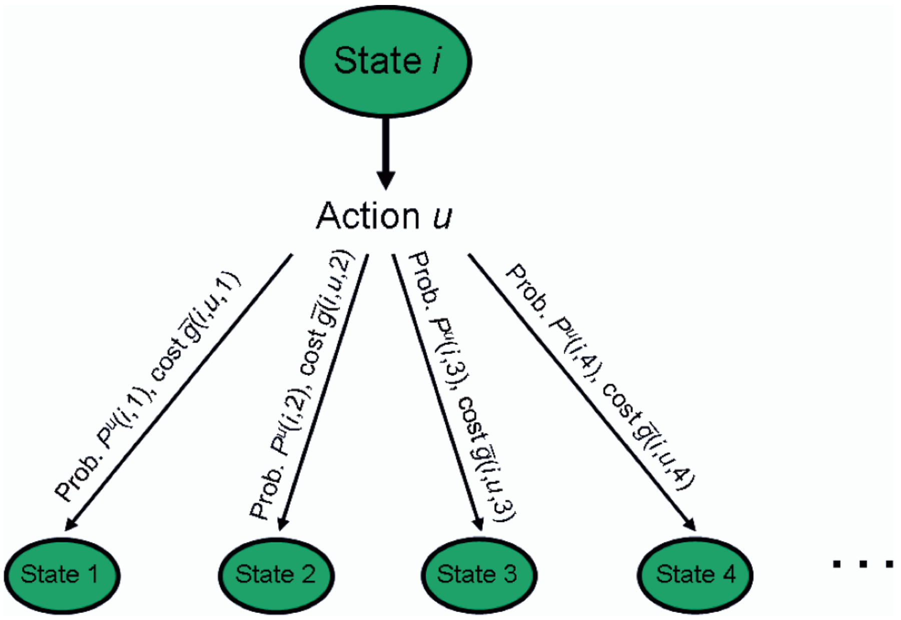
State i
Action u
Prob. $P^u(i,1)$, cost $\bar{g}(i,u,1)$
Prob. $P^u(i,2)$, cost $\bar{g}(i,u,2)$
Prob. $P^u(i,3)$, cost $\bar{g}(i,u,3)$
Prob. $P^u(i,4)$, cost $\bar{g}(i,u,4)$
State 1
State 2
State 3
State 4
...

Fig. 15.5 Diagram of the possible transitions from taking action u in state i, along with their probabilities and expected costs. The Q-value for a policy μ, Q^μ, may be computed by taking a probability-weighted combination of all the transition costs plus discounted value functions of the resulting states, J^μ, as given by (15.14). The optimal Q-value, Q^*, is similarly computed from the optimal value function, J^*, in (15.15).

it. Furthermore, the optimal policy for an MDP can be determined directly from its optimal Q-value, denoted Q^*, without requiring a model of the environment. Formally, we define the Q-value for a policy μ by

$$Q^\mu(i,u) = E\left[g(i_0,u,i_1) + \sum_{t=1}^{\infty} \alpha^t g(i_t,\mu(i_t),i_{t+1}) \;\middle|\; i_0 = i\right] \tag{15.13}$$

Here the expectation is taken over all possible trajectories and transition costs under the policy μ after action u is taken in state i. Note that (15.13) is different from (15.1) only in that the action taken in state i_0 is not necessarily the one prescribed by the policy μ; however, $Q^\mu(i,\ \mu(i)) = J^\mu(i)$. The Q-value for a stochastic policy π is defined similarly to (15.13), but with all actions following the first one being chosen according to π.

As is true for the value function, there is a consistency equation for Q-values that follows immediately from their definition and the definition of the value function. For any deterministic policy μ,

$$\begin{aligned} Q^\mu(i,u) &= E_j\left[g(i,u,j) + \alpha J^\mu(j)\right] \\ &= \sum_{j\in S} P^u(i,j)\,\left(\bar{g}(i,u,j) + \alpha J^\mu(j)\right) \end{aligned} \tag{15.14}$$

for each state $i \in \boldsymbol{S}$ and action $u \in U(i)$, where the expectation is over the random cost g and next state j. A similar equation holds for stochastic policies. The right side of (15.14) may be thought of as following all the possible state transitions and costs that can result from taking action u in state i and weighting the results – the transition costs plus the discounted value function of the resultant state – according to their probabilities. This is diagrammed in Fig. 15.5. Equation (15.13) could also be represented by an infinitely large tree of the possible trajectories, where the discounted costs of each transition weighted by their probabilities are added up over all the branches.

An *optimal Q-value* is one which has the smallest achievable value for each state-action pair. That is, if we denote the optimal Q-value function by Q^*, then for all states $i \in \boldsymbol{S}$ and actions $u \in U(i)$,

$Q^*(i,u) \le Q^\pi(i,u)$ for all (deterministic or stochastic) policies π and $Q^*(i,u) = Q^\pi(i,u)$ for some policy π. It follows from (15.14) and the definition of the optimal value function that the optimal Q-value satisfies the equation

$$Q^*(i,u) = E_j\left[g(i,u,j) + \alpha J^*(j)\right] \qquad (15.15)$$

for all states $i \in \boldsymbol{S}$ and actions $u \in U(i)$. Thus (15.6) may be written in terms of the optimal Q-value as

$$J^*(i) = \min_{u\in U(i)} Q^*(i,u) \qquad (15.16)$$

The Bellman Optimality Equation for value functions can also be stated in terms of Q-values. In particular, under the hypotheses for the Bellman Optimality Equation, the optimal value function Q^* exists, has finite components, and is the unique solution to the equations

$$\begin{aligned} Q^*(i,u) &= E_j\left[g(i,u,j) + \alpha \min_{v\in U(j)} Q^*(j,v)\right] \\ &= \sum_{j\in S} P^u(i,j)\,(\bar{g}(i,u,j) \\ &\quad + \alpha \min_{v\in U(j)} Q^*(j,v)) \end{aligned} \qquad (15.17)$$

for all states $i \in \boldsymbol{S}$ and actions $u \in U(i)$. Furthermore, it follows from (15.12) and (15.15) that the deterministic policy μ is optimal if and only if

$$Q^*(i,\mu(i)) = \min_{u\in U(i)} Q^*(i,u) \qquad (15.18)$$

for all states $i \in \boldsymbol{S}$. According to this characterization, an optimal policy can be determined directly from the optimal Q-value even in the absence of a model of the environment by simply selecting an action in each state that has the smallest value of Q^*. This is a powerful advantage for a wide range of problems.

In words, (15.15) or (15.17) define $Q^*(i,u)$ as the total of the future discounted costs expected from taking action u in state i, assuming that the optimal policy is followed for every action thereafter. The "robot maze" problem described earlier and illustrated in Fig. 15.4 allows a particularly concrete interpretation: $Q^*(i,u)$ is the expected minimum number of steps from state i to the exit (state 0), including the result of action u. The values of Q^* for this problem are depicted in Fig. 15.6. For example, $Q^*(2, \text{East}) = 2$ because trying to move east from State 2 (and bouncing off the wall) uses one step, and then moving south and out of the maze requires a second step. The shortest path to the exit can be found by simply moving in the direction associated with the smallest value of Q^* in each state. Thus, finding the optimal Q-value is tantamount to finding the optimal policy.

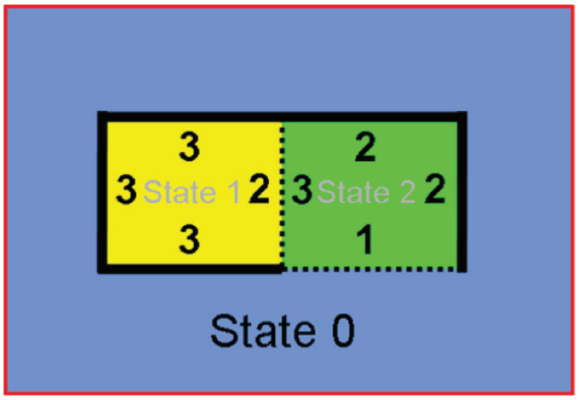

Fig. 15.6 The three-state robot maze with optimal Q-values displayed for each state in the direction of the associated move. For this problem, the Q^* values represent the minimum number of steps to the "exit" including the given action.

15.7 Policy Improvement

Even when a model of the environment is available, there are various reasons to seek other solution methods for an MDP than value iteration using (15.6) or the linear program in (15.11). Value iteration can be computationally intensive, and other methods may often be faster. Linear programming techniques are effective for small state and action spaces, but may not work well for large problems. And a model of the MDP that is estimated from experience may have errors that lead to a poor solution with either approach. Finally, if something is known *a priori* about a good policy, a learning framework that can make use of that knowledge is desirable. For these reasons, a wide variety of reinforcement learning algorithms that make direct use of experience obtained from interacting with the environment have been proposed. Many of these use some sort of *policy iteration*. At the root of this method is the observation that, from (15.14) and the fact that $J^\mu(i) = Q^\mu(i,\mu(i))$ it follows that for any two policies μ and μ', if $Q^\mu(i,\mu'(i)) \le Q^\mu(i,\mu(i))$ for all states $i \in \boldsymbol{S}$, then $J^{\mu'}(i) \le J^\mu(i)$ for all $i \in \boldsymbol{S}$; if in addition $Q^\mu(i,\mu'(i)) < Q^\mu(i,\mu(i))$ for some state $i \in \boldsymbol{S}$, then $J^{\mu'}(i) < J^\mu(i)$ for that state. This result is called the Policy Improvement Theorem. Policy

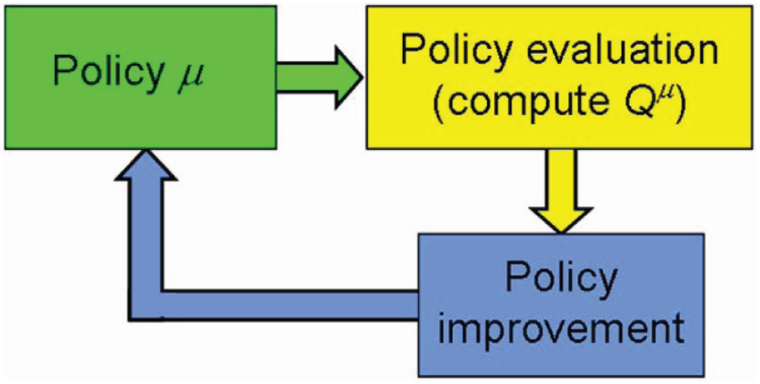

Fig. 15.7 A high-level diagram illustrating policy iteration: the cycle of improvement may continue until an optimal policy is found. Many practical reinforcement learning algorithms have this form.

iteration operates by iteratively improving a policy until it becomes optimal. This is done by evaluating the Q-value for a policy and then choosing a new policy that, for each state, uses an action that appears better (or at least equally good) based on the Q-value. It may be described more precisely as follows.

Policy Iteration Algorithm

Select an initial policy μ_0, either at random or based on *a priori* knowledge of a good solution to the MDP. At each subsequent time $t \geq 0$, evaluate Q^{μ_t} exactly and update the policy according to

$$\mu_{t+1}(i) = \begin{cases} \mu_t(i) \text{ if } Q^{\mu_t}(i, \mu_t(i)) \\ \quad = \min_{u \in U(i)} Q^{\mu_t}(i, u), \text{ or} \\ \text{any element } v \text{ for which } Q^{\mu_t}(i, v) \\ \quad = \min_{u \in U(i)} Q^{\mu_t}(i, u) \text{ otherwise} \end{cases} \tag{15.19}$$

for all states $i \in \boldsymbol{S}$. Repeat this process until $\mu_{t+1} = \mu_t$, then stop. μ_{t+1} is an optimal policy.

A high-level diagram of the Policy Iteration Algorithm is shown in Fig. 15.7. The Policy Iteration Algorithm will eventually end because there are only a finite number of possible policies, and at each step the policy improvement theorem guarantees that $J^{\mu'}(i) \leq J^{\mu}(i)$ for all $i \in \boldsymbol{S}$ and $J^{\mu'}(i) < J^{\mu}(i)$ for at least one state $i \in \boldsymbol{S}$, preventing any policy from being repeated.

Policy iteration provides a method for improving a policy once its Q-values for all state-action pairs are known, but it does not address how these values are to be obtained. The process of computing the value function or Q-value for a policy is called *policy evaluation* or *prediction*. If a model of the environment is known, computing the value function is sufficient since each Q-value component can be obtained from a "one-step lookahead" using (15.14). In this case, there are a number of classical methods in dynamic programming for policy evaluation, including the iterative solution of (15.3). Another possibility is to use a Monte-Carlo approach: execute the policy many times starting from each state and average the returns, as suggested by (15.1); the law of large numbers ensures that this average eventually converges to the value function. The temporal difference algorithms discussed in the next section provide a parameterized combination of these two approaches that may be used in the model-free case.

Algorithms that use the framework of policy iteration but do not evaluate the Q-value exactly before updating the policy are commonly called *generalized* or *optimistic* policy iteration algorithms. Many practical reinforcement learning algorithms are of this type.

15.8 Temporal Difference Learning

Temporal difference (TD) algorithms provide a way for a learning agent to learn from experience with the environment. The basic idea is to update an estimate of the MDP's value function based on the difference of successive estimates of the value of a state as new experience is obtained. For instance, suppose an estimate at time t of the value of state i_t for the policy μ is $J_t(i)$. Starting in state i_t and executing the action

$\mu(i_t)$ may result in a transition to state j accompanied by a cost $g(i_t, \mu(i_t), j)$, and the new state j will have an associated value function estimate $J_t(j)$. The quantity $g(i_t, \mu(i_t), j) + \alpha J_t(j)$ is an estimate of the right-hand side of (15.3), called the *one-step return*. Of course, this estimate involves only one sample of what may be a noisy random process, and it is also impacted by whether or not $J_t(j)$ is a good estimate of $J^\mu(j)$. Therefore, it is not advisable to immediately replace $J_t(i_t)$ with this new estimate, but rather to "nudge" $J_t(i_t)$ a small step toward it. This may be accomplished by choosing a small step-size γ_t between 0 and 1 and making the assignment

$$\begin{aligned} J_{t+1}(i_t) &= (1-\gamma_t)J_t(i_t) \\ &\quad + \gamma_t\left(g(i_t, \mu(i_t), j) + \alpha J_t(j)\right) \\ &= J_t(i_t) + \gamma_t(g(i_t, \mu(i_t), j) \\ &\quad + \alpha J_t(j) - J_t(i_t)) \end{aligned} \tag{15.20}$$

This equation represents moving $J_t(i_t)$ a fraction γ_t of the distance toward $g(i_t, \mu(i_t), j) + \alpha J_t(j)$. The quantity $g(i_t, \mu(i_t), j) + \alpha J_t(j) - J_t(i_t)$ is called a *temporal difference*, and the update method just described is called a *one-step temporal difference* algorithm, or TD(0). If the iteration (15.20) is performed repeatedly for all possible initial states $i_t \in \boldsymbol{S}$ and the values of γ_t converge to 0 in an appropriate fashion, the estimates J_t will converge to J^μ. This iteration of successive "nudges" is called a *stochastic approximation* or *Robbins-Monro* approach to the solution of the consistency equation (15.3); see Robbins and Monro (1951) and Kushner and Yin (1997). A familiar example of stochastic approximation is the incremental calculation of a population mean as new sample values are obtained. For instance, suppose $M_t = \text{mean}(\{x_1, x_2, \ldots, x_t\})$ and now a new sample x_{t+1} has become available. Then the new mean, M_{t+1}, may be written

$$\begin{aligned} M_{t+1} &= \frac{1}{t+1}\sum_{n=1}^{t+1} x_n = \frac{1}{t+1}\left(t\,M_t + x_{t+1}\right) \\ &= M_t + \frac{1}{t+1}\left(x_{t+1} - M_t\right) \end{aligned} \tag{15.21}$$

This equation has the same form as (15.20), with the sample value x_{t+1} representing a new (though noisy) estimate for the population mean and the step-size $\gamma_t = (t+1)^{-1}$. Of course, as $t \to \infty$, the law of large numbers guarantees that the values of M_{t+1} defined by (15.21) will converge to the mean of the distribution from which the x_t are drawn (assuming the distribution is well-behaved, e.g., bounded). In fact, convergence to the mean will occur for any sequence γ_t between 0 and 1 so long as $\sum_{t=0}^{\infty} \gamma_t = \infty$ and $\sum_{t=0}^{\infty} \gamma_t^2 < \infty$. (Note that the second condition implies that $\gamma_t \to 0$.) These are standard step-size conditions for stochastic approximation methods. For many more results and applications of stochastic approximation, see the text by Kushner and Yin (1997).

Analogs of (15.20) can be formulated using N-step returns, that is, results from a sequence of N successive actions instead of just one, along with the value of the final state. Formally, an *N-step return* beginning at time t in state i_t is defined by

$$R_t^{(N)} = \sum_{k=0}^{N-1} \alpha^k g(i_{t+k}, u_{t+k}, i_{t+k+1}) + \alpha^N J_{t+N}(i_{t+N}) \tag{15.22}$$

As is true for one-step returns, the N-step return provides a new estimate for the value function of the initial state. Larger values of N generally will produce higher variance N-step returns since they involve more random transitions, but lower bias since the value function of the final state has less influence. In fact, if N is allowed to go to infinity, the N-step return approaches an unbiased Monte-Carlo return. An algorithm having the form of (15.20) but using N-step returns under a fixed policy might be called an N-step temporal difference algorithm. However, such algorithms are rarely used in practice. Much more common is to use a weighted average of *all* the N-step returns, parameterized by a value λ between 0 and 1. The λ-return beginning at time t in state i_t is defined by

$$R_t^\lambda = (1-\lambda)\sum_{N=1}^{\infty} \lambda^{N-1} R_t^{(N)} \tag{15.23}$$

Recall that $(1-\lambda)\sum_{N=1}^{\infty} \lambda^{N-1} = 1$ for $0 \le \lambda < 1$, so (15.23) represents a weighted average of the $R_t^{(N)}$ terms with coefficients that sum to 1. Of course, the λ-return for a state i_t cannot generally be computed at the original time the state is visited, since future experience is required to determine the N-step returns. Fortunately, the λ-return update can be decomposed in a clever way that makes it possible to continue nudging the value function towards the λ-return as that future experience is obtained. For λ between 0 and 1, we may write the λ-return temporal difference as shown below. (Here the time subscripts on the value function estimates J have been omitted for conciseness.)

$$\begin{aligned}
R_t^\lambda - J(i_t) &= (1-\lambda)\sum_{N=1}^{\infty}\lambda^{N-1}R_t^{(N)} - J(i_t)\\
&= (1-\lambda)\sum_{N=1}^{\infty}\lambda^{N-1}\left(\sum_{k=0}^{N-1}\alpha^k g(i_{t+k},u_{t+k},i_{t+k+1}) + \alpha^N J(i_{t+N})\right) - J(i_t)\\
&= (1-\lambda)\sum_{N=1}^{\infty}\sum_{k=0}^{N-1}\lambda^{N-1}\alpha^k g(i_{t+k},u_{t+k},i_{t+k+1}) + (1-\lambda)\sum_{N=1}^{\infty}\lambda^{N-1}\alpha^N J(i_{t+N}) - J(i_t)\\
&= (1-\lambda)\sum_{k=0}^{\infty}\sum_{N=k+1}^{\infty}\lambda^{N-1}\alpha^k g(i_{t+k},u_{t+k},i_{t+k+1}) + (1-\lambda)\sum_{N=1}^{\infty}\lambda^{N-1}\alpha^N J(i_{t+N}) - J(i_t)\\
&= \sum_{k=0}^{\infty}\lambda^k\alpha^k g(i_{t+k},u_{t+k},i_{t+k+1}) + \sum_{k=0}^{\infty}\lambda^k\alpha^{k+1}J(i_{t+k+1}) - \sum_{k=0}^{\infty}\lambda^k\alpha^k J(i_{t+k})\\
&= \sum_{k=0}^{\infty}\lambda^k\alpha^k\left(g(i_{t+k},u_{t+k},i_{t+k+1}) + \alpha J(i_{t+k+1}) - J(i_{t+k})\right)
\end{aligned} \tag{15.24}$$

Thus, the difference between the λ-return and the initial value function estimate for state i_t may be written as a weighted sum of quantities very close to one-step temporal differences, except that they are based on the original estimate J instead of the estimate available when the action is taken. Denoting the one-step temporal difference at time t by

$$\delta_t = g(i_t,u_t,i_{t+1}) + \alpha J_t(i_{t+1}) - J_t(i_t) \tag{15.25}$$

the formula in (15.24) can be approximated as

$$R_t^\lambda - J_t(i_t) \approx \sum_{k=0}^{\infty}\lambda^k\alpha^k\delta_{t+k} \tag{15.26}$$

The one-step temporal difference update in (15.20) may be replaced with one based on λ-returns,

$$J_{t+1}(i_t) = J_t(i_t) + \gamma_t\sum_{k=1}^{\infty}(\lambda\alpha)^k\delta_{t+k} \tag{15.27}$$

The full sequence of one-step temporal differences is not immediately known when state i_t is visited, but (15.27) can be used to continue adjusting the current estimate of $J(i_t)$ as additional experiences (and hence, additional terms in the sum) are gathered. The basic idea of this approach is illustrated in Fig. 15.8: at each timestep, the latest temporal difference may be used to adjust the value functions for all states visited so far. In particular, k steps after time t, $J_{t+k}(i_t)$ can be incremented by $\gamma_t(\lambda\alpha)^k\delta_{t+k}$. In a practical application of this approach, it is convenient to keep track of the temporal difference discount factor $(\lambda\alpha)^k$ for each state using a so-called *eligibility trace*, which starts out at one when a state is visited and then "decays" by a factor $\lambda\alpha$ at each subsequent timestep. These ideas lead to the formulation of a temporal difference algorithm known as TD(λ), described formally below; the timestep subscripts on e and γ are omitted for simplicity.

TD(λ) Algorithm

Suppose that a stepsize γ between 0 and 1 has been chosen. Initialize J_0 arbitrarily, and let $e(i) = 0$ for all states $i \in \boldsymbol{S}$. Simulate the MDP, selecting actions u_t as prescribed by the fixed policy μ. After each transition to a new state i_{t+1}, perform the following updates:

$$\begin{aligned}
&\delta_t = g(i_t,u_t,i_{t+1}) + \alpha J_t(i_{t+1}) - J_t(i_t)\\
&e(i_{t+1}) = e(i_{t+1}) + 1\\
&\qquad\text{(or "replacing traces" variant: } e(i_{t+1}) = 1)\\
&J_{t+1}(i) = J_t(i) + \gamma\, e(i)\,\delta_t \text{ for all } i \in S\\
&e(i) = \lambda\,\alpha\, e(i) \text{ for all } i \in S
\end{aligned} \tag{15.28}$$

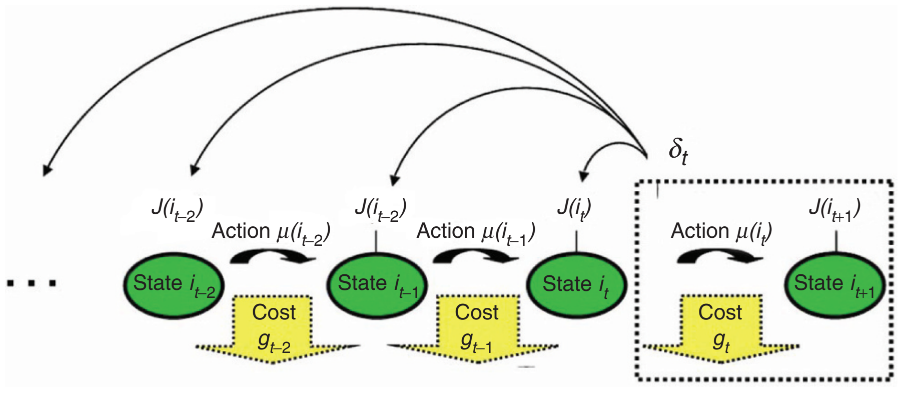

Fig. 15.8 Illustration of temporal difference learning of the value function for a policy μ. The estimate of J for every state visited so far is updated based on the temporal difference obtained from the latest timestep, $\delta_t = g_t + \alpha J(i_{t+1}) - J(i_t)$, where $g_t = g(i_t, \mu(i_t), i_{t+1})$.

The variant of TD(λ) in which the eligibility trace for a state is reset to one after each visit is called a *first-visit* or *replacing traces* method, in contrast to the *every-visit* or *accumulating traces* method in which the eligibility trace is always incremented by one. If $\lambda = 0$, this algorithm reduces to the one-step temporal difference algorithm, TD(0), described earlier. If $\lambda = 1$, it becomes an *on-line Monte-Carlo* algorithm. When the fixed stepsize γ is replaced by an appropriately decreasing sequence γ_t and every state is visited infinitely often, TD(λ) has been shown to converge to the optimal value function under fairly general conditions (Bertsekas and Tsitsiklis 1996; Dayan and Sejnowski 1994; Jaakkola et al. 1994). Several variants of TD(λ) have also been proposed; for instance, λ may be changed or "tuned" during learning to improve performance, or the eligibility traces may be updated differently.

One weakness of TD(λ) is that it learns the policy's value function, J^{μ}, not its Q-value. This is a significant deficiency in the model-free case, since policy improvement via (15.19) requires knowing the Q-value. In general, temporal difference methods for learning Q-values directly in the context of policy iteration can be problematic, both practically and theoretically. However, an algorithm that uses one-step temporal differences to learn the optimal Q-value directly is described in the next section.

15.9 *Q*-Learning

The temporal difference algorithms described in the previous section were stochastic approximation methods for solving the consistency equation (15.3) for a fixed policy. In contrast, Q-learning is a stochastic approximation to value iteration for solving (15.17), thus learning the optimal Q-value, Q^*, directly. This algorithm, proposed by Chris Watkins in his 1989 Ph.D. dissertation (Watkins 1989), was a major breakthrough in reinforcement learning theory. By learning Q^*, Q-learning immediately yields the optimal policy via (15.18). Another powerful feature of Q-learning is that it allows the learning agent to learn about the *optimal* policy while executing a completely *arbitrary* policy.

The basic idea of Q-learning is to update an estimate Q_t of Q^* at every timestep based on the one-step temporal difference $\delta_t = g(i_t, u_t, i_{t+1}) + \alpha \min_{v \in U(i_{t+1})} Q_t(i_{t+1}, v) - Q_t(i_t, u_t)$. The quantity $g(i_t, u_t, i_{t+1}) + \alpha \min_{v \in U(i_{t+1})} Q_t(i_{t+1}, v)$ from the latest interaction with the environment provides a new (but noisy and possibly biased) estimate for $Q^*(i_t, u_t)$, and so the stochastic approximation "nudging" approach described in the previous section is used to refine the current Q^* estimate based on this new information. This process is illustrated in Fig. 15.9 and described formally below.

Q-Learning Algorithm

Suppose $\gamma_t(i, u) \geq 0$ for all times t, states $i \in \boldsymbol{S}$ and actions $u \in U(i)$, and initialize Q_0 arbitrarily. Simulate the MDP under any policy, not necessarily deterministic or even stationary. After each

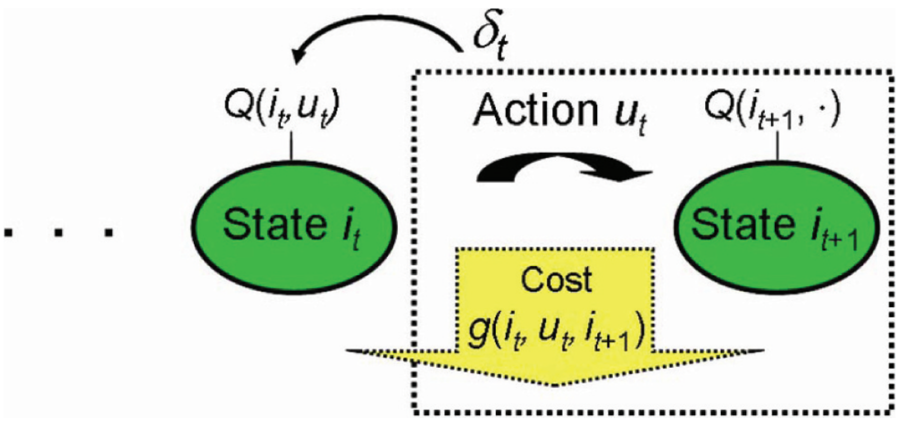

Fig. 15.9 Diagram of the Q-learning algorithm update. A one-step temporal difference based on the latest transition cost and minimum Q-value in the new state is used to update the optimal Q-value estimate for the action taken in the previous state.

transition from a state i_t to a new state i_{t+1} under action u_t, perform the following update:

$$Q_{t+1}(i_t, u_t) = Q_t(i_t, u_t) + \gamma_t(i_t, u_t)\big(g(i_t, u_t, i_{t+1}) + \alpha \min_{v \in U(i_{t+1})} Q_t(i_{t+1}, v) - Q_t(i_t, u_t)\big) \tag{15.29}$$

If the stepsizes γ_t get small in an appropriate fashion as $t \to \infty$ and each state-action pair (i,u) is visited infinitely often, then it can be shown that $\lim_{t\to\infty} Q_t = Q^*$. To guarantee that every action is explored in every state infinitely often as the learning agent interacts with the environment, Q-learning requires that a stochastic or non-stationary policy be used. On the other hand, to maintain low transition costs and perhaps speed convergence, it is common to give priority to those actions that appear best under the current Q-estimate (the so-called *greedy* actions). One strategy for achieving this balance between exploration and exploitation is to execute an *ε-greedy* policy which in state i takes an action having the smallest Q-value with probability $1 - \varepsilon$ and an action chosen randomly from $U(i)$ with probability ε, where ε is between 0 and 1. Another is to use the *Boltzmann* policy, defined for all states $i \in \boldsymbol{S}$ and all $u \in U(i)$ by

$$\pi_t(i, u) = \frac{\exp(-Q_t(i, u)/T)}{\sum_{v \in U(i)} \exp(-Q_t(i, v)/T)} \tag{15.30}$$

where $T > 0$ is called the *temperature*. An equivalent but numerically more stable alternative is to use the "advantages" $A_t(i, u) = Q_t(i, u) - \min_{v \in U(i)} Q_t(i, v)$ in place of the Q-values in (15.30). By taking $\varepsilon \to 0$ or $T \to 0$ sufficiently slowly under an ε-greedy or Boltzmann policy, one can ensure that the Q-learning algorithm converges to Q^* while also allowing the policy being executed to approach the optimal policy (see Bertsekas and Tsitsiklis 1996, p. 251). Choosing a schedule for ε or T that provides a good rate of convergence to Q^* for a particular MDP may require experimentation or tuning using a separate optimization technique. In some practical on-line applications, convergence may not be desired: choosing an appropriate fixed value of ε or T allows the learning agent to continue exploring, which may be quite useful for adapting to changes when the MDP being solved isn't truly stationary. In this case, the choice of a good fixed value for T depends on the magnitude of the values of Q and the desired amount of exploration.

15.10 Temporal Difference Control

The Q-learning algorithm described in the previous section is, in a sense, a complete solution to the classical reinforcement learning problem. It learns the optimal Q-value for an MDP, and hence the optimal policy, while offering significant flexibility in the policy actually followed. However, Q-learning uses only a one-step temporal difference return; not only does this lead to potentially high bias in an update when the next state's Q-estimate is lousy, but it also means that only the very next experience following a visit to a state-action pair is used to improve its Q-estimate. The rate of convergence of Q-learning is therefore slower than might be achieved by using more experience (longer sequences of actions and transitions) per update. For this reason, a number of algorithms have been proposed that combine ideas from eligibility traces, policy iteration, and Q-learning with the goal of providing faster convergence; among these are Sarsa(λ) and several versions of $Q(\lambda)$. The main idea of these methods is to use eligibility traces and temporal differences to update the Q-value estimate; some also use this estimate to attempt to improve the current policy.

The two $Q(\lambda)$ algorithms described below attempt to duplicate Q-learning's ability to learn the optimal Q-value while following an arbitrary policy. The first of these is due to Watkins (1989). Motivated by an expansion of (15.17), his algorithm makes use of all N-step returns beginning from a state-action pair (i, u) in a manner similar to TD(λ). However, these returns are terminated as soon as a non-greedy action is selected.

Watkins' Q *(λ)*

Let the stepsizes $\gamma(i, u)$ be between 0 and 1, $e(i, u) = 0$, and initialize Q_0 arbitrarily. Simulate the MDP under any policy, not necessarily deterministic or even stationary. After each transition to a new state i_{t+1} and selection of the next action u_{t+1}, perform the following updates:

$$\begin{aligned}
\delta_t &= g(i_t, u_t, i_{t+1}) + \alpha \min\nolimits_{v \in U(i_{t+1})} Q_t(i_{t+1}, v) - Q_t(i_t, u_t) \\
e(i_t, u_t) &= e(i_t, u_t) + 1 \text{ (or “replacing traces” variant: } e(i_t, u_t) = 1) \\
Q_{t+1}(i, u) &= Q_t(i, u) + \gamma(i, u)\, e(i, u)\, \delta_t \text{ for all } i \in S \text{ and } u \in U(i) \\
e(i, u) &= \begin{cases} \lambda\, \alpha\, e(i, u) \text{ if } Q_t(i_{t+1}, u_{t+1}) = \min_v Q_t(i_{t+1}, v), \text{ or} \\ 0 \text{ otherwise} \qquad\qquad\qquad \text{for all } i \in S \text{ and } u \in U(i) \end{cases}
\end{aligned} \tag{15.31}$$

For the replacing traces variant in this and other algorithms that use eligibility traces to learn Q-values, a slight enhancement is available. In addition to setting $e(i_t, u_t) = 1$, one may set $e(i_t, u) = 0$ for all $u \neq u_t$, thus terminating all N-step returns when a state is revisited. This strategy has been shown empirically to outperform both the every-visit and standard replacing traces methods (Singh and Sutton 1996).

The main weakness of Watkins' $Q(\lambda)$ algorithm is that the lengths of the backups will be very short when many exploratory actions are taken. If actions are selected based on an ε-greedy or Boltzmann policy with ε or T decreasing with time, this will often be the case early in the learning process. A second “optimistic” or “naïve” $Q(\lambda)$ algorithm, proposed by Sutton and Barto (1998) as a simpler alternative to a similar algorithm due to Peng and Williams (1996), ameliorates this problem by allowing all returns to be used – even those following an exploratory action. Although this will result in some technically incorrect updates being performed, the initial rate of learning should be higher than in Watkins' $Q(\lambda)$ algorithm, and the algorithms become nearly identical as the probability of exploratory actions is decreased.

Naïve Q*(λ)*

Let the stepsizes $\gamma(i,u)$ be between 0 and 1, $e(i,u) = 0$, and initialize Q_0 arbitrarily. Simulate the MDP under any policy, not necessarily deterministic or even stationary. After each transition to a new state i_{t+1}, perform the following updates:

$$\begin{aligned}
\delta_t &= g(i_t, u_t, i_{t+1}) + \alpha \min_{v \in U(i_{t+1})} Q_t(i_{t+1}, v) - Q_t(i_t, u_t) \\
e(i_t, u_t) &= e(i_t, u_t) + 1 \text{ (or ``replacing traces'' variant: } e(i_t, u_t) = 1) \\
Q_{t+1}(i, u) &= Q_t(i, u) + \gamma(i, u)\, e(i, u)\, \delta_t \text{ for all } i \in S \text{ and } u \in U(i) \\
e(i, u) &= \lambda \alpha e(i, u) \text{ for all } i \in S \text{ and } u \in U(i)
\end{aligned} \tag{15.32}$$

Note that $Q(0)$ under either Watkins' or Naïve $Q(\lambda)$ is simply ordinary Q-learning. In order for Q_t to converge to Q^*, it is generally necessary to replace the stepsizes $\gamma(i,u)$ with sequences decreasing to zero in an appropriate fashion, and to guarantee that each state-action pair is visited infinitely often; additional conditions may also be required. Under any of these methods, once Q^* is found, an optimal policy may be determined via (15.18).

A final temporal difference control algorithm, Sarsa(λ), is so-named because it uses temporal differences based on *S*tate-*a*ction-*r*eward(cost)-*s*tate-*a*ction sequences. Unlike the Q-learning or $Q(\lambda)$ algorithms, its temporal differences do not use the minimum Q-value of the next state, but rather the Q-value for the action actually selected. For this reason, Sarsa(λ) is called an *on-policy* algorithm; it learns the Q-value of the policy being executed, not the optimal Q-value. In particular, as in the Naïve Q (λ) algorithm, returns are not terminated when non-greedy actions are taken. Indeed, the N-step returns used by Sarsa(λ) can be thought of as samples of a multi-step version of the Q-value consistency equation 15.14 rather than of the Bellman Optimality equation 15.17.

Sarsa(λ)

Let the stepsizes $\gamma(i,u)$ be between 0 and 1, $e(i,u) = 0$, and initialize Q_0 and π_0 arbitrarily. Simulate the MDP, selecting actions u_t at each timestep as prescribed by π_t. After each transition to a new state i_{t+1} and selection of the next action u_{t+1}, perform the following updates:

$$\begin{aligned}
\delta_t &= g(i_t, u_t, i_{t+1}) + \alpha Q_t(i_{t+1}, u_{t+1}) - Q_t(i_t, u_t) \\
e(i_t, u_t) &= e(i_t, u_t) + 1 \text{ (or ``replacing traces'' variant: } e(i_t, u_t) = 1) \\
Q_{t+1}(i, u) &= Q_t(i, u) + \gamma(i, u)\, e(i, u)\, \delta_t \text{ for all } i \in S \text{ and } u \in U(i) \\
e(i, u) &= \lambda\, \alpha\, e(i, u) \text{ for all } i \in S \text{ and } u \in U(i)
\end{aligned} \tag{15.33}$$

and determine a new policy π_{t+1} using some function of $t + 1$ and Q_{t+1}.

There are many ways that the new policy π_{t+1} may be determined; for instance, it may be an ε-greedy or a Boltzmann policy, where ε or T decreases with time. In order that $\lim_{t\to\infty} Q_t = Q^*$, the step sizes γ should be replaced with an appropriately decreasing sequence. The sequence of policies must eventually assign positive probabilities only to actions that are greedy with respect to the Q-estimate while ensuring that each state-action pair is visited infinitely often. Such a sequence of policies is called *greedy in the limit with infinite exploration* (GLIE) in Singh et al. (2000), where it is proven that one-step Sarsa, Sarsa(0), converges under GLIE policy sequences. Additional results on the convergence of optimistic policy iteration algorithms may be found in Tsitsiklis (2002).

15.11 Partially Observable MDPs

The techniques presented in this chapter are effective for solving MDPs, but in some practical applications, the exact state of the environment may not

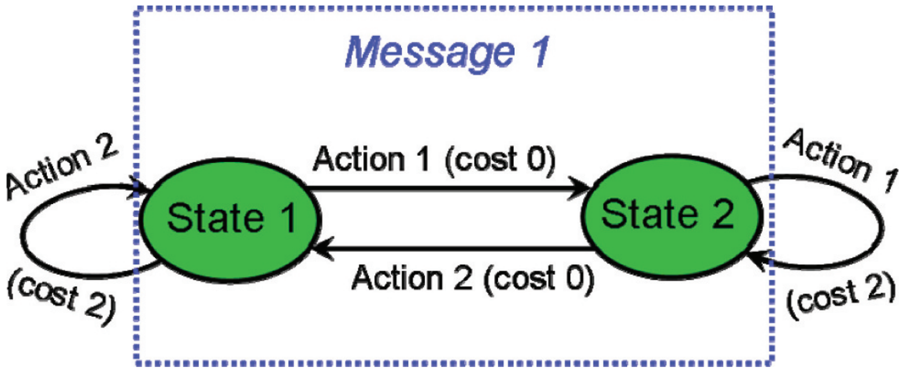

Fig. 15.10 Example POMDP. The underlying MDP has two distinct states, two available actions and deterministic state transitions represented by the arrows; costs associated with the state transitions are indicated in parentheses. However, the learning agent cannot sense the precise states but only a single message that fails to distinguish them.

easily be determined. With only an estimate of the state available, the history of past actions may influence future costs and transition probabilities, violating the Markov hypothesis. When these effects are small or the problem is otherwise well-behaved, traditional reinforcement learning methods such as Q-learning may still provide good solutions. However, in general these Partially Observable Markov Decision Processes (POMDPs) may be much more difficult than MDPs to solve (Lovejoy 1991).

To formalize the notion of a POMDP, we assume that the environment has the structure of an MDP but the learning agent perceives only an observation, or *message*, from a set $\mathbf{M} = \{m_1, m_2, \dots m_M\}$, with the message determined according to a conditional probability distribution $P(m|i)$ for message m given the true state i. The learner will in general not have a model for the underlying MDP or even know the number of true states. Thus, while actions continue to drive state transitions in the MDP, only the costs incurred and the current message will be available to the learning agent.

One approach to solving POMDPs is to use the history of observations, actions, and costs to try to obtain an improved estimate of the true, hidden state (e.g., Chrisman 1992), but such methods can be computationally expensive and may not scale well for large problems. An alternative approach is to learn the best stochastic memoryless policy, i.e., a policy that prescribes a probability distribution over available actions based only on the immediately available message. Such policies are the natural solution to a wide class of problems including games and their economic analogs, since an intelligent opponent could adapt to exploit any fixed, deterministic policy. A simple example of a POMDP for which the optimal memoryless policy is stochastic is depicted in Fig. 15.10. Although there are two distinct states in the underlying MDP, the learning agent receives a message that fails to distinguish them. If action 1 is always taken, a cost of 2 will be incurred at every timestep following the first one. The same is true if the policy is to always take action 2. Thus, either deterministic policy will result in an average cost per timestep of 2. However, a stochastic policy that prescribes taking action 1 with probability 0.5 and action 2 with probability 0.5 will result in an average cost per timestep of 1; this is the optimal memoryless policy for the POMDP. Since stochastic policies include deterministic policies, which simply set the probability of all but one action in each state to 0, it always makes sense to look for the optimal memoryless policy for a POMDP in the class of stochastic policies.

Unlike in an MDP, where there exists an optimal policy that simultaneously produces the minimum possible value function for each state, there may not be such a policy for a POMDP. Thus, a measure of performance other than the expected sum of discounted costs (the measure used for MDPs) may be required. One such measure is given by the asymptotic average cost per timestep that a stochastic policy π achieves, as described in the example above. A method for computing estimates of Q-values related to this quantity and using them to incrementally improve a stochastic policy is described in Jaakkola et al. (1995). A modified version of this method was used in Williams and

Singh (1999) to learn stochastic policies for several problems including a matrix game, a robot navigating a maze with imperfect state sensor, and the dynamic assignment of jobs in a server queue. An alternative approach, which uses the idea of gradient descent in a parameterized stochastic policy space, may be found in Baxter and Bartlett (2000).

15.12 Learning with Function Approximation

Although the theory and algorithms described so far in this chapter have assumed that the states and actions are discrete and sufficiently small in number that their value function or Q-values could be stored and updated in "table-lookup" fashion, this may not be true for many practical control problems. For example, realistic navigation problems do not take place within a simple maze in which only four moves are possible; rather, the locations may be described via distances from certain landmarks, the terrain may be variable, and movements in any direction may be possible. In cases of very large, infinite, or continuous state and action spaces, it becomes necessary to represent value functions and Q-values via a parameterized function approximator – e.g., a linear approximation or a neural network. When the reinforcement algorithm being utilized calls for a stochastic approximation Q-value update, the function approximator's parameters are adjusted in such a fashion that the Q-value it represents for the relevant state and action is nudged moves in the direction of the new estimate.

To make this more precise, let us consider on-line learning algorithms that operate on Q-values and suppose that $Q(i, u) = f_{\mathbf{w}}(i, u)$, where $\mathbf{w}$ is a vector of parameters for the approximation function f. For instance, if f were a neural network, $\mathbf{w}$ might represent the vector of all connection weights and activation thresholds; in a polynomial approximation, it might represent the set of all coefficients. At time t, the vector $\mathbf{w}_t$ yields a Q-value $Q_t(i_t, u_t) = f_{\mathbf{w}_t}(i_t, u_t)$ and the learning algorithm produces a new estimate $\hat{Q}_{t+1}(i_t, u_t)$ for $Q_{t+1}(i_t, u_t)$. The vector $\mathbf{w}_t$ must now be incrementally changed so that $f_{\mathbf{w}_{t+1}}(i_t, u_t)$ is "nudged" towards $\hat{Q}_{t+1}(i_t, u_t)$. Here nudging is used in place of simply adjusting $\mathbf{w}_t$ to make $f_{\mathbf{w}_{t+1}}(i_t, u_t) = \hat{Q}_{t+1}(i_t, u_t)$ because the latter could unduly degrade the Q-value representations of other nearby state-action pairs. In fact, the manner in which state-action pairs (i_t, u_t) are chosen for updates can be quite important in determining whether the approximation $f_{\mathbf{w}_t}$ behaves as desired (Tsitsiklis and Van Roy 1997; Tadic 2001). Interaction with a real or simulated environment while following a GLIE policy seems to be a good approach to producing a suitable sampling of state-action pairs, but choosing an appropriate schedule for diminishing exploration may require experimentation.

Once a method for generating actions is chosen, $\mathbf{w}_t$ may be adjusted by performing a single step of gradient descent on the squared difference between the function approximator output and the new Q-value estimate:

$$\begin{aligned}\mathbf{w}_{t+1} &= \mathbf{w}_t - \frac{1}{2}\xi_t \, \nabla_{\mathbf{w}_t} \left[f_{\mathbf{w}_t}(i_t, u_t) - \hat{Q}_{t+1}(i_t, u_t) \right]^2 \\ &= \mathbf{w}_t - \xi_t \left[f_{\mathbf{w}_t}(i_t, u_t) - \hat{Q}_{t+1}(i_t, u_t) \right] \nabla_{\mathbf{w}_t} f_{\mathbf{w}_t}(i_t, u_t)\end{aligned} \tag{15.34}$$

where ξ_t is a stepsize parameter between 0 and 1 and $\nabla_{\mathbf{w}} f$ is the gradient of f – the vector of partial derivatives of f with respect to the components of $\mathbf{w}$. If f is a neural network, the update of $\mathbf{w}_t$ may be performed using standard on-line backpropagation, where the association $(i_t, u_t) \mapsto \hat{Q}_{t+1}(i_t, u_t)$ is considered a training example. In a learning algorithm that produces a new Q estimate of the form $\hat{Q}_{t+1}(i_t, u_t) = Q_t(i_t, u_t) + \gamma_t \, \delta_t$, (15.34) becomes

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \xi_t \, \gamma_t \, \delta_t \, \nabla_{\mathbf{w}_t} f_{\mathbf{w}_t}(i_t, u_t) \tag{15.35}$$

Both ξ_t and γ_t must diminish towards zero in an appropriate fashion as learning continues in order for $\mathbf{w}_{t+1}$ to converge. If eligibility traces are used in the learning algorithm, as in $Q(\lambda)$ or Sarsa(λ), they must now be replaced with eligibility traces over the components of $\mathbf{w}$. A common approach is to update the vector of eligibility traces, $\mathbf{e}$, as

$$\mathbf{e}_{t+1} = \lambda \, \alpha \, \mathbf{e}_t + \nabla_{\mathbf{w}} f(i_t, u_t) \tag{15.36}$$

so that future adjustments via $\mathbf{w}_{t+1} = \mathbf{w}_t + \xi_t \, \gamma_t \, \delta_t \, \mathbf{e}_t$ will continue to update the components of $\mathbf{w}$ as new information is obtained, thus speeding learning.

It should be noted that there are very few theoretical results that guarantee the convergence of reinforcement learning algorithms when function approximation is used, and even for the case of linear function approximation with Q-learning there are counterexamples for which the Q-values diverge to infinity (Baird 1995; Precup et al. 2001). Nevertheless, there are also many success stories demonstrating the potential usefulness of this approach, and a number of techniques and rules of thumb have been developed for handling optimal stopping and other problems that arise (Bertsekas and Tsitsiklis 1996). As is always the case in using neural networks or other function approximators, it is very important that a data representation be carefully chosen. In particular, it is desirable to derive features of the states and actions for use as inputs to the function approximator; these should be chosen so that, to the extent possible, "nearby" features lead to similar costs and state transitions. Experience and experimentation will likely be necessary to get the best results.

15.13 Applications of Reinforcement Learning

This section presents applications of reinforcement learning to three sample problems relevant to environmental science: dynamic routing of sensor data in a wireless array, control of a scanning remote sensor, and aircraft routing in an environment for which probabilistic weather hazard information is available.

15.13.1 Dynamic Routing in Wireless Sensor Arrays

Wireless arrays of small, battery or solar-powered *in situ* sensors are beginning to provide an exciting new technology for environmental science research. Unlike many traditional research deployments, such arrays can be quickly and easily deployed in an *ad hoc* fashion to measure biogeochemical and other environmental processes at a high temporal and spatial resolution. In order to be most flexible and efficient, the sensors are placed in the locations of scientific interest and then self-organize their communications to relay or "hop" measurements back to a base station that records them. This approach, known as *mesh networking*, allows the sensor network to work well even when many of the sensors are out of line-of-sight with the base station (e.g., behind a tree or over the crest of a hill); moreover, it allows the use of smaller radios and lower-power transmissions than would be necessary for each sensor node to communicate directly with the base station.

Equipping a wireless sensor array to learn a good network communications structure and adapt it as conditions change can be accomplished using reinforcement learning techniques. One way to formulate the problem would be to consider it an SSPP similar to the "robot maze", where the goal is to relay a message from the originating node to the base station in the minimum number of steps. However, the sensor array communications routing problem is not really an SSPP because the connection strengths between nodes, available battery power, and network traffic are all variable and may change over time, and occasionally a node may fail altogether. These changes will alter the cost of transmission and the probability that a message will be successfully received. For example, if the shortest path to the base station is used repeatedly, some nodes may experience much more traffic than others and hence greater battery drain and more rapid failure. In order to ensure a robust network, the routing scheme should balance the transmission loads on the different nodes to the extent possible; thus, the performance metric to be optimized should incorporate not only the number of hops a message has to take, but also the traffic on the most power-limited nodes in the network.

There are many ways to formulate this problem, and we will use a fairly simple one. We choose as "states" the nodes in the network, with actions given by the choice of which target node to attempt to transmit to. If the message is received, the receiving node will include information on its battery power and its current minimum Q-value in the acknowledgement signal; in practice, this acknowledgement of reception may be obtained by "overhearing" the message's retransmission. We define the cost of a transmission to be 1/(fraction of battery power remaining) for the *receiving* node. If no acknowledgement signal is received in a certain time period, then the transmission is deemed to have failed, presumably because the targeted receiving node is out of range, has insufficient battery power, or is busy with another transmission. In this case, the state returns to the sending node, which is then also

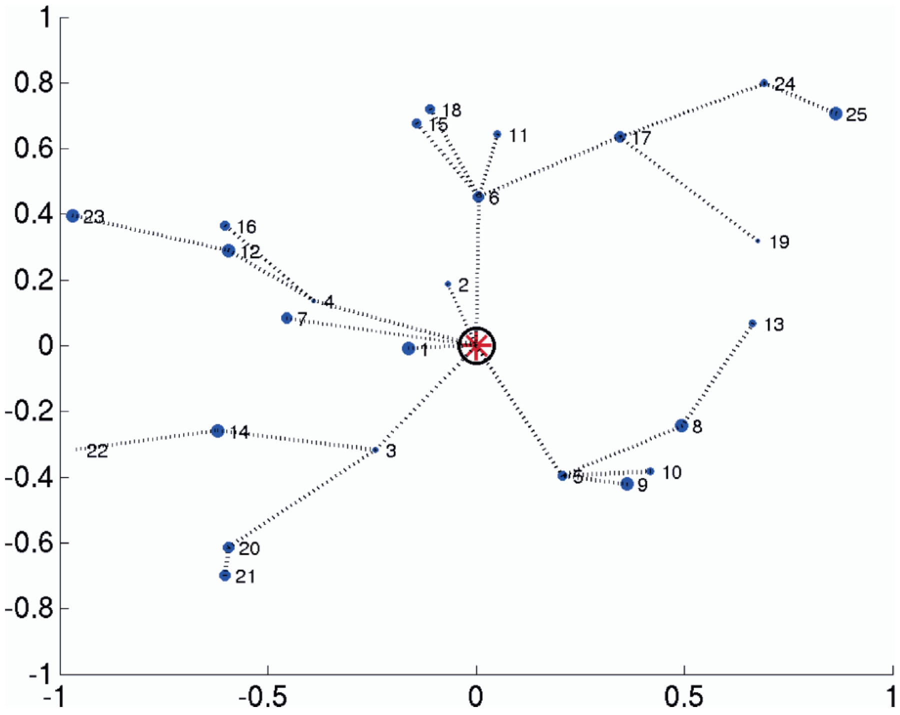

Fig. 15.11 Random deployment of a wireless sensor array consisting of 25 nodes, with the base station at the origin. The size of the blue circle representing each node is proportional to the frequency with which messages originate at that node. Nodes are numbered in order from closest to farthest from the base station, and the shortest-path routes (those with the lowest mean number of transmissions required to get a message to the base station) are indicated by dotted lines.

considered the receiving node for the purpose of determining the cost of the transmission. This formulation is not truly an MDP because the transition probabilities and costs are not stationary; rather, they change in time based on the history of past transmissions. Nevertheless, we will demonstrate that a straightforward application of Q-learning is capable of finding good policies for routing messages through the network and adapting the policies as conditions change.

We create a scenario for testing this approach via simulation by randomly placing 25 sensor nodes in a 2×2 unit domain with the base station in the center, as shown in Fig. 15.11. Each node is assigned a randomly-chosen probability for producing a message at each timestep. This probability accounts for the fact that sensor nodes may employ adaptive reporting strategies, with each node transmitting measurements more frequently when "interesting" phenomena are detected at its location. To model uniform time-based reporting, these probabilities could be set to 100%, or a regular reporting schedule could be implemented for each node. The probability that a message will be received by a target node and successfully acknowledged is determined by a function of the distance between the two nodes, as shown in Fig. 15.12. (This function was chosen for the purpose of this didactical example and does not necessarily represent the performance of any specific radio technology.) Each transmission reduces the sending node's remaining battery power by a fixed amount, and we assume that each battery begins with the capacity for sending 100,000 messages. Each node maintains a set of Q-values that are used to determine the transmission policy as described below and are updated after each transmission based on information received from the acknowledgment signal, or from the sending node if no acknowledgment is received.

The Q-learning algorithm described in Section 15.9 is a good choice for solving this on-line learning problem because information from the next state (node) is naturally available via the acknowledgment signal, whereas the multi-step backups needed

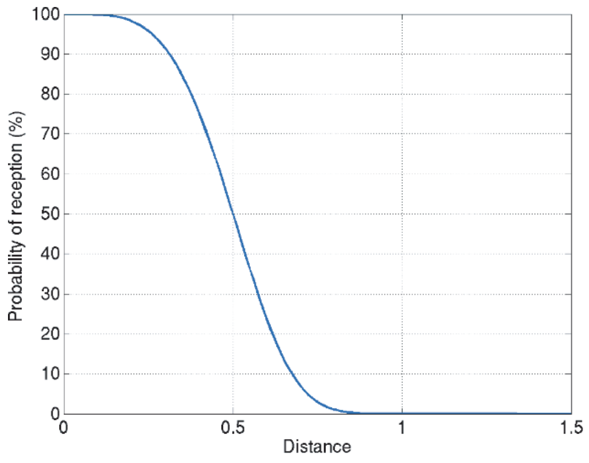

Fig. 15.12 Probability of a successful transmission from one sensor node to another, including receipt of an acknowledgement, as a function of the distance between them.

by $Q(\lambda)$ or Sarsa(λ) would require additional transmissions. We use the Boltzmann exploration policy from (15.30) with advantages $A_t(i,u) = Q_t(i,u) - \min_{v \in U(i)} Q_t(i,v)$ replacing Q-values and a choice of $T = 0.05$. Note that smaller values of T would tend to lead to fewer random "exploratory" transmissions and less ability to adapt to changing conditions, while larger values of T lead to more exploratory transmissions and lower network efficiency. The learning rate was chosen as $\gamma = 0.2$. Smaller values of γ would lead to more stable and possibly more accurate Q-values and a more consistent transmission strategy, while larger values allow quicker adaptation to changing conditions. The best choice of exploration and learning rates depends in part on the size of the network and the timescales of the changes that affect sensor reporting frequencies and transmission success; experimentation or tuning via simulation may be necessary to obtain optimal results.

Figure 15.13 shows comparative results from the fixed shortest-path routes (those with, on average, the smallest number of transmissions required to reach the base station) indicated by the dotted lines in Fig. 15.11 and the online adaptive Q-learning strategy described above. For each episode of the simulation, an originating node is chosen at random according to the origination probabilities, the message is transmitted through the network according to the current routing policy, and the episode ends when the message is received at the base station. The left panels show a running average of the total number of transmissions required for each message on a log-log plot, and the right panel shows the battery power remaining for each of the 25 sensors as a function of the cumulative number of messages received at the base station. While the number of transmissions per message remains constant over time for the static routing strategy (with some fluctuation due to the randomness in the simulation) and the battery power for each node decreases linearly with time, the results for the adaptive method shows an initial reduction in the average number of transmissions for each message as the network learns a good strategy and then an eventual increase as the network reorganizes to reduce the strain on the most-utilized nodes. The static routing strategy results in the failure of node 6 due to exhausted battery power after 213,967 messages are received at the base station, whereas the adaptive method successfully transmits 507,715 messages before failure. Thus, the adaptive approach is able to lengthen the lifetime of the entire network by more than a factor of two for this scenario. Of course, these results represent only single simulation runs using each method, and results can be expected to vary somewhat due to the random elements of the simulation.

Traditional mesh networks reorganize their routing topology periodically to adapt to changing conditions or the failure of some of the network nodes, and so avoid the stark failure illustrated here. A standard technique for reorganizing is to send a flood of transmissions, or "beacon", from the base station down to the nodes, which results in additional drain on the node batteries. The adaptive Q-learning method achieves the same objective while avoiding the extra transmission costs for periodic wholesale network reformation. A precise comparison of the two methods would require a more realistic simulation or *in situ* test for a specific application, but it is clear from this example that a technique based on reinforcement learning has the potential to produce efficient adaptive network routing strategies.

15.13.2 Adaptive Scanning Strategy for a Remote Sensor

An optimal control problem that arises frequently in environmental science research is how to target

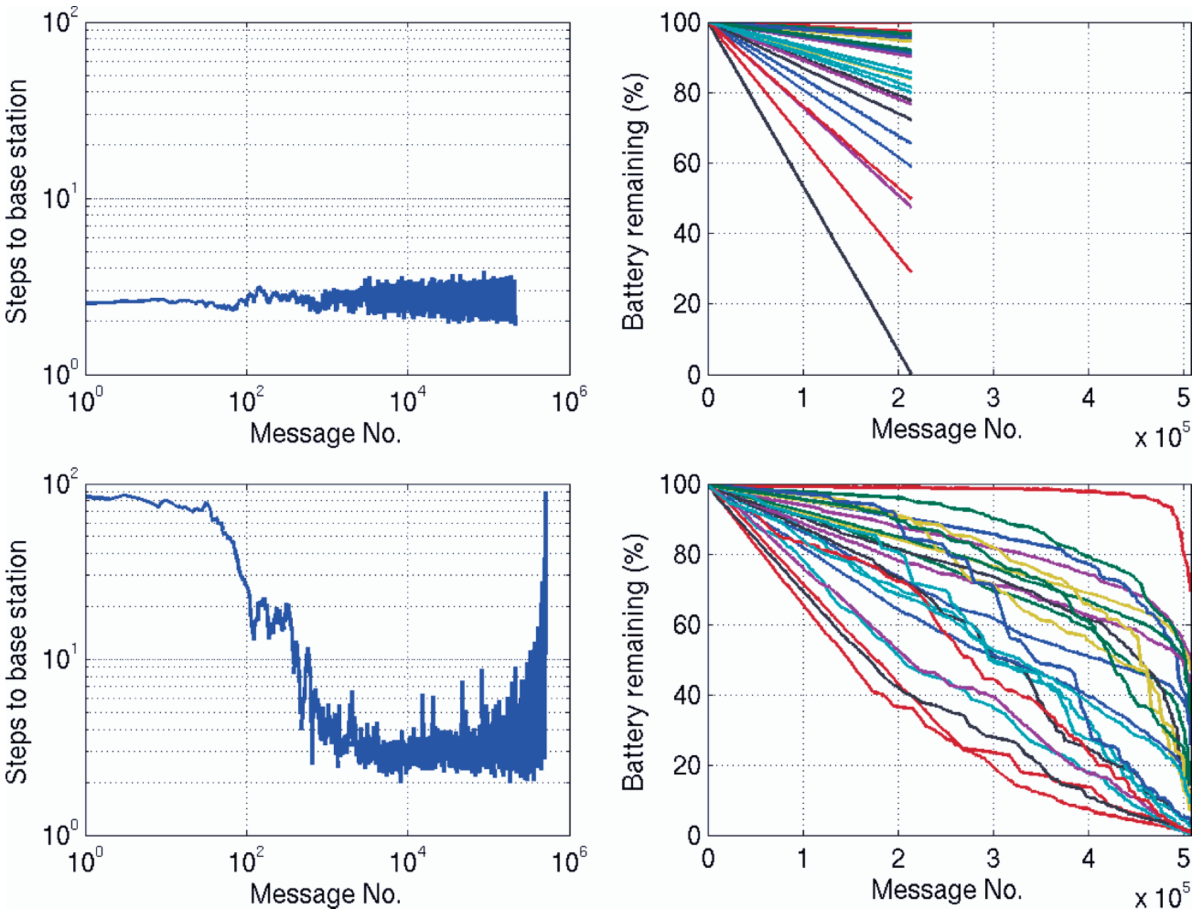

Fig. 15.13 (Top row) Results from using the shortest path routes indicated by the dotted lines in Fig. 15.11. (Bottom row) results from using dynamic routing based on Q-learning as described in the text. The plots on the left show a running average over 20 messages of the number of transmissions utilized for the message to reach the base station. The right hand side shows the remaining battery power in each node as a function of the message number. In order from the least to greatest battery power remaining after message number 2×10^5, the traces in the upper plot are for nodes 6, 5, 3, 4, 17, 20, 12, 24, 8, 14, 7, 23, 1, 9, 25, 21, 16, 18, 15, 13, 10, 11, 19, 2, and 22, respectively, and in the lower plot are for nodes 5, 6, 2, 1, 3, 17, 14, 7, 4, 8, 20, 24, 9, 16, 12, 23, 25, 21, 10, 15, 18, 13, 11, 19, and 22, respectively.

observations to produce the greatest benefit in understanding environmental processes, supplying data for modeling or forecasting, or providing timely warning of hazardous conditions. For example, adaptive scanning is employed by spaceborne remote sensors to improve their ability to capture significant meteorological phenomena using limited resources (Atlas 1982). Developers of the National Science Foundation's Collaborative Adaptive Sensing of the Atmosphere (CASA) X-band radar network plan to use coordinated adaptive sensing techniques to better capture precipitation events and detect flash floods or tornadoes (McLaughlin et al. 2005). And the United States' operational NEXRAD Doppler radars employ a number of volume coverage patterns (VCPs) appropriate to different weather scenarios, which may be selected automatically or by a human operator. Reinforcement learning can be used to develop an adaptive sampling strategy for a scanning remote sensor that balances a need for enhanced dwell-times over significant events with maintaining adequate temporal or spatial scanning coverage to quickly capture new developments.

In particular, we consider the problem of controlling a scanning remote sensor such as a radar, lidar, or scanning radiometer capable of taking measurements in one of four directions: north, east, south or west. At each timestep, the sensor may rotate clockwise, maintain its current orientation, or rotate counterclockwise; it then detects the state of the atmosphere in the new sector, characterized as "clear" (e.g., no clouds), "developing" (e.g., significant clouds), or "hazardous" (e.g., tornadic supercell). The sensor memory stores the most recent observations made in each sector and the elapsed times since they were last scanned. This information, coupled with the current observation, comprise the system state and will be the basis for the decision of which direction the sensor should rotate in the next timestep. In order to ensure a minimal temporal coverage and also limit the number of possible states in the MDP, the sensor reverts to a standard clockwise scan strategy whenever the elapsed time for any sector reaches 10 or more. Thus, the state is determined by the last observation in each of the four directions and the elapsed time in the directions not currently being observed, for $3^4 \cdot 12^3 = 139{,}968$ possible states. Of course, not all of these states are actually "reachable" – for instance, no two sectors can have the same elapsed times in practice. Moreover, symmetry can be used to further reduce the effective number of states; in essence, we assume that the sensor only rotates clockwise but the order of the sectors can be reversed.

Our goal is to train the sensor to quickly detect significant events (e.g., hazardous weather) and dwell on them to the degree possible to improve the quality of their characterization and reduce the lead time for warnings to the public. To achieve this, we impose a cost function that at each timestep charges a penalty for each sector in which there is hazardous weather. The amount of the penalty is based on the elapsed time, Δt, since the sector was scanned: for $\Delta t = 0$, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 timesteps, we define $g(\Delta t) = 0, 1, 2, 3, 4, 5, 6, 8, 10, 12, 14, 17,$ 20, respectively. Note that Δt is encoded in the MDP's state, so the cost function g is a deterministic function of the state. We choose a value of α close to 1 so that future costs are not discounted too much, say $\alpha = 0.995$. Finally, the manner in which the weather, w_t, changes with time may be specified via a conditional probability matrix such as

$$P(w_{t+1}|w_t) = \begin{bmatrix} 0.98 & 0.17 & 0.00 \\ 0.02 & 0.79 & 0.08 \\ 0.00 & 0.04 & 0.92 \end{bmatrix} \tag{15.37}$$

which may be interpreted as follows: if $w_t = 1$ ("clear"), the probability is 98% that $w_{t+1} = 1$, 2% that $w_{t+1} = 2$ ("developing"), and 0% that $w_{t+1} = 3$ ("hazardous"); if $w_t = 2$ the probabilities are 17%, 79%, and 4%; and if $w_t = 3$ the probabilities are 0%, 8%, and 92%, respectively. If the conditional probability matrix P doesn't change with time, a standard result from Markov theory tells us that at any later time $t + k$,

$$P(w_{t+k}|w_t) = P(w_{t+1}|w_t)^k \tag{15.38}$$

where the exponent by k, a positive integer, represents the matrix multiplied by itself k times. In fact, as $k \to \infty$, we can compute

$$\lim_{k\to\infty} P(w_{t+k}|w_t) = \lim_{k\to\infty} P(w_{t+1}|w_t)^k$$
$$= \begin{bmatrix} 0.85 & 0.85 & 0.85 \\ 0.10 & 0.10 & 0.10 \\ 0.05 & 0.05 & 0.05 \end{bmatrix} \tag{15.39}$$

showing that for the choice of P given by (15.37) the weather probability distribution will asymptotically reach the hypothetical climatological averages: "clear" 85% of the time, "developing" 10% of the time, and "hazardous" 5% of the time, regardless of the initial weather conditions.

We have now specified a model of the MDP, which comprises all the information needed to simulate the sensor observations, actions, state transitions and costs in order to use Q-learning, $Q(\lambda)$ or Sarsa(λ) to find the optimal Q-value and hence the optimal scanning policy. However, the large number of states and the relative rareness of hazardous weather means that these methods can be quite slow and very sensitive to the choice of learning rates and exploration strategy. The computational requirements could be reduced by employing state aggregation – i.e., grouping the elapsed times into categories such as "short", "long" and "very long" – to reduce the effective number of states, or by using function approximation, but either reduces the problem to a POMDP and may result in learning a good but not optimal policy. A faster and more accurate alternative is to utilize the MDP model in value iteration, solving (15.6) directly and then computing the optimal policy using (15.12). For every initial state and action, we compute the probability distribution over weather conditions in all four directions via (15.38) using the last recorded observations and the elapsed times. These in turn determine the

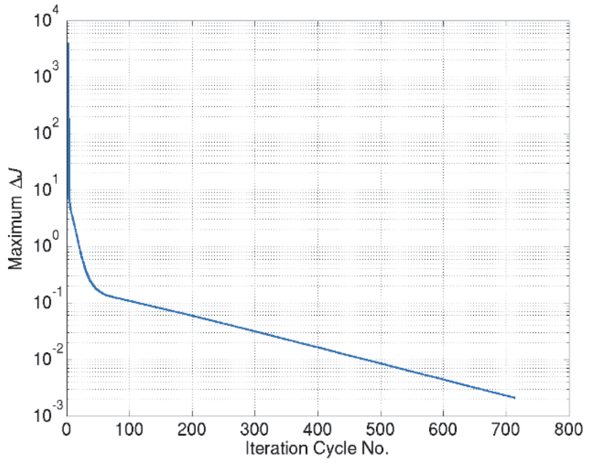

Fig. 15.14 Maximum change in the optimal value function estimate, J, after each cycle of updates during the value iteration process used to solve the remote sensor adaptive scanning problem. Because the estimate is initialized as $J = 0$, the changes are always positive.

probabilities of the three possible state transitions for each action (the new state is given by the deterministic elapsed times and random weather in the new scanning direction) and the mean cost of each transition (based on the random weather in all the other sectors). The value function J is initialized as 0 for all states, so that value iteration causes a monotonic increase in J for all states; if an elapsed time greater than 12 in any sector occurs, the value of that "state" is taken as $20(1 - \alpha)^{-1} = 4{,}000$ for the purpose of computing the right hand side of (15.6). Figure 15.14 shows the rate of convergence of the value iteration, as measured by the maximum change in the value function, $\max_{i \in S} |J_{t+1}(i) - J_t(i)|$, after each pass through all states. After rapid initial progress, the rate of convergence becomes exponential.

To evaluate the policy obtained from the final J-iterate via (15.12), separate 10^7-step simulations were performed for both the learned policy and a standard scan that simply rotated clockwise at every timestep. The average costs per timestep were found to be 0.16 and 0.30, respectively, showing that the learned policy improved performance by nearly a factor of two over the standard scan. In addition, for each occurrence of "hazardous" weather in any sector and at any timestep, the total number of timesteps elapsed since the sector containing it was last scanned were tabulated. As suggested by (15.39), "hazardous" weather occurred about 5% of the time, meaning that about 2×10^6 instances occurred in the four sectors over the course of each simulation. The results in Fig. 15.15 show that an elapsed time of 0 – the ideal value – was achieved about 2.5 times more frequently by the learned policy than the standard scan, while elapsed times of 1, 2 or 3 occurred less frequently. However, the learned policy did very occasionally, about 2.5% of the time, yield elapsed times of 4 or greater. The tradeoff between increased dwell over sectors with hazardous weather and the potential for occasional instances of long elapsed times is determined by the cost function, g, which can be altered to obtain different behavior.

Of course, this formulation of the scanning problem solved here is significantly simplified relative to those often encountered in practice, where weather or another phenomenon being measured is correlated from sector to sector and the scan strategy includes selection of elevations as well as azimuths. Nevertheless, this example provides a suggestion of how reinforcement learning might be used to help develop improved observing strategies.

15.13.3 *Aircraft Routing Using Probabilistic Forecasts*

An important goal of meteorology and environmental science is to provide reliable forecasts to aid planning and decision making. For instance, the U.S. Federal Aviation Administration's Joint Planning and Development Office (JPDO) has developed a Next Generation Air Transportation System vision (JPDO 2006) that requires probabilistic forecast grids to guide the routing of aircraft around potentially hazardous weather and improve the safety and efficiency of the National Airspace System (NAS).

The specific problem we consider is how to use probabilistic forecast grids at various lead times to plan aircraft routes that balance a desire to conserve fuel use with the need to avoid potential accidents. We focus on the problem of routing a single aircraft to a target airport, though in the NAS the safe separation of multiple aircraft must be considered as well. For the sake of simplicity, we use only 2-D Cartesian probabilistic forecast grids of aviation hazards (e.g., thunderstorms or turbulence) at a specified flight

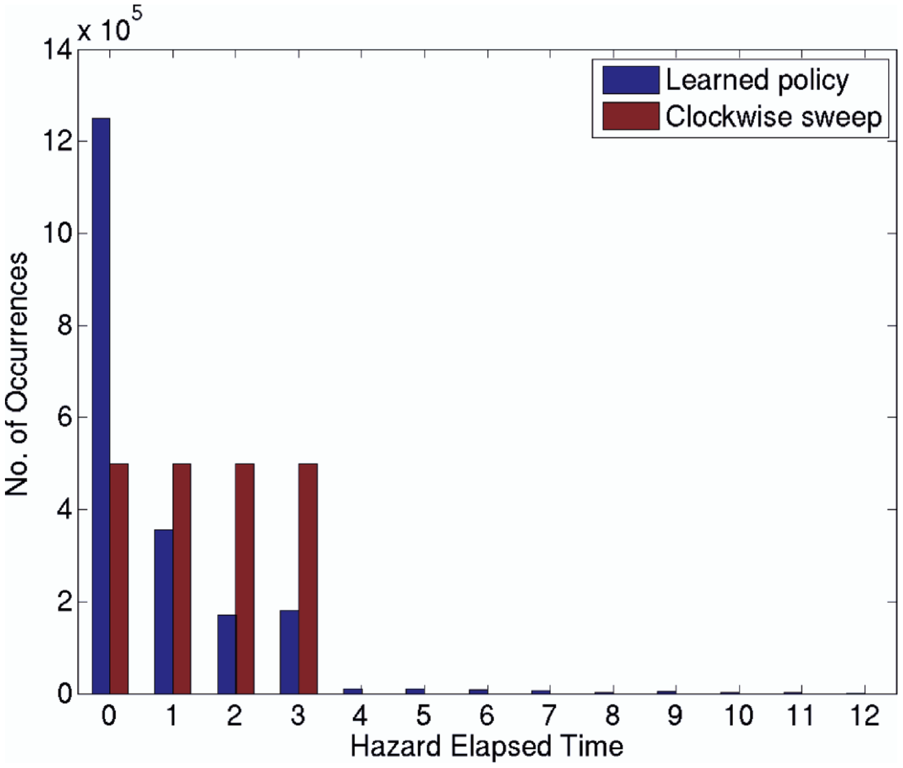

Fig. 15.15 Results from 10^7-step simulations of the scanning remote sensor using the optimal policy determined via reinforcement learning (blue) and a standard clockwise rotation at each timestep (red). The histograms show the total number of occurrences in all sectors and timesteps of the elapsed time since hazardous weather was scanned, where lower elapsed times are better; hazardous weather existed in each sector about 5% of the time, so the total number of occurrences was about 2×10^6 for each simulation.

level, but the approach illustrated here easily generalizes to 3-D grids in an appropriate map projection. The balance between fuel use and accident risk will depend on factors like the type of aircraft being flown; for instance, large aircraft are less fuel efficient but may be better equipped to mitigate hazards posed by atmospheric turbulence, icing, lightning or windshear. For a given flight, the probabilistic weather grids must be mapped to an assessment of the risk of a significant hazard to the aircraft's operation, which may be naturally quantified in terms of its potential economic impact to the airline; this allows it to be compared with other risks and impacts to form a basis for decision making. The MDP's state is given by the time and location of the aircraft. For the purpose of this example, a timestep is 5 min, the aircraft flies at a constant speed of 480 knots (40 nautical miles, nmi, per timestep), the cost of fuel is \$500 per timestep, and the weather forecast grids at each time t are scaled into the probability f_t of an accident costing \$2,500,000 for every timestep (5 min) spent in those conditions. For planning purposes, the cost of each flight segment is estimated as the sum of the fuel cost and the cost of an accident times its probability (that is, its "expected cost"). To be precise, the mean cost incurred at time t along a one-timestep flight segment from $\mathbf{x}_t$ along a vector $\mathbf{r}$ having length 40 nmi is given by

$$\begin{aligned}&\bar{g}\left(\{\mathbf{x}_t, t\}, \mathbf{r}, \{\mathbf{x}_t + \mathbf{r}, t+5\}\right)\\ &\quad = \$500 + \$2{,}500{,}000 \int_0^1 f_t\left(\mathbf{x}_t + \tau\,\mathbf{r}\right) d\tau \qquad (15.40)\end{aligned}$$

More complex versions of (15.40) could be developed to incorporate the cost of a delay after the flight exceeds a certain duration, or probabilities of multiple types of accidents or incidents and their associated costs due to, for example, taking an aircraft out of service for inspection or repair, providing worker's

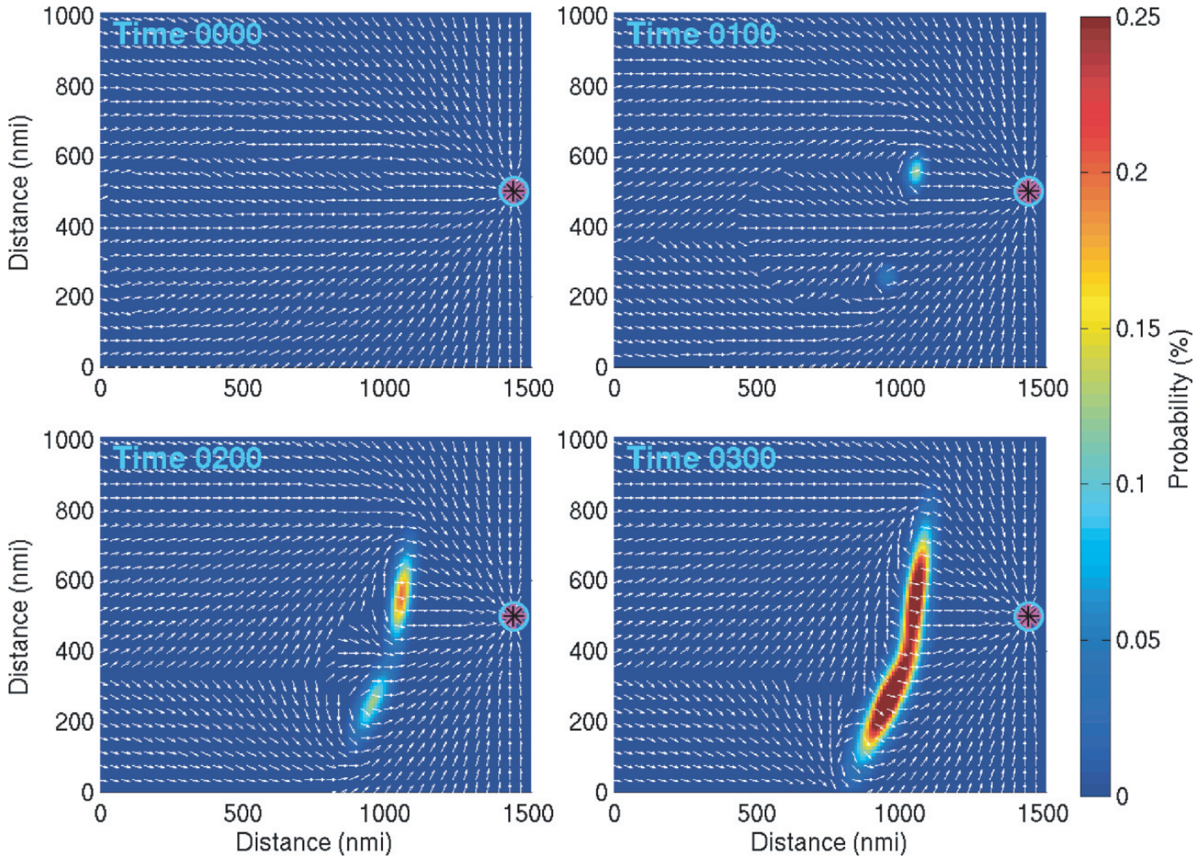

Fig. 15.16 Weather hazard probability forecasts and learned route vectors for the aircraft routing problem over a period of 3 h. The color scale gives the probability that an accident costing \$2.5 million will occur during 5 min of flight through the conditions forecast at that location and time, and the destination airport is depicted on the east side of the domain.

compensation for an injured flight attendant, settling a passenger injury claim, or sustaining damage to the airline's reputation that reduces future ticket sales. An example set of forecasts f_t shown for $t = 0$, 1, 2 and 3 h (times 0000, 0100, 0200 and 0300) is given by the color-scaled grids in Fig. 15.16. This example depicts a scenario in which a line of thunderstorms is forecast to develop and block a direct route from the west to an airport in the east. Grids of f_t are available at 5 min intervals between 0 and 3 h, after which we assume the forecasts don't change; that is, we take $f_t = f_{0300}$ for $t \geq 0300$.

The aircraft routing problem as defined above is an SSPP with states given by the aircraft's position and the time, actions given by the direction of travel, and deterministic state transitions accompanied by costs prescribed by (15.40). Because the possible states and actions are not discrete but occupy a continuum, it is necessary to employ function approximation in solving this MDP. We use a lookup table representation of the value function on a grid $\{(i\,\hat{\mathbf{x}},\, j\,\hat{\mathbf{y}}) \mid i = 0, \ldots, 37;\ j = 0, \ldots, 25\}$, where $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ are vectors of length 40 nmi pointing east and north, respectively; the value function at intermediate locations is estimated by linear interpolation. More precisely, the value function for the state $\{a\,\hat{\mathbf{x}} + b\,\hat{\mathbf{y}}, t\}$, where t is a non-negative multiple of 5 min and a and b are real numbers with $0 \leq a \leq 37$ and $0 \leq b \leq 25$, respectively, is approximated by

$$\begin{aligned}
&\tilde{J}(\{a\,\hat{\mathbf{x}} + b\,\hat{\mathbf{y}}, t\}) \\
&\quad = (\lceil a \rceil - a)\,(\lceil b \rceil - b)\;\; J\left(\{\lfloor a \rfloor\,\hat{\mathbf{x}} + \lfloor b \rfloor\,\hat{\mathbf{y}}, t\}\right) \\
&\qquad + (\lceil a \rceil - a)\,(b - \lfloor b \rfloor)\;\; J\left(\{\lfloor a \rfloor\,\hat{\mathbf{x}} + \lceil b \rceil\,\hat{\mathbf{y}}, t\}\right) \\
&\qquad + (a - \lfloor a \rfloor)\,(\lceil b \rceil - b)\;\; J\left(\{\lceil a \rceil\,\hat{\mathbf{x}} + \lfloor b \rfloor\,\hat{\mathbf{y}}, t\}\right) \\
&\qquad + (a - \lfloor a \rfloor)\,(b - \lfloor b \rfloor)\;\; J\left(\{\lceil a \rceil\,\hat{\mathbf{x}} + \lceil b \rceil\,\hat{\mathbf{y}}, t\}\right)
\end{aligned} \tag{15.41}$$

Here J denotes the value function at a grid point, the ceiling function $\lceil a \rceil$ represents the smallest integer greater than or equal to a, and the floor function $\lfloor a \rfloor$ represents the largest integer less than or equal to a. The optimal value function, J^*, may be found using value iteration, with the right hand side of (15.6) approximated by the minimum over directions $0°, 10°, 20°, 30°, \ldots, 350°$, the transition costs determined by (15.40), and the value function of the new state approximated by (15.41). We first perform value iteration to find J^* for the static SSPP with time "frozen" at $t = 0300$. In order to speed convergence, we initialize J as the cost of flying directly to the destination airport in the absence of weather hazards (which is easily computed from the distance) and then update J at grid points chosen in order of increasing distance from the airport during each iteration. Next we reduce t by 5 min intervals and perform just one sweep through all grid points, computing J^* for that time using (15.6). The reason that only one iteration at each time is now sufficient is that the optimal value function for the next time – and, therefore, the next state – has already been determined. These calculations are continued until $t = 0$. Finally, the learned policy for each state (position and time) may be computed using (15.12), that is, we choose a policy μ such that

$$\begin{aligned} &\bar{g}\left(\{a\,\hat{\mathbf{x}} + b\,\hat{\mathbf{y}}, t\}, \mu(\{a\,\hat{\mathbf{x}} + b\,\hat{\mathbf{y}}, t\}), \{a\,\hat{\mathbf{x}} + b\,\hat{\mathbf{y}} + \mathbf{r}_\mu, t+5\}\right) \\ &\quad + \tilde{J}^*\left(\{a\,\hat{\mathbf{x}} + b\,\hat{\mathbf{y}} + \mathbf{r}_\mu, t+5\}\right) \\ &= \min_{\{\mathbf{r}\,|\,\|\mathbf{r}\|=40\text{ nmi}\}} \Big[\bar{g}\left(\{a\,\hat{\mathbf{x}} + b\,\hat{\mathbf{y}}, t\}, \mathbf{r}, \{a\,\hat{\mathbf{x}} + b\,\hat{\mathbf{y}} + \mathbf{r}, t+5\}\right) \\ &\quad + \tilde{J}^*\left(\{a\,\hat{\mathbf{x}} + b\,\hat{\mathbf{y}} + \mathbf{r}, t+5\}\right)\Big] \end{aligned} \tag{15.42}$$

where $\tilde{J}^*$ represents the linear interpolation of the learned optimal value function and $\mathbf{r}_\mu$ is shorthand for the displacement traveled in 5 min of flight in the direction specified by the policy μ. The learned policy μ may not strictly be optimal due to the use of function approximation, under which Bellman's Equality is not guaranteed to hold. Nevertheless, the policy should be near-optimal if the grid on which the value function J is represented has an appropriate scale. The vectors in Fig. 15.16 show the learned policy determined via (15.42) for each point on the 40 nmi grid at $t = 0000$, 0100, 0200 and 0300, where again for simplicity we have limited consideration to directions specified at 10° increments.

The learned policy shown in Fig. 15.16 may be interpreted as follows. At each time shown, the vectors show the direction that an aircraft starting or continuing from that location should take based on the sequence of probabilistic forecasts of future hazards. The optimal route is determined by following this "flow" as it changes at each subsequent time. Note that the domain has been chosen so that an aircraft could nearly traverse its width (1,480 nmi) in 3 h. At time 0000, the vectors in the western 2/3 of the domain are already adjusting routes to avoid the weather hazard that is forecast to develop. By time 0100, the western region shows three distinct routing strategies depending on location: go to the north of the developing line of storms, south of it or, if within about 500 nmi, fly through the gap. By 0200, the area in which aircraft are directed towards the gap has shrunk as it begins to close, and by 0300 the line of storms has fully developed and aircraft are directed around it or attempt to exit it as quickly as possible.

In addition to finding the optimal route, another important consideration may be whether to fly at all. Figure 15.17 shows the result of subtracting the learned value function from the optimal value function found in the absence of weather, a difference that represents the expected increased cost due to the weather hazard of a flight originating or continuing from the indicated location and time, assuming that it follows the prescribed optimal route from that point onwards. At time 0000, a slight increase in cost may be seen, primarily in the western part of the domain, due to the slight elongation of the optimal flight path or enhanced likelihood of an accident. By time 0100 this cost has grown substantially, particularly in the region in the west from which the aircraft is routed around the north or south of the line. The costs increase further at times 0200 and 0300, particularly for aircraft that will have to deviate substantially to avoid the storm or those that are already in it. An airline dispatcher might use this marginal "cost-to-go" information to help determine whether it would be worthwhile to delay or cancel a flight, or even divert an en route flight to another airport.

The method and results presented for this aircraft routing example can be extended in a straightforward way to more complicated scenarios. For instance, current routing in the NAS requires aircraft to fly along pre-specified "jetways" or between defined waypoints.

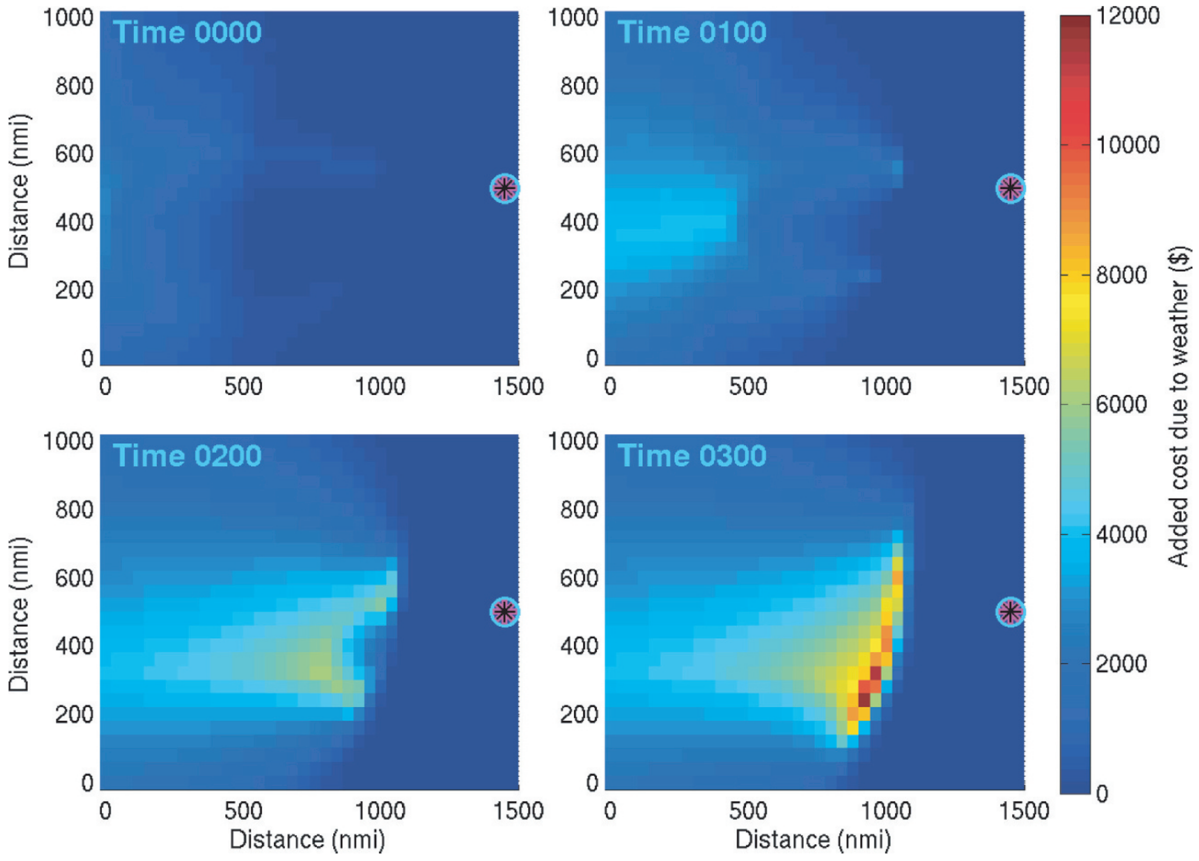

Fig. 15.17 The added (marginal) flight cost due to the weather hazard for flights originating or continuing from the indicated location and time. These values were obtained by subtracting the optimal value function, J^*, for a no-hazard situation from the optimal value function learned for the scenario depicted in Fig. 15.16 using the method described in the text.

Restricting routes to certain paths would actually substantially simplify the MDP by reducing the number of states and actions, allowing tabular representation of the value function and obviating the need for function approximation. As previously mentioned, a third dimension (altitude) can easily be added to the forecast grids and routes, and the cost function can be modified to account for the increased cost of flying at lower levels where aerodynamics are less favorable. The effect of winds, including the jet stream, can be accommodated by adding the wind vector to the aircraft's velocity at each timestep. The cost function would be specific to each aircraft type, since models have different capabilities, fuel use and ability to withstand weather and other hazards. Finally, the effects of congestion (unsafe density of aircraft) may be included as an additional hazard whose expected cost is added in (15.40). Starting with an initial forecast of congestion, the method described above may be used to find the optimal routes for all aircraft desiring to occupy the NAS, or to determine that a ground delay program or cancellation is appropriate. The optimal paths of all those flights may then be traced to revise the congestion forecast for each location and time. Then new optimal routes may be chosen based on this new cost information, and the process repeated. If this iteration is done carefully (e.g., as a relaxation method), a good overall set of routes may be obtained. Marginal costs like those shown in Fig. 15.17 may again be used to determine what flights should be delayed, cancelled or diverted to avoid undue costs and risks. Thus, reinforcement learning may have a lot to offer in designing practical solutions to this important and difficult problem. For a more detailed treatment of the topic of routing aircraft given weather hazard information, the reader is invited to consult Bertsimas and Patterson (1998), Evans et al. (2006), and Krozel et al. (2006).

15.14 Conclusion

Reinforcment learning builds on ideas from the fields of mathematics, engineering and psychology to develop algorithms that identify optimal or near-optimal control policies based on simulated or real interaction with a stochastic environment. Unlike traditional methods that require the underlying dynamical system to be modeled by a set of probability transition matrices or differential equations, reinforcement learning techniques can be applied to complex problems for which no model exists, This chapter has presented an introduction to the theory underlying reinforcement learning and the Markov Decision Processes (MDPs) to which they apply, described several practical reinforcement learning algorithms, and presented three sample applications that demonstrated the powerful potential of reinforcement learning to solve problems that arise in the environmental sciences.

Acknowledgements Satinder Singh first introduced me to reinforcement learning, and my Ph.D. advisor, Kent Goodrich, generously allowed me to explore some of its mathematical aspects for my doctoral dissertation despite its unfamiliarity to him. The idea for the aircraft routing example came from discussions with Bill Myers, whose dissertation (Myers 2000) examines how the depiction of forecast hazard data can affect humans' routing decisions. The wireless sensor array application was inspired by conversations with Lynette Laffea. Several friends and colleagues provided valuable feedback on a draft of this chapter, including Kent Goodrich, Lynette Laffea, Emily Mankin, and Matthias Steiner. I very much appreciate their assistance.

References

Atlas, D. (1982). Adaptively pointing spacebome radar for precipitation measurements. *Journal of Applied Meteorology*, *21*, 429–443.

Baird, L. C. (1995). Residual algorithms: Reinforcement learning with function approximation. In A. Prieditis & S. J. Russell (Eds.), *Proceedings of the Twelfth International Conference on Machine Learning* (pp. 30–37). 9–12 July 1995. Tahoe City, CA/San Francisco: Morgan Kaufmann.

Baxter, J., & Bartlett, P. L. (2000). Reinforcement learning in POMDP via direct gradient ascent. *Proceedings of the 17th International Conference on Machine Learning* (pp. 41–48). 29 June–2 July 2000. Stanford, CA/San Francisco: Morgan Kaufmann.

Bellman, R. E. (1957). *Dynamic programming* (342 pp.). Princeton, NJ: Princeton University Press.

Bertsekas, D. P. (1995). *Dynamic programming and optimal control* (Vol. 1, Vol. 2, 387 pp., 292 pp.). Belmont, MA: Athena Scientific.

Bertsekas, D. P., & Tsitsiklis, J. N. (1996). *Neuro-dynamic programming* (491 pp.). Belmont, MA: Athena Scientific.

Bertsimas, D., & Patterson, S. S. (1998). The air traffic flow management problem with enroute capacities. *Operations Research*, *46*, 406–422.

Chrisman, L. (1992). Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. *Proceedings of the Tenth National Conference on Artificial Intelligence* (pp. 183–188). 12–16 July 1992. San Jose/Menlo Park, CA: AAAI Press.

Dayan, P., & Sejnowski, T. (1994). TD(0) converges with probability 1. *Machine Learning*, *14*, 295–301.

Evans, J. E., Weber, M. E., & Moser, W. R. (2006). Integrating advanced weather forecast technologies into air traffic management decision support. *Lincoln Laboratory Journal*, *16*, 81–96.

Hamilton, W. R. (1835). Second essay on a general method in dynamics. *Philosophical Transactions of the Royal Society*, Part I for 1835, 95–144.

Jaakkola, T., Jordan, M., & Singh, S. (1994). On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, *6*, 1185–1201.

Jaakkola, T., Singh, S., & Jordan, M. (1995). Reinforcement learning algorithm for partially observable Markov decision problems. In G. Tesauro, D. S. Touretzky, & T. Leen (Eds.), *Advances in neural information processing systems: Proceedings of the 1994 Conference* (pp. 345–352). Cambridge, MA: MIT Press.

Joint Planning and Development Office (JPDO). (2006). *Next generation air transportation system (NGATS)—weather concept of operations* (30 pp.). Washington, DC: Weather Integration Product Team.

Krozel, J., Andre, A. D. & Smith, P. (2006). Future air traffic management requirements for dynamic weather avoidance routing. *Preprints, 25th Digital Avionics Systems Conference* (pp. 1–9). October 2006. Portland, OR: IEEE/AIAA.

Kushner, H. J., & Yin, G. G. (1997). *Stochastic approximation algorithms and applications* (417 pp.). New York: Springer.

Lovejoy, W. S. (1991). A survey of algorithmic methods for partially observable Markov decision processes. *Annals of Operations Research*, *28*, 47–66.

McLaughlin, D. J., Chandrasekar, V., Droegemeier, K., Frasier, S., Kurose, J., Junyent, F., et al. (2005). Distributed Collaborative Adaptive Sensing (DCAS) for improved detection, understanding, and prediction of atmospheric hazards. *Preprints-CD, AMS Ninth Symposium on Integrated Observing and Assimilation Systems for the Atmosphere, Oceans, and Land Surface*. 10–13 January 2005. Paper 11.3. San Diego, CA.

Myers, W. L. (2000). *Effects of visual representations of dynamic hazard worlds on human navigational performance*. Ph.D. thesis, Department of Computer Science, University of Colorado, 64 pp.

Peng, J., & Williams, R. J. (1996). Incremental multi-step *Q*-learning. *Machine Learning*, *22*, 283–290.

Precup, D., Sutton, R. S., & Dasgupta, S. (2001). Off-policy temporal-difference learning with function approximation. In C. E. Brodley and A. P. Danylok (Eds.), *Proceedings of the 18th International Conference on Machine Learning* (pp. 417–424). 28 June–1 July 2001. Williamstown, MA/San Francisco, CA: Morgan Kaufmann.

Puterman, M. L. (2005). *Markov decision processes: Discrete stochastic dynamic programming* (649 pp.). Hoboken, NJ: Wiley Interscience.

Robbins, H., & Monro, S. (1951). A stochastic approximation method. *Annals of Mathematical Statistics*, *22*, 400–407.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development*, *3*, 211–229.

Si, J., Barto, A. G., Powell, W. B., & Wunsch, D. (Eds.). (2004). *Handbook of learning and approximate dynamic programming* (644 pp.). Piscataway, NJ: Wiley-Interscience.

Singh, S. P., & Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning*, *22*, 123–158.

Singh, S. P., Jaakkola, T., Littman, M. L., & Szepasvari, C. (2000). Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, *38*, 287–308.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction* (322 pp.). Cambridge, MA: MIT Press.

Tadic, V. (2001). On the convergence of temporal-difference learning with linear function approximation. *Machine Learning*, *42*, 241–267.

Tsitsiklis, J. N. (2002). On the convergence of optimistic policy iteration. *Journal of Machine Learning Research*, *3*, 59–72.

Tsitsiklis, J. N., & Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, *42*, 674–690.

Turing, A. M. (1948). Intelligent machinery, National Physical Laboratory report. In D. C. Ince (Ed.). 1992, *Collected works of A. M. Turing: Mechanical intelligence* (227 pp.). New York: Elsevier Science.

Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, *59*, 433–460.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. Ph.D. thesis, King's College, Cambridge University, Cambridge, 234 pp.

Watkins, C. J. C. H., & Dayan, P. (1992). *Q*-learning. *Machine Learning*, *8*, 279–292.

Williams, J. K. (2000). *On the convergence of model-free policy iteration algorithms for reinforcement learning: Stochastic approximation under discontinuous mean dynamics*. Ph.D. thesis, Department of Mathematics, University of Colorado, Colorado, 173 pp.

Williams, J. K., & Singh, S. (1999). Experimental results on learning stochastic memoryless policies for partially observable Markov decision processes. In M. S. Kearns, S. A. Solla, and D. A. Cohn (Eds.), *Advances in neural information processing systems 11. Proceedings of the 1998 Conference* (pp. 1073–1079). Cambridge, MA: MIT Press.