Reinforcement Learning: A Tutorial Survey and Recent Advances

Abhijit Gosavi
Department of Engineering Management and Systems Engineering
219 Engineering Management
Missouri University of Science and Technology
Rolla, MO 65409
Email: gosavia@mst.edu

Abstract

In the last few years, Reinforcement Learning (RL), also called adaptive (or approximate) dynamic programming (ADP), 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, allowing 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.

Keywords: artificial intelligence, dynamic programming, simulation, reinforcement learning.

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 fifties. 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 large-scale and/or complex problems.

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 article 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 Section 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 an MDP, in a subset of states (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 pre-determined 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 online supplement). 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:*

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

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 Eqn.(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 as the trajectory of states is random.

Definition 2 *The average reward of a policy $\hat{d}$ starting at state i is:*

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

where x_s, like 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 2-state MDP in which 2 actions are permitted in each state. The relevant data is supplied in Figure 1. The example illustrates the nature of a *generic* MDP. Theoretically speaking, underlying any

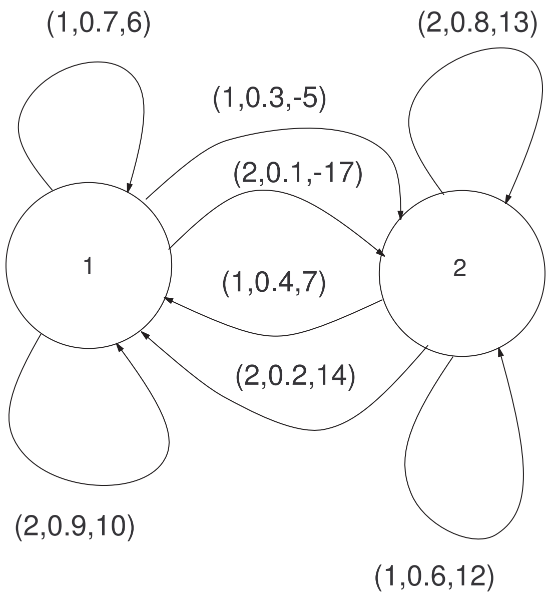

Figure 1: A 2-state MDP with (x, y, z) on each arc denoting the action (x), transition probability (y), and immediate reward (z) associated with the transition.

MDP, there exist data with a structure similar to this 2-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 2-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 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:

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

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: 1 in state 2 and 2 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 out of two actions, 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, while 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 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 (since 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.*

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

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* $\vec{J}_{\hat{d}} \equiv \{J_{\hat{d}}(1), J_{\hat{d}}(2), \ldots, J_{\hat{d}}(|\mathcal{S}|)\}$ *such that the following system of linear equations is satisfied.*

$$J_{\hat{d}}(i) = \left[\bar{r}(i,d(i)) + \gamma \sum_{j=1}^{|\mathcal{S}|} p(i,d(i),j) J_{\hat{d}}(j) \right] \text{ for all } i \in \mathcal{S}. \tag{2}$$

The (unknown) quantities, $J^*(.)$ and $J_{\hat{d}}(.)$, in the Bellman equations are referred to as *value* functions. The value function in Eqn.(1) is called the "optimal" value function, while that in Eqn.(2) is called the "policy" value

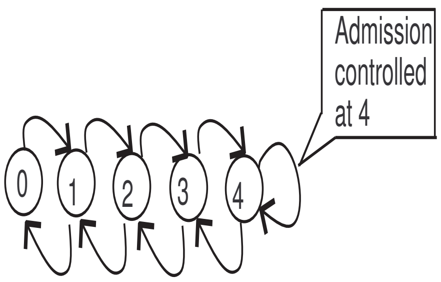

Figure 2: The admissions-control MDP: The figure shows a Markov chain associated with a policy that disallows more than 4 customers in the queue. The number inside the circle denotes the state of the system, i.e., the number of customers in the system.

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 $J^*(.)$ defined in Eqn.(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.*

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

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 $J_{\hat{d}}(.)$ defined in Eqn.(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.*

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

We next derive Q-value versions of the Bellman equations, i.e., Eqns.(1) and (2). Eqns.(1) and (3) imply that for all $i \in \mathcal{S}$:$J^*(i) = \max_{a\in\mathcal{A}(i)} Q(i,a)$.

Using this fact in Eqn.(3), one can obtain the Q-value version of Eqn.(1) as follows:

$$Q(i,a) = \sum_{j=1}^{|\mathcal{S}|} p(i,a,j)\left[r(i,a,j) + \gamma \max_{b\in\mathcal{A}(j)} Q(j,b)\right] \text{ for all } i \in \mathcal{S} \text{ and } a \in \mathcal{A}(i). \tag{5}$$

Similarly, Eqns.(2) and (4) imply that for all $i \in \mathcal{S}$: $J_{\hat{d}}(i) = Q_{\hat{d}}(i, d(i))$, which implies that Eqn.(4) can be written as:

$$Q_{\hat{d}}(i,a) = \sum_{j=1}^{|\mathcal{S}|} p(i,a,j)\left[r(i,a,j) + \gamma Q_{\hat{d}}(j, d(j))\right] \text{ for all } i \in \mathcal{S} \text{ and } a \in \mathcal{A}(i). \tag{6}$$

This is the Q-value version of Eqn.(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., Eqn.(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

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

Step 2: For each $i \in \mathcal{S}$, compute:

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

Step 3: Calculate for each $i \in \mathcal{S}$,

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

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

Figure 3: Steps in VI based on Q-values

iteration. I.e., 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., Eqn.(6). (iii) In Step 3, we set $\hat{d}_{k+1} = \hat{d}_k$, *if possible.* The reason is multiple policies could be optimal, 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.

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

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

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

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

If possible, set $\hat{d}_{k+1} = \hat{d}_k$. See also Remark 4 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

(1992) is: $\mu_m = T_1/\left(1 + \frac{(m-1)^2}{1+T_2}\right)$, with e.g., $T_1 = 0.1$ and $T_2 = 10^6$.

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:

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

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

TD-Learning: Learning with temporal differences (TD) (Sutton, 1984) generalizes beyond the RM algorithm. If W^k denotes the iterate in the kth iteration, the TD update is:

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

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, while 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

Step 1. Initialize all Q-values to 0. That is, for all (l, u), where $l \in \mathcal{S}$ and $u \in \mathcal{A}(l)$, 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 $1/|\mathcal{A}(i)|$. The action that maximizes the Q-value for current state is called a *greedy* action for the current state. (See text 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):

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

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

Figure 6: Steps in Q-Learning based on VI

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 from 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 Eqn.(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 Eqns.(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 visited is haphazard. Thus, a possible trajectory for a 3-state MDP is: $1,2,2,3,2,1,2,3,2,3,3,1,1,1,\ldots$. Clearly, here, when updates are performed for a state, the values used for other states have not been updated the same number of times. Such updates are called *asynchronous* updates. In classical VI, usually, 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.

```
• Initialize all the $Q$-values, $Q(l,u)$ for all $l \in \mathcal{S}$ and $u \in \mathcal{A}(l)$, to arbitrary values.
• Repeat for each episode:
    – Initialize $i$ and select action $a \in \mathcal{A}(i)$ 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 $b \in \mathcal{A}(j)$ via limiting-greedy
          exploration. Then update $Q(i,a)$ using:
              $Q(i,a) \leftarrow (1-\mu)Q(i,a) + \mu\left[r(i,a,j) + \gamma Q(j,b)\right].$    (12)
        * If $j$ is a terminal state, end current episode. Otherwise continue within
          episode.
```

Figure 7: SARSA

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 towards the greedy action, is often used. With an exploratory strategy, in the kth iteration, one selects the greedy action $\arg\max_{u \in \mathcal{A}(i)} Q(i,u)$ with a probability p^k and any one of the remaining actions with probability $\frac{1-p^k}{|\mathcal{A}(i)|-1}$. A possible rule for the probability p^k is: $p^k = 1 - \frac{B}{k}$, where B for instance could equal 0.5. With such a rule, the probability of selecting non-greedy 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: $\alpha Q(i,a) + \frac{c(i)}{\sum_j \tilde{p}(i,a,j)c(j)}$, 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 $\tilde{p}(.,.,)$ denotes the TP, which can be esti-

Step 1. Initialize all J-values and H-values to 0. That is, for all l, where $l \in \mathcal{S}$, and $u \in \mathcal{A}(l)$, 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 $e^{H(i,a)} / \sum_{b \in \mathcal{A}(i)} e^{H(i,b)}$. (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:

$$J(i) \leftarrow (1-\mu)J(i) + \mu \left[r(i,a,j) + \gamma J(j) \right]. \tag{13}$$

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

$$H(i,a) \leftarrow H(i,a) + \beta \left[r(i,a,j) + \gamma J(j) - J(i) \right]. \tag{14}$$

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

Figure 8: Steps in an actor-critic algorithm

mated via model-building (see 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).

Step 1. Initialize all the P-values, $P(l,u)$ for all $l \in \mathcal{S}$ and $u \in \mathcal{A}(l)$, 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 2a. Simulate action $a \in \mathcal{A}(i)$ with probability $1/|\mathcal{A}(i)|$.

Step 2b. 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:

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

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

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

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

Figure 9: Q-P-Learning for discounted-reward MDPs.

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 the learning. This is 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 Eqn. (12) of which 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 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 text-books (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 (Eqns. (13) & 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, 2004b, 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 Eqn. (15) in Figure 9).

3.2 Average reward

RL algorithms for average reward (Schwartz, 1993; Singh, 1994) have been studied for some time. Average 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, and 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 1, the average reward is a more suitable performance metric because discounted VI (or modified PI) with a large value for the discount factor can take very long for convergence (see pgs. 163-164 of Puterman (1994)). 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 $\vec{J^*} \equiv \{J^*(1), J^*(2), \ldots, J^*(|\mathcal{S}|)\}$ and a scalar ρ^*, which denotes the optimal average reward, such that the following system of linear equations is satisfied.*

$$J^*(i) = \max_{a \in \mathcal{A}(i)} \left[\bar{r}(i,a) - \rho^* + \sum_{j=1}^{|\mathcal{S}|} p(i,a,j) J^*(j) \right] \quad \textit{for all } i \in \mathcal{S}. \tag{16}$$

Like its discounted counterpart, an appealing feature of Eqn.(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 $\vec{J_{\hat{d}}} \equiv \{J_{\hat{d}}(1), J_{\hat{d}}(2), \ldots, J_{\hat{d}}(|\mathcal{S}|)\}$ 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.*

$$J_{\hat{d}}(i) = \left[\bar{r}(i,d(i)) - \rho_{\hat{d}} + \sum_{j=1}^{|\mathcal{S}|} p(i,d(i),j) J_{\hat{d}}(j) \right] \quad \textit{for all } i \in \mathcal{S}. \tag{17}$$

Again, like its discounted counterpart, Eqn.(17) can be exploited within a PI scheme to generate optimal solutions. Note, however, that to utilize Eqns.(16) and (17), one has to have access to ρ^* and $\rho_{\hat{d}}$, respectively. Since 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 2b of Q-P-Learning (Figure 9), updating is performed using the following:

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

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 non-linear 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 $(n+1) << |\mathcal{Q}_a|$, 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 non-linear 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 beed 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 sub-grids 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.

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) (pgs. 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, 2004a) that 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 along with T_r and T_t, which 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 4a, which follows Step 4, is used to update ρ. This step (4a) is carried out only if a *greedy* action is selected in Step 2 (see Figure 6 for a definition of greedy actions). Step 4a 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 2b, use the following update:

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

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.

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: $p : \mathcal{S} \times \mathcal{A}^1 \times \mathcal{A}^2 \times \ldots \times \mathcal{A}^n \times \mathcal{S} \rightarrow \Re$. 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:

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

The most challenging and exciting version of the problem is the most general case where each player wishes to optimize his/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 2 players. Consider a strategy-tuple $(\hat{d}^1_*, \hat{d}^2_*)$ such that for all $i \in \mathcal{S}$

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

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).

Hierarchical MDPs: Some MDPs have a special structure that make them amenable to a so-called hierarchical decomposition (Forestier and Varaiya, 1978). With such a decomposition, one can essentially solve MDPs with

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

Table 1: The first entry in any round bracket is the Q-value for the row player and the second for the column player. The Nash equilibrium is $(2, 3)$. The solution $(7, 7)$ will be unstable, though it is mutually beneficial, because both players have incentive to deviate.

smaller state spaces at a lower level which supply solutions to a control-optimization problem at a higher level. In other words, the state space can be effectively divided into disjoint sets where 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 which 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. Since 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, while, 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:

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

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. (1997, 1995), 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 use gradient ascent to optimize the function; the gradient is computed with respect to the probability, which is stored as a pre-defined (linear or non-linear) 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 the output is a probability. In addition to $\vec{w}$, one stores two other vectors, $\vec{\psi}$ and $\vec{\Delta}$, both of size L. Thus, in all, this algorithm requires the storage of only $3L$ scalars. Every element of $\vec{\psi}$ and $\vec{\Delta}$ is initialized to 0. 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: For every $l = 1, 2, \ldots, L$,

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

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

where $\kappa \in (0, 1)$ is fixed at 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).

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, which can be used to come up with a guess of 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 III 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 of being able to develop high-quality solutions to POMDPs. They also explain the notion of "policy-trees" that can be employed to solve POMDPs.

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 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 (section 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.

Non-stationary 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, and 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., are *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.

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 that 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 non-expansive 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 apriori 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 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.

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, 2004a) suitable. Other applications of these algorithms include 'voice-over-packet" networks in Akar and Sahin (2003) (R-SMART) and vendor selection for supply chains in Pontrandolfo et al. (2002) (SMART). 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, 2004b; 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, 2004b) and λ-SMART (Gosavi et al., 2002) were applied. The work related to hyper-heuristics (Burke et al., 2003a,b; Nareyek, 2003) can be used when RL is to be used dynamically to select a meta-heuristic. 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, since 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. In order to familiarize the reader to 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 Section 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., 2005) 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, 2002), and analyzing computational complexity (Evan-Dar and Mansour, 2003).

References

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

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

N. Akar and C. Sahin. Reinforcement learning as a means of dynamic aggregate QoS provisioning. *Lecture Notes in Comp. Science, Springer, Heidelberg*, 2698:100–114, 2003.

A. Arapostathis, V. Borkar, E. Fernandez-Gaucherand, M. Ghosh, and S. Marcus. Discrete-time controlled Markov processes with average cost criterion: A survey. *SIAM Journal of Control and Optimization*, 31(2): 282–344, 1993.

L. C. Baird. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the 12th Int. Conf. on Machine Learning.* 1995.

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

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

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

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

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

D. Bertsekas. *Dynamic Programming and Optimal Control.* Athena, MA, 1995.

D. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming.* Athena, MA, 1996.

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

V. Borkar. Q-learning for risk-sensitive control. *Mathematics of Operations Research*, 27(2):294–311, 2002.

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

V. S. Borkar. Asynchronous stochastic approximation. *SIAM J. Control Optim.*, 36(3):840–851, 1998.

V. S. Borkar and S.P. Meyn. The ODE method for convergence of stochastic approximation and reinforcement learning. *SIAM J. Control Optim*, 38(2): 447–469, 2000.

V.S. Borkar and K. Soumyanath. A new analog parallel scheme for fixed point computation, part I: Theory. *IEEE Transactions on Circuits and Systems I: Theory and Applications*, 44:351–355, 1997.

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

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

J.A. Boyan and A.W. Moore. Generalization in Reinforcement Learning: Safely Approximating the Value Function. *Advances in Neural Information Processing Systems*, pages 369–376, 1995.

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

S.J. Bradtke and M. Duff. Reinforcement learning methods for continuous-time Markov decision problems. In *Advances in Neural Information Processing Systems 7*. MIT Press, Cambridge, MA, USA, 1995.

E. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg. Hyper-heuristics: An emerging direction in modern search technology (chapter 16). In *Handbook of Meta-heuristics*. Kluwer Academic Publishers, 2003a.

E. K. Burke, G. Kendall, and E. Soubeiga. A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics*, 9:451–470, 2003b.

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

D. Chapman and L. Kaelbling. Input generalization in delayed reinforcement learning. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 726–731. Morgan Kaufmann, 1991.

R. Crites and A. Barto. Improving elevator performance using reinforcement learning. In *Proceedings on Advances in NIPS*, pages 1017–1023. MIT Press, 1995.

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

C. Darken, J. Chang, and J. Moody. Learning rate schedules for faster stochastic gradient search. In D.A. White and D.A. Sofge, editors, *Neural Networks for Signal Processing 2 - Proceedings of the 1992 IEEE Workshop*. IEEE Press, Piscataway, NJ, 1992.

T.K. Das, A. Gosavi, S. Mahadevan, and N. Marchalleck. Solving semi-Markov decision problems using average reward reinforcement learning. *Management Science*, 45(4):560–574, 1999.

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

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

D. P. Defarias and B. van Roy. The linear programming approach to approximate dynamic programming. *Operations Research*, 51:850–865, 2003.

T. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13: 227–303, 2000.

E. Evan-Dar and Y. Mansour. Learning rates for Q-learning. *Journal of Machine Learning Research*, 5:1–25, 2003.

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

P. Geibel and F. Wysotzki. Risk-sensitive reinforcement learning applied to control under constraints. *Journal of Artificial Intelligence Research*, 24: 81–108, 2005.

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

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

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

A. Gosavi. Reinforcement learning for long-run average cost. *European Journal of Operational Research*, 155:654–674, 2004a.

A. Gosavi. A reinforcement learning algorithm based on policy iteration for average reward: Empirical results with yield management and convergence analysis. *Machine Learning*, 55(1):5–29, 2004b.

A. Gosavi. Boundedness of iterates in Q-learning. *Systems and Control Letters*, 55:347–349, 2006.

A. Gosavi, N. Bandla, and T. K. Das. A reinforcement learning approach to a single leg airline revenue management problem with multiple fare classes and overbooking. *IIE Transactions*, 34(9):729–742, 2002.

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

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

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

M. Hauskrecht. Value-function approximations for partially observable Markov decision processes. *Journal of Artificial Intelligence Research*, 13: 33–94, 2000.

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

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

J. Hu and M.P. Wellman. Nash Q-Learning for general-sum stochastic games. *Journal of Machine Learning Research*, 4:1039–1069, 2003.

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

L.P. Kaelbling, M.L. Littman, and A.W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

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

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

S. Keerthi and B. Ravindran. A tutorial survey of reinforcement learning. *Sadhana*, 19(6):851–889, 1994.

H. Kimura, M. Yamamura, and S. Kobayashi. Reinforcement learning by stochastic hill climbing on discounted reward. In *Proceedings of the 12th Int. Conf. on Machine Learning*, pages 295–303, 1995.

H. Kimura, K. Miyazaki, and S. Kobayashi. Reinforcement learning in POMDPs with function approximation. In *Proceedings of the 14th Int. Conf. on Machine Learning*, pages 152–160, 1997.

V. Konda and J. Tsitsiklis. Actor-critic algorithms. In *NIPS 12*, pages 1008–1014. MIT Press, 2000.

V.R. Konda and V. S. Borkar. Actor-critic type learning algorithms for Markov decision processes. *SIAM J. Control Optim.*, 38(1):94–123, 1999.

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

Z. Lin, J. Bean, and C. White III. A hybrid genetic/optimization algorithm for finite horizon, partially observed Markov decision process. *INFORMS Journal on Computing*, 16(1):27–38, 2004.

M. Littman. Friend or foe Q-Learning in general-sum games. In *Proceedings of the 18th Int. Conf. on Machine Learning*, 2001.

M. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of 11th Int. Conf. on Machine Learning*, pages 157–163, 1994.

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

D. Luce. *Individual choice behavior.* Wiley, NY, USA, 1959.

S. Mahadevan. Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine Learning*, 22(1):159–95, 1996.

S. Mahadevan and M. Maggioni. Value function approximation using diffusion wavelets and Laplacian eigenfunction. In *NIPS.* MIT Press, 2006.

R. Makar, S. Mahadevan, and M. Ghavamzadeh. Hierarchical multi-agent reinforcement learning. In *Proceedings of the 5th Int. Conf. on Autonomous Agents*, pages 246–253. 2001.

P. Marbach and J. Tsitsiklis. Simulation-based optimization of Markov reward processes: Implementation issues. In *Proceedings of the 38th IEEE Conference on Decision and Control, Phoenix, AZ, USA*, pages 1769–1774, 1999.

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

S. Murthy, S. Kasif, and S. Salzberg. A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 2:1–33, 1994.

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

A. Nareyek. Choosing search heuristics by non-stationary reinforcement learning. In *Meta-heuristics: Computer Decision-making (Edited by M. Resende and J. de Sousa)*, pages 523–544. Kluwer Academic Publishers, 2003.

A. Ng and M. Jordan. PEGASUS: A policy search method for large MDPs and POMDPs. In *Proceedings of 16th Conference on Uncertainty in Artificial Intelligence.* 2000.

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

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

R. Parr and S. Russell. Reinforcement learning with hierarchies of machines. In *NIPS.* MIT Press, 1998.

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

D. L. Pepyne, D.P. Looze, C.G. Cassandras, and T.E. Djaferis. Application of Q-learning to elevator dispatching. Unpublished Report, 1996.

L. Peshkin. *Reinforcement Learning by Policy Search.* PhD thesis, MIT, Cambridge, MA, USA, 2001.

P. Pontrandolfo, A. Gosavi, O.G. Okogbaa, and T.K. Das. Global supply chain management: A reinforcement learning approach. *International Journal of Production Research*, 40(6):1299–1317, 2002.

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

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

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

J. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

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

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

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

J. Satia and R. Lave. Markovian decision processes with uncertain transition probabilities. *Operations Research*, 21:728–740, 1973.

M. Sato, K. Abe, and H. Takeda. Learning control of finite Markov chains with explicit trade-off between estimation and control. In *Connectionist Models, Proceedings of the 1990 Summer School.* Morgan Kaufmann, 1990.

D. Schuurmans and R. Patrascu. Advances in nips 14. MIT Press, 2002.

A. Schwartz. A reinforcement learning method for maximizing undiscounted rewards. *Proceedings of the 10th Annual Conference on Machine Learning*, pages 298–305, 1993.

S. Singh. Reinforcement learning algorithms for average-payoff Markovian decision processes. In *Proceedings of the 12th AAAI.* MIT Press, 1994.

S. Singh and D. Bertsekas. Reinforcement learning for dynamic channel allocation in cellular telephone systems. In *Advances in Neural Information Processing Systems (1996)*, pages 974–980. 1997.

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

S. Singh, V. Tadic, and A. Doucet. A policy-gradient method for semi-Markov decision processes with application to call admission control. Technical Report CUED/F-INFENG/TR 523.Cambridge University, 2005.

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

R. Smallwood and E. Sondik. The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, 21:1071–1088, 1973.

W. Smart and L. Kaelbling. Practical reinforcement learning in continuous spaces. In *Proceedings of the 17th Int. Conf. on Machine Learning*, pages 903–910, 2000.

E. Sondik. The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs. *Operations Research*, 26(2): 282–304, 1978.

R. Sutton. Reinforcement Learning. *Machine Learning (Special Issue)*, 8(3), 1992.

R. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynmaic programming. In *Proceedings of the 7th Int. Conf. on Machine Learning*, pages 216–224. 1990.

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

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

R. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings on Advances in NIPS*, pages 1057–1063. MIT Press, 2000.

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

R.S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *NIPS 8*. MIT Press, Cambridge, MA, 1996.

I. Szita, B. Takacs, and A. Lorincz. ϵ-MDPs: Learning in varying environments. *Journal of Machine Learning Research*, 3:145–174, 2002.

P. Tadepalli and D. Ok. Model-based Average Reward Reinforcement Learning Algorithms. *Artificial Intelligence*, 100:177–224, 1998.

G. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8(3), 1992.

T. Tezcan and A. Gosavi. Optimal buffer allocation in production lines using an automata search. *Proceedings of the Institute of Industrial Engineering Conference, Dallas*, 2001.

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

S. Thrun and K. Moller. Active exploration in dynamic environments. In *Advances in NIPS*. Morgan Kaufman, 1992.

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

J. N. Tsitsiklis and B. van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, 1997.

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

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

B. van Roy, D.P. Bertsekas, Y. Lee, and J.N. Tsitsiklis. A neuro-dynamic programming approach to retailer inventory management. In *Proceedings of the IEEE Conference on Decision and Control*, 1997.

C.J. Watkins. *Learning from Delayed Rewards*. PhD thesis, Kings College, Cambridge, England, May 1989.

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

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

R. M. Wheeler and K. S. Narendra. Decentralized learning in finite Markov chains. *IEEE Transactions on Automatic Control*, 31(6):373–376, 1986.

C. White and H. Eldeib. Markov decision processes with imprecise transition probabilities. *Operations Research*, 43:739–749, 1994.

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

C White III and W Scherer. Solution procedures for partially observed Markov decision processes. *Operations Research*, 37(5):791–797, 1989.

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

S. Whiteson and P. Stone. Evolutionary function approximation for reinforcement learning. *Journal of Machine Learning Research*, 7:877–917, 2006.

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

R. Williams and L. Baird. 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, MA, 1993.

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

W. Zhang and T. Dietterich. A reinforcement learning approach to job-shop scheduling. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*. 1995.

A Online Supplement

A.1 Regular Markov chains:

For a *regular* Markov chain, there exists a finite value for k, where $k > 0$ and k is an integer, such that if $\mathbf{P}$ denotes the one-step TPM of the Markov chain concerned, then every element of $\mathbf{P}^k$ is strictly greater than 0. (See Grinstead and Snell (1997) for an in-depth discussion.)

A.2 Q-value methods for SMDPs:

For continuous-time SMDPs, in which the time of transition has an exponential distribution (Bertsekas, 1995), Bradtke and Duff (1995) presented the following updating rule for the Q-values, which can be used in place of Step 4 in Figure 6:

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

where $t(i,a,j)$ denotes the time spent in the transition from i to j under action a. For the more general case, the following two updating rules were proposed in Gosavi (2003) (pgs 245-247). The first applies to Q-Learning, as described in Figure 6, with Step 4 replaced by:

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

and the second to Q-P-Learning, as described in Figure 9, with Step 2b replaced by:

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

A.3 Model building

The idea of building a model via *straightforward counting* can be formalized as follows. Let $N(i,a)$ denote the number of times action a is tried in state i, and let $N_j(i,a)$ denote the number of times the selection of action a in state i leads the system to state j in the next decision-making epoch. All of these counters are initialized to 0 in the beginning. In a model-based algorithm, one keeps estimates of the value function, $h(i)$, and the expected immediate reward, $\bar{r}(i,a)$, all of which are initialized to 0 in the beginning. When the system transitions to state j after action a is selected in state i, the following updates are carried out in the order given below:

$$N(i,a) \leftarrow N(i,a)+1, \qquad N_j(i,a) \leftarrow N_j(i,a)+1$$

$$\tilde{p}(i,a,l) \leftarrow N_l(i,a)/N(i,a), \text{ for all } l \in \mathcal{S} \text{ and } \bar{r}(i,a) \leftarrow \bar{r}(i,a)+\frac{[r(i,a,j)-\bar{r}(i,a)]}{N(i,a)}.$$

Then, the value function, h, of DP is updated with estimates of the TPs and expected immediate rewards. The main updating rule for discounting from Barto et al. (1995) (see Tadepalli and Ok (1998) for average reward) is:

$$h(i) \leftarrow \max_{u \in \mathcal{A}(i)} \left[\bar{r}(i,u) + \gamma \sum_{l \in \mathcal{S}} \tilde{p}(i,u,l) h(l) \right].$$

A.4 Neural networks

A back-propagation neural network has at least three layers, the input layer, the hidden layer, and an output layer, and associated with each layer is a finite number of nodes. See Figure 10. The unique output node predicts the value of the Q-value for a given state. Usually, a separate net is used for each action. Two classes of weights are used for representation: weights from the input layer to the hidden layer and weights from the hidden layer to the output layer. These weights are gradually adjusted in an *incremental* manner as data become available, i.e., every time a Q-value is updated.

Let $W_1(j,h)$ denote the weights from the jth node in the input layer to the hth node in the hidden layer, and let $W_2(h)$ denote the weight from the hth hidden node to the output node. Let n denote the number of input nodes, H the number of hidden nodes, and let Q_{new} be the *updated* (new) value for the Q-value. All the weights are initialized to very small values before the RL algorithm starts operation. Every time a Q-value is updated by the RL algorithm, the following "sweep" of computations is carried out to update the weights in the neural network (to accommodate the change). Each sweep consists of the following four steps, and the number of sweeps required per iteration of the RL algorithm has to be determined by trial and error with respect to the task at hand.

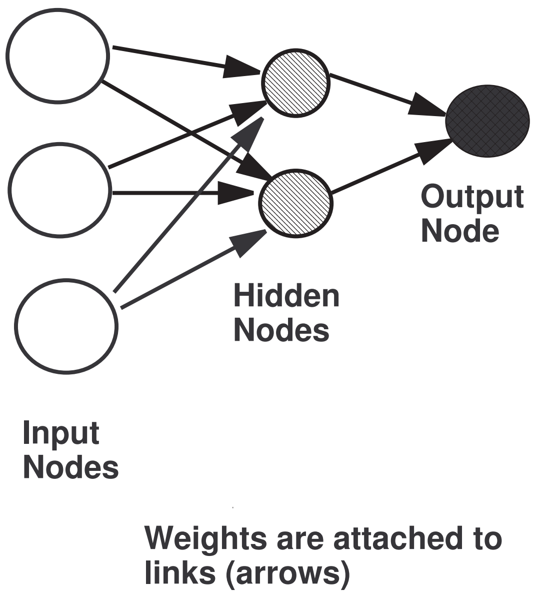

Figure 10: A schematic view of a multi-layer neural network with three input nodes, two hidden nodes, and one output node.

Step 1. Compute $v(h)$ for $h = 1, 2, \ldots, H$, using the following function:

$$v(h) \leftarrow \frac{1}{1 + e^{-v^*(h)}}, \text{ where } v^*(h) \leftarrow \sum_{j=1}^{n} W_1(j,h)x(j),$$

in which $x(j)$ denotes the jth state variable of the Q-value being updated.

Step 2. Now, compute

$$Q_{old} \leftarrow \sum_{h=1}^{H} W_2(h)v(h).$$

Step 3. For each $h = 1, 2, \ldots, H$ and each $j = 1, 2, \ldots, n$, update the weights as follows:

$$W_1(j,h) \leftarrow W_1(j,h) + \mu_{NET}(Q_{new} - Q_{old})W_2(h)v(h)(1 - v(h))x(j),$$

where μ_{NET} is the learning rate of the neural network.

Step 4. For each $h = 1, 2, \ldots, H$, update

$$W_2(h) \leftarrow W_2(h) + \mu_{NET}(Q_{new} - Q_{old})v(h).$$

A.5 Kernel methods:

An example that assigns a distance-based weight to every Q-value would work as follows: Employ for action a, the function $\tilde{w}(i,l,a)$, which, for all given points i and l in the state space, is defined as:

$$\tilde{w}(i,l,a) = D\left(\frac{|i-l|}{\omega_a}\right), \text{ in which}$$

ω_a is a smoothing parameter that has to be tuned with respect to the task at hand, and

$$D(s) = 3/4(1-s^2) \text{ if } |s| \leq 1 \text{ and } D(s) = 0 \text{ otherwise.}$$

The Q-value at any state l and for action a in such a representation is computed as:

$$Q(l,a) = \frac{\sum_i \tilde{w}(i,l,a)Q(i,a)}{\sum_i \tilde{w}(i,l,a)}, \text{ where the summation is over all the exemplar points.}$$