Reinforcement Learning: A Tutorial Survey and Recent Advances

Abhijit Gosavi

Department of Engineering Management and Systems Engineering, Missouri University of
Science and Technology, Rolla, Missouri 65409, gosavia@mst.edu

In the last few years, reinforcement learning (RL), also called adaptive (or approximate) dynamic programming, has emerged as a powerful tool for solving complex sequential decision-making problems in control theory. Although seminal research in this area was performed in the artificial intelligence (AI) community, more recently it has attracted the attention of optimization theorists because of several noteworthy success stories from operations management. It is on *large-scale* and complex problems of dynamic optimization, in particular the Markov decision problem (MDP) and its variants, that the power of RL becomes more obvious. It has been known for many years that on large-scale MDPs, the *curse of dimensionality* and the *curse of modeling* render classical dynamic programming (DP) ineffective. The excitement in RL stems from its direct attack on these curses, which allows it to solve problems that were considered intractable via classical DP in the past. The success of RL is due to its strong mathematical roots in the principles of DP, Monte Carlo simulation, function approximation, and AI. Topics treated in some detail in this survey are temporal differences, *Q*-learning, semi-MDPs, and stochastic games. Several recent advances in RL, e.g., policy gradients and hierarchical RL, are covered along with references. Pointers to numerous examples of applications are provided. This overview is aimed at *uncovering the mathematical roots* of this science so that readers gain a clear understanding of the core concepts and are able to use them in their own research. The survey points to more than 100 references from the literature.

Key words: artificial intelligence; dynamic programming; simulation; reinforcement learning
History: Accepted by Michel Gendreau, Area Editor for Heuristic Search and Learning; received April 2004; revised January 2005, December 2005, August 2006, December 2006; accepted August 2008. Published online in *Articles in Advance* December 19, 2008.

1. Introduction

Markov decision problems (MDPs) are problems of sequential decision making in which a *control* (action) has to be selected in each decision-making state visited by the concerned system. Such problems are widespread in stochastic control theory, and their roots can be traced to the pioneering work of Richard Bellman in the 1950s. The main contribution of Bellman's work was to show that the computational burden of an MDP could be dramatically reduced via what is now well known as *dynamic programming* (DP). However, it was also recognized quite early in the historical evolution of this problem domain that on large-scale and complex MDPs, methods of classical DP, namely, policy iteration (PI) and value iteration (VI), break down. The requirement of computing, storing, and manipulating the so-called transition probability matrices (TPMs) is responsible for this breakdown in classical DP. In problems involving complex systems with several governing random variables, it is usually difficult to *compute* the values of the transition probabilities (TPs). This phenomenon is

called the *curse of modeling*. In problems with a large dimension, *storing* or *manipulating* the elements of the so-called value function—needed in DP—becomes challenging. This is called the *curse of dimensionality*. As such, classical DP even today is rather ineffective on problems that are large scale, complex, or both.

The power of reinforcement learning (RL) or adaptive (or approximate) DP (ADP) lies in its ability to solve, near-optimally, complex and large-scale MDPs on which classical DP breaks down. RL emerged as a tool in the artificial intelligence (AI) and neural research communities, where combining DP with derivative-based adaptive function approximations (Werbös 1987) and learning-based methods (Barto et al. 1983) was advocated in the mid-1980s. The modern science of RL has emerged from a synthesis of notions from four different fields: classical DP, AI (temporal differences), stochastic approximation (simulation), and function approximation (regression, Bellman error, and neural networks). In this survey, we will discuss the main ideas in RL, with special attention to the underlying mathematical principles.

We will describe a few important algorithms along with pointers to some case studies. Outside the MDP, we will also present some recent advances in solving other problems such as semi-Markov decision problems (SMDPs), competitive MDPs (also called stochastic games), and hierarchical MDPs. The survey points to more than 100 references from the existing literature, and it is hoped that new ideas for research will be stimulated from reading this paper.

The rest of this paper is organized as follows. Section 2 introduces the MDP framework. Section 3 presents an overview of DP-based RL conducted in a simulation environment. Extensions of RL to other domains, along with recent developments and some applications, are presented in §4. Section 5 concludes this survey with pointers to active areas of current research.

2. MDPs

The MDP framework is used extensively in stochastic control theory (Bertsekas 1995, Puterman 1994) of discrete-event systems. In an MDP, the system under consideration is assumed to be driven by underlying Markov chains. In a Markov chain, the system jumps randomly from one state to another in discrete-time steps, and the probability of transition from the current state to the next depends only on the current state and not on where the system has been before. Further, in a subset of states in an MDP (called the set of decision-making states), the system is required to choose an action or a control from a set of actions. A policy defines the action to be chosen in every state; it is a mapping from the set of states to the set of actions. An immediate reward, which may be positive, negative, or zero, is earned in transitioning from one state to another under the influence of an action. The performance metric of a policy is usually a function (the objective function) of the immediate rewards earned when the associated policy is followed over a predetermined time horizon. The time horizon could be finite or infinite, depending on what is intended by the designer of the system. The MDP is all about finding the *optimal* policy (policies) that optimizes a given performance metric. A separate Markov chain is associated to each policy. Furthermore, it should be noted that the MDP described above considers events that occur in discrete time.

We will assume that the state space, $\mathcal{S}$, and the action space in state i, $\mathcal{A}(i)$, for every $i \in \mathcal{S}$ are finite sets. Further, the Markov chain associated with every action in the MDP is *regular* (Grinstead and Snell 1997; see the Online Supplement, available at http://joc.pubs.informs.org/ecompanion.html). Finally, the time horizon for measuring the performance metric is of infinite duration. Two popular performance metrics

that we define below are *discounted reward* and *average reward*. The discounted reward is the sum of the discounted rewards earned over the entire time horizon when the associated policy is pursued, while the average reward is the expected reward earned in one step.

Let $d(i)$ denote the action chosen in state i when policy $\hat{d}$ is pursued; note that $\hat{d}$ contains $|\mathcal{S}|$ elements. Let $r(i, a, j)$ denote the immediate reward earned in going from state i to state j under action a, and let $p(i, a, j)$ denote the probability of the same transition.

DEFINITION 1. The discounted reward of a policy $\hat{d}$ starting at state i is

$$J_{\hat{d}}(i) \equiv \lim_{k \to \infty} \mathsf{E}\left[\sum_{s=1}^{k} \gamma^{s-1} r(x_s, d(x_s), x_{s+1}) \,\Big|\, x_1 = i \right],$$

where x_s is the state occupied before the sth transition and γ denotes the discount factor.

Intuitively, $J_{\hat{d}}(i)$ denotes the total discounted reward earned along an infinitely long trajectory starting at i, if policy $\hat{d}$ is pursued throughout the trajectory. Similarly, $J^*(i)$, which will be one of the variables in Equation (1) below, will denote the total discounted reward earned along an infinitely long trajectory starting at state i when the optimal policy is pursued throughout the trajectory. The expectation operator, E, is necessary in the previous definition and in the next definition because the trajectory of states is random.

DEFINITION 2. The average reward of a policy $\hat{d}$ starting at state i is

$$\rho_{\hat{d}}(i) \equiv \lim_{k \to \infty} \frac{\mathsf{E}\left[\sum_{s=1}^{k} r(x_s, d(x_s), x_{s+1}) \mid x_1 = i \right]}{k},$$

where x_s, as above, is the state occupied before the sth transition. For MDPs in which all Markov chains are regular, the average reward is independent of the starting state.

We now define an MDP and provide two simple examples.

DEFINITION 3. For discounted reward, the MDP is to determine $\hat{d}^*$ so that $J_{\hat{d}^*}(i) \geq J_{\hat{d}}(i)$ for all $\hat{d}$ and every $i \in \mathcal{S}$. For average reward, the MDP is to determine $\hat{d}^*$ so that $\rho_{\hat{d}^*}(i) \geq \rho_{\hat{d}}(i)$ for all $\hat{d}$ and every $i \in \mathcal{S}$.

EXAMPLE 1. Consider a two-state MDP in which two actions are permitted in each state. The relevant data are supplied in Figure 1. The example illustrates the nature of a generic MDP. Theoretically speaking, underlying any MDP exist data with a structure similar to this two-state MDP; for large-scale MDPs, usually the TPs cannot be determined easily. However, simple examples such as these can serve as test-beds for numerically testing a newly designed RL algorithm. This two-state MDP can be solved by exhaustive evaluation of all its policies. Let $\mathbf{P}_{\hat{d}}$ denote the matrix whose element in the ith row and jth column
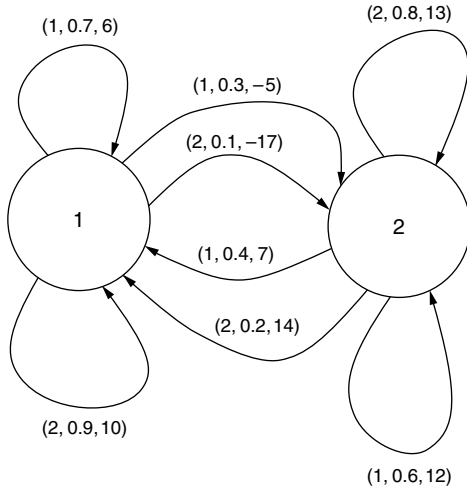
Figure 1 **Two-State MDP with (x, y, z) on Each Arc Denoting the Action (x), Transition Probability (y), and Immediate Reward (z) Associated with the Transition**

is $p(i, d(i), j)$, and let $\bar{r}(i, a) = \sum_{j=1}^{|\mathcal{S}|} p(i, a, j) r(i, a, j)$ denote the *expected* immediate reward in state i when action a is chosen in it. $\mathbf{P}_{\hat{d}}$ is called the one-step TPM associated with policy $\hat{d}$. Now the average reward of a policy $\hat{d}$ can be calculated as follows: $\rho_{\hat{d}} = \sum_{i \in \mathcal{S}} \pi_{\hat{d}}(i) \bar{r}(i, d(i))$, where $\vec{\pi}_{\hat{d}}$—the column vector of limiting probabilities of the Markov chain associated with $\hat{d}$—can be determined by solving the following system of linear equations:

$$[\pi_{\hat{d}}(1), \pi_{\hat{d}}(2), \ldots, \pi_{\hat{d}}(|\mathcal{S}|)]\mathbf{P}_{\hat{d}}$$
$$= [\pi_{\hat{d}}(1), \pi_{\hat{d}}(2), \ldots, \pi_{\hat{d}}(|\mathcal{S}|)] \quad \text{and} \quad \sum_{i \in \mathcal{S}} \pi_{\hat{d}}(i) = 1.$$

By searching over all values of $\rho_{\hat{d}}$ in the space of policies, one can easily identify the optimal solution for small problems. The optimal actions here are one in state 2 and two in state 1.

EXAMPLE 2. The classical admissions-control problem can be cast as an MDP. See Figure 2. In its simplest version, there is a single-server single-channel queue, and the state of the system is defined by the number of entities in the system; when a new entity enters, the manager can select one of two actions,
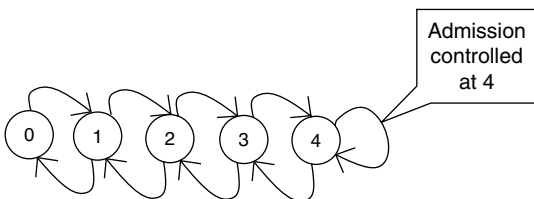

Figure 2 **Admissions-Control MDP**
Notes. This shows a Markov chain associated with a policy that disallows more than four customers in the queue. The number inside the circle denotes the state of the system, i.e., the number of customers in the system.

which are (i) accept the entity and (ii) reject the entity. Associated with a rejection action is a cost. It also costs money continuously (on an hourly basis, say) to keep entities waiting. Allowing too many customers leads to increased waiting costs, whereas rejecting too many entities produces high rejection costs. This poses an optimization problem in which one must determine when to stop accepting new customers. Under suitable assumptions, the system state transitions have a Markov property. Solving the MDP leads to a solution for the underlying optimization problem.

3. Reinforcement Learning with Q-Values

RL is based on discrete-event DP. Classical DP is based on two forms of the Bellman equation: the Bellman optimality equation (BOE) and the Bellman policy equation (BPE). Associated with them are the two algorithms of DP: VI and PI, respectively (Bellman 1954). The framework can be studied separately for discounted and average reward. We will first present the Bellman equations needed in DP algorithms, then present their Q-value (also called Q-factor) versions, and then finally derive RL algorithms from their DP counterparts. For the average-reward case, we will present the Bellman equations, skip the derivations of the RL algorithms (because they are analogous to those of the discounted reward case), and then present the associated RL algorithms.

3.1. Discounted Reward
In the rest of this paper, the ith element of a vector $\vec{x}$, or an n-tuple $\hat{x}$, will be denoted by $x(i)$. A result that employs the BOE and has detailed proofs (Bertsekas 1995) is stated next. In VI, one starts with guessed estimates of the value function and uses the BOE successively to generate ϵ-optimal estimates of the value function—from which the ϵ-optimal policy can be obtained.

THEOREM 1. *For a discounted reward MDP in which all Markov chains are regular, there exists a vector $\vec{J}^* \equiv \{J^*(1), J^*(2), \ldots, J^*(|\mathcal{S}|)\}$ such that the following system of linear equations is satisfied*:

$$J^*(i) = \max_{a \in \mathcal{A}(i)} \left[\bar{r}(i, a) + \gamma \sum_{j=1}^{|\mathcal{S}|} p(i, a, j) J^*(j) \right]$$

for all $i \in \mathcal{S}$. (1)

The following result employs the BPE (also called the Poisson equation). Theorems 1 and 2 are proved in, e.g., Bertsekas (1995). The usefulness of BPE is exploited in the method of PI, in which it is solved repeatedly until the optimal solution is obtained.

THEOREM 2. *For a discounted reward MDP in which the Markov chain associated with a policy $\hat{d}$ is regular, there exists a vector $\vec{J}_{\hat{d}} \equiv \{J_{\hat{d}}(1), J_{\hat{d}}(2), \dots, J_{\hat{d}}(|\mathscr{S}|)\}$ such that the following system of linear equations is satisfied:*

$$J_{\hat{d}}(i) = \left[\bar{r}(i, d(i)) + \gamma \sum_{j=1}^{|\mathscr{S}|} p(i, d(i), j) J_{\hat{d}}(j) \right]$$

$$\text{for all } i \in \mathscr{S}. \quad (2)$$

The (unknown) quantities, $J^*(\cdot)$ and $J_{\hat{d}}(\cdot)$, in the Bellman equations are referred to as *value* functions. The value function in Equation (1) is called the "optimal" value function, whereas that in Equation (2) is called the "policy" value function. We next define Q-values. The optimality Q-value for a state i and action a represents the value of the performance metric obtained when action a is selected in state i and the optimal policy is pursued thereafter.

DEFINITION 4. Consider $J^*(\cdot)$, defined in Equation (1). The optimality Q-value of a given state-action pair, (i, a), where i denotes the state and a the action associated with the optimal policy, can be defined in terms of the value function as follows:

$$Q(i, a) = \sum_{j=1}^{|\mathscr{S}|} p(i, a, j)[r(i, a, j) + \gamma J^*(j)]. \quad (3)$$

The policy Q-value for a state i and action a represents the performance metric's value if action a is selected in state i and the policy in question is pursued thereafter.

DEFINITION 5. Consider $J_{\hat{d}}(\cdot)$, defined in Equation (2). The policy Q-value of a given state-action pair, (i, a), where i denotes the state and a the action associated with the policy $\hat{d}$, can be defined in terms of the value function as follows:

$$Q_{\hat{d}}(i, a) = \sum_{j=1}^{|\mathscr{S}|} p(i, a, j)[r(i, a, j) + \gamma J_{\hat{d}}(j)]. \quad (4)$$

We next derive Q-value versions of the Bellman equations, i.e., Equations (1) and (2). Equations (1) and (3) imply that for all $i \in \mathscr{S}$, $J^*(i) = \max_{a \in \mathscr{A}(i)} Q(i, a)$. Using this fact in Equation (3), one can obtain the Q-value version of Equation (1) as follows:

$$Q(i, a) = \sum_{j=1}^{|\mathscr{S}|} p(i, a, j) \left[r(i, a, j) + \gamma \max_{b \in \mathscr{A}(j)} Q(j, b) \right]$$

$$\text{for all } i \in \mathscr{S} \text{ and } a \in \mathscr{A}(i). \quad (5)$$

Similarly, Equations (2) and (4) imply that for all $i \in \mathscr{S}$, $J_{\hat{d}}(i) = Q_{\hat{d}}(i, d(i))$, which implies that Equation (4) can be written as

$$Q_{\hat{d}}(i, a) = \sum_{j=1}^{|\mathscr{S}|} p(i, a, j)[r(i, a, j) + \gamma Q_{\hat{d}}(j, d(j))]$$

$$\text{for all } i \in \mathscr{S} \text{ and } a \in \mathscr{A}(i). \quad (6)$$

Step 1. Set $k = 1$ and for all $i \in \mathscr{S}$ and $a \in \mathscr{A}(i)$, $Q^k(i, a) = 0$. Specify $\epsilon > 0$.

Step 2. For each $i \in \mathscr{S}$, compute

$$Q^{k+1}(i) \leftarrow \sum_{j=1}^{|\mathscr{S}|} p(i, a, j) \left[r(i, a, j) + \gamma \max_{b \in \mathscr{A}(j)} Q^k(j, b) \right]. \quad (7)$$

Step 3. Calculate for each $i \in \mathscr{S}$,

$$J^{k+1}(i) = \max_{a \in \mathscr{A}(i)} Q^{k+1}(i, a) \quad \text{and} \quad J^k(i) = \max_{a \in \mathscr{A}(i)} Q^k(i, a).$$

Then, if $||(\vec{J}^{k+1} - \vec{J}^k)|| < \epsilon(1 - \gamma)/2\gamma$, go to Step 4. Otherwise, increase k by 1 and return to Step 2.

Step 4. For each $i \in \mathscr{S}$, choose $d_\epsilon(i) \in \arg\max_{b \in \mathscr{A}(i)} Q^k(i, b)$, where $\hat{d}_\epsilon$ denotes the ϵ-optimal policy, and stop.

Figure 3   Steps in VI Based on Q-Values

This is the Q-value version of Equation (2). We now present VI and PI algorithms based on the Q-value versions of the Bellman equation.

VI. In VI, one starts with arbitrary values for the value function vector (or Q-values) and applies the Bellman optimality transformation repeatedly until one obtains the optimal policy. In our descriptions, W^K will denote the value of the iterate W in the kth iteration of the algorithm. Figure 3 outlines the steps in a VI algorithm based on Q-values. The transformation in Step 2 of Figure 3 is based on the BOE of Q-values, i.e., Equation (5). Step 3 in the algorithm is based on the fact that the termination criterion ensures that the policy obtained is indeed ϵ-optimal (see Puterman 1994 for more details).

PI. In PI, the general scheme of operation is as follows. One starts with an arbitrary policy. Then the BPE is solved to obtain the value function associated with the policy. The value function is then used to determine a better policy. This continues until the algorithm obtains the optimal policy. We will present a so-called modified PI algorithm (van Nunen 1976). In modified PI, the BPE is solved via a VI-like scheme in an ϵ-approximate sense, i.e., one starts with arbitrary values for the elements of the value-function vector, and the transformation based on the BPE is used repeatedly until the vector converges, in an ϵ-approximate sense, to a fixed point. In Figure 4, we outline the main steps in a Q-value version of modified PI.

In regards to the description in Figure 4, we note the following: (i) Step 2 consists of a set of iterations in which transformation (8) is applied in every iteration. That is, one starts with arbitrary values (e.g., 0) for $Q_{\hat{d}_k}(i, a) \; \forall (i, a)$ and repeatedly applies (8) until the Q-values converge in an ϵ-sense. (ii) The transformation (8) is based on the BPE of Q-values, i.e., Equation (6). (iii) In Step 3, we set $\hat{d}_{k+1} = \hat{d}_k$, *if possible*. The reason is multiple policies could be optimal,

Step 1. Set $k = 1$ and select an arbitrary policy $\hat{d}_k$. ($\hat{d}_k$ will denote the policy in the kth iteration.)

Step 2. (Policy evaluation [PE]) Initialize the Q-values for all states and actions to arbitrary values (preferably 0). Use the following transformation repeatedly until the Q-values converge in an ϵ-approximate sense:

- For all $(i, a) \in \mathcal{S} \times \mathcal{A}(i)$

$$Q_{\hat{d}_k}(i, a) \leftarrow \sum_{j=1}^{|\mathcal{S}|} p(i, a, j)[r(i, a, j) + \gamma Q_{\hat{d}_k}(j, d_k(j))]. \quad (8)$$

Step 3. (Policy Improvement) Generate a new policy $\hat{d}_{k+1}$ using the following relation:

$$d_{k+1}(i) \in \arg\max_{u \in \mathcal{A}(i)} Q_{\hat{d}_k}(i, u) \quad \text{for all } i \in \mathcal{S}.$$

If possible, set $\hat{d}_{k+1} = \hat{d}_k$. See also Remark (iii) in main text.

Step 4. If policy $\hat{d}_{k+1}$ is identical to policy $\hat{d}_k$, stop. Otherwise, set $k \leftarrow k + 1$ and return to Step 2.

Figure 4   Outline of Modified PI Based on Q-Values

and the algorithm could cycle *indefinitely* in the loop of Steps 2 and 3 otherwise.

Robbins-Monro (RM) Stochastic Approximation. RL uses a stochastic approximation (Robbins and Monro 1951) version of the transformations (7) or (8) within simulators. Consider a random variable X, and suppose that an estimate of its mean, $\mathsf{E}[X]$, is to be obtained from its samples, with Y^m denoting the estimate of the mean in the mth iteration. The RM algorithm described in Figure 5 can be used for this purpose. The RM algorithm follows from the strong law of large numbers and ensures that with probability 1, $\lim_{m \to \infty} Y^m = \mathsf{E}[X]$. The algorithm is guaranteed to converge if $\lim_{M \to \infty} \sum_{m=1}^{M} \mu_m = \infty$ and $\lim_{M \to \infty} \sum_{m=1}^{M} \mu_m^2 < \infty$, where μ_m denotes the step size in the mth iteration. A number of other rules obey these conditions, e.g., $\mu_m = T_1/(m + T_2)$, where $T_1 > 0$ and $T_2 \geq 0$. Another rule (Darken et al. 1992) is $\mu_m = T_1/(1 + (m - 1)^2/(1 + T_2))$, with, e.g., $T_1 = 0.1$ and $T_2 = 10^6$.

TD-Learning. Learning with temporal differences (TD) (Sutton 1984) generalizes beyond the RM

Step 1. Let X_i denote the ith sample of X, i.e., the sample obtained in the ith iteration, and Y^m the estimate of $\mathsf{E}[X]$ in the mth iteration. Set $m = 1$, Y^0 to any arbitrary number and specify $\epsilon > 0$.

Step 2. Update Y, using X_m, the current sample of X, as follows:

$$Y^m \leftarrow (1 - \mu_m)Y^{m-1} + \mu_m X_m, \quad (9)$$

where $\mu_m = 1/(1 + m)$ is one rule for selecting the step size, μ, in the mth iteration. Other rules are discussed in the text.

Step 3. If $|Y^m - Y^{m-1}| < \epsilon$, stop. Otherwise, increase m by 1 and return to Step 2.

Figure 5   The Robbins-Monro Algorithm

algorithm. If W^k denotes the iterate in the kth iteration, the TD update is

$$W^{k+1} \leftarrow (1 - \mu)W^k + \mu[\text{feedback}], \quad (10)$$

where the feedback takes different forms. This is a recency-weighing updating scheme in which the old estimate is gradually changed with current feedback from the system. The feedback is usually a function of the immediate reward, $r(i, a, j)$, and the values of the iterates for other states. In most of the algorithms that we will cover, a unique W will be associated with each state-action pair. Positive (i.e., large to maximize rewards) feedback strengthens action a by increasing its W, whereas negative feedback weakens the action. By obtaining feedback with a sufficiently large number of trials, the algorithm "learns" the optimal policy. This is the thread that underlies the TD machinery and holds it together. However, with respect to their working mechanisms and areas of application, there are many differences in the algorithms considered below. When the feedback used is from one state transition of the underlying Markov chain, we refer to the algorithm as a TD(0) algorithm, and this is equivalent to the RM algorithm. When the feedback is from multiple transitions, the algorithm is called a TD(λ) algorithm. Such algorithms become effective in the so-called *episodic* tasks, which we will describe later. In TD(λ), we have that feedback $= R_k + \lambda R_{k+1} + \lambda^2 R_{k+2} + \cdots$, where R_k is the immediate reward received in the kth iteration and $\lambda \in [0, 1]$. Oftentimes, a "family" of algorithms can be generated by using different values for λ from the closed interval $[0, 1]$. We shall attempt to relate many of the algorithms to be discussed to this generalized framework.

Q-Learning. The well-known Q-learning algorithm can be derived from the definition of Q-values and the RM (or TD(0)) algorithm. Note from Equation (5) that $Q(i, a)$ for any (i, a) is an expectation. Let the estimate of $Q(i, a)$ in the mth iteration be denoted by $Q^m(i, a)$. Then by setting $Q(i, a) = \mathsf{E}[X]$ and $Q^m(i, a) = Y^m$ in the RM algorithm, the algorithm for estimating $Q(i, a)$ becomes

$$Q^m(i, a) \leftarrow (1 - \mu_m)Q^{m-1}(i, a)$$
$$+ \mu_m\left[r(i, a, j) + \gamma \max_{b \in \mathcal{A}(j)} Q^{m-1}(j, b)\right], \quad (11)$$

which follows directly from Equations (9) and (5). This is the main transformation of Q-learning (Watkins 1989), which is described in detail in Figure 6.

It turns out that transformation (11) can be used in a simulator. In a simulator, one can generate a sufficiently long sample trajectory of states so that the Q-values being estimated converge. What is interesting is that in simulators, the order in which states are

Step 1. Initialize all Q-values to 0. That is, for all (l, u), where $l \in \mathscr{S}$ and $u \in \mathscr{A}(l)$, set $Q(l, u) \leftarrow 0$. Set k, the number of state changes, to 0. Run the algorithm for $k_{\max}$ iterations, where $k_{\max}$ is chosen to be a sufficiently large number. Start system simulation at any arbitrary state.

Step 2. Let the current state be i. Select action a with a probability of $1/|\mathscr{A}(i)|$. The action that maximizes the Q-value for current state is called a *greedy* action for the current state. (See §3.1 for some other action-selection rules.)

Step 3. Simulate action a. Let the next state be j. Let $r(i, a, j)$ be the immediate reward earned in transiting to j from i under a's influence. Increment k by 1.

Step 4. Update $Q(i, a)$ with μ as a function of k (see step-size rules of RM):

$$Q(i, a) \leftarrow (1 - \mu)Q(i, a) + \mu\left[r(i, a, j) + \gamma \max_{b \in \mathscr{A}(j)} Q(j, b)\right].$$

Step 5. If $k < k_{\max}$, set $i \leftarrow j$ and then go to Step 2. Otherwise, go to Step 6.

Step 6. For each $l \in \mathscr{S}$, select $d(l) \in \arg\max_{b \in \mathscr{A}(l)} Q(l, b)$. The policy (solution) generated by the algorithm is $\hat{d}$. Stop.

Figure 6   Steps in Q-Learning Based on VI

visited is haphazard. Thus, a possible trajectory for a three-state MDP is 1, 2, 2, 3, 2, 1, 2, 3, 2, 3, 3, 1, 1, 1, ... Clearly, when updates are performed here for a state, the values used for other states have not been updated the same number of times. Such updates are called *asynchronous* updates. Usually in classical VI, values used in every iteration have been updated the same number of times. Such updates are called *synchronous* updates. Fortunately, it can be proved that under certain conditions, asynchronous transformations can also generate the optimal solutions (Borkar 1998) generated by their synchronous counterparts.

The interest in transformations such as (11) stems from the fact that they do *not* need the TPs! Hence they are called *model-free* transformations; i.e., the TP model is *not* required. However, they must be run in simulators of systems, which can be easily constructed from the distributions of the governing random variables. It is well known that simulating a complex system is *considerably easier* than generating the TPs of the system. This is also why RL is said to *avoid* the curse of modeling.

In the simulator, one may choose each action with the same probability. This usually ensures that all samples are gathered properly and that all state-action pairs are updated infinitely often; the latter is a requirement for convergence to the correct values of the Q-values. In practice, a so-called *exploratory* policy, with a bias toward the greedy action, is often used. With an exploratory strategy, in the kth iteration, one selects the greedy action $\arg\max_{u \in \mathscr{A}(i)} Q(i, u)$ with a probability p^k and any one of the remaining actions with probability $(1 - p^k)/(|\mathscr{A}(i)| - 1)$. A possible rule for the probability p^k is $p^k = 1 - B/k$, where B, for

instance, could equal 0.5. With such a rule, the probability of selecting nongreedy actions is automatically decayed to 0 with increasing k. This is also called *limiting-greedy* exploration.

Random, *undirected* exploration, discussed above, can cause the algorithm to take time exponential in the number of states to converge (Whitehead 1991). *Directed exploration* strategies—counter-based (Sato et al. 1990), error-and-counter-based (Thrun and Moller 1992), and recency-based (Sutton 1990)—can overcome this drawback. A counter-based directed exploration strategy works as follows: In state i, the action that maximizes the following is selected: $\alpha Q(i, a) + c(i)/(\sum_j \tilde{p}(i, a, j)c(j))$, where $\alpha > 0$ is called the exploration-versus-exploitation gain factor, $c(i)$ denotes the number of times state i has been visited thus far, and $\tilde{p}(\cdot, \cdot, \cdot)$ denotes the TP, which can be estimated via model-building (see the Online Supplement). A number of exploration strategies have been discussed and experimented with in the literature: the Boltzmann strategy (Luce 1959), the E^3 strategy (Kearns and Singh 2002), and the external-source strategy (Smart and Kaelbling 2000).

SARSA. SARSA (Rummery and Niranjan 1994, Sutton 1996, Sutton and Barto 1998) is a well-known algorithm based on an "on-policy" control. In on-policy control, a unique policy is evaluated for some time during learning—unlike Q-learning, which does "off-policy" control, in which the policy being evaluated can change in every iteration. SARSA uses the concept of learning in *episodes*, in which there is a "terminal" state and the episode terminates when the terminal state is reached. Details are given in Figure 7 from Equation (12); it is not hard to see why it is a TD(0) algorithm. TD(λ) can also be used in SARSA (see SARSA(λ) of Sutton 1996), especially when the learning is episodic. An important notion of *eligibility traces*, discussed in Singh and Sutton (1996), can be used to increase the power of TD(λ) methods by attaching variable weights to the reinforcements in the updating strategy. When function approximation can

- Initialize all the Q-values, $Q(l, u)$ for all $l \in \mathscr{S}$ and $u \in \mathscr{A}(l)$, to arbitrary values.
- Repeat for each episode:
  — Initialize i and select action $a \in \mathscr{A}(i)$ via limiting-greedy exploration.
  — Repeat for each step of episode:
    * Simulate action a and let the next state be j with $r(i, a, j)$ being the immediate reward. Select action $b \in \mathscr{A}(j)$ via limiting-greedy exploration. Then update $Q(i, a)$ using

$$Q(i, a) \leftarrow (1 - \mu)Q(i, a) + \mu[r(i, a, j) + \gamma Q(j, b)]. \quad (12)$$

    * If j is a terminal state, end current episode. Otherwise, continue within episode.

Figure 7   SARSA

Step 1. Initialize all J-values and H-values to 0. That is, for all l, where $l \in \mathscr{S}$ and $u \in \mathscr{A}(l)$, set $J(l) \leftarrow 0$ and $H(l, u) \leftarrow 0$. Set k, the number of state changes, to 0. Run the algorithm for $k_{\max}$ iterations, where $k_{\max}$ is chosen to be a sufficiently large number. Start system simulation at any arbitrary state.

Step 2. Let the current state be i. Select action a with a probability of $e^{H(i,a)} / \sum_{b \in \mathscr{A}(i)} e^{H(i,b)}$. (This is called the Gibbs softmax method.)

Step 3. Simulate action a. Let the next state be j. Let $r(i, a, j)$ be the immediate reward earned in going to j from i under a. Increment k by 1. Update $J(i)$ using the following equation:

$$J(i) \leftarrow (1 - \mu)J(i) + \mu[r(i, a, j) + \gamma J(j)]. \qquad (13)$$

Step 4. Update $H(i, a)$ using a step size, β, much smaller than μ:

$$H(i, a) \leftarrow H(i, a) + \beta[r(i, a, j) + \gamma J(j) - J(i)]. \qquad (14)$$

Step 5. If $k < k_{\max}$, set $i \leftarrow j$ and then go to Step 2. Otherwise, go to Step 6.

Step 6. For each $l \in \mathscr{S}$, select $d(l) \in \arg\max_{b \in \mathscr{A}(l)} H(l, b)$. The policy (solution) generated by the algorithm is $\hat{d}$. Stop.

Figure 8   Steps in an Actor-Critic Algorithm

be performed more easily with on-policy updates, an on-policy algorithm like SARSA becomes more effective than Q-learning.

Adaptive Critics. Adaptive-critic, also called actor-critic, algorithms (Witten 1977, Barto et al. 1983, Werbös 1992) are based on combing notions of PI (Howard 1960) and adaptive function approximation (Werbös 1987). PI has two steps: policy evaluation (PE) and policy improvement, in which a very approximate PE is conducted with just *one* update. The main idea here is as follows: The actor selects a policy (does policy improvement) and the critic enacts or simulates the policy (does PE). See Figure 8 for details. Implementational details are discussed in textbooks (Sutton and Barto 1998, Bertsekas and Tsitsiklis 1996), and analytical issues, in some papers (Konda and Borkar 1999, Konda and Tsitsiklis 2000). From the details, again, it is not hard to see that the main updates (Equations (13) and (14)) correspond to a TD(0) update with off-policy control. Actor critics are really useful when one seeks stochastic policies (Sutton et al. 2000), e.g., in partially observable domains.

Q-P-**Learning.** Q-P-learning (Gosavi 2004a, 2003) follows the scheme of modified PI in which a policy is chosen, its value function is estimated (PE), and using the value function, a better policy is selected (policy improvement). This continues until the algorithm cannot produce a better policy. See Figure 9 for details. The algorithm is especially useful in average reward and semi-Markov problems, which we will discuss below. The algorithm is an on-policy algorithm based on a TD(0) update (see Equation (15) in Figure 9).

3.2. Average Reward
RL algorithms for average reward (Schwartz 1993, Singh 1994) have been studied for some time. Average

Step 1. Initialize all the P-values, $P(l, u)$ for all $l \in \mathscr{S}$ and $u \in \mathscr{A}(l)$, to arbitrary values. Set all the visit-values, $N(l, u)$, to 0. Set E, the number of policy evaluations (PEs), to 1. Initialize $E_{\max}$ and $k_{\max}$ to large numbers.

Step 2 (PE). Start fresh simulation. Set all the Q-values, $Q(l, u)$, to 0. Let the current system state be i. Set k, the number of iterations within a PE, to 1.

Step 2(a). Simulate action $a \in \mathscr{A}(i)$ with probability $1/|\mathscr{A}(i)|$.

Step 2(b). Let the next state encountered in the simulator be j. Increment $N(i, a)$ by 1. Compute $r(i, a, j)$ from the simulator. Set μ as a function of $N(i, a)$ (discussed before). Then update $Q(i, a)$ using

$$Q(i, a) \leftarrow (1 - \mu)Q(i, a) + \mu\left[r(i, a, j) + \gamma Q(j, \underset{b \in \mathscr{A}(j)}{\arg\max} P(j, b))\right]. \qquad (15)$$

Step 2(c). Set $k \leftarrow k + 1$. If $k < k_{\max}$, set $i \leftarrow j$ and go to Step 2(a); else go to Step 3.

Step 3 (Policy Improvement). Set for all $l \in \mathscr{S}, u \in \mathscr{A}(l)$, $P(l, u) \leftarrow Q(l, u)$. $E \leftarrow E + 1$. If E equals $E_{\max}$, go to Step 4. Otherwise, go to Step 2.

Step 4. For each $l \in \mathscr{S}$, select $d(l) \in \arg\max_{b \in \mathscr{A}(l)} Q(l, b)$. The policy (solution) generated by the algorithm is $\hat{d}$. Stop.

Figure 9   *Q-P*-Learning for Discounted Reward MDPs

reward differs considerably from discounted reward. It gives equal weight to reward earned over all parts of the time horizon—unlike discounted reward, which, for all practical purposes, ignores rewards beyond a certain distance in the time horizon, i.e., where the discount factor renders the effective reward negligibly small. In real-life situations, the discount factor may be unknown; under such conditions the average reward is a more suitable choice, as is evident from its widespread use in operations management. Also, if the discount factor is very close to one, the average reward is a more suitable performance metric because a discounted VI (or modified PI) with a large value for the discount factor can take very long for convergence (see Puterman 1994, pp. 163–164). Our discussion here is limited to the relevant Bellman equations and RL algorithms. The following is the average-reward analog of Theorem 1.

THEOREM 3. *For an average-reward MDP in which all Markov chains are regular, there exists a vector $\vec{J}^* \equiv \{J^*(1), J^*(2), \dots, J^*(|\mathscr{S}|)\}$ and a scalar ρ^*, which denotes the optimal average reward, such that the following system of linear equations is satisfied:*

$$J^*(i) = \max_{a \in \mathscr{A}(i)}\left[\bar{r}(i, a) - \rho^* + \sum_{j=1}^{|\mathscr{S}|} p(i, a, j)J^*(j)\right]$$

$$\text{for all } i \in \mathscr{S}. \quad (16)$$

Like its discounted counterpart, an appealing feature of Equation (16) is that it can be exploited to generate the optimal policy via VI. The following is the analog of Theorem 2 for average reward.

THEOREM 4. *For an average-reward MDP in which the Markov chain associated with a policy $\hat{d}$ is regular,*

there exists a vector $\vec{J}_{\hat{d}} \equiv \{J_{\hat{d}}(1), J_{\hat{d}}(2), \ldots, J_{\hat{d}}(|\mathscr{S}|)\}$ and a scalar $\rho_{\hat{d}}$, which denotes the average reward of policy $\hat{d}$, such that the following system of linear equations is satisfied:

$$J_{\hat{d}}(i) = \left[\bar{r}(i, d(i)) - \rho_{\hat{d}} + \sum_{j=1}^{|\mathscr{S}|} p(i, d(i), j) J_{\hat{d}}(j) \right]$$

$$\text{for all } i \in \mathscr{S}. \quad (17)$$

Again, like its discounted counterpart, Equation (17) can be exploited within a PI scheme to generate optimal solutions. Note, however, that to utilize Equations (16) and (17), one has to have access to ρ^* and $\rho_{\hat{d}}$, respectively. Because these quantities are generally not available, a way to work around this difficulty is to use *relative* versions of the BOE in the VI algorithm. Relative Q-learning, analyzed in Abounadi et al. (2001), is based on VI, and relative Q-P-learning (Gosavi 2003) is based on modified PI. In both algorithms, one has to choose a distinguished state-action pair (i^*, a^*) at the beginning. The algorithms are similar to those described in Figures 6 and 9 with the following difference:

• In Step 4 of Q-learning (Figure 6), updating is performed using the following:

$$Q(i, a) \leftarrow (1 - \mu)Q(i, a)$$
$$+ \mu \left[r(i, a, j) + \max_{b \in \mathcal{A}(j)} Q(j, b) - Q(i^*, a^*) \right].$$

• In Step 2(b) of Q-P-learning (Figure 9), updating is performed using the following:

$$Q(i, a) \leftarrow (1 - \mu)Q(i, a) + \mu \left[r(i, a, j) - Q(i^*, a^*) \right.$$
$$\left. + Q\left(j, \arg\max_{b \in \mathcal{A}(j)} P(j, b) \right) \right].$$

3.3. Scaling Up
For large-scale problems with millions of state-action pairs, it is difficult to explicitly store all the Q-values. This causes the curse of dimensionality. In a *look-up table*, all Q-values are stored separately. A way of working around the curse of dimensionality is to approximate Q-values using regression or neural networks. The idea is to store all the Q-values for every given action in the form of a small number of scalars, where the scalars are the coefficients of regression parameters or the "weights" of a neural network. Consider the state defined by an n-tuple, $\hat{x} = \{x(1), x(2), \ldots, x(n)\}$. A linear representation for action a would be $Q_{\vec{w}}(\hat{x}, a) = w_a(0) + \sum_{i=1}^{n} w_a(i)x(i)$, where $\vec{w}_a = \{w_a(0), w_a(1), \ldots, w_a(n)\}$ is the vector of weights for a. A nonlinear representation could be $Q_{\vec{w}}(\hat{x}, a) = w_a(0) + \sum_{i=1}^{n} w_a(i)x^{\theta(i)}(i)$, where $\theta(i)$ is the

power to which the ith state variable is raised. In either case, if $\mathcal{Q}_a$ denotes the set of Q-values for a, then one needs only $(n+1)$ scalars, with $(n+1) \ll |\mathcal{Q}_a|$, for defining *all* the Q-values of a.

Many implementations of RL in real-world problems have used neural networks (Crites and Barto 1995, Tesauro 1992, Singh and Bertsekas 1997, Das et al. 1999, Gosavi et al. 2002). But divergence to suboptimality with linear or nonlinear function approximations has also been reported (Baird 1995, Boyan and Moore 1995, Tsitsiklis and van Roy 1997, Ormoneit and Sen 2002). Outside of function approximation, a robust scheme called local regression uses "exemplars," i.e., representative states whose Q-values are stored *explicitly* (Tadepalli and Ok 1998). Clearly, the number of exemplars is much smaller than the state space. Then kernel methods, nearest-neighbor algorithms, decision trees, or interpolation (Hastie et al. 2001) can be used to estimate (predict) the value of any Q-value in the state space. Kernel methods (Hastie et al. 2001) assign a distance-based weight to every Q-value stored. Decision trees have also been used for the same task, and the central idea underlying their use (Chapman and Kaelbling 1991, Pyeatt and Howe 1998) is to split the state space, via *leaf nodes*, into dissimilar grids, and the splitting depends on how often a part of the state space has been visited. Within each grid, the decision to form subgrids is based on metrics from information theory or statistics, e.g., the notion of information gain (Quinlan 1986), the Gini index (Murthy et al. 1994), and statistical criteria like the t-statistic (Chapman and Kaelbling 1991). Several other papers (Gordon 1995, Hauskrecht 2000, Dayan 1992, Tsitsiklis and van Roy 1996) discuss the issue of function approximation. See the Online Supplement for more details of neural networks and kernel methods.

4. Extensions and Recent Advances
In this section, we present several extensions of the foundational RL ideas to other problem domains and algorithms. Some of the developments we will discuss are recent advances in the field of RL. We will conclude this section by enumerating a few RL case studies in operations management.

4.1. Semi-Markov Decision Problems
In a semi-MDP (SMDP), unlike an MDP, the time spent in any transition is not irrelevant but is an integral part of the objective function. Let $t(i, a, j)$ denote the time spent in the transition to state j from state i when action a is selected in state i. The MDP algorithms for discounted reward can be extended to the SMDP using in place of γ, the discount factor, the following: $e^{-Rt(i, a, j)}$, where R denotes the continuous rate of interest. For the discounted case, solving

continuous-time MDPs (where $t(i, a, j)$ has the exponential distribution Bertsekas 1995) via Q-learning (Bradtke and Duff 1995) and random-time SMDPs via Q-learning and Q-P-learning algorithms (Gosavi 2003, pp. 245–247) is discussed in the Online Supplement. For average reward, a popular approach called the vanishing-discount approach (Arapostathis et al. 1993) employs any discounted RL algorithm with a very small positive value for R. Alternatively, one can use R-SMART (Gosavi 2004b), which differs from Q-learning (Figure 6) for MDPs as follows. In Step 1, ρ, which denotes the current estimate of the optimal average reward, is set to 0, and T_r and T_t are also set to 0. Step 4 uses the following update:

$$Q(i, a) \leftarrow (1 - \mu)Q(i, a)$$
$$+ \mu\left[r(i, a, j) - \rho t(i, a, j) + \max_{b \in \mathcal{A}(j)} Q(j, b)\right].$$

Step 4(a), which follows Step 4, is used to update ρ. Step 4(a) is carried out only if a *greedy* action is selected in Step 2 (see Figure 6 for a definition of greedy actions). Step 4(a) is as follows: $T_r \leftarrow T_r + r(i, a, j)$; $T_t \leftarrow T_t + t(i, a, j)$; and $\rho \leftarrow (1 - \beta)\rho + \beta T_r/T_t$, where β, the step size used for updating ρ, should be smaller than μ (Borkar 1997). The algorithm in Das et al. (1999) uses $\beta = 1$. The Q-P-learning algorithm for SMDPs differs from the same for MDPs (Figure 9) as follows. In Step 2(b), use the following update:

$$Q(i, a) \leftarrow (1 - \mu)Q(i, a) + \mu\left[r(i, a, j) - \rho t(i, a, j)\right.$$
$$\left. + Q\left(j, \arg\max_{b \in \mathcal{A}(j)} P(j, b)\right)\right],$$

where ρ denotes the average reward of the policy being evaluated in Step 2. The value of ρ can be estimated in Step 2 via simulating the system for a long time, using in state m the action given by $\arg\max_{b \in A(m)} P(m, b)$, at the end of which ρ is calculated as the sum of the immediate rewards earned in simulation divided by the simulation time.

4.2. Stochastic Games
The stochastic game (or a competitive Markov decision problem) is a sequential decision-making problem with multiple decision makers. We first formalize the framework needed for stochastic games.

Let n denote the number of players and $\mathcal{A}^l$ the set of actions associated with the lth player. In a stochastic game, the immediate rewards and the TPs depend on the current state and the actions chosen by *all* the players (agents). Thus for n players, the TP function is the following map: $p: \mathcal{S} \times \mathcal{A}^1 \times \mathcal{A}^2 \times \cdots \times \mathcal{A}^n \times \mathcal{S} \to \Re$. The immediate reward will, however, be a *different* function for each player. For the lth player, in the transition from state i to state j when the mth player, for

$m = 1, 2 \ldots, n$, choses action a^m, the immediate reward will be denoted by $r^l(i, a^1, a^2, \ldots, a^n, j)$. The policy of a decision maker in an MDP is referred to as a *strategy* of the player in game theoretic literature. If $\hat{d}^l$ denotes the strategy of the lth player, $d^l(i)$ will denote the corresponding action chosen in state i by the lth player. Clearly, the performance metric here is a function of the strategies selected by *all* players. The discounted reward for the lth player associated with the strategy-tuple, $(\hat{d}^1, \hat{d}^2, \ldots, \hat{d}^n)$, when i is the starting state and x_s the state before the sth transition, can be defined as

$$J^l(i, \hat{d}^1, \hat{d}^2, \ldots, \hat{d}^n)$$
$$\equiv \lim_{k \to \infty} \mathsf{E}\left[\sum_{s=1}^k \gamma^{s-1} r^l(x_s, d^1(x_s), d^2(x_s), \ldots, d^n(x_s), x_{s+1}) \,\Big|\, x_1 = i\right].$$

The most challenging and exciting version of the problem is the most general case where each player wishes to optimize his or her own objective function. For this game, an appealing and well-known solution called Nash equilibrium is of the following form. We discuss it for the case of two players. Consider a strategy-tuple $(\hat{d}^1_*, \hat{d}^2_*)$ such that for all $i \in \mathcal{S}$

$$J^1(i, \hat{d}^1_*, \hat{d}^2_*) \geq J^1(i, \hat{d}^1, \hat{d}^2_*) \quad \text{for all } \hat{d}^1 \quad \text{and}$$
$$J^2(i, \hat{d}^1_*, \hat{d}^2_*) \geq J^2(i, \hat{d}^1_*, \hat{d}^2) \quad \text{for all } \hat{d}^2.$$

Any *unilateral* deviations (deviations made without telling other players) from $(\hat{d}^1_*, \hat{d}^2_*)$ either by player 1 or 2 will lead to losses for the one who deviates. Let $Q^l(i, a^1, a^2)$ for $l = 1, 2$ denote the Q-value of the lth player associated with player 1 choosing action a^1 and player 2 chosing action a^2 in state i. A Q-learning algorithm can be implemented in a simulator in which one simulates transitions from one state to another. After each transition from i to j, the following rule for updating can be used. For $l = \{1, 2\}$,

$$Q^l(i, a^1, a^2) \leftarrow (1 - \mu)Q^l(i, a^1, a^2)$$
$$+ \mu\left[r(i, a^1, a^2, j) + \gamma Q^l_{\text{next}}\right].$$

Hu and Wellman (2003) developed a Nash-Q-learning algorithm for discounted reward, with

$$Q^l_{\text{next}} = \text{Nash}_{b^1 \in \mathcal{A}^1(j),\, b^2 \in \mathcal{A}^2(j),\, \ldots,\, b^n \in \mathcal{A}^n(j)} Q^l_k(j, b^1, b^2, \ldots, b^n),$$

where the Nash operator over the Q-values of a given player for the state j is the Q-value associated with a Nash-equilibrium solution. (Table 1 provides a simple example.) RL algorithms have been developed for rational games (Bowling and Veloso 2002), zero-sum games (Littman 1994), friend-and-foe games (Littman 2001), and average reward games (Ravulapati et al. 2004).

Table 1 The First Entry in Any Round Bracket Is the Q-Value for the Row Player and the Second for the Column Player

	Column player	
Row player	Action 1	Action 2
Action 1	$(2, 3)$	$(9, -1)$
Action 2	$(-1, 9)$	$(7, 7)$

Notes. The Nash equilibrium is $(2, 3)$. The solution $(7, 7)$ will be unstable, although it is mutually beneficial, because both players have incentive to deviate.

4.3. Hierarchical MDPs

Some MDPs have a special structure that makes them amenable to a so-called hierarchical decomposition (Forestier and Varaiya 1978). With such a decomposition, one can essentially solve MDPs with smaller state spaces at a lower level that supply solutions to a control-optimization problem at a higher level. In other words, the state space can be effectively divided into disjoint sets in which an MDP is solved in each set without (sometimes with; see, e.g., Makar et al. 2001) consideration of the MDP in the other. These solutions form the input to a higher-level problem that does not attempt to make decisions at the lower-level states. Considerable research has been performed in developing hierarchical methods of RL. In fact, this appears to be a challenging frontier of research in RL. The options framework (Sutton et al. 1998), the MAXQ framework (Dietterich 2000), and the hierarchical-abstract-machines (HAMs) framework (Parr 1998, Parr and Russell 1998) are some of the recent developments in this area. See also Barto and Mahadevan (2003) for a review of recent work in this area and Makar et al. (2001) for an application. We now describe options and HAMs; both of these rely on SMDP models.

An option is essentially a fixed policy. In an options framework, the system is restricted to a set of options in every disjoint subset of the state space. In any state, one has to choose an option from the set of options available in that state. When the system enters a new subset of the state space, a new set of options becomes available. Because each option is like a fixed policy, it is composed of "primitive" actions, e.g., accept customer and reject customer. Broadly speaking, the underlying goal in such a setting is to search over a restricted set of policies in each subset of the state space. The Q-values for selecting an option, o, in a state i can be learned in the following style for discounted reward:

$$Q(i, o) \leftarrow (1 - \mu)Q(i, o)$$
$$+ \mu \left[r(i, o, j) + \gamma^{t(i, o, j)} \max_{o' \in \mathcal{O}(j)} Q(j, o') \right],$$

where j is a state in which the current option o terminates and a new set of options becomes available;

in the transition from i to j, $r(i, o, j)$ and $t(i, o, j)$ denote the immediate reward earned and the time spent, respectively. In the above, $\mathcal{O}(j)$ denotes the set of options available in state j. The updating is thus performed when an option terminates.

In a HAMs framework, in addition to the system state and actions, one considers the paradigm of machine states, which belong to the following set: $\{Choice, Call, Act, Stop\}$. Whenever the lower-level controller reaches a state at which higher-level control is needed, a machine state is also entered and a *choice point* is said to have been reached. At the choice point, the action (which could be a composite policy over internal states of the low-level controller) is selected. The choice point is hence defined by $\langle i, z \rangle$, where i denotes the system state and z the machine state. If the last choice point encountered at time s was $\langle i, z \rangle$ and the current choice point at time $s + \tau$ is $\langle j, z' \rangle$, then if r_{s+l} denotes the immediate reward in time step $s + l$, the update for the Q-values is

$$Q(\langle i, z \rangle, a) \leftarrow (1 - \mu)Q(\langle i, z \rangle, a)$$
$$+ \mu \left[\sum_{l=1}^{\tau} \gamma^{l-1} r_{s+l} + \gamma^{\tau} \max_b Q(\langle j, z' \rangle, b) \right].$$

4.4. Learning Automata and Policy Gradient Algorithms

A number of paradigms in RL are not based on notions of DP. Two prominent paradigms are the learning automata (LA) and the policy gradient (PG). In both frameworks, one starts with a stochastic policy that selects actions randomly in the beginning. The probability of selecting an action in a given state is updated generally after one or more transitions, based on feedback from the simulator, until the optimal policy is identified.

Most LA algorithms consider the average reward performance metric (Narendra and Thathachar 1989). An important algorithm for solving MDPs (Wheeler and Narendra 1986) has been extended to SMDPs (Gosavi et al. 2004). LA concepts have also been extended to combinatorial optimization (Thathachar and Sastry 1987, Poznyak and Najim 1997, Tezcan and Gosavi 2001).

PG algorithms have gained popularity recently and have become an active area of research. The papers of Williams (1992), Kimura et al. (1995, 1997), and Baxter and Bartlett (2001) are some of the initial works of research in this area; see also Peshkin (2001) and Ng and Jordan (2000). The main idea underlying a PG algorithm is to treat the performance metric of the MDP as a function of the probability of selecting actions in states and to use gradient ascent to optimize the function; the gradient is computed with respect to the probability, which is stored as a predefined (linear or nonlinear) function of a vector $\vec{w}$, the

action and the state. As the learning progresses, the vector $\vec{w}$ is updated, which results in updating the probabilities. The size of $\vec{w}$, L, is much smaller than the state-action space. Let $q(i, a, \vec{w})$ denote a function whose inputs are the action a, state i, and current values of the vector $\vec{w}$ and whose output is a probability. In addition to $\vec{w}$, one stores two other vectors, $\vec{\psi}$ and $\vec{\Delta}$, both of size L. Thus this algorithm requires the storage of a total of only $3L$ scalars. Every element of $\vec{\psi}$ and $\vec{\Delta}$ is initialized to zero. In any iteration, if the current state is i, action a is selected with probability $q(i, a, \vec{w})$. If the kth transition in the simulator is from state i to state j under action a, the updates to be carried out in the following order are these: For every $l = 1, 2, \ldots, L$,

$$\psi(l) \leftarrow \kappa \psi(l) + \frac{\partial q(i, a, \vec{w})/\partial w(l)}{q(i, a, \vec{w})},$$

$$\Delta(l) \leftarrow \Delta(l) + \frac{1}{k}[r(i, a, j)\psi(l) - \Delta(l)], \quad \text{and}$$

$$w(l) \leftarrow w(l) + \mu\Delta(l),$$

where $\kappa \in (0, 1)$ is fixed at the start. $\vec{\Delta}$, which estimates the gradient, can be computed in a number of different ways (Cao and Guo 2004, Marbach and Tsitsiklis 1999). The gradient computations can also be integrated within those of a neural network (Sutton et al. 2000).

4.5. Partially Observable MDPs (POMDPs)
These are variants of the MDP in which the state of the system is only partially visible to the decision maker. What is available is a set of signals that can be used to come up with a guess about the current state. The Markov property does not hold in these circumstances. In such problems, it can be shown (Lovejoy 1991, Sondik 1978, Smallwood and Sondik 1973, Monahan 1982, White and Scherer 1989, Lin et al. 2004) that it is sufficient to keep a probability distribution called the *belief state distribution* of the current state. The most challenging aspect of this problem is that the state space becomes continuous, and consequently it is difficult to solve the problem *exactly* even for a handful of states. In an excellent paper, Kaelbling et al. (1999) present a great deal of POMDP theory in the context of RL. In their work, RL has shown considerable promise in being able to develop high-quality solutions to POMDPs. They also explain the notion of "policy trees" that can be employed to solve POMDPs.

4.6. Factored MDPs and Model-Based RL
Model-based RL makes use of the TP model. Model-free RL, as stated above, uses the RM scheme in a simulator and bypasses the need for a TP model. The central idea in model-based RL is to generate the TP model, either by a straightforward counting procedure in a simulator (see the Online Supplement) or else by exploiting the structure of the problem to generate a so-called dynamic Bayesian network (Dean and Kanazawa 1989). Within simulators, it is possible to construct the TPMs of all the actions and then perform a DP-like update using the TPs. Real-time DP (Barto et al. 1995) and *H*-learning (Tadepalli and Ok 1998) are model-based algorithms for discounted and average reward, respectively. See Chapter 9 (§10) of Gosavi (2003) for discussions of other model-based RL algorithms. Use of Bayesian networks to represent the TPs in a compact manner was pioneered by Boutilier et al. (1999) and was called a factored MDP. The TPMs are learned as functions of a few scalars via a Bayesian network. Thereafter, RL algorithms can be used in conjunction with the TP model learned.

4.7. Nonstationary RL
RL has been widely applied in *on-line* applications in the domain of robotics with considerable success (Sutton and Barto 1998). In on-line algorithms, a simulator is not used; instead, the agent is allowed to gather experience in the real-world system. In such implementations, information about the system becomes available gradually with time. This is in contrast to simulation-based applications, in which the distributions of the underlying random variables are assumed to be known. Algorithms in such simulation-based applications are referred to as *off-line* algorithms because the learning phase of these algorithms is not used on the real-world system. Off-line algorithms have the advantage of not requiring costly runs of the system with sub-optimal policies. However, if the distribution functions of the underlying random variables cannot be estimated accurately and if trial runs of the real-world system are not too expensive, on-line algorithms are more suitable. Also, on-line algorithms may be useful in environments that vary with time, i.e., *not* stationary. In such environments, the TPMs are either noisy (Givan et al. 2000, Satia and Lave 1973, White and Eldeib 1994) or assumed to change with time (Szita et al. 2002). This is an exciting topic because it promises to make the MDP model more general, but it is likely to be significantly more challenging than the time-stationary MDP.

4.8. Convergence
Study of convergence properties of an algorithm enables one to better understand its behavior—in particular, the ability of the algorithm to obtain an optimal solution. We will cite some important references and not attempt to present any results.

RL algorithms are guaranteed to converge to optimal solutions for the look-up table case. The convergence is generally established with probability 1

(w.p.1) because of the RM basis of RL. The first results for asynchronous convergence of discounted Q-learning (Jaakola et al. 1994, Tsitsiklis 1994) were based on norm contractions. The idea of ordinary differential equations (ODEs) for proving convergence under asynchronous conditions was proposed in Borkar (1998), where it was shown that the iterate tracks an ODE, which is much slower than the one shown to exist under synchronous conditions (Kushner and Clark 1978). This result was subsequently pursued for the average reward algorithms (Abounadi et al. 2001), which also exploited results for nonexpansive mappings (Borkar and Soumyanath 1997). A more general ODE result that can be used for both discounted and average reward cases was proposed later (Borkar and Meyn 2000); this result employs notions of fluid limits (Dai and Meyn 1995). Most of these results (see, however, Abounadi et al. 2002) require showing a priori boundedness of the iterate, which is possible under some conditions (Bertsekas and Tsitsiklis 1996, Borkar and Meyn 2000, Gosavi 2006), and the existence of some asymptotic properties of the ODE. Once this is accomplished, a critical lemma from Hirsch (1989) is employed to prove convergence w.p.1.

The picture is somewhat incomplete for the analysis when function approximation is used (Chapter 6 of Bertsekas and Tsitsiklis 1996). Some of the initial works in bounding the behavior of greedy policies obtained via function approximation are Williams and Baird (1993) and Singh and Yee (1994). TD methods using function approximation are known to converge (Sutton 1984, 1992). Function approximation with state aggregation has also been analyzed (Tsitsiklis and van Roy 1996) in the literature. For PE, linear function approximation has been shown to converge in Bradtke and Barto (1996). Baird (1995) discusses a combination of TD with gradient-descent-based function approximation.

4.9. Applications
RL has been applied in a large number of domains successfully. Here, we enumerate a few case studies related to operations management. In particular, we describe the special features of the algorithms that made them suitable for the domain of application. Continuous-time discounted algorithms (Bradtke and Duff 1995) were employed for elevator scheduling (Crites and Barto 1995, Pepyne et al. 1996) because the problem structure had a continuous-time Markov chain underlying it. The job-shop scheduling problem in Zhang and Dietterich (1995) had an episodic nature, and hence TD(λ) became preferable (Sutton 1984, Peng and Williams 1993). Preventive maintenance problems are usually SMDPs with an undiscounted objective function, which make SMART (Das

et al. 1999) and R-SMART (Gosavi 2004b) suitable. Other applications of these algorithms include (for R-SMART) "voice-over-packet" networks in Akar and Sahin (2003) and (for SMART) vendor selection for supply chains in Pontrandolfo et al. (2002). The AGV routing problem in Tadepalli and Ok (1998) is one of the few case studies of model-building RL for large-scale problems. The model they learn is able to capture the complex dynamics of the AGV problem. A well-known "revenue management problem" can be set up as an average reward SMDP (Gosavi 2004a, Gosavi et al. 2002), but it has a unique reward structure, with much of the reward concentrated in certain states that makes SMART, which is TD(0), unstable. Hence, Q-P-learning (Gosavi 2004a) and λ-SMART (Gosavi et al. 2002) were applied. The work related to hyperheuristics (Burke et al. 2003a, b; Nareyek 2003) can be used when RL is to be used dynamically to select a metaheuristic. An SMDP with a discount factor was employed for a cell phone network management problem (Singh and Bertsekas 1997) that allowed handy combination with a neuron-based function approximator. The retailer-inventory management problem in van Roy et al. (1997) used regular Q-learning with a vanishing discount factor. Makar et al. (2001) employ hierarchical RL because the AGV scheduling task they consider has controllers at multiple levels. It is likely that the field of applied RL will explode in the coming years because of RL's ability to solve problems previously considered intractable.

5. Conclusions
The MDP (Bellman 1954, Howard 1960) has found numerous applications in industrial settings (White 1985). However, the curses of modeling and dimensionality have plagued it for many years. Because of these curses, DP has not been very effective in solving many realistic MDPs in the real world. The advent of RL has engendered a significant change in our ability to solve MDPs—especially those that have complex transition mechanisms and large state spaces.

RL is unique in its use of the combination of four distinct machineries: stochastic approximation, DP, AI, and function approximation. By exploiting these machineries, RL has opened the avenue for solving large-scale MDPs (and its variants) in discrete-event systems, which were considered intractable in the past. RL has outperformed industrial heuristics in several of the published case studies of industrial importance. RL as a science is relatively young and has already made a considerable impact on operations research. The optimism expressed about RL in the early surveys (Keerthi and Ravindran 1994, Kaelbling et al. 1996, Mahadevan 1996) has been bolstered by several success stories.

Although a great deal of work in algorithmic development of RL has already occurred, RL continues to attract research attention. To familiarize the reader with open problems in RL, we enumerate some areas undergoing active research. Function approximation (Mahadevan and Maggioni 2006, Whiteson and Stone 2006) to this day remains an unsolved problem because of the lack of stability of known approximators (discussed in §3.3). Hierarchical RL (Bhatnagar and Panigrahi 2006) also promises to solve large-scale problems and those with special multi-layer structures. Policy gradients (Singh et al. 2007) has emerged as a new area in RL research with a gradient-descent approach to the MDP. Interest in linear programming (LP) for solving MDPs has been renewed (Defarias and van Roy 2003) because of the well-known stability properties of LP solvers. Other areas that are attracting interest are risk-sensitive RL (Borkar 2002, Geibel and Wysotzki 2005), factored MDPs (Schuurmans and Patrascu 2001), and analyzing computational complexity (Evan-Dar and Mansour 2003).

Acknowledgments

References

Abounadi, J., D. Bertsekas, V. Borkar. 2001. Learning algorithms for Markov decision processes with average cost. *SIAM J. Control Optim.* **40** 681–698.

Abounadi, J., D. Bertsekas, V. Borkar. 2002. Stochastic approximations for non-expansive maps: Application to Q-learning algorithms. *SIAM J. Control Optim.* **41** 1–22.

Akar, N., C. Sahin. 2003. Reinforcement learning as a means of dynamic aggregate QoS provisioning. W. Burakowski, B. Koch, A. Beben, eds. *Architectures for Quality of Service in the Internet*. Lecture Notes in Computer Science, Vol. 2698. Springer, Heidelberg, Germany, 100–114.

Arapostathis, A., V. Borkar, E. Fernandez-Gaucherand, M. Ghosh, S. Marcus. 1993. Discrete-time controlled Markov processes with average cost criterion: A survey. *SIAM J. Control Optim.* **31** 282–344.

Baird, L. 1995. Residual algorithms: Reinforcement learning with function approximation. *Proc. 12th Internat. Conf. Machine Learn.*, Morgan Kaufmann, San Francisco, 30–37.

Barto, A. G., S. Mahadevan. 2003. Recent advances in hierarchical reinforcement learning. *Discrete-Event Dynam. Systems: Theory Appl.* **13** 41–77.

Barto, A. G., S. J. Bradtke, S. P. Singh. 1995. Learning to act using real-time dynamic programming. *Artificial Intelligence* **72** 81–138.

Barto, A. G., R. S. Sutton, C. W. Anderson. 1983. Neuronlike elements that can solve difficult learning control problems. *IEEE Trans. Systems, Man, Cybernetics* **13** 835–846.

Baxter, J., P. Bartlett. 2001. Infinite-horizon policy-gradient estimation. *J. Artificial Intelligence* **15** 319–350.

Bellman, R. 1954. The theory of dynamic programming. *Bull. Amer. Math. Soc.* **60** 503–516.

Bertsekas, D. 1995. *Dynamic Programming and Optimal Control*. Athena Scientific, Nashua, NH.

Bertsekas, D., J. Tsitsiklis. 1996. *Neuro-Dynamic Programming*. Athena Scientific, Nashua, NH.

Bhatnagar, S., J. R. Panigrahi. 2006. Actor-critic algorithms for heirarchical Markov decision processes. *Automatica* **42** 637–644.

Borkar, V. S. 1997. Stochastic approximation with two-time scales. *Systems Control Lett.* **29** 291–294.

Borkar, V. S. 1998. Asynchronous stochastic approximation. *SIAM J. Control Optim.* **36** 840–851.

Borkar, V. S. 2002. Q-learning for risk-sensitive control. *Math. Oper. Res.* **27** 294–311.

Borkar, V. S., S. P. Meyn. 2000. The ODE method for convergence of stochastic approximation and reinforcement learning. *SIAM J. Control Optim.* **38** 447–469.

Borkar, V. S., K. Soumyanath. 1997. A new analog parallel scheme for fixed point computation, Part I: Theory. *IEEE Trans. Circuits Systems I: Theory Appl.* **44** 351–355.

Boutilier, C., T. Dean, S. Hanks. 1999. Decision-theoretic planning: Structural assumptions and computational leverage. *J. Artificial Intelligence Res.* **11** 1–94.

Bowling, M., M. Veloso. 2002. Multi-agent learning using a variable learning rate. *Artificial Intelligence* **136** 215–250.

Boyan, J. A., A. W. Moore. 1995. Generalization in reinforcement learning: Safely approximating the value function. D. Touretzky, M. Mozer, M. Hasselmo, eds. *Advances in Neural Information Processing Systems*. MIT Press, Cambridge, MA, 369–376.

Bradtke, S., A. Barto. 1996. Linear least-squares algorithms for temporal difference learning. *Machine Learn.* **22** 33–57.

Bradtke, S. J., M. Duff. 1995. Reinforcement learning methods for continuous-time Markov decision problems. G. Tesauro, D. Touretzky, T. Leen, eds. *Advances in Neural Information Processing Systems*, Vol. 7. MIT Press, Cambridge, MA, 393–400.

Burke, E. K., G. Kendall, E. Soubeiga. 2003a. A tabu-search hyper-heuristic for timetabling and rostering. *J. Heuristics* **9** 451–470.

Burke, E., G. Kendall, J. Newall, E. Hart, P. Ross, S. Schulenburg. 2003b. Hyper-heuristics: An emerging direction in modern search technology. F. Glover, G. Kochenberger, eds. *Handbook of Meta-heuristics*, Chapter 16. Kluwer Academic Publishers, Boston, 457–474.

Cao, X.-R., X. Guo. 2004. A unified approach to Markov decision problems and performance sensitivity analysis with discounted and average criteria. *Automatica* **40** 1749–1759.

Chapman, D., L. Kaelbling. 1991. Input generalization in delayed reinforcement learning. *Proc. 12th Internat. Joint Conf. Artificial Intelligence*, Morgan Kaufmann, San Francisco, 726–731.

Crites, R., A. Barto. 1995. Improving elevator performance using reinforcement learning. *Proc. Adv. Neural Information Processing Systems*, MIT Press, Cambridge, MA, 1017–1023.

Dai, J., S. P. Meyn. 1995. Stability and convergence of moments for multiclass queuing networks via fluid limits. *IEEE Trans. Automatic Control* **40** 1889–1904.

Darken, C., J. Chang, J. Moody. 1992. Learning rate schedules for faster stochastic gradient search. D. A. White, D. A. Sofge, eds. *Neural Networks for Signal Processing 2—Proc. 1992 IEEE Workshop*, IEEE Press, Piscataway, NJ.

Das, T. K., A. Gosavi, S. Mahadevan, N. Marchalleck. 1999. Solving semi-Markov decision problems using average reward reinforcement learning. *Management Sci.* **45** 560–574.

Dayan, P. 1992. The convergence of TD(λ) for general λ. *Machine Learn.* **8** 341–362.

Dean, T., K. Kanazawa. 1989. A model for reasoning about persistence and causation. *Computational Intelligence* **5** 142–150.

Defarias, D. P., B. van Roy. 2003. The linear programming approach to approximate dynamic programming. *Oper. Res.* **51** 850–865.

Dietterich, T. 2000. Hierarchical reinforcement learning with the MAXQ value function decomposition. *J. Artificial Intelligence Res.* **13** 227–303.

Evan-Dar, E., Y. Mansour. 2003. Learning rates for *Q*-learning. *J. Machine Learn. Res.* **5** 1–25.

Forestier, J., P. Varaiya. 1978. Multilayer control of large Markov chains. *IEEE Trans. Automatic Control* **23** 298–304.

Geibel, P., F. Wysotzki. 2005. Risk-sensitive reinforcement learning applied to control under constraints. *J. Artificial Intelligence Res.* **24** 81–108.

Givan, R., S. Leach, T. Dean. 2000. Bounded parameter Markov decision processes. *Artificial Intelligence* **122** 71–109.

Gordon, G. 1995. Stable function approximation in dynamic programming. Technical Report CMU-CS-95-103, Carnegie Mellon University, Pittsburgh.

Gosavi, A. 2003. *Simulation-Based Optimization: Parametric Optimization Techniques and Reinforcement Learning.* Kluwer Academic Publishers, Boston.

Gosavi, A. 2004a. A reinforcement learning algorithm based on policy iteration for average reward: Empirical results with yield management and convergence analysis. *Machine Learn.* **55** 5–29.

Gosavi, A. 2004b. Reinforcement learning for long-run average cost. *Eur. J. Oper. Res.* **155** 654–674.

Gosavi, A. 2006. Boundedness of iterates in *Q*-learning. *Systems Control Lett.* **55** 347-349.

Gosavi, A., N. Bandla, T. K. Das. 2002. A reinforcement learning approach to a single leg airline revenue management problem with multiple fare classes and overbooking. *IIE Trans.* **34** 729–742.

Gosavi, A., T. K. Das, S. Sarkar. 2004. A simulation-based learning automata framework for solving semi-Markov decision problems. *IIE Trans.* **36** 1–11.

Grinstead, C. M., J. L. Snell. 1997. *Introduction to Probability.* American Mathematical Society, Providence, RI.

Hastie, T., R. Tibshirani, J. Friedman. 2001. *The Elements of Statistical Learning.* Springer, New York.

Hauskrecht, M. 2000. Value-function approximations for partially observable Markov decision processes. *J. Artificial Intelligence Res.* **13** 33–94.

Hirsch, M. 1989. Convergent activation dynamics in continuous time networks. *Neural Networks* **2** 331–349.

Howard, R. 1960. *Dynamic Programming and Markov Processes.* MIT Press, Cambridge, MA.

Hu, J., M. P. Wellman. 2003. Nash *Q*-learning for general-sum stochastic games. *J. Machine Learn. Res.* **4** 1039–1069.

Jaakola, T., M. Jordan, S. P. Singh. 1994. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Comput.* **6** 1185–1201.

Kaelbling, L. P., M. L. Littman, A. R. Cassandra. 1999. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* **101** 99–134.

Kaelbling, L. P., M. L. Littman, A. W. Moore. 1996. Reinforcement learning: A survey. *J. Artificial Intelligence Res.* **4** 237–285.

Kearns, M., S. P. Singh. 2002. Near-optimal reinforcement learning in polynomial time. *Machine Learn.* **49** 209–232.

Keerthi, S., B. Ravindran. 1994. A tutorial survey of reinforcement learning. *Sadhana* **19** 851–889.

Kimura, H., K. Miyazaki, S. Kobayashi. 1997. Reinforcement learning in POMDPs with function approximation. *Proc. 14th Internat. Conf. Machine Learn.,* Morgan Kaufmann, San Francisco, 152–160.

Kimura, H., M. Yamamura, S. Kobayashi. 1995. Reinforcement learning by stochastic hill climbing on discounted reward. *Proc. 12th Internat. Conf. Machine Learn., Tahoe City, CA,* Morgan Kaufmann, San Francisco, 295–303.

Konda, V. R., J. Tsitsiklis. 2000. Actor-critic algorithms. *Neural Information Processing Systems 12.* MIT Press, Cambridge, MA, 1008–1014.

Konda, V. R., V. S. Borkar. 1999. Actor-critic type learning algorithms for Markov decision processes. *SIAM J. Control Optim.* **38** 94–123.

Kushner, H. J., D. S. Clark. 1978. *Stochastic Approximation Methods for Constrained and Unconstrained Systems.* Springer-Verlag, New York.

Lin, Z., J. Bean, C. C. White III. 2004. A hybrid genetic/optimization algorithm for finite horizon, partially observed Markov decision process. *INFORMS J. Comput.* **16** 27–38.

Littman, M. 1994. Markov games as a framework for multi-agent reinforcement learning. W. W. Cohen, H. Hirsch, eds. *Proc. 11th Internat. Conf. Machine Learn., New Brunswick, NJ,* Morgan Kaufmann, San Francisco, 157–163.

Littman, M. 2001. Friend or foe *Q*-learning in general-sum games. C. E. Brodley, A. P. Danyluk, eds. *Proc. 18th Internat. Conf. Machine Learn., Williamston, MA,* Morgan Kaufmann, San Francisco, 322–328.

Lovejoy, W. 1991. A survey of algorithmic methods for partially observable Markov decision processes. *Ann. Oper. Res.* **28** 47–66.

Luce, D. 1959. *Individual Choice Behavior.* John Wiley & Sons, New York.

Mahadevan, S. 1996. Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine Learn.* **22** 159–95.

Mahadevan, S., M. Maggioni. 2006. Value function approximation using diffusion wavelets and Laplacian eigenfunction. *Neural Information Processing Systems.* MIT Press, Cambridge, MA.

Makar, R., S. Mahadevan, M. Ghavamzadeh. 2001. Hierarchical multi-agent reinforcement learning. *Proc. 5th Internat. Conf. Autonomous Agents, Montréal,* ACM, New York, 246–253.

Marbach, P., J. Tsitsiklis. 1999. Simulation-based optimization of Markov reward processes: Implementation issues. *Proc. 38th IEEE Conf. Decision and Control, Phoenix, AZ,* IEEE, Washington, D.C., 1769–1774.

Monahan, G. 1982. A survey of partially observable Markov decision processes: Theory, models, and algorithms. *Management Sci.* **28** 1–16.

Murthy, S., S. Kasif, S. Salzberg. 1994. A system for induction of oblique decision trees. *J. Artificial Intelligence Res.* **2** 1–33.

Narendra, K. S., M. A. L. Thathachar. 1989. *Learning Automata: An Introduction.* Prentice Hall, Englewood Cliffs, NJ.

Nareyek, A. 2003. Choosing search heuristics by non-stationary reinforcement learning. M. G. Resende, J. P. de Sousa, eds. *Metaheuristics: Computer Decision-Making.* Kluwer Academic Publishers, Dordrecht, The Netherlands, 523–544.

Ng, A., M. Jordan. 2000. PEGASUS: A policy search method for large MDPs and POMDPs. C. Boutilier, M. Goldszmidt, Eds. *Proc. 16th Conf. Uncertainty in Artificial Intelligence,* Morgan Kaufmann, San Mateo, CA, 406–415.

Ormoneit, D., S. Sen. 2002. Kernel-based reinforcement learning. *Machine Learn.* **49** 161–178.

Parr, R. 1998. Hierarchical control and learning for Markov decision processes. Unpublished Ph.D. dissertation, University of California, Berkeley, Berkeley.

Parr, R., S. Russell. 1998. Reinforcement learning with hierarchies of machines. *Neural Information Processing Systems.* MIT Press, Cambridge, MA, 1043–1049.

Peng, J., R. J. Williams. 1993. Efficient learning and planning with the DYNA framework. *Adapt. Behav.* **1** 437–454.

Pepyne, D. L., D. P. Looze, C. G. Cassandras, T. E. Djaferis. 1996. Application of *Q*-learning to elevator dispatching. *Proc. 13th Internat. Federation Automatic Control World Congress, San Francisco.* Pergamon, San Francisco.

Peshkin, L. 2001. Reinforcement learning by policy search. Ph.D. thesis, MIT, Cambridge, MA.

Pontrandolfo, P., A. Gosavi, O. G. Okogbaa, T. K. Das. 2002. Global supply chain management: A reinforcement learning approach. *Internat. J. Production Res.* **40** 1299–1317.

Poznyak, A. S., K. Najim. 1997. *Learning Automata and Stochastic Optimization.* Lecture Notes in Control and Information Sciences, Vol. 225. Springer-Verlag, London.

Puterman, M. L. 1994. *Markov Decision Processes.* Wiley Interscience, New York.

Pyeatt, L., A. Howe. 1998. Decision tree function approximation in reinforcement learning. Technical Report CS-98-112, Colorado State University, Fort Collins, CO.

Quinlan, J. 1986. Induction of decision trees. *Machine Learn.* **1** 81–106.

Ravulapati, K. K., J. Rao, T. K. Das. 2004. A reinforcement learning approach to stochastic business games. *IIE Trans.* **36** 373–385.

Robbins, H., S. Monro. 1951. A stochastic approximation method. *Ann. Math. Statist.* **22** 400–407.

Rummery, G. A., M. Niranjan. 1994. On-line *Q*-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166, Cambridge University, Cambridge, UK.

Satia, J. K., R. E. Lave, Jr. 1973. Markovian decision processes with uncertain transition probabilities. *Oper. Res.* **21** 728–740.

Sato, M., K. Abe, H. Takeda. 1990. Learning control of finite Markov chains with explicit trade-off between estimation and control. D. S. Touretsky, J. L. Elman, T. J. Sejnowski, eds. *Connectionist Models: Proc. 1990 Summer School,* Morgan Kaufmann, San Francisco, 287–300.

Schuurmans, D., R. Patrascu. 2001. Direct value-approximation for factored MDPs. *Neural Information Processing Systems,* Vol. 14. MIT Press, Cambridge, MA, 1579–1586.

Schwartz, A. 1993. A reinforcement learning method for maximizing undiscounted rewards. *Proc. 10th Ann. Conf. Machine Learn.,* Morgan Kaufmann, San Francisco, 298–305.

Singh, S. P. 1994. Reinforcement learning algorithms for average-payoff Markovian decision processes. *Proc. 12th AAAI,* MIT Press, Cambridge, MA, 700–705.

Singh, S. P., D. Bertsekas. 1997. Reinforcement learning for dynamic channel allocation in cellular telephone systems. *Adv. Neural Inform. Processing Systems* **9** 974–980.

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

Singh, S. P., R. Yee. 1994. An upper bound on the loss from approximate optimal value functions. *Machine Learn.* **16** 227–233.

Singh, S. S., V. Tadic, A. Doucet. 2007. A policy-gradient method for semi-Markov decision processes with application to call admission control. *Eur. J. Oper. Res.* **178** 808–818.

Smallwood, R. D., E. J. Sondik. 1973. The optimal control of partially observable Markov processes over a finite horizon. *Oper. Res.* **21** 1071–1088.

Smart, W., L. Kaelbling. 2000. Practical reinforcement learning in continuous spaces. *Proc. 17th Internat. Conf. Machine Learn., Stanford, CA,* Morgan Kaufmann, San Francisco, 903–910.

Sondik, E. 1978. The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs. *Oper. Res.* **26** 282–304.

Sutton, R. S. 1984. Temporal credit assignment in reinforcement learning. Unpublished Ph.D. dissertation, University of Massachusetts, Amherst.

Sutton, R. S. 1990. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. *Proc. 7th Internat. Conf. Machine Learn., Austin, TX,* Morgan Kaufmann, San Francisco, 216–224.

Sutton, R. S. 1992. Reinforcement learning. *Machine Learn.* **8** 225–227.

Sutton, R. S. 1996. Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Neural Information Processing Systems,* Vol. 8. MIT Press, Cambridge, MA, 1038–1044.

Sutton, R., A. G. Barto. 1998. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA.

Sutton, R., D. Precup, S. P. Singh. 1998. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence* **112** 181–211.

Sutton, R., D. McAllester, S. P. Singh, Y. Mansour. 2000. Policy gradient methods for reinforcement learning with function approximation. *Proc. Adv. Neural Information Processing Systems,* MIT Press, Cambridge, MA, 1057–1063.

Szita, I., B. Takacs, A. Lorincz. 2002. ϵ-MDPs: Learning in varying environments. *J. Machine Learn. Res.* **3** 145–174.

Tadepalli, P., D. Ok. 1998. Model-based average reward reinforcement learning algorithms. *Artificial Intelligence* **100** 177–224.

Tesauro, G. 1992. Practical issues in temporal difference learning. *Machine Learn.* **8** 257–277.

Tezcan, T., A. Gosavi. 2001. Optimal buffer allocation in production lines using an automata search. *Proc. Inst. Indust. Engrg. Conf., Dallas.* Institute of Industrial Engineers, Norwell, GA.

Thathachar, M. A. L., P. S. Sastry. 1987. Learning optimal discriminant functions through a cooperative game of automata. *IEEE Trans. Systems, Man, Cybernetics* **17** 73-85.

Thrun, S., K. Moller. 1992. Active exploration in dynamic environments. *Advances in Neural Information Processing Systems.* Morgan Kaufmann, San Francisco, 531–538.

Tsitsiklis, J. N., 1994. Asynchronous stochastic approximation and *Q*-learning. *Machine Learn.* **16** 185–202.

Tsitsiklis, J. N., B. van Roy. 1996. Feature-based methods for large-scale dynamic programming. *Machine Learn.* **22** 59–94.

Tsitsiklis, J. N., B. van Roy. 1997. An analysis of temporal-difference learning with function approximation. *IEEE Trans. Automatic Control* **42** 674–690.

van Nunen, J. A. E. E. 1976. A set of successive approximation methods for discounted Markovian decision problems. *Z. Oper. Res.* **20** 203–208.

van Roy, B., D. P. Bertsekas, Y. Lee, J. N. Tsitsiklis. 1997. A neuro-dynamic programming approach to retailer inventory management. *Proc. IEEE Conf. Decision and Control, San Diego,* IEEE, Washington, D.C., 4052–4057.

Watkins, C. J. 1989. Learning from delayed rewards. Ph.D. thesis, Kings College, Cambridge, UK.

Werbös, P. J. 1987. Building and understanding adaptive systems: A statistical/numerical approach to factory automation and brain research. *IEEE Trans. Systems, Man, Cybernetics* **17** 7–20.

Werbös, P. J. 1992. Approximate dynamic programming for real-time control and neural modeling. D. A. White, D. A. Sofge, eds. *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches.* Van Nostrand, NY, 493–525.

Wheeler, R. M., K. S. Narendra. 1986. Decentralized learning in finite Markov chains. *IEEE Trans. Automatic Control* **31** 373–376.

White, D. J. 1985. Real applications of Markov decision processes. *Interfaces* **15** 73–83.

White III, C. C., H. K. Eldeib. 1994. Markov decision processes with imprecise transition probabilities. *Oper. Res.* **42** 739–749.

White III, C. C., W. T. Scherer. 1989. Solution procedures for partially observed Markov decision processes. *Oper. Res.* **37** 791–797.

Whitehead, S. 1991. A study of cooperative mechanisms for faster reinforcement learning. Technical Report 365, Computer Science Department, University of Rochester, Rochester, NY.

Whiteson, S., P. Stone. 2006. Evolutionary function approximation for reinforcement learning. *J. Machine Learn. Res.* **7** 877–917.

Williams, R. 1992. Simple statistical gradient following algorithms for connectionist reinforcement learning. *Machine Learn.* **8** 229–256.

Williams, R., L. Baird. 1993. Analysis of some incremental variants of policy iteration: First steps toward understanding actor-critic learning systems. Technical Report NU-CCS-93-11, College of Computer Science, Northeastern University, Boston.

Witten, I. H. 1977. An adaptive optimal controller for discrete time Markov environments. *Inform. Control* **34** 286–295.

Zhang, W., T. Dietterich. 1995. A reinforcement learning approach to job-shop scheduling. *Proc. 14th Internat. Joint Conf. Artificial Intelligence,* Morgan Kaufmann, San Francisco, 1114–1120.