Reinforcement Learning for Long-Run Average Cost

Abhijit Gosavi
Assistant Professor
261, Technology Bldg, 2200 Bonforte Blvd
Colorado State University, Pueblo, Pueblo, CO 81001
Email: gosavi@uscolo.edu, Phone: (719)549-2788

December 6, 2002

Abstract

A large class of sequential decision-making problems under uncertainty can be modeled as Markov and Semi-Markov Decision Problems, when their underlying probability structure has a Markov chain. They may be solved by using classical dynamic programming methods. However, dynamic programming methods suffer from the *curse of dimensionality* and break down rapidly in face of large state spaces. In addition, dynamic programming methods require the exact computation of the so-called transition probabilities, which are often hard to obtain and are hence said to suffer from the *curse of modeling* as well. In recent years, a simulation-based method, called reinforcement learning, has emerged in the literature. It can, to a great extent, alleviate stochastic dynamic programming of its curses by generating near-optimal solutions to problems having large state-spaces and complex transition mechanisms. In this paper, a simulation-based algorithm that solves Markov and Semi-Markov decision problems is presented, along with its convergence analysis. The algorithm involves a step-size based transformation on two time scales. Its convergence analysis is based on a recent result on asynchronous convergence of iterates on two time scales. We present numerical results from the new algorithm on a classical preventive maintenance case study of a reasonable size, where results on the optimal policy are also available. In addition, we present a tutorial that explains the framework of reinforcement learning in the context of semi-Markov decision problems for long-run average cost.

Keywords: Stochastic Processes, Reinforcement Learning, Two time scales.

1 Introduction

Many real-life decision-making problems are found in *stochastic* environments, the uncertainty of which adds to the complexity of their analysis. A subset of these stochastic problems can be formulated as Markov or Semi-Markov Decision Problems. Examples of stochastic systems abound in the service and manufacturing industries. Some examples are as follows: finding optimal inspection policies in a quality control problem [9], optimal control of a queuing problem with two customer types [29], part selection in an FMS [27], optimal maintenance of a production-inventory system [11], a stochastic economic lot-size scheduling problem with two part types [12], and Sennott [28] for some queuing problems.

An MDP (Markov Decision Problem) or an SMDP (Semi-Markov Decision Problem) may be solved by either iteratively solving a linear system of equations developed by Bellman [2] (the method is called policy iteration) or by using the Bellman transformation in an iterative style to compute the optimal *value function* (the method is called value iteration). *Dynamic programming* methods [24] such as value iteration and policy iteration have traditionally been used for solving MDPs and SMDPs. These methods require the exact computation and storage of the so-called transition probability matrices. Many real-life problems have large state-spaces (of the order of millions of states or even larger numbers) where computation or even storage of these matrices is difficult. Dynamic programming (DP) is hence said to have been cursed by the *curse of dimensionality* and the *curse of modeling*. In solving problems with a large state-space, one is forced to resort to problem-specific heuristics to obtain solutions, examples of which are ubiquitous in operations research (see [27] in relation to an FMS problem and [12] in the context of stochastic economic lot-size scheduling problem).

In recent years, a method called Reinforcement Learning (RL) has emerged in the literature. It combines concepts from dynamic programming, stochastic approximation via simulation, and function approximation (which may be performed with the help of regression or neu-

ral networks). Essentially, it employs a step-size version of the Bellman transformation, *within a simulator.* It has been shown to supply optimal or near-optimal solutions to large MDPs/SMDPs even when they are considered intractable via traditional dynamic programming methods. As a result, there is great excitement about RL methods in the operations research community. A noteworthy feature of RL is that it can be very easily integrated into modern simulation packages. Compared to dynamic programming, which involves the tedious (and often very difficult) first step of writing the expressions for the transition probabilities associated with each state-action pair, followed by value iteration or policy iteration, the computational effort and the time spent in implementing an RL model in C or even in a modern simulation package is *extremely* low. *This adds to the attractiveness of this approach.* Textbook treatment on this subject can be found in Sutton and Barto [32] and Bertsekas and Tsitsiklis [4]. Convergence properties of RL algorithms have been studied in existing literature through (i) the use of contraction mappings of transformations in the algorithms [4] and (ii) the use of ordinary differential equations (ODE) that track the transformations (see Abounadi, Bertsekas and Borkar [1], Borkar and Meyn [7] and Konda and Borkar [17]).

This paper makes two contributions to the current literature. First, it presents a new RL algorithm for solving MDPs and SMDPs under the long-run average cost criterion. An analysis of the convergence of the new algorithm, using a method that tracks ODEs (Ordinary Differential Equations) underlying the transformations (required in the algorithm), has also been presented. Second, this paper provides numerical evidence of its convergence on an SMDP of a reasonable size and a complex transition mechanism (obtained from the literature in preventive maintenance).

The remainder of this paper is organized as follows. Section 2 provides a brief overview of the existing literature on the subject of MDPs, SMDPs, and RL. Section 3 contains an account on MDPs, SMDPs, RL, and the underlying framework of dynamic programming. In Section 4, a mathematically concise version of the new RL algorithm is presented. Section

5 deals with the convergence issues related to RL algorithms, the underlying framework of the ODE method and the convergence proof of our new algorithm. Section 6 contains numerical results, obtained from a case study related to decision-making in a preventive maintenance domain. Section 7 discusses other average cost RL algorithms in the literature. The implications of this research and some general conclusions have been placed in Section 8.

2 Overview of the Related Literature

Markov decision problems involve a decision maker, which can influence the behavior of the system as it evolves through time over a finite set of states. Optimal control is exercised by the decision-maker by selecting the optimal action in each state; optimality is measured in terms of a pre-determined performance criterion like total discounted cost (summed over the time horizon) or the long-run average cost (average cost per unit time over the entire time horizon). Outside linear programming, two methods for solving MDPs and SMDPs are: policy iteration and value iteration. We shall focus on the latter under the long-run average cost criterion. Textbooks written by Bertsekas [3] and Puterman [24] discuss this topic in great detail. Some of the pioneers of RL are Sutton [31] and Watkins [35]. For a detailed account on early average cost RL (ARL) algorithms, the reader is referred to Mahadevan [21]. A number of algorithms, such as R-Learning [26] , [33] and [30], exist for solving average cost MDPs, but they do not appear to have convergence proofs. SMART (Semi-Markov Average Reward Technique) [10], which is designed for SMDPs, also does not have a convergence proof. The first convergent ARL algorithm for MDPs appeared in [1]. The first result in using an ODE method to analyze stochastic approximation schemes, such as the ones encountered in RL, is due to Ljung [20]. An alternative analysis [20] was presented in Kushner and Clark [18]. Some of the first applications of the ODE framework

in RL have appeared in [1], [17], [7].

In dynamic programming all variables (for all the different states) are updated in every iteration simultaneously, i.e., *synchronously*. However, in RL, the system is simulated and the order in which variables are updated depends on the order in which variables (states) are visited in the simulator. The haphazard order of visiting states does not ensure that the variables associated with the states are updated *once* in *every* cycle as in a synchronous update. This is called *asynchronous* updating. Using the ODE (Ordinary Differential Equation) method, Borkar [5] analyzed stochastic approximation operating in a synchronous style, on a two-time scale, in a very general context. These results were subsequently extended to the asynchronous case (see Borkar [6], Borkar and Meyn [7], Abounadi *et al.* [1]).

3 Underlying Framework of Markov Decision Theory

This section gives an outline of the framework underlying Markov decision problems and Semi-Markov decision problems. It concludes with an analysis of how the framework of RL is rooted in that of dynamic programming.

3.1 MDPs and SMDPs

Formally an MDP can be defined as follows. Let

$$\mathbf{X} = \{X_n : n \in \mathcal{N}, X_n \in S\} \tag{1}$$

denote the underlying Markov chain of an MDP, where X_n denotes the system state at the nth decision making epoch, S denotes the state space, and $\mathcal{N}$ denotes the set of integers. At any decision making epoch n, where $X_n = i \in S$, the action taken is denoted by $A_n = a \in A(i)$. $A(i)$ denotes the set of possible actions in state i and $\cup_{A(i)} = A$. Associated with any action

$a \in A$ is a transition matrix $P(a) = \{p(i,a,j)\}$ of the Markov chain $\mathbf{X}$, where $p(i,a,j)$ represents the probability of moving from state i to state j under action a in one step. A cost function is defined as $r : S \times A \rightarrow \Re$, where $\Re$ denotes the real line, and $r(i,a)$ is the expected cost for taking action a in state i. It is assumed that the state space of the Markov chain considered in this paper is finite. Also, for the sake of simplicity, only those problems which have aperiodic and unichain Markov chains are considered. Dynamic programming methods can find a stationary deterministic policy π^* (which is a mapping $\pi^* : S \rightarrow A$), that is optimal with respect to the average cost criterion. (A stationary policy is one that is independent of time.) Under the assumption of a finite state-space, the average cost can be shown to be finite.

When sojourn times in an MDP are drawn from general probability distributions, the problem can often be modeled as a Semi-Markov Decision Problem (SMDP). Every SMDP is characterized by two processes namely the *natural process* and the *decision process*. Suppose, for each $n \in \mathcal{N}$, a random variable X_n assumes values from a countable set S and a random variable T_n takes values in $\Re^+ = [0, \infty]$, such that $0 = T_0 \leq T_1 \leq T_2 \cdots$. The decision process $(\mathbf{X}, \mathbf{T}) = \{X_n, T_n : n \in \mathcal{N}\}$ is said to be Semi-Markovian with state space S, because for all $n \in \mathcal{N}$, $j \in S$, and $t \in \Re^+$, the following Semi-Markov condition is satisfied:

$$P\{X_{n+1} = j, T_{n+1} - T_n \leq t | X_0, \cdots, X_n, T_0, \cdots, T_n\} = P\{X_{n+1} = j, T_{n+1} - T_n \leq t | X_n, T_n\}, \tag{2}$$

where the notation $P\{.|.\}$ denotes a conditional probability in which the values of the variables in the condition are assumed to be known. The natural process $\mathbf{Y} = \{Y_t : t \in \Re^+\}$, where Y_t denotes the system state at time t and where $Y_t = X_n$ if $t = T_n$, can be viewed as a continuous time process that contains information about the system state at all times including the decision-making instants. The natural process $\mathbf{Y}$ and the decision process (X, T) agree at the decision-making instants but may not agree at other times. Let, for $i \in S$ and $a \in A_i$, $g(i,a,j)$ denote the immediate cost (or one-step transition cost) incurred when i

is the decision-making state, in which an action a is chosen and the next decision-making state is j. Let $t(i, a, j)$ denote the time spent in the state transition. Also let i_k represent the state visited in the kth decision-making epoch and a_k represent the action taken in that epoch. Then the maximum average cost of an SMDP under a policy π (that dictates the path followed by the Markov chain) for an infinite time period (T), which is to be minimized, can be given as

$$\rho^{\pi(i)} = \lim_{T\to\infty} \sup \frac{\sum_{k=1}^{T} g(i_k, a_k, i_{k+1})}{\sum_{k=1}^{T} t(i_k, a_k, i_{k+1})}. \tag{3}$$

We next present the Bellman equation, which forms the foundation stone of stochastic dynamic programming and so also of RL.

3.1.1 The Bellman Optimality Equation for Average Cost MDPs and SMDPs

Theorem 1: Under considerations of average cost for an infinite time horizon for any finite unichain SMDP, there exists a scalar ρ^* and a value function R^* satisfying the system of equations for all $i \in S$,

$$R^*(i) = \min_{a\in A_i} \left(r(i, a) - \rho^* y(i, a) + \sum_{j\in S} p(i, a, j) R^*(j) \right),$$

such that the greedy policy π^* formed by selecting actions that minimize the right-hand side of the above equation is average cost optimal*, where $r(i, a)$ is the expected immediate cost when an action a is taken in state i, $y(i, a)$ is the expected sojourn time in state i when action a is taken in it, and $p(i, a, j)$ is the probability of transition from state i to state j under action a in one step.*

For the MDP case, the Bellman equation can be obtained by setting $y(i, a) = 1$ for values of (i, a). We next present the value iteration algorithm that uses the Bellman equation to generate an optimal policy in case of MDPs.

3.1.2 Value Iteration for MDPs

Let $R^m(i)$ be the total expected value of evolving through m stages starting at state i, and ψ be the space of bounded real valued functions on S.

1. Select $R^0 \in \psi$, specify $\epsilon > 0$ and set $m = 0$.

2. For each $i \in S$, compute $R^{m+1}(i)$ by

$$R^{m+1}(i) = \min_{a \in A_i} \left\{ r(i,a) + \sum_{j \in S} p(i,a,j) R^m(j) \right\}. \tag{4}$$

3. If $sp(R^{m+1} - R^m) < \epsilon$, go to step 4. Else increment n by 1 and return to step 2. Here *sp* denotes *span*, which for a vector b is defined as $sp(b) = \max_{i \in S} b(i) - \min_{i \in S} b(i)$ (Puterman [24]).

4. For each $i \in S$, choose the action $d_\epsilon(i)$ as

$$d_\epsilon(i) \in \text{argmin}_{a \in A_i} \left\{ r(i,a) + \sum_{j \in S} p(i,a,j) R^m(j) \right\} \tag{5}$$

and stop.

Value iteration has been shown to be convergent. However it is numerically unstable (see Puterman [24]) and hence in practice a *relative* value iteration method is used, which we describe next.

3.1.3 Relative Value Iteration for MDPs

1. Select $R^0 \in \psi$, choose $k^* \in S$ arbitrarily, specify $\epsilon > 0$, and set $m = 0$.

2. For each $i \in S$, compute $R^{m+1}(i)$ by

$$R^{m+1}(i) = \min_{a \in A_i} \left\{ r(i,a) - R^m(k^*) + \sum_{j \in S} p(i,a,j) R^m(j) \right\}. \tag{6}$$

3. If $sp(R^{m+1} - R^m) < \epsilon$, go to step 4. Otherwise increment m by 1 and return to step 2.

4. For each $i \in S$, choose the action $d_\epsilon(i)$ as

$$d_\epsilon(i) \in \text{argmin}_{a \in A_i} \left\{ r(i,a) + \sum_{j \in S} p(i,a,j) R^m(j) \right\} \tag{7}$$

and stop.

It is clear that this algorithm differs from value iteration in subtracting the value function of some state ($R(k^*)$)in each update. The reason this algorithm converges to the same solution as value iteration is: essentially the vectors (of value function in each state) generated by the two methods differ from each other by a constant ($R(k^*)$), but their relative values are same and so is the sequence of minimizing actions in each case. Hence one may subtract any value in place of $R(k^*)$and still get an optimal algorithm. This is why one can also subtract the optimal average cost (ρ^*). In fact, this would give us a transformation identical to the Bellman transformation. We shall discuss this in more detail in the context of our algorithm. But let us now briefly discuss how such algorithms may be implemented in a simulator, which is the central idea under RL.

3.2 RL

RL is a simulation-based method where the optimal value function is approximated using simulation. We shall first present an intuitive explanation of value-iteration based RL, which is also called Q-Learning. Thereafter we shall discuss the value-iteration roots of Q-Learning.

3.2.1 The Working Mechanism of Q-Learning

An RL model consists of four elements (see Figure 1), which are namely an environment, a learning agent with its knowledge base, a set of actions that the agent can choose from and

the response from the environment to the different actions in different states. The knowledge base is made up of the so-called Q-factors for each *state-action* pair. After examining the numerical values (which may be in terms of costs or rewards) of these Q-factors (we shall define them shortly) for the *different actions* that may be taken in the *current state*, the agent decides which action to take in that state. To every action, the possible response is simulated. To be able to do this, complete knowledge of the random variables that govern the system dynamics is necessary to the simulator.

In what follows, we describe the mode in which "learning" (stochastic approximation) occurs. Before learning begins, the values $Q(i, u)$ for all states i and all actions u are set to the same value. When the system is in a decision-making state i, the learning agent examines the Q-factors for all actions in state i and selects the action b with the minimum Q-factor (if values are in terms of cost). This leads the system along a path till the system encounters another decision-making state (j). During the travel over this path, i.e. a state-transition from i to j, which is simulated in the simulator, the agent gathers information from the environment about the immediate costs incurred and the time spent during the state-change. This information is used by the agent with the help of its learning algorithm to update the factor $Q(i, b)$. A poor action results in an increase of this value while a good action that results in low cost causes the value to be decreased (and *vice-versa* if the value is in terms of rewards). Of course the exact change is determined by the algorithm which is developed from the Bellman equation. In other words, the performance of an action in a state serves as an experience to the agent which is used to update it knowledge base. Thus every piece of experience makes the agent a trifle smarter in its perception of the environment than it was before. As all state-action pairs are encountered (theoretically an infinite number of times), the agent learns the optimal actions in each state.

When the system visits a state, the agent selects the action with the lowest Q-factor (or the highest if it is in terms of rewards) for that state-action pair. Therefore in the event of

that action producing a low immediate cost, the updating mechanism causes the agent to be partial to that action. Similarly, the opposite is possible when the action produces a high immediate cost. This can cause considerable trouble in learning the right policy because the short-term effect of an action can be misleading. Consequently it becomes necessary to try all actions in all states. Therefore occasionally the agent has to divert from its policy of selecting the most preferred action (greedy strategy) and instead select some other action. This is called *exploration*. As good actions are rewarded and bad actions punished over time, some actions tend to be more and more preferable and some less. The learning phase ends when a clear trend appears in the knowledge base and an optimal or near-optimal policy is learned. Of course, by this point, exploration must cease. We next explain how the updating equation is based on the Bellman equation.

3.3 Dynamic Programming Roots of RL

Q-Learning essentially solves the Bellman equation iteratively in an asynchronous style to obtain the optimal value function. In what follows, we shall present only the transformations related to a version of Q-learning based on average cost value iteration for MDPs (discussed in Section 3.1.2). The idea is to show its Bellman equation links.

The optimal Q-factor, $Q(i,u)$ for a pair (i,u) with $i \in S$ and $u \in A(i)$ may defined as follows in the context of the average cost MDP:

$$Q(i,u) = \sum_{j \in S} p(i,u,j)[g(i,u,j) + R^*(j)], \tag{8}$$

where $g(i,u,j)$ is the immediate cost incurred in transition from state i to state j under the action u. Now the Bellman equation for average cost that is used in value iteration is as follows:

$$R^*(i) = \min_{u \in A(i)} \sum_{j \in S} p(i,u,j)[g(i,u,j) + R^*(j)] \ \forall i. \tag{9}$$

Using equation (8) and equation (9), we have that

$$R^*(i) = \min_{u \in A(i)} Q(i,u) \ \forall i. \tag{10}$$

From equations (8) and (10), it follows that

$$Q(i,u) = \sum_{j \in S} p(i,u,j)[g(i,u,j) + \min_{v \in A(j)} Q(j,v)] \ \forall (i,u). \tag{11}$$

RL requires the updating to be gradual, for at least two reasons. One reason is that RL is necessarily asynchronous and can diverge without proper learning rates [6]. The second reason is that avoiding the computation of the transition probabilities is a major goal of RL, which too, we shall see shortly, requires a learning rate. The next equation gives a step-size or learning rate version of equation (11). For example for a step-size γ, where $\gamma \in (0,1]$ the updating may be given as follows:

$$Q(i,u) \leftarrow (1-\gamma)Q(i,u) + \gamma \sum_{j \in S} p(i,u,j)[g(i,u,j) + \min_{v \in A(j)} Q(j,v)] \forall (i,u). \tag{12}$$

Equation (12) gives the updating equation for a model-based Q-Learning algorithm. The Robbins-Monro [25] stochastic approximation scheme (i.e. updating with a learning rate) is used to approximate the mean of a random variable with methods such as simulation. It can be expressed as follows: $Q \leftarrow (1-\gamma)Q + \gamma[f(Q)]$, (notice the similarity with equation (12)) where the term in the square brackets is a function of the Q vector. It can be shown that a suitable learning rate ($1/k$ if k denotes the number of iterations) produces an averaging effect in the sense that if it is used over a sufficiently long updating period, we obtain on the left hand side an average of $f(Q)$ (in case $f(Q)$ is stochastic). Since the term $f(Q)$ is averaged in any case, one can replace the expectation (over j) in equation (12) by a sample. And this gives us a Robbins-Monro version of Q Learning, which can be written as follows:

$$Q(i,u) \leftarrow (1-\gamma)Q(i,u) + \gamma[g(i,u,j) + \min_{v \in A_j} Q(j,v)]. \tag{13}$$

This is the model-free (free of transition probabilities) Q-Learning analogue of average cost value iteration (the original Q-Learning was presented in the context of discounted cost by

Watkins [35]). However, this version is based on the average cost value iteration algorithm (just as Watkins' Q-Learning was based on discounted cost value iteration) and average cost value iteration is numerically unstable (see both Bertsekas [3] and Puterman [24] for a clear explanation of this phenomenon). For a numerically stable version of Q-Learning, one has to use a relative value iteration method, which may have to run on two time scales in the simulator.

The motivation for the above discussion was that *many* average cost value iteration RL algorithms may be similarly connected to DP. It must be emphasized here that RL algorithms must converge *asynchronously*, to be of any use. This is because in RL, the updating is performed in a simulator. In DP, the updating can be synchronous (where all states are updated simultaneously in any given iteration), which considerably simplifies its convergence analysis. We should also mention at this point that these Q-factors can be suitably stored using neural networks or regression, which gives RL the power to be able to handle large state-spaces. Using a neural network, for example, one may be able to store a million Q-factors using a hundred weights (values). In the next section, we present a new RL algorithm for SMDPs for optimizing average cost, over an infinite time horizon.

4 The New Algorithm

The new algorithm has its roots in a relative value iteration method in which the average cost forms the subtraction term. The average cost is approximated on one time scale and the Q-factor on the other. In this section, we shall discuss the core of the new algorithm leaving the implementational details to Section 8. We first introduce some notation that we shall need.

Let Q^k be the vector of Q-values at the kth iteration. Let $\{e^k\}$ be the sequence of states visited in the simulation till the kth epoch. Now $\{e^k\}$ can also be viewed as a sequence of

vector-valued random variables over S (the set of decision-making states), subject to the discrete probability distribution, $Probability(e^k_{iu} = j) = p(i, u, j)$ where $p(i, u, j)$ is the one-step transition probability of going from state i to state j under action u. The notation e^k_{iu} denotes the the state in the $(k+1)$th epoch if the state in the kth epoch was i and the action taken was u. Two learning rates on two time scales will be used and they are denoted by α and β respectively. The learning rate α at any decision-epoch for each state-action pair depends on the function $m(.)$ which is the number of times the state-action pair was tried till that decision epoch. And β depends on the number of decision epochs. In what follows, first a version of the algorithm suitable for use with MDPs is presented. Thereafter the SMDP extension is discussed.

4.1 MDP Version

The core of the new algorithm for MDPs may be given by the following two updating equations: For all $(i, u) \in (S \times A)$,

$$Q^{k+1}(i,u) = Q^k(i,u) + \alpha(m(k,i,u))[g(i,u,e^k_{iu}) + \min_v Q^k(e^k_{iu}, v) - \rho^k - Q^k(i,u)]I((i,u) = \phi^k), \tag{14}$$

$$\rho^{k+1} = (1 - \beta(k))\rho^k + \beta(k)\frac{[(J(k)\rho^k) + g(i,u,e^k_{iu})]}{J(k+1)}, \tag{15}$$

where $J(k)$ is the number of state-transitions up to the kth iteration (for MDPs $J(k) = k$), $\phi = \{\phi^1, \phi^2, \ldots\}$ is the sequence of state-action pairs tried in the learning process, and ϕ^k is the state-action pair tried in the kth epoch. Now equation (15) has a Robbins-Monro version:

$$\rho^{k+1} = (1 - \beta(k))\rho^k + \beta(k)g(i,u,e^k_{iu}). \tag{16}$$

The Robbins-Monro version of the new algorithm for MDPs would use equations (14) and (16). The equivalence of both forms follows from the validity of the Robbins-Monro stochastic approximation scheme.

4.2 SMDP Version

The new Q-Learning algorithm for SMDPs may be written with the following two equations: For all $(i, u) \in (S \times A)$,

$$Q^{k+1}(i,u) = Q^k(i,u) + \alpha(m(k,i,u))[g(i,u,e^k_{iu}) + \min_v Q^k(e^k_{iu}, v) - \rho^k t(i,u,e^k_{iu}) - Q^k(i,u)]I((i,u) = \phi^k) \quad (17)$$

$$\rho^{k+1} = (1 - \beta(k))\rho^k + \beta(k)\frac{[(T(k)\rho^k) + g(i,u,e^k_{iu})]}{T(k+1)} \quad (18)$$

where $T(k)$ denotes the sum of the time spent in all states visited till the kth iteration.

The SMDP version of the algorithm given by equations (17) and (18) has a Robbins-Monro version. It also makes use of the renewal reward theorem. The term ρ^k in equation (17) would have to be replaced by C^k/T^k and instead of equation (18) the following equations would be used to update the terms C^k and T^k.

$$C^{k+1} = (1 - \beta(k))C^k + \beta(k)g(i,u,e^k_{iu}), \quad (19)$$

and

$$T^{k+1} = (1 - \beta(k))T^k + \beta(k)t(i,u,e^k_{iu}). \quad (20)$$

The next section deals with the convergence analysis of the new algorithm.

5 The ODE Framework and Stochastic Approximation

We next discuss the framework of ODEs in relation to stochastic approximation schemes. If one is interested in solving a system of equations of the form

$$F(r) = r, \quad (21)$$

where F is a function from $\mathcal{R}^n$ into itself and r is a vector, one possible algorithm to solve this system is

$$r \leftarrow (1-\gamma)r + \gamma F(r), \tag{22}$$

where γ is a step-size or learning rate, usually chosen to be smaller than 1. Sometimes the form of $F(r)$ is not known completely. The example of interest to us here is that of the Bellman equation without its transition probabilities. An estimate of $F(r)$ can however be obtained through simulation. In other words, it is possible to gain access to a noisy random variable s, where $s = F(r) + w$ and w is a random noise. Both simulation and real experiments usually have such a noise associated with their outcomes. Therefore it is reasonable to use s in place of $F(r)$ and the resulting algorithm would then become

$$r \leftarrow (1-\gamma)r + \gamma(F(r) + w). \tag{23}$$

The algorithm is called a *stochastic approximation* (SA) algorithm. The Robbins-Monro stochastic approximation method serves as the foundation of model-free RL algorithms (which do not use transition probabilities). Suppose that $F(r) = E[g(r)]$ (E here stands for expectation while $g(r)$ is a random variable). One way to estimate $F(r)$ then would be to obtain several samples $g_1(r), g_2(r), ..., g_k(r)$ and then use

$$F(r) = \frac{\sum_{i=1}^{k} g_i(r)}{k}. \tag{24}$$

As k becomes large, this sample mean on the right hand side of the equation (24) starts converging to the true mean $E[g(r)]$. At the other extreme, if $k = 1$ then the estimate of $F(r)$ is based on a single sample. The stochastic approximation algorithm then becomes:

$$r \leftarrow (1-\gamma)r + \gamma g(r). \tag{25}$$

This algorithm is called the Robbins-Monro algorithm. It is okay to use $k = 1$, since (under certain restriction on γ) it has been shown that r anyway tends to $E[g(r)]$ due to the averaging nature of the SA scheme. The SA scheme given in (22) may also be written as

follows.

$$r_{k+1} = r_k + \gamma(F(r_k) - r_k). \tag{26}$$

This may be rewritten as :

$$\frac{r_{k+1} - r_k}{\gamma} = F(r_k) - r_k, \tag{27}$$

which suggests a continuous time version

$$\frac{dr(t)}{dt} = F(r(t)) - r(t), \tag{28}$$

using an ODE.

Consider a situation where stochastic approximations are taking place on two different time scales in the same simulation environment. Let $I(.)$ be an identity function which takes the value of 1 when the condition within the round brackets (.) is satisfied and 0 when it is not. Let $\{x^k\}$ and $\{y^k\}$ be sequences in $\Re^n$ and $\Re^l$ respectively, generated according to the following schemes on two different time scales.

For $i = 1, ..., n$ and $j = 1, ..., l$:

$$x_i^{k+1} = x_i^k + a(q(k,i))(h_i(x^k, y^k) + w_{1_i}^k)I(i = \phi_1^k), \tag{29}$$

$$y_j^{k+1} = y_j^k + b(s(k,j))(f_j(x^k, y^k) + w_{2_j}^k)I(j = \phi_2^k), \tag{30}$$

where $\{\phi_1^k\}$ and $\{\phi_2^k\}$ are random processes that take values over the sets $S_1 = \{1, 2, ..., n\}$ and $S_2 = \{1, 2, ..., l\}$ respectively , $h(.,.)$ and $f(.,.)$ are arbitrary functions on (x^k, y^k), w^k denotes the noise term and

$$q(k,i) = \sum_{m=0}^{k} I(i = \phi_1^m), \tag{31}$$

$$s(k,j) = \sum_{m=0}^{k} I(j = \phi_2^m). \tag{32}$$

In context of ARL, x^k and y^k represent the action values and the average cost respectively at the kth iteration. So n equals the number of state-action pairs and l would equal 1. Hence at every state change, the second sequence registers $j = 1$. Now ϕ_1^k denotes the

state-action pair visited at the kth state change and so depending on the state-action pair visited, the respective action value will undergo a change; others will remain unchanged. Step-sizes a and b depend on q and s respectively. (See Remark 4 in Section 6.2 on how such a dependence may be ensured roughly in RL.) Now $q(k,i)$ for a given state-action pair represents the number of times the state i has been visited till the kth iteration of the simulation and $s(k,j)$ denotes the number of times state-changes have taken place till the kth iteration in the simulation. (It may be noted that since $l = 1$, $s(k,j)$ is always $s(k,1)$ in the ARL context.) The mathematical definitions of the sequences given above help to formalize the dynamics of the stochastic approximation schemes underlying average cost RL algorithms. In addition, the following set of assumptions are also needed.

5.1 Assumptions

1. The functions h and f are Lipschitz continuous (see Appendix for a definition of Lipschitz continuity).

2. There exist $A > 0$ and $B > 0$ such that for all $i = 1, 2, .., n$ and $j = 1, 2, .., l$ with probability 1

$$\lim_{k\to\infty} \inf \frac{q(k,i)}{k+1} \geq A, \tag{33}$$

and

$$\lim_{k\to\infty} \inf \frac{s(k,j)}{k+1} \geq B. \tag{34}$$

3. The step-sizes a and b satisfy the standard step-size conditions given in the Appendix.

In addition they must satisfy the following condition:

$$\lim_{k\to\infty} \sup \frac{b(k)}{a(k)} = 0. \tag{35}$$

4. For some constants $A_1, B_1, D_1, A_2, B_2, D_2$ the following condition is satisfied :

$$E[w_1^k|(x^k, ..., x^0, y^k, ..., y^0, w^{k-1}, ..., w^0)] = 0, \tag{36}$$

$$E[||w_1^k||^2|(x^k, ..., x^0, y^k, ..., y^0, w^{k-1}, ..., w^0)] \leq A_1 + B_1||x^k||^2 + D_1||y^k||^2, \tag{37}$$

$$E[w_2^k|(x^k, ..., x^0, y^k, ..., y^0, w^{k-1}, ..., w^0)] = 0, \tag{38}$$

and

$$E[||w_2^k||^2|(x^k, ..., x^0, y^k, ..., y^0, w^{k-1}, ..., w^0)] \leq A_2 + B_2||x^k||^2 + D_2||y^k||^2. \tag{39}$$

5. For all $y \in \Re^l$, the ODE

$$\frac{dx}{dt} = h(x(t), y) \tag{40}$$

has an asymptotically stable critical point $G(y)$ (see Appendix for a definition) such that the map G is Lipschitz continuous.

6. The ODE

$$\frac{dy}{dt} = f(G(y(t)), y(t)) \tag{41}$$

has a global asymptotically stable critical point y^*.

Here are the intuitive meanings of some of these assumptions. The second assumption says that each state-action pair is visited after a finite time interval. The third assumption is very crucial for these schemes to work. It says that the second iteration is much slower than the first because of its smaller step-size. The third assumption implies that the fast iteration in x sees the slower iteration in y as a constant and hence converges, while the slower iteration sees the faster iteration as having converged. The limiting behavior of the slower iteration is given by the ODE in assumption 6 while that of the faster one is given by that in assumption 5. As a result, $\{y^k\}$ should hence converge to y^*, x^k should converge to $f(y^*)$ and together the sequences should converge to $(f(y^*), y^*)$. In fact, this is a result due to Borkar [5] which we state next.

5.1.1 Borkar's Lemma

Consider the coupled sequence $\{(x^k, y^k)\}$ *generated as in equations (29 and 30) under the assumptions stated in Section (5.1). If the iterates are bounded, then* $\{(x^k, y^k)\}$ *converges to* $(f(y^*), y^*)$ *with probability one.*

This result will be used to prove the convergence of the new algorithm.

6 Convergence Analysis of the New Algorithm

We now come to our main result.

Theorem 2: The algorithms presented in Sections 4.1 and 4.2 converge to near optimal solutions under the assumptions made in Section 5.1 and under the following additional assumptions:

1. *The iterates* Q *and* ρ *are bounded.*

2. *A unique learning rate* α *is associated with each* Q *factor which is decayed in a fashion that depends upon the number of times the particular* Q*-factor was tried. There is also a unique learning rate* β *associated with the scalar* ρ *that depends on the number of times* ρ *is updated.*

3. *There exists a state* s *in the Markov chain such that for some integer* m*, and for all initial states and all stationary policies,* s *is visited with a positive probability at least once within the first* m *stages.*

6.1 Proof

Convergence follows from Borkar's lemma (in Section 5.1.1) with some additional work. First, we need to show that the transformations in the algorithm are of the form presented in Borkar's lemma and that they track ODEs.

6.1.1 Relevant ODEs

The iterations in the algorithm can be shown to relate to a set of mappings which are defined next. Let H_1 denote a mapping that transforms the vector (Q^k) to a new vector $(H_1(Q^k)$ as follows.

$$(H_1(Q^k))(i,u) = \sum_j p(i,u,j)[g(i,u,j) - \rho^k t(i,u,j) + \min_v Q^k(j,v)]. \tag{42}$$

Now a vector $(H_2(Q^k)$ can be defined as follows.

$$(H_2(Q^k))(i,u) = [g(i,u,e^k_{iu}) - \rho^k t(i,u,e^k_{iu}) + \min_v Q^k(e^k_{iu},v)]. \tag{43}$$

We defines mappings F_1 and F_2 for SMDPs as follows:

$$F_1(\rho^k) = \sum_j p(i,u,j)[g(i,u,j) + T(k)\rho^k]/T(k+1),$$

$$F_2(\rho^k) = [g(i,u,e^k_{iu}) + T(k)\rho^k]/T(k+1).$$

Using these transformations, the algorithm for SMDPs (17 and 18) may be re-written using the following equations:

$$Q^{k+1}(i,u) = Q^k(i,u) + \alpha(m(k,i,u))[H_1(Q^k)(i,u) - Q^k(i,u) + w^k_1(i,u)]I((i,u) = \phi^k) \tag{44}$$

and

$$\rho^{k+1} = \rho^k + \beta(k)[F_1(\rho^k) - \rho^k + w^k_2], \tag{45}$$

where the noise term w^k_1 is given by:

$$w^k_1 = H_2(Q^k) - H_1(Q^k), \tag{46}$$

and the noise term w_2^k is given by:

$$w_2^k = F_2(\rho^k) - F_1(\rho^k). \tag{47}$$

Now the equation (44) can be written as:

$$Q^{k+1}(i,u) = Q^k(i,u) + \alpha(m(k,i,u))[h(Q^k(i,u),\rho^k) + w^k]I((i,u) = \phi^k), \tag{48}$$

where $h(Q^k) = H_1(Q^k) - Q^k$. Also equation (45) can be written as:

$$\rho^{k+1} = \rho^k + \beta(k)[f(\rho^k) + w_2^k], \tag{49}$$

where $f(\rho^k) = F_1(\rho^k) - \rho^k$. Note that the equations (48) and (49) for SMDPs form a special case of the general class of algorithms (29) and (30) analyzed using the lemma (5.1.1). Then the corresponding limiting ODEs for sequences defined in equations (48) and (49) would be respectively

$$\frac{dQ(t)}{dt} = h(Q(t),\rho), \tag{50}$$

and

$$\frac{d\rho}{dt} = f(\rho(t)). \tag{51}$$

The analysis for MDPs to obtain the limiting differential equations (50) and (51) follows from the analysis of SMDPs by setting the terms $t(i,u,j)$ and $t(i,u,e_{iu})$ to 1 in the mappings H_1 and H_2 and replacing $T(k)$ by $J(k)$ in the mappings F_1 and F_2. We next show that the iterates Q and ρ are bounded.

6.1.2 Boundedness of Iterates

In the context of boundedness, it is necessary to define one more transformation which is not used in the algorithm directly. The transformation is denoted by T and is defined as:

$$(TQ)(i,u) = \sum_{j=1}^{n} p(i,u,j)[g(i,u,j) + \min_v Q(j,v)].$$

Now T is contractive with respect to some weighted sup-norm under Assumption 3 of Theorem 2 (see Remark 1 in Section 6.2). Then it can be shown [4] that there exists a vector r, and scalars $\delta \in (0,1)$ and $D > 0$, such that

$$||TQ||_r \leq \delta ||Q||_r + D,$$

where $||s||_r$ denotes the weighted sup-norm of a vector s with respect to the vector r and is defined as follows:

$$||s||_r = \max_{i=1,\ldots,n} \frac{|s(i)|}{r(i)}.$$

Note that

$$(H_1 Q)(i,u) = (TQ)(i,u) - \rho \sum_j p(i,u,j)t(i,u,j),$$

which implies that:

$$|(H_1 Q)(i,u)| \leq |(TQ)(i,u)| + |\rho \sum_j p(i,u,j)t(i,u,j)|,$$

Also it is clear that ρ is finite (it is the average cost of a stationary policy of a Markov chain with finite costs and finite number of states) and so is $\sum_j p(i,u,j)t(i,u,j)$ for any value of (i,u). This ensures that for $D_1 > 0$,

$$||H_1 Q||_r \leq ||TQ||_r + D_1.$$

But since $||TQ||_r \leq \delta ||Q||_r + D$, we have that:

$$||H_1 Q||_r - D_1 \leq ||TQ||_r \leq \delta ||Q||_r + D,$$

which implies that:

$$||H_1 Q||_r \leq \delta ||Q||_r + D',$$

where $D' = (D + D_1) > 0$. Then by using a result in [34], it is straightforward to show that H_1 keeps the iterate Q bounded. Having shown the boundedness of both the Q vector and ρ, the stage is set to examine the assumptions in the lemma (5.1.1).

6.1.3 Verification of Assumptions in Section 5.1

The limiting ODEs (equations (50) and (51)) in this algorithm can be related to forms given in Borkar's lemma (see Section 5.1.1) by observing that here Q stands for x and ρ stands for y in equations (29) and (30). The assumptions for the lemma are given in the Section 5.1. The lemma also assumes that the iterates remain bounded which has been already proved in the previous section. The assumptions 2 and 3 in Section 5.1 place restrictions on the learning process. The assumption 4 on the noise terms is satisfied because it can be seen from the definition of the noise terms (46 and 47) that the noise represents the difference between the sample and a conditional mean. The conditional mean of this difference tends to 0 as number of samples tends to infinity (using Martingale convergence theory [4]). The variance of this difference has to be bounded because the costs are bounded (and the iterates are bounded, which has been proved) thus satisfying the second clause in assumption 4 (see [4]). The remaining assumptions that need to be verified are assumption 1, 5, and 6. The mapping h is indeed Lipschitz continuous because the mapping is linear everywhere. An examination of the mapping h in equation (42) reveals this fact. The transformation f is also Lipschitz continuous because of the same reason. The assumption 5 may be verified as follows. For a fixed ρ, the mapping $H_1(Q)$ which may be written as $H_\rho(Q)$ is non-expansive with respect to the sup-norm (see Remark 5 in Section 6.2). Using a remarkably powerful result from Borkar and Soumyanath [8] that holds for non-expansive mappings and does not need a contraction property, it can be shown that the solution of the differential equation converges to an asymptotically stable critical point. This point will be henceforth called Q_ρ. This point Q_ρ (the term corresponding to $G(y)$ in the statement of assumption 5 of Borkar's lemma) is a function of ρ, the Lipschitz continuity of which can be proved by the fact that each component of the Q vector is Lipschitz continuous in ρ (see Bertsekas [4]). The second ODE converges to the average cost of a policy. It is clear that as soon as The Q values stabilize, the policy becomes stationary since it is the Q values that dictate the policy. The

average cost of a stationary policy is Lipschitz continuous (see [4]). Besides, the average cost of a given policy is a finite constant for a stationary policy with finite costs and transition times [24]. It remains to be shown however that the policy to which the algorithm converges is indeed the optimal policy. This needs some more work, that we have adapted from Gosavi [14] but we shall present it here for the sake of completeness. The work that is needed is presented via Theorem 3. Theorem 3 completes the proof of Theorem 2.

It is sufficient to show that the Q factors converge to their optimal values since once a policy is learned (all the information related to the policy can be extracted from the Q-values), the optimal average cost can be determined by re-simulating the system with the policy that was learned. In other words, we do not concern ourselves with what value ρ converges to.

We shall first consider the MDP case. In case of the MDP, the Bellman transformation for value iteration is as given in equation (9) which is rarely used in practice since one of the Q values can become very large or very small [3] and hence a relative value iteration method is used where the transformation is as follows.

$$R^*(i) = \min_{u \in A_i} \sum_{j \in S} p(i,u,j)[g(i,u,j) - R(t) + R^*(j)], \tag{52}$$

where $R(t)$ is the value function of some state in the MDP. However both transformations lead to the same policy since the values in one differ from the corresponding values in the other by a constant and hence their "relative values" are same. This is the mechanism used to show that relative value iteration converges to an optimal solution and it is sound since value iteration has been demonstrated to be optimal. We shall use the same mechanism to establish that the new algorithm converges to the same relative values as value iteration.

Consider the transformation (14) that updates the Q-factors in our algorithm. It must be understood that were ρ to be a constant our proof would terminate at this point because the value function produced by such an algorithm would differ from the value function produced by the optimal value iteration algorithm by a constant (in each element) and would generate

the same policy. However, in the new algorithm, ρ keeps changing. So in order to demonstrate equivalence to value iteration, we need to show that in spite of the changing value of ρ, the value function produced by the new algorithm is identical to that produced by a fixed ρ. (Of course, an important question that arises at this point is as follows. If a constant ρ can generate an optimal solution, why do we bother to update ρ ? After all, once the optimal Q factors are known, it is not difficult to obtain a good approximation of the 'optimal' value of ρ using simulation. The answer to this is that from the current estimate of the value function, one has to subtract, in every iteration, a certain quantity that keeps increasing with the Q factors else one or more of the Q factors can become very large or very small Bertsekas [3].)

We shall next state the result that will establish the equivalence of the transformation (14) and the value iteration transformation (13) for the MDP case.

Theorem 3: The transformation (14) and transformation (13) both produce the same Q factors. (In other words, transformation (42) generates an optimal solution for the MDP.)

Proof: For the proof of this theorem, we need the following three lemmas.

Lemma 1: If $r(k)$ denotes the immediate cost incurred in the kth update of ρ, the value of the scalar ρ at the kth iteration ($k = 0, 1, 2...$) can be given as

$$\rho^k = A(k) + B(k), \tag{53}$$

where

$$A(k) = (1 - \beta(1))(1 - \beta(2)) \dots (1 - \beta(k-1))(1 - \beta(k))\rho^0$$

and

$$B(k) = \beta(k)r(k) + \sum_{d=1}^{k-1} C(d),$$

where

$$C(d) = \beta(d)r(d)(1 - \beta(d+1))(1 - \beta(d+2)) \dots (1 - \beta(k)).$$

Proof: We shall use an induction argument. We verify it for $k = 1$. Now ρ in our algorithm is generated by transformation (16). Hence $\rho^1 = (1-\beta(1))\rho^0 + \beta(1)r(1)$. Now from equation (53) we have : $A(1) = (1 - \beta(1))\rho^0$ and $B(k) = \beta(1)r(1)$ thereby satisfying relation (53) for $k = 1$. (Notice that this case is not very interesting as the C terms are 0, but the case for k=2 can also be easily verified.) We next assume that the result is true for $k = n$. Hence $\rho^n = A(n) + B(n)$. Now using transformation (16), we have

$$\rho^{n+1} = (1 - \beta(n+1))\rho^n + \beta(n+1)r(n+1)$$

$$= (1 - \beta(n+1))[A(n) + B(n)] + \beta(n+1)r(n+1)$$

$$= (1 - \beta(n+1))[(1 - \beta(1))(1 - \beta(2)\ldots(1 - \beta(n))\rho^0]$$

$$+ (1 - \beta(n+1))[\beta(n)r(n) + \sum_{d=1}^{n-1} C(d)] + \beta(n+1)r(n+1)$$

$$= A(n+1) + B(n+1).$$

The proof is from Gosavi [14]. *Q.E.D.*

We next show that the difference between value of the scalar ρ at its ith iteration and $(i+\delta)$th iteration is of the order of β.

Lemma 2: The difference between the value of the iterate ρ at the ith iteration and that at the $(i + \delta)$th iteration is of the order of β.

Proof: Using lemma 1, we have

$$\rho^{i+\delta} - \rho^i = A(i + \delta) - A(i) + B(i + \delta) - B(i).$$

It is clear from their definitions that the B terms are already of the order of β. We also have that

$$A(i+\delta) - A(i) = (1-\beta(1))(1-\beta(2))\ldots(1-\beta(i+\delta))\rho^0 - (1-\beta(1))(1-\beta(2)\ldots(1-\beta(i))\rho^0$$

$$\simeq -(\beta(i+1) + \beta(i+2) + \ldots + \beta(i+\delta))\rho^0,$$

where the last approximate equality is obtained after neglecting terms of the order of β raised to 2 and higher integers. It is also clear thus that the difference $\rho^{i+\delta} - \rho^i$ is of the order of β. *Q.E.D.*

We next prove that the difference between a Q factor obtained from subtracting $\rho^{i+\delta}$ and that obtained from subtracting ρ^i is of the order of $\alpha\beta$.

Lemma 3: The difference in the Q factor obtained from in $\rho^{i+\delta}$ as opposed to ρ^i using transformation (14) once is of the order of $\alpha\beta$.

Proof: Suppressing some parts of the notation for the sake of clarity, we denote by Q_s the Q factor obtained by subtracting ρ^s in transformation (14). Hence the error e introduced in the Q factor by subtracting $\rho^{i+\delta}$ as opposed to ρ^i can be written (using some simple algebra with transformation 14):

$$e = Q_{i+\delta} - Q_i = \alpha(\rho^i - \rho^{i+\delta})$$

Using lemma 2, we can write e as:

$$e = O(\alpha\beta).$$

Q.E.D.

(The result can be extended to the case of multiple applications of transformation 14.) From Lemma 3, one can see that the error introduced in the Q factor by the changing value of ρ is negligible (because both α and β are much smaller than 1) and the error tends to 0 asymptotically as α and β tend to 0. In other words this proves Theorem 3. *Q.E.D.*

The proof of Theorem 2 for the MDP case follows from Theorem 3.

For the SMDP case, relative value iteration cannot be used unless the optimal value of ρ is known beforehand. For SMDPs hence, under average cost, an approximate procedure for value iteration has been suggested (see both Bertsekas [3] and Puterman [24]). This approximation approach requires the knowledge of the transition probabilities and hence is

ruled out for RL, where the challenge is to generate a near optimal solution without the transition function. Under these circumstances if we must use a value iteration method, the best we can do is to start with a good guess of the optimal value of ρ (and this is not as difficult as it seems) since we can still obtain a 'near-optimal' solution. For the SMDP, our updating is given by transformations (17) and (18). The strategy that will ensure a 'near-optimal' solution is to start with a value of ρ that is known to be close to the optimal value, in equation (18). This will ensure that the Bellman equation (see Theorem 1 in Section 3.1.1) is approximately satisfied by the transformation (17). As in the MDP case, a changing value of ρ will not affect the policy to which the Q-factors will converge.

The difficulty of finding an estimate of the optimal cost beforehand is also encountered in other learning paradigms such as learning automata (see Narendra and Thathachar [23] and Gosavi, Das and Sarkar [16]), which hinges on a good estimate of the 'feedback' of a given action that needs the knowledge of the optimal cost beforehand. A commonly used strategy is to use a fraction of the average cost produced by some good heuristic. If no heuristic is available, a strategy that has been used in automata literature is to choose a randomized policy (that takes each action in a given state with equal probability) and then use a fraction of the average cost of the randomized policy as estimate of the average cost. In any case, in RL several trials are often necessary with tuning learning rates to obtain superior performance. Hence if a chosen value does not work (in the sense that it fails to outperform a known heuristic), one has to choose a lower value (in case of average cost) or a higher value (in case of average reward). Empirical evidence suggests that this problem does not cause too much trouble in practice.

6.2 Some Remarks on the Convergence Behavior

We next make come comments on the convergence proof shown above.

- Remark 1: That the transformation (42) is contractive is a proven result. See appendix A.5 for a proof.

- Remark 2: The role of exploration in any Q-learning algorithm is to ensure that all state-action pairs are tried. Unless all state-action pairs are tried and good estimates of all the Q-factors are obtained, Q-learning, which is essentially a form of value iteration can diverge.

- Remark 3: Assumption 2 in Section 5.1 says that every state-action pair must be tried after a finite interval of time. A mechanism in RL that roughly guarantees this, is function approximation. Function approximation schemes such as neural networks or local regression (or even local neural networks (see Gosavi, Bandla and Das [15])) ensure with probability 1 that all states are visited after a finite time interval and that no one state is 'forgotten' for a very long time. (The reason for this phenomenon is that whenever a state is visited, some neighboring states are also updated (in case of local regression / local neural networks) or all the states in the system are updated in case of a global state-space-wide neural network / regression.) This is perhaps why a function-approximation-coupled approach usually outperforms a look-up table even when the state-space can be managed with a look-up table ([15]).

- Remark 4: It is difficult to satisfy, in a naive sense, in RL, the assumption that every Q factor should have its own learning rate and that it be decayed depending on the number of visits paid to it. This is because that would require keeping track of the number of times every state-action pair is tried and that would defeat the purpose of RL since this would require too much storage in problems with a huge state-space (unless some compact methods are devised for storing counters that indicate these frequencies). However, this condition too can be roughly satisfied under a function approximation approach where all states are visited more or less regularly and hence one can use the same learning rate (and consequently same decaying scheme) for all

the state-action pairs.

- Remark 5: We next show that the mapping (42) is non-expansive with respect to the sup-norm. Using the map (42), it is clear that

$$H_1Q_1(i,u) - H_1Q_2(i,u) = \sum_j p(i,u,j)[min_v Q_1(j,v) - min_v Q_2(j,v)]. \tag{54}$$

Then we have that:

$$\begin{aligned} |H_1Q_1(i,u) - H_1Q_2(i,u)| &\leq \sum_j p(i,u,j)|\min_v Q_1(j,v) - \min_v Q_2(j,v)| \\ &\leq \sum_j p(i,u,j) \max_v |Q_1(j,v) - Q_2(j,v)| \\ &\leq ||Q_1 - Q_2||_\infty, \end{aligned}$$

which implies that

$$\max_{i,u} |H_1Q_1(i,u) - H_1Q_2(i,u)| \leq ||Q_1 - Q_2||_\infty,$$

which can be written as:

$$||H_1Q_1 - H_1Q_2||_\infty \leq ||Q_1 - Q_2||_\infty.$$

- Remark 6: The SMART algorithm [10] for the MDP case can also be expressed with a step-size in its updating of ρ. The updating in SMART is done as follows.

Initialize k and TOTAL_COST to 0 in the beginning. Perform update of ρ as follows:

$$\text{TOTAL\_COST} \leftarrow \text{TOTAL\_COST} + g(i,u,j)$$

$$\rho \leftarrow \text{TOTAL\_COST}/k + 1$$

It is not hard to show that this update of ρ is equivalent to:

$$\rho \leftarrow (1-\beta)\rho + \beta g(i,u,j)$$

in which $\beta = 1/(k+1)$. Using this argument, it is possible to prove convergence of SMART in a manner similar to that shown above. However, there are two problems: 1) This step-size rule of $1/(k+1)$ cannot be controlled (as required in Borkar's result) and may cause divergence. Also, it may be difficult to ensure that β is smaller than α. 2) The starting value of ρ used in SMART is 0 and this may be far away from the optimal value of ρ. Since the algorithm presented in this paper uses a relaxation scheme in its update of ρ, but is otherwise similar to SMART, it was named Relaxed-SMART in [13].

7 Numerical Results

First, we present the Robbins-Monro version of the algorithm for SMDPs in a step by step format, with all the implementational details. Thereafter a description of the preventive maintenance case-study [11] and the numerical results on several example problems from it are presented.

The step-by-step details of the algorithm, for a numerical implementation are supplied in Figure 2. Let S be the set of all states and $A(i)$ be the set of actions in state i. The learning rates (α and β) are decayed to 0 as the learning progresses. The exploration probability (p) also has to be suitably decayed. Typically a decaying scheme works as follows. If $d(m)$ denotes the variable at the mth iteration, then $d(m)$ may be defined as follows: $d(m) = M/m$ where M is some predetermined constant (for example 0.1). Also it may be noted here that even though the algorithm will be presented in a form to be used with the more general SMDP, it is obvious that by setting the transition times to one in every case, the algorithm can be used for an MDP (which is a special case of an SMDP) as well. The value of ρ^s, which is the starting value of ρ can be obtained from using the renewal reward heuristic on the PM case study [10]. In case of MDPs, ρ^s can be 0 but in case of SMDPs, ρ^s should

preferably be the average cost of some heuristic. (See Section 6 for another mechanism.)

7.1 Case Study

We next describe the preventive maintenance case study chosen to benchmark the new algorithm. A production inventory system is considered which produces a single product type to satisfy an external demand process. The system is prone to failures; the time between failures, a random variable, is not exponentially distributed. Hence preventive maintenance [19] is an option to reduce the downtime of the machine. Demands that arrive when the inventory buffer is empty are not backordered and are, therefore, lost. Provision for a finished-product inventory buffer of the product between the system and the demands is kept so as to minimize the chances of losing demand when the machine is down due to breakdown or maintenance. The system stops production when the buffer inventory reaches $\mathcal{S}$ and the production resumes when the inventory drops to s. During its vacation, i.e. when production stops due to the inventory reaching its upper limit $\mathcal{S}$, the system is assumed not to age or fail. Preventive maintenance decisions are made only at the completion of a part production. The problem may be set up as an SMDP which is discussed next. An infinite supply of raw material is assumed. When the system produces a part, the part goes into the inventory-buffer. When a demand arrives, the buffer is depleted by one unit; if the buffer is empty that particular demand goes away never to return. As the machine ages during production, it can fail. The repair time and the production times are random variables. During the vacations, the machine is assumed not to age. After the end of the repair or maintenance, the machine is assumed to be as good as new. When the machine completes the manufacture of one part, two options are available : (i) to produce one more part and (ii) to schedule a maintenance. The decision needs to be taken based on the age of the machine, a measure of which is the number of parts produced since last repair or maintenance (which will be denoted by c), and the number of parts in the buffer (w). The

decision-making state-space of the system may hence be denoted by (w, c). For more details on the Semi-Markov model, the Semi-Markov property with the chosen state-space and the method used to obtain the optimal policies, the reader is referred to the excellent paper written by Das and Sarkar [11].

Comment: It needs to be emphasized here that in the Das and Sarkar [11] paper, the assumption that the production time and the time between failure were *both* gamma-distributed made it considerably easy to compute the exact transition probabilities. They were able to make use of the property that sum of gamma is gamma. With a gamma distributed production time and some other distribution for the time for failure or repair, computing the transition probabilities becomes much more difficult. The complexity of the transition function is usually (though not always) dependent on the number of random variables in the system and how they interact. Sometimes, even problems with a small state-space have very complex transition mechanisms (this is especially true of the SMDP) and obtaining expressions for the transition probabilities can be quite difficult. And it is no exaggeration to say that under general assumptions for the distributions, computing the transition probabilities can be very difficult for problems even with a few hundred states such as some of those studied in [11], where there are five random variables involved. It should be noted that a change in distribution can be easily accommodated in a simulation model such as that in RL and a large number of random variables do not make the simulation model any harder to construct.

The demand arrival process (with batch size of 1) is Poisson (γ). The unit production time, machine repair time, and the time between machine failures are gamma distributed with parameters (d, λ), (r, δ), and (k, μ) respectively. The time for preventive maintenance has a uniform (a, b) distribution. The values of the input parameters used for numerical tests with the algorithm are shown in Table 1. For each of the nine parameter sets (shown in Table 1), the cost parameters considered are denoted by: C_d (net revenue per demand serviced),

C_r (cost per repair), and C_m (cost per maintenance). In the experiments performed, $\mathcal{S}$ is assumed to be 3 and s is assumed to be 2. The optimal results (average cost) for the single product inventory system obtained numerically from the theoretical model and the RL algorithm are summarized in Table 2. Results show that the RL algorithm performs extremely well. A statistical test comparing two means: the average cost obtained from the policy generated by the RL algorithm and that obtained from the optimal policy [11] reveals no significant difference at a 95 % confidence level.

8 Other average cost RL algorithms

There are a number of other RL algorithms that have been proposed in the literature. SMART was discussed above. We would like to discuss R-Learning [26], which was the first average cost RL algorithm to have appeared in the literature, and the contracting Q-Learning algorithm of Abounadi, Bertsekas and Borkar [1], which was the first convergent average cost RL algorithm to have appeared in the literature. These algorithms were designed for MDPs and hence we shall discuss them in that context.

R-Learning. Refer to Figure 2. Use $t(i,u,j) = 1$ for all values of i, j and u in Step 4. Step 5 in R-Learning is:

$$\rho \leftarrow (1-\beta)\rho + \beta[g(i,u,j) + \min_{v \in A(j)} Q(j,v) - \min_{w \in A(i)} Q(i,w)].$$

Other steps of the algorithm are as in Figure 2.

Contracting Q-Learning Again, refer to Figure 2 and use $t(i,u,j) = 1$ for all i, u and j. Step 4 would be replaced by:

$$Q(i,u) \leftarrow (1-\alpha)Q(i,u) + \alpha[g(i,u,j) - \rho + \min_{v \in A(j)} \hat{Q}(j,v)]$$

where $\hat{Q}(j,v) = Q(j,v)$, if $i \neq j$, and $\hat{Q}(j,v) = 0$ otherwise. Step 5 would be:

$$\rho \leftarrow \rho + \beta \min_{v \in A(s^*)} Q(s^*, v)$$

where s^* is a special state that is selected in the beginning of the algorithm. The special state could be any state in the system, but cannot be changed once selected. This contracting Q-learning algorithm has the flavor of the relative value iteration algorithm.

Although numerical experience with these algorithms is not available on large-scale problems, the strong convergence properties of the algorithm in [1] suggest that this algorithm is a good candidate for use in large-scale problems.

9 Conclusions

RL is a relatively new field that developed in the mid-nineties even though the theory of dynamic programming for solving Markov Decision Problems is very old. The emergence of RL based algorithms has given the field of stochastic dynamic programming the power it did not have previously. Only very recently have commercial applications of RL started appearing. The theory of average cost RL is considerably harder than that of discounted RL and a lot of fundamental work remains to be done in this area. Model-free average cost RL algorithms can be classified into two categories: (i) algorithms based on relative value iteration, (ii) algorithms based on Bellman equation. Three algorithms based on relative value iteration have been proposed, the first that appeared in literature R-Learning, and the other two were proposed in Singh [30] and [1]. In the second category, we found two algorithms: SMART [10] and the new algorithm (presented in this paper). Some of the work presented here comes from a dissertation [13]. Some of the open research issues are envisioned as follows: (i) Study of MDPs and SMDPs under the finite horizon in ARL. (ii) Development of an RL algorithm using Sennott's Approximating Sequence Method [28]. (iii) Application of the new algorithm to difficult large-sized problems such as the Stochastic

Economic Lot-size Scheduling Problem (SELSP) using the SMDP model.

A Appendix

A.1 Sup-norm of a Vector

The sup-norm or max-norm of a vector x, is defined as $||x||_\infty = \max_i |x_i|$, where $|x_i|$ denotes the absolute value of the vector's ith component.

A.2 Lipschitz Continuity

A function $f(x)$ is said to be Lipschitz continuous in $x \in R^n$ if there exists a finite value for L that satisfies:

$$||f(x_2) - f(x_1)|| \leq L||x_2 - x_1||,$$

for any x_1 and x_2 in R^n and $||.||$ denotes a norm.

A.3 Critical Point and Asymptotically Stable Critical Point of an ODE

Consider the differential equation $\frac{dx}{dt} = f(x, t)$.

Definition 1. The *critical point* (or equilibrium point) is defined as the vector x^* that such that $f(x^*, t) = 0$ in the differential equation given above.

Definition 2. Let x^* be the critical point of the ODE. Let $B(x^*, R)$ be the ball (open sphere) of radius R centered around x^*. Then x^* is said to be an *asymptotically stable critical point* if and only if for all values of $\epsilon > 0$, there exists a scalar quantity $R_\epsilon > 0$ such that if $x(0) \in B(x^*, R_\epsilon)$, then $x(t) \in B(x^*, \epsilon)$ for all t, and $\lim_{t\to\infty} x(t) = x^*$.

A.4 Standard Step-Size Conditions

Let $a(k)$ be the step-size at the kth iteration. The standard conditions for convergence that $a(k)$ must satisfy are as follows: (i). $\Sigma_{k=0}^{\infty} a(k) = \infty$, (ii) $a(k+1) \leq a(k)$ for k large enough, (iii) There exists $r \in (0,1)$ such that for all $q \geq r$, $\Sigma_{k=0}^{\infty} a^{1+q}(k) < \infty$. (iv) Let $A(m) = \Sigma_{k=0}^{m} a(k)$ then for all $r \in (0,1)$, $sup_k \frac{a([rk])}{a(k)} < \infty$, and the following holds uniformly in $y \in [r,1)$ for all $r \in (0,1)$, $\lim_{k\to\infty} \frac{A([yk])}{A(k)} = 1$. A step-size that satisfies all of the above conditions is $\{\frac{1}{k}\}$.

A.5 Contraction Property of Transformation H_1 (42)

Consider the transformation in equation (42) for a Markov chain under Assumption 3 of Theorem 2.. There exists a vector ξ such that mapping H_1 is a contraction mapping with respected to a weighted sup-norm $||.||_{\xi}$.

Comment: It must be emphasized that the result we are about to present holds for the SMDP in which we use the transformation H_1 (42) and that we are *not* concerned with whether it holds for the fictitious shortest stochastic path problem that will be invoked subsequently. A shortest stochastic path problem is invoked only for the sake of establishing a relation for the transition probabilities of our SMDP - a relation that can be used to prove the result. The proof is due to Littman [22] and was also independently discovered by Bertsekas and Tsitsiklis [4] (pg. 23, Proposition 2.2). Even though the proof has been quoted in the shortest stochastic path context in [4], it has been used elsewhere because it pertains to the mapping.

Proof: We first consider a *new, fictitious* shortest stochastic path problem, whose transition probabilities are identical to the SMDP under consideration, but all the immediate transition costs are -1 and the immediate transition times are 1.

Then we have that for all $i = 1, 2, \ldots, n$ and stationary policies μ,

$$D(i,u) = -1 + \min_{v \in U(j)} \sum_j p(i,u,j) D(j,v) \leq -1 + \sum_j p(i,\mu(i),j) D(j,\mu(j)), \tag{55}$$

where $D(i,u)$ is the Q-factor for the state-action pair (i,u). We shall make our notation a little more compact by replacing (i,u) by k and $(j,\mu(j))$ by l, where k and l denote the state-action pairs and take values in $1, 2, 3, \ldots, N$. Thus $D(i,u)$ will be replaced by $D(k)$. Let us now define a vector ξ as follows $\forall k$:

$$\xi(k) = -D(k).$$

Then for all k, we have from the definition of $D(k)$ above (equation (55)), $D(k) \leq -1$, which implies that $\xi(k) \geq 1$ for $k = 1, 2, 3, \ldots, N$. For *all* stationary policies μ, we then have from (55) that for all k,

$$\sum_j p(i,\mu(i),j) \xi(l) \leq \xi(k) - 1 \leq \zeta \xi(k), \tag{56}$$

where ζ is defined by:

$$\zeta = \max_{k=1,2,\ldots,N} \frac{\xi(k) - 1}{\xi(k)} < 1, \tag{57}$$

where the last inequality follows from the fact that $\xi(k) \geq 1$. Thus we have :

$$\sum_j p(i,\mu(i),j) \xi(l) \leq \zeta \xi(k), \tag{58}$$

which is true of the transition probabilities of the SMDP. We now turn our attention to the SMDP in question. From transformation H_1 in equation (42), we can write :

$$H_1 Q_1(i,u) - H_1 Q_2(i,u) = \sum_j p(i,u,j) [\min_v Q_1(j,v) - \min_v Q_2(j,v)]. \tag{59}$$

Using our compact notation, we can write the above as:

$$H_1 Q_1(k) - H_1 Q_2(k) = \sum_j p(i,u,j) [\min_l Q_1(l) - \min_l Q_2(l)]. \tag{60}$$

Then we have that:

$$|H_1 Q_1(k) - H_1 Q_2(k)| \leq \sum_j p(i,u,j) |\min_l Q_1(l) - \min_l Q_2(l)|$$

$$\leq \sum_j p(i,u,j) \max_l |Q_1(l) - Q_2(l)|$$
$$= \sum_j p(i,u,j)\xi(l)||Q_1 - Q_2||_\xi$$
$$\leq \zeta\xi(k)||Q_1 - Q_2||_\xi,$$

where we define the weighted max norm of $X(k)$ with respect to ξ as:

$$||X||_\xi = \max_k \frac{|X(k)|}{\xi(k)}.$$

From the above, we can write:

$$\frac{|H_1Q_1(k) - H_1Q_2(k)|}{\xi(k)} \leq \zeta||Q_1 - Q_2||_\xi.$$

Since the above holds for all values of k, it also holds for the value of k, for which its left hand side is maximized. In other words, we have that:

$$||H_1Q_1 - H_1Q_2||_\xi \leq \zeta||Q_1 - Q_2||_\xi,$$

which proves the result.

References

[1] J. Abounadi, D. Bertsekas, and V. Borkar. Ode analysis for Q-learning algorithms. LIDS Report, MIT, Cambridge, MA, 1996.

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

[3] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, Belmont, Massachusetts, 1995.

[4] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

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

[6] V. S. Borkar. Asynchronous stochastic approximation. *SIAM J. Control Optim.*, 36 No 3:840–851, 1998.

[7] V. S. Borkar and S.P. Meyn. The ode method for convergence of stochastic approximation and reinforcement learning. Working Paper.

[8] V. S. Borkar and K. Soumyanath. An analog scheme for fixed point computation, Part i:theory. *IEEE Transactions Circuits and Systems I. Fundamental Theory and Appl.*, 44:351–354, 1997.

[9] C.G. Cassandras and Youngnam Han. Optimal inspection policies for a manufacturing station. *European Journal of Operational Research*, 63:35–53, 1992.

[10] 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.

[11] T. K. Das and S. Sarkar. Optimal preventive maintenance in a production inventory system. *IIE Transactions on Quality and Reliability*, 31. (in press).

[12] M. Elhafsi and S. Bai. Optimal and near-optimal control of a two-part stochastic manufacturing system with dynamic setups. Research Report 95-10, at the Department of Industrial Engineering, University of Florida, Gainesville, 1997.

[13] A. Gosavi. An algorithm for solving semi-markov decision problems using reinforcement learning: Convergence analysis and numerical results. Unpublished Ph.D. Dissertation, Department of Industrial and Management Systems Engineering, University of South Florida, Tampa, FL, May,1999.

[14] A. Gosavi. On the Convergence of Some Reinforcement Learning Algorithms. Working paper, Department of Engineering, University of Southern Colorado, Pueblo, 2000.

[15] A. Gosavi, N. Bandla, and T. K. Das. Airline seat allocation among Multiple Fare Classes with Overbooking. To appear in IIE Transactions.

[16] A. Gosavi, T. K. Das, and S. Sarkar. A Simulation-Based Learning Automata Framework for Solving Semi-Markov Decision Problems under Long-Run Average Reward. Under review, Department of Engineering, University of Southern Colorado.

[17] V.R. Konda and V. S. Borkar. Actor-critic type learning algorithms for markov decision processes. Working Paper, Indian Institute of Sciences, Bangalore, India.

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

[19] E. E. Lewis. *Introduction to Reliability Engineering*. John Wiley and Sons, Inc, New York, 1994.

[20] L. Ljung. Analysis of recursive stochastic algorithms. *I.E.E.E. Transactions on Automatic Control*, 22:551–575, 1977.

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

[22] M.L.Littman. Algorithms for sequential decision-making. Unpublished Ph.D. Thesis, Brown University, Providence, R.I. 1996.

[23] Kumapati Narendra and M.A.L. Thatachar. *Learning Automata: An Introduction*. Prentice Hall, Englewood Cliffs, New Jersey, 1989.

[24] M. L. Puterman. *Markov Decision Processes*. Wiley Interscience, New York, USA, 1994.

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

[26] A. Schwartz. A reinforcement learning method for maximizing undiscounted rewards. *Proceeding of the Tenth Annual Conference on Machine Learning*, pages 298–305, 1993.

[27] A. Seidmann and P.J. Schweitzer. Part selection policy for a flexible manufacturing cell feeding several production lines. *IIE Transactions*, 16, number 4:355–362, Decemeber, 1984.

[28] L. Sennott. *Stochastic Dynamic Programming and the Control of Queueing Systems*. John Wiley and Sons, 1999.

[29] T. Shioyama. Optimal control of a queuing network system with two types of customers. *European Journal of Operational Research*, 52:367–372, 1991.

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

[31] R. Sutton. Reinforcement learning. *Special Issue of Machine Learning Journal*, 8(3), 1992.

[32] R. Sutton and A. G. Barto. *Reinforcement Learning*. The MIT Press, Cambridge, Massachusetts, 1998.

[33] P. Tadepalli and D. Ok. Scaling up average reward reinforcement learning by approximating the domain models and the value function. In *Proceedings of the Thirteenth International Machine Learning Conference*, pages 471–479. Morgan Kaufmann, 1996.

[34] J. Tsitsiklis. Asynchronous stochastic approximation and q-learning. *Machine Learning*, 16:185–202, 1994.

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