Solving Semi-Markov Decision Problems using Average Reward Reinforcement Learning

Tapas K. Das and Abhijit Gosavi
Department of Industrial and Management Systems Engineering
University of South Florida, Tampa, FL 33620

Sridhar Mahadevan
Department of Computer Science
Michigan State University, East Lansing, MI 48824

Nicholas Marchalleck
Department of Computer Science and Engineering
University of South Florida, Tampa, FL 33620

September 20, 1998

Abstract

A large class of problems of sequential decision making under uncertainty, of which the underlying probability structure is a Markov process, can be modeled as stochastic dynamic programs (referred to, in general, as Markov decision problems or MDPs). However, the computational complexity of the classical MDP algorithms, such as value iteration and policy iteration, is prohibitive and can grow intractably with the size of the problem and its related data. Furthermore, these techniques require for each action the one step transition probability and reward matrices, obtaining which is often unrealistic for large and complex systems. Recently, there has been much interest in a simulation-based stochastic approximation framework called reinforcement learning (RL), for computing near optimal policies for MDPs. RL has been successfully applied to very large problems, such as elevator scheduling, and dynamic channel allocation of cellular telephone systems.

In this paper, we extend RL to a more general class of decision tasks that are referred to as semi-Markov decision problems (SMDPs). In particular, we focus on SMDPs under the average-reward criterion. We present a new model-free RL algorithm called SMART (Semi-Markov Average Reward Technique). We present a detailed study of this algorithm on a combinatorially large problem of determining the optimal preventive maintenance schedule of a production inventory system. Numerical results from both the theoretical model and the RL algorithm are presented and compared.

1 Introduction

A wide variety of problems in diverse areas, ranging from manufacturing to computer communications, involve sequential decision making under uncertainty. A subset of these problems, which are amenable to Markovian analysis (Markov chains, in particular), are referred to as Markov decision problems (MDPs). MDPs have been studied extensively in the stochastic dynamic programming literature. Some examples of such problems that can be studied under the stochastic DP framework are: inventory management, preventive maintenance, polling systems in computer networks, AGV path planning, repair allocation, call routing in cellular telephone networks, job shop scheduling, etc.

The framework of dynamic programming and MDPs, originally proposed by Bellman [4] and subsequently extended by Karlin [16], Howard [14], Blackwell [6], to name a few, is quite extensive and rigorous. Well known algorithms, such as value iteration, policy iteration, and linear programming find optimal solutions (i.e., optimal policies) of MDPs. However, the main drawback of these classical algorithms is that they require, for every decision, computation of the corresponding one step transition probability matrix and the one step transition reward matrix using the distributions of the random variables that govern the stochastic processes underlying the system. For complex systems with large state spaces, the burden of developing the expressions for transition probabilities and rewards could be enormous. Also, the computational requirement of storing these matrices (e.g., of size 10^{11} for the problem considered later in this paper) would make the problem intractable. This is commonly referred to as *curse of dimensionality* in the DP literature. In absence of better approaches, problem-specific heuristic algorithms are often used that attempt to reach acceptable near-optimal solutions.

Computer simulation based reinforcement learning (RL) methods of stochastic approximation, such as decentralized learning automata (Wheeler and Narendra [35]), method of temporal differences, TD(λ) (Sutton [29]), and Q-learning (Watkins [34]) have been proposed in recent years as viable alternatives for obtaining near optimal policies for large scale MDPs with considerably less computational effort than what is required for DP algorithms. RL has two distinct advantages over DP. First, it avoids the need for computing the transition probability and the reward matrices. The reason being that it uses discrete event simulation (Law and Kelton [17]) as its modeling tool, which requires only the probability distributions of the process random variables (and not the one step transition probabilities). Secondly, RL methods can handle problems with very large state spaces (e.g., with 10^{12} states) since its computational burden is related only to value function estimation, for which it can effectively use various function approximation methods (such as, regression, and neural networks).

Most of the published research in reinforcement learning is focused on the discounted sum of rewards as the optimality metric (Kaelbling et al. [15], Van Roy et al. [24]). However, in many engineering design and decision making problems, performance measures are not easily described in economic terms, and hence it may be preferable to compare policies on the basis of their time averaged expected reward rather than their expected total discounted reward. Some examples of such measures are, average waiting time of a job in the queue, average number of customers in the system, average throughput of a machine, and average percentage of satisfied demands in an inventory system. Applications of RL under the average reward criterion have been limited to MDPs only with a small state space (Tadepalli [32]).

Bradtke and Duff [7] discuss how RL algorithms developed for MDPs can be extended to generalized version of MDPs, in particular semi-Markov decision problems (SMDPs), under the discounted cost criterion. Recently, applications of RL have been extended to semi-Markov decision problems with the discounted cost criterion. These include a discounted RL (Q-learning) algorithm for elevator control (Crites and Barto [8]) and a Q-learning algorithm for dynamic channel allocation in cellular telephone systems (Singh and Bertsekas [26]). The latter presents experimental results for a special case of SMDPs, namely continuous-time Markov decision problems (CTMDPs), where the sojourn times at the decision making states are always exponentially distributed with parameters dependent on the initial state.

In this paper, we first discuss the basic concepts underlying the reinforcement learning approach and its roots in algorithms such as Bellman's value iteration algorithm [4] and Sutton's temporal difference (TD) algorithm [29]. We then present an *average reward* RL algorithm, named SMART (**s**emi-**M**arkov **a**verage **r**eward **t**echnique), for semi-Markov decision problems (SMDPs). The algorithm presented here can be applied to a much larger class of problems for which stochastic approximation algorithms based on average reward reinforcement learning have not been applied.

We present experimental results from the proposed algorithm via a challenging problem scenario involving optimal preventive maintenance of a production inventory system. The optimal results, obtained numerically, from its semi-Markov model for a set of problems (with a limited state space) are compared with the results obtained from the SMART algorithm. The speed and accuracy of the SMART algorithm is extremely encouraging. To demonstrate the performance of RL algorithm for problems with a large state space, we subsequently expand the preventive maintenance problem to represent a more real life situation (with state space of $7 \times 10^7 \times |\Re^+|$), and compare the results from SMART with those obtained from two heuristic algorithms that are based on well known mathematical principles.

2 Semi-Markov Decision Problems

Many sequential decision making problems have underlying probability structures that can not be characterized solely by Markov chains, for example, systems with sojourn times that are drawn from general probability distributions having parameters dependent on (perhaps) both the initial and final states of the sojourns. Such problems can often be modeled as semi-Markov decision Processes (SMDPs) embedded on continuous time semi-Markov processes

(SMPs).

Suppose, for each $n \in \mathcal{N}$ (the set of integers), a random variable X_n takes values in a countable set $\mathcal{E}$ 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 stochastic process $(\mathbf{X}, \mathbf{T}) = \{X_n, T_n : n \in \mathcal{N}\}$ is said to be a Markov renewal process (MRP) with state space $\mathcal{E}$, when for all $n \in \mathcal{N}$, $j \in \mathcal{E}$, and $t \in \Re^+$, the following condition is satisfied.

$$P\{X_{n+1} = j, T_{n+1} - T_n \leq t | X_0, \cdots, X_n T_0, \cdots, T_n\} = P\{X_{n+1} = j, T_{n+1} - T_n \leq t | X_n\}. \quad (1)$$

Define $L = \sup_n T_n$, then L is the lifetime of $(\mathbf{X}, \mathbf{T})$. If $\mathcal{E}$ is finite and the Markov chain $\mathbf{X} = \{X_n : n \in \mathcal{N}\}$ is irreducible, then $L = +\infty$ almost surely. Define a process $\mathbf{Y} = \{Y_t : t \in \Re^+\}$, where $Y_t = X_n$, if $T_n \leq t < T_{n+1}$. The process $\mathbf{Y}$ is called a semi-Markov process associated with the MRP $(\mathbf{X}, \mathbf{T})$.

Clearly, for SMDPs, decision epochs are not restricted to discrete time epochs (like in MDPs) but are all time epochs at which the system enters a new decision making state. (Decisions are only made at specific system state change epochs that are relevant to the decision maker.) That is, the system state may change several times between two decision epochs. We will refer to the Markov renewal process embedded at the decision making epochs of $\mathbf{Y}$ as the *decision process*, and the complete process $\mathbf{Y}$ as the *natural process*. The decision process can be formalized as follows. Let $\hat{T}_m$, $m \in \mathcal{N}$, be the time of the m^{th} decision epoch of $\mathbf{Y}$ and the system state immediately following $\hat{T}_m$ be $\hat{X}_m$. Define $\hat{\mathbf{X}} = \{\hat{X}_m : m \in \mathcal{N}\}$ and $\hat{\mathbf{T}} = \{\hat{T}_m : m \in \mathcal{N}\}$. Then we have the *decision process* (SMDP) as $(\hat{\mathbf{X}}, \hat{\mathbf{T}}) = \{(\hat{X}_m, \hat{T}_m) : m \in \mathcal{N}\}$. For an SMDP, we assume that sojourn times are finite and a finite number of transitions take place within a finite time period. For any state i, $\mathcal{A}_i$ denotes the set of possible actions, and $\cup_{\mathcal{A}_i} = \mathcal{A}$ (the action space). For the Markov chain $\hat{\mathbf{X}}$ underlying the SMDP, and for any action $a \in \mathcal{A}$, there is a state transition matrix $P(a) = \{P_{ij}(a) : i, j \in \mathcal{E}\}$, where $P_{ij}(a)$ represents the probability of moving from state i to state j under action a.

The reward structure for SMDPs that includes most applications can be given as follows. For $i \in \mathcal{E}$, when action $a \in \mathcal{A}_i$ is chosen, a lump sum reward of $k(i,a)$ is received. Further accrual of reward occurs at a rate $c(l,i,a)$, for all $l \in \mathcal{E}$, as long as the natural process $\mathbf{Y}$ occupies state l during a transition of the embedded decision process from state i caused by action a. That is, between two transitions of the decision process, the continuous reward accrual rate may vary with the natural process. The total expected reward between m^{th} and $(m+1)^{st}$ decision epochs ($r(i,a)$), when the system state at the m^{th} epoch is i and action taken is a, can be calculated as

$$r(i,a) = k(i,a) + E\left\{\int_{\hat{T}_m}^{\hat{T}_{m+1}} c(Y_t, i, a)dt\right\}. \tag{2}$$

Clearly for stationary processes, the sojourn time $\hat{T}_{m+1} - \hat{T}_m$ depends only on the state-action pair (i,a) at the starting epoch, and is independent of m. Let, for $i \in \mathcal{E}$ and $a \in \mathcal{A}_i$, $y(i,a)$ denote the expected length of time until the next decision epoch, given that action a is chosen when the system state at the m^{th} decision epoch is i. Then we have that

$$y(i,a) \equiv E_i^a\{\hat{T}_{m+1} - \hat{T}_m\} = \int_0^\infty t \sum_{j\in\mathcal{E}} Q(i,j,dt|a), \tag{3}$$

where $Q(i,j,t|a) = P\{\hat{X}_{m+1} = j, \hat{T}_{m+1} - \hat{T}_m \leq t | \hat{X}_m = i, A_m = a\}$. Then the gain (average reward rate) of an SMDP starting at state i and continuing with policy π can be given (using the renewal reward theorem (Ross [23]) as

$$g^\pi(i) = \lim_{N\to\infty} \frac{E_i^\pi\left\{\sum_{m=0}^N \left[k(\hat{X}_m, A_m) + \int_{\hat{T}_m}^{\hat{T}_{m+1}} c(Y_t, X_m, A_m)dt\right]\right\}}{E_i^\pi\left\{\sum_{m=0}^N (\hat{T}_{m+1} - \hat{T}_m)\right\}}. \tag{4}$$

In this paper, we focus on problems with average reward performance measure, where, for every stationary policy, the embedded Markov chain $\hat{\mathbf{X}}$ has a *unichain* transition probability matrix (i.e., every *stationary deterministic policy* π^*, which is a mapping $\pi^* : \mathcal{E} \to \mathcal{A}$, has a single recurrent class plus a possibly empty set of transient states, Puterman [21], pp. 348). Under this assumption, the expected average reward of every stationary policy does not vary with the initial state.

2.1 Bellman Optimality Equation for Average Reward SMDPs

The Bellman optimality equation for SMDPs can be stated as follows.

Theorem 1 *For any finite unichain SMDP, there exists a scalar g^* and a value function R^* satisfying the system of equations*

$$R^*(i) = \max_{a \in \mathcal{A}_i} \left(r(i,a) - g^* y(i,a) + \sum_{j \in \mathcal{E}} P_{ij}(a) R^*(j) \right), \quad \forall i \in \mathcal{E}, \tag{5}$$

where $y(i,a)$ is the average sojourn time of the SMDP in state i under action a, and the "greedy" policy π^* formed by selecting actions that maximize the right-hand side of the above equation is gain-optimal.

A proof of the above theorem can be found in Puterman [21].

A value iteration algorithm (similar to the one for MDPs) can be derived from the above Bellman equation for solving SMDPs. We note, however, that the immediate reward $r(i,a)$ calculation for SMDPs is quite difficult, since it requires calculation of the limiting probabilities of the natural process ($\mathbf{Y}$) in which SMDPs are embedded. In addition to the above mentioned difficulty, value iteration sweeps through the whole state space and simultaneously updates the values of all the states in every iteration. This creates a considerable computational challenge for value iteration, especially, for problems with large state spaces. Even under favorable conditions, convergence of the average reward value iteration algorithm is very slow. Also, since v^n (the value at the n^{th} iteration) diverges linearly in n, the average reward value iteration becomes numerically unstable. A relative value iteration algorithm given below (White [36]) avoids the difficulty with divergence, but does not enhance the rate of convergence.

2.2 Relative Value Iteration Algorithm

1. Select $v^0 \in V$, choose $k^* \in \mathcal{E}$, specify $\epsilon > 0$, and set $m = 0$.

2. For each $i \in \mathcal{E}$, compute $v^{m+1}(i)$ by

$$v^{m+1}(i) = \max_{a \in \mathcal{A}_i} \left\{ r(i,a) - v^m(k^*) + \sum_{j \in \mathcal{E}} P_{ij}(a) v^m(j) \right\}.$$

3. If $sp(v^{m+1} - v^m) < \epsilon$, go to step 4. Otherwise increment m by 1 and return to step 2. sp denotes *span*, which for a vector v is defined as $span(v) = \max_{i \in \mathcal{E}} v(i) - \min_{i \in \mathcal{E}} v(i)$ (Puterman [21]).

4. For each $i \in \mathcal{E}$, choose ϵ-optimal action in steady state, $d_\epsilon(i)$, as

$$d_\epsilon(i) \in \text{argmax}_{a \in \mathcal{A}_i} \left\{ r(i,a) - v^m(k^*) + \sum_{j \in \mathcal{E}} P_{ij}(a) v^m(j) \right\}$$

   and stop.

This algorithm differs from value iteration in that it normalizes the value by substracting a constant after each iteration. The relative value iteration is a *synchronous* algorithm, in that it conducts an exhaustive sweep over the whole state space at each step. An alternative approach is to update the value of one state at a time. This can be accomplished by using an asynchronous version of the relative value iteration along with system simulation. The simulation provides, at every state transition epoch, the new state whose value is updated. The values of the other states are kept the same.

2.3 Asynchronous Relative Value Iteration

1. Select $v^0 \in V$, choose any $k^* \in \mathcal{E}$, specify $\epsilon > 0$, and set $m = 0$.

2. Let the system state (obtained from simulation) at the m^{th} decision making epoch be $i \in \mathcal{E}$. Compute $v^{m+1}(i)$ by

$$v^{m+1}(i) = \max_{a \in \mathcal{A}_i} \left\{ r(i,a) - v^m(k^*) + \sum_{j \in \mathcal{E}} P_{ij}(a) v^m(j) \right\}.$$

3. Update the values at the $(m+1)^{st}$ epoch for all states $j \in \mathcal{E}$ as follows:

$$v^{m+1}(j) = \begin{cases} v^{m+1}(i) & \text{if } j = i \\ v^m(j) & \text{if } j \neq i \end{cases}$$

Steps 4 and 5 are same as the steps 3 and 4 of the relative value iteration algorithm.

It can be shown very easily that this algorithm can diverge, a learning based version of this algorithm has no known counter example showing divergence (Abounadi [1]). In the learning based version, the step (2) of the asynchronous relative value iteration will be as follows:

$$v^{m+1}(i) = (1-\alpha) v^m(i) + \alpha \max_{a \in \mathcal{A}_i} \left\{ r(i,a) - v^m(k^*) + \sum_{j \in \mathcal{E}} P_{ij}(a) v^m(j) \right\},$$

where α is the learning rate with a small starting value which is decayed suitably to zero. Many variants of the relative value iteration algorithm, given above, have been used to develop reinforcement learning algorithms (Abounadi [2], Bertsekas and Tsitsiklis [5]).

3 Reinforcement Learning

Reinforcement learning (RL) is a way of teaching agents (decision makers) optimal control policies. This is accomplished by assigning rewards and punishments for their actions based on the temporal feedback obtained during active interactions of the learning agents with dynamic systems. Such an incremental learning procedure specialized for prediction and control problems was developed by Sutton [28] and is referred to as *temporal-difference (TD)* methods.

Any learning model basically contains 4 elements which are the environment, the learning agent, a set of actions, and the environmental response (sensory input). The learning agent selects an action for the system which leads the system along a unique path till the system encounters another decision-making state. At that time, the system needs to consult with the learning agent for the next action. During a state transition, the agent gathers sensory inputs from the environment, and from it derives information about the new state, immediate reward, and the time spent during the state-transition. Using this information and the algorithm, the agent updates its knowledge base and selects the next action. This completes one step in the iteration process. As this process repeats, the learning agent continues to improve its performance.

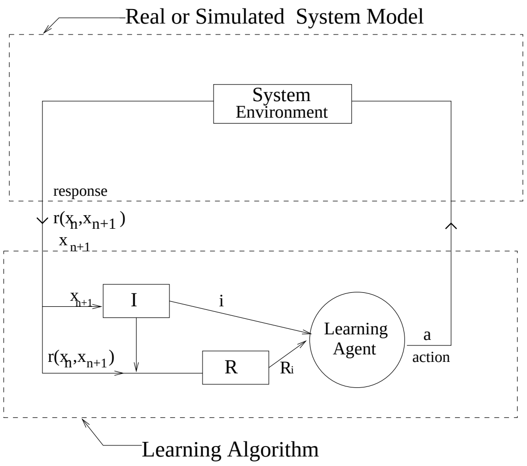

Figure 1: A Reinforcement Learning Model.

A reinforcement learning model is depicted in Figure 1. On the n^{th} step of interaction, based on the system state $x_n = i$ and the action values $R(i) = \{R(i,k) : k \in \mathcal{A}_i\}$ for the k available actions, the agent takes an action a. The system evolves stochastically in response to the input state-action (i, a) pair, and results in responses concerning the next system state x_{n+1} and the reward (or punishment) $r(x_n, x_{n+1})$ obtained during the transition. These system responses serve as the sensory inputs for the agent. From the input x_{n+1}, the function I helps the agent in perceiving the new system state. (The perceived new state could be different from x_{n+1} only for partially observable processes.) Using the information about the new state (from I) and the sensory input $r(x_n, x_{n+1})$, the reinforcement function R accomplishes the following: 1) calculates the new action value $R(i, a)$ for the previous state action pair $(x_n = i,\ A_n = a)$, and 2) updates the value function for action a. For a greedy policy, at any state i, the decision maker chooses the action with the highest (or lowest for minimization) action value.

Initially, the action values for all state-action pairs are assigned arbitrary equal values (e.g., zeros). When a system visits a state for the first time, a random action gets selected because all the action values are equal. As the system revisits the state, the learning agent selects the action based on the current action values which are no longer equal. For ergodic processes, the states continue to be revisited and consequently the agent gets many opportunities to refine the action values and the corresponding decision making process. Sometimes, the learning agent chooses an action other than that suggested by the greedy policy. This is called exploration. As the good actions are rewarded and the bad actions are punished over time, for every state, the action values of a smaller subset (one or more) of all the available actions become dominant. The learning phase ends when a clear trend appears, and the dominant actions constitute the decision policy vector.

A detailed account of average-reward reinforcement learning (ARL) can be found in Mahadevan [18]. For an overview of work in reinforcement learning using the discounted criterion, see the excellent survey paper by Kaelbling [15].

3.1 Model-free RL

Note that both the synchronous and the asynchronous algorithms for value iteration are model based, since they require the one step transition probabilities ($P_{ij}(a)$'s, for all $i, j \in \mathcal{E}$ and $a \in \mathcal{A}$). The model based RL algorithms estimate the transition probabilities from sample paths produced during system simulation (which, as mentioned in Section 1, requires only the probability laws governing the random variables associated with the system). These probabilities are used to estimate the action values using variants of asynchronous value iteration (e.g., Sutton [31]). A key strength of RL is that it can also be applied to infer optimal policies without the need for transition probabilities; such algorithms are referred to as model-free algorithms. The model-free algorithms can infer the action values for all state-action pairs (on which the policy depends) directly from the sample paths generated by simulation. Some applications of such model-free methods can be found in studies, such as controlling a team of elevators in a multi-storey building (Crites and Barto [8]), NASA space-shuttle job-shop scheduling (Zhang and Dietterich [37]), and dynamic channel allocation in cellular telephone systems (Singh and Bertsekas [26]).

A problem with reinforcement learning methods is *how to assign temporal credit* (utility value or reinforcement) to an *action* that might have far reaching effects. It is often practically impossible to wait infinitely long to truly assess the credit of an action. Using the insights from value iteration, RL algorithms assign credit to a state-action pair based on the immediate reward and the action value of the next state. A subset of the temporal difference algorithms, referred to as $TD(\lambda)$ in RL literature (Sutton and Barto [27]), forms a class of $TD(0)$ algorithms, where $\lambda = 0$ indicates a single step updating. In TD(0) algorithms, assigning *utility value* (reinforcement) to an action is done based on one step performance only. We describe the action value updating rule of a infinite horizon discounted MDP (Sutton [30]) to build intuition for the value updating part of the SMART algorithm described later. Suppose that when action a is chosen in state i, it results in immediate reward $r_{imm}(i,a,j)$ and the system makes a transition to state j. Then, value updating for the state-action pair (i,a) is done as follows:

$$R_{new}(i,a) \; = R_{old}(i,a) + \alpha \left[r_{imm}(i,a,j) + \gamma \max_b R(j,b) - R_{old}(i,a) \right],$$

where γ is the discounting factor and α is the learning rate. It may be noted that the first two terms within the square brackets of the above equation constitute an estimate of the action value for taking action a in state i. This approach of assessing action value from a single sample path is known as *Robbins-Monro* stochastic approximation (Robbins and Monro [22]). We also note that, the component within square brackets of the updating equation (i.e., newly assessed action value minus the old action value of a state-action pair) denotes the one step temporal difference $TD(0)$, of which a small (α) fraction is added to the old action value to get the new action value.

The above updating equation can be rewritten as

$$R_{new}(i,a) \ = (1-\alpha)R_{old}(i,a) + \alpha \left[r_{imm}(i,a,j) + \gamma \max_b R(j,b) \right].$$

The SMART algorithm, which is developed next, is based on the average reward Bellman equation, and is similar in spirit to the $TD(0)$ approach in its action value updating.

3.2 SMART: A Model-Free Average Reward RL Algorithm for SMDPs

Let $R^*(i,a)$ be the expected average adjusted value of taking action a in state i, and then continuing infinitely by choosing actions optimally. Then $R^*(i)$, the value of state i when the initial action is also optimal, can be written as $R^*(i) = \max_a R^*(i,a)$. The average reward Bellman equation for SMDPs (5) can be rewritten in action value form as

$$R^*(i,a) = r(i,a) - g^* y(i,a) + \sum_{j \in \mathcal{E}} P_{ij}(a) \max_{b \in \mathcal{A}_j} R^*(j,b), \quad \forall i \in \mathcal{E}. \tag{6}$$

Then, we have $\pi^*(i) = \text{argmax}_a R^*(i,a)$ as an optimal policy. Note that, since the $R(\cdot,\cdot)$ function makes the action explicit, the SMART algorithm estimates R values on-line using a temporal difference method, and then uses them to define a policy. The value of the state-action pair (i,a) visited at the m^{th} decision making epoch is updated by SMART as follows.

Let simulation of action a in state i result in a system transition to state j at the subsequent decision epoch, then we can write that

$$R_{new}(i,a) = (1-\alpha_m)R_{old}(i,a) + \alpha_m \left\{ r_{imm}(i,j,a) - g_m \tau(i,j,a) + \max_b R_{old}(j,b) \right\}, \quad (7)$$

where $r_{imm}(i,j,a)$ is the actual cumulative reward earned between two successive decision epochs starting in state i (with action a) and ending in state j respectively. $\tau(i,j,a)$ is the actual sojourn time between the decision epochs, and α_m is a learning rate parameter for updating of the action value of a state-action pair of the m^{th} decision epoch. Note that g_m is the reward rate (or, gain, as defined in (4)), which is estimated by taking the ratio of the total reward earned and the total simulation time till the m^{th} decision making epoch, as shown below.

$$g_m = \frac{\sum_{k=0}^{m} r_{imm}(X_k, X_{k+1}, A_k)}{\sum_{k=0}^{m} \tau(X_k, X_{k+1}, A_k)} = \frac{c_m}{t_m}. \quad (8)$$

Details of the algorithm are given in Figure 2. A small percentage of the time, decisions other than that with the highest reinforcement value were taken. This is referred to in the RL literature as *exploration* (Bertsekas and Tsitsiklis [5]). Exploration plays an important role in ensuring that all the states of the underlying Markov chain are visited by the system and all the potentially beneficial actions in each state are tried out. When all the system states are visited, the Markov chain remains irreducible. It is also required that each state is visited infinitely often, which is a precondition for asymptotic convergence of the learning algorithms. The learning rate α_m and the exploration rate p_m are both decayed slowly to 0 according to a Darken-Chang-Moody search-then-converge procedure ([9]) as follows. Note that, in the expression below, Θ can be substituted by α and p for learning and exploration respectively. $\Theta_m = \frac{\Theta_0}{1+u}$, where $u = \frac{m^2}{\Theta_\tau + m}$, where Θ_0 and Θ_τ are constants.

3.2.1 Reinforcement Value Function Approximation in SMART

In most real life SMDPs, the number of states is usually so large as to rule out tabular representation of the reinforcement action values. Following [8], we used a feed-forward net

ALGORITHM:

1. Let time step $m = 0$. Initialize action values $R_{old}(i, a) = R_{new}(i, a) = 0$ for all $i \in \mathcal{E}$ and $a \in \mathcal{A}_i$. Set the cumulative reward $c_{new} = 0$, the total time be $t_{new} = 0$, and the reward rate $g_{new} = 0$. Also, initialize the input parameters for the Darken-Chang-Moody (DCM) scheme (α_0, α_τ, p_0, and p_τ). Start system simulation.

2. While $m <$ MAX_STEPS do

   If the system state at the time step m is $i \in \mathcal{E}$,

   (a) Calculate p_m and α_m using the DCM scheme.

   (b) With high probability $1 - p_m$, simulate an action $a \in \mathcal{A}_i$ that maximizes $R_{new}(i, a)$, otherwise choose a random (exploratory) action from the set $\{\mathcal{A}_i \setminus a\}$.

   (c) Simulate the chosen action. Let the system state at the next decision epoch be j. Also let $\tau(i, j, a)$ be the transition time, and $r_{imm}(i, j, a)$ be the immediate reward earned as a result of taking action a in state i.

   (d) Find $R_{new}(i, a)$ using :

$$
\begin{aligned}
R_{new}(i,a) = \quad & (1-\alpha_m)R_{old}(i,a) + \alpha_m \{r_{imm}(i,j,a) \\
& -g_{new}\tau(i,j,a) + \max_b R_{old}(j,b)\}.
\end{aligned}
$$

   (e) In case a nonexploratory action was chosen in step 2(a)

   - Update total reward $c_{new} \longleftarrow c_{new} + r_{imm}(i, j, a)$
   - Update total time $t_{new} \longleftarrow t_{new} + \tau(i, j, a)$
   - Update average reward $g_{new} \longleftarrow \frac{c_{new}}{t_{new}}$

   Else, go to Step (f).

   (f) Set $R_{old}(i, a) \longleftarrow R_{new}(i, a)$

   (g) Set current state i to new state j, and $m \leftarrow m + 1$.

Figure 2: SMART: A model-free algorithm for computing gain-optimal policies for SMDPs.

to represent the action value function. Equation (7) used in SMART is replaced by a step which involves updating the weights of the net. So after each action choice, in step 2(c) of the algorithm, the weights of the corresponding action net are updated, using the well known backpropagation algorithm, as follows:

$$\Delta\phi = \alpha_m \epsilon(i, j, a, r_{imm}(i, j, a), \phi) \bigtriangledown_\phi R_m(i, a, \phi), \tag{9}$$

where $\epsilon(i, j, a, r_{imm}(i, j, a), \phi)$ is the temporal difference error

$$\left[r_{imm}(i, j, a) - g_{new}\tau(i, j, a) + \max_b R_m(j, b, \phi) - R_m(i, a, \phi) \right].$$

The symbol ϕ denotes the vector of weights of the net, α_m is the learning rate used at the m^{th} iteration, and $\bigtriangledown_\phi R_m(i, a, \phi)$ is the vector of partial derivatives of $R_m(i, a, \phi)$ with respect to each component of ϕ.

4 Case Study: Preventive Maintenance of a Production Inventory System

We first consider an unreliable discrete part production inventory system with limited state space, that is modeled as a SMDP (Das and Sarkar [10]). Numerical results for the problem obtained from both the theoretical model and SMART are presented and compared. For the expanded problem, we use results from two well designed heuristic approaches to compare with SMART results.

4.1 Single product inventory system

The production inventory system considered here produces a single product type to satisfy an exogenous demand process. To hedge against the uncertainties in both the production and the demand processes, provision for a inventory buffer of the product between the system and the demands is kept. Demands that arrive when the inventory buffer is empty are not backordered and are, therefore, lost. (The assumption here is that the customers are not loyal and thus use alternate manufacturing facilities to meet their needs.)

It seems logical that some form of preventive maintenance of the system may improve the performance of the production process by avoiding unscheduled and usually long interruptions caused by the system failures. However, since preventive maintenance also interrupts the production process, the important question is: *should the system be maintained? If so, how often?* The issue of determining the optimal level of preventive maintenance that maximizes the system performance is addressed here. The measure of system performance considered for optimization is the average reward G (\$/unit time), which is a function of the service level (% of satisfied demands), numbers of repair and maintenance, and the cost parameters of the system. The average reward rate is obtained as the rate of revenue earned from demand serviced minus the repair cost rate and the maintenance cost rate. This performance measure is used as the basis to optimally determine the maintenance criteria, which can be stated as: *if the current inventory is i and the production count (number of products made since the last repair/maintenance) is at least N_i, then maintain the machine.*

4.1.1 Numerical Results on the Single Product Problem

A fairly general numerical example problem is considered as a vehicle for obtaining optimal results that can be used to compare with the results obtained from the SMART algorithm. The characteristics and the input parameters of the example problem are as follows. 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 buffer inventory is maintained by an ($S = 3$, $s = 2$) policy.

System	Time Bet. Demands (γ)	Time Bet. Failure (κ, μ)	Time bet. Product. (d, λ)	Maint. Time (a, b)	Repair Time (r, δ)
1	1/10	(8, 0.08)	(8, 0.8)	(5, 20)	(2, 0.01)
2	1/10	**(8, 0.008)**	(8, 0.8)	(5, 20)	(2, 0.01)
3	**1/7**	(8, 0.08)	(8, 0.8)	(5, 20)	(2, 0.01)
4	**1/15**	(8, 0.08)	(8, 0.8)	(5, 20)	(2, 0.01)
5	**1/15**	(8, 0.08)	(8, 0.8)	**(25, 40)**	(2,0.01)
6	**1/15**	(8, 0.08)	(8, 0.8)	(5, 20)	**(2, 0.02)**
7	**1/15**	**(8, 0.08)**	(8, 0.8)	(5, 20)	**(4, 0.02)**
8	**1/15**	**(8, 0.01)**	(8, 0.8)	(5, 20)	**(4, 0.02)**
9	**1/20**	**(8, 0.04)**	**(8, 0.4)**	(5, 20)	**(4, 0.02)**

Table 1: Input parameters for nine sample numerical examples with single product

The system stops production when the buffer inventory reaches S (such a period of *inaction* is often referred to as *server vacation* in queueing literature), and the production resumes when the inventory drops to s. During its vacation, the system stays in cold standby mode and does not age or fail. Preventive maintenance decision is made only at the completion epoch of a part production.

The following values of the input parameters were used: $\gamma = 0.10$, $d = 8.0, \lambda = 0.80$, $k = 8.0, \mu = 0.80$, $a = 5.0, b = 20.0$, $r = 2.0, \delta = 0.01$. Table 1 shows various other combinations of the input parameters. For each of the nine parameter sets (shown in Table 1), the cost parameters considered were: C_d (net revenue per demand serviced) = \$1, C_r (cost per repair) = \$5, C_m (cost per maintenance) = \$2.

The optimal results (average reward) for the single product inventory system obtained numerically from the theoretical model and SMART are summarized in Table 2. Results clearly show that the reinforcement learning algorithm (SMART) performs extremely well (the mean values lie within 4% of the optimal values).

$C_d = 1, C_r = 5, C_m = 2$		
System	Optimal Average Reward	RL Average Reward Mean$\pm$3(std. dev.)
1	0.034	$0.033\pm1.25\times10^{-3}$
2	0.076	$0.073\pm1.35\times10^{-3}$
3	0.035	$0.036\pm3.5\times10^{-4}$
4	0.028	$0.028\pm4.0\times10^{-4}$
5	0.025	$0.025\pm3.5\times10^{-4}$
6	0.031	$0.03\pm2.0\times10^{-4}$
7	0.028	$0.028\pm5.5\times10^{-4}$
8	0.057	$0.057\pm3.5\times10^{-4}$
9	0.020	$0.020\pm5.5\times10^{-4}$

Table 2: Results of RL algorithm. The results were averaged over 40 runs, where each simulation run lasted for 1.0 million time units.

4.2 Multiple product inventory problem

To further demonstrate the power of the SMART algorithm, we extended the system to accommodate five (5) different products each having its own separate $(S_i, s_i : i = 1, \cdots, 5)$ inventory buffer. Each product has its separate Poisson demand arrival process, and random production times. The service policy used is as follows. The machine continues to produce a product type until the buffer reaches level S_i. At that time the machine has two options, either to begin production of another product type, or to take a vacation (period of no production). The first option is selected if one or more product buffers are at or below their s_i levels. When more than one buffer needs replenishment, buffer with the lowest index (i) is given priority. The machine vacation ends as soon as any of the buffer levels drop to s_i.

The state of the system is an 8-tuple consisting of the buffer levels of all five products, the age of the production system since the last maintenance/repair, machine operating condition, and the product type which the machine is currently setup for. Hence the cardinality of the state space for the five product problem can be approximated by $|\mathcal{E}| = (5N_s \prod_{i=1}^{5}(S_i+1)+5)\times|\Re^+|$, where $N_s = 4$ indicates the number of operating status of the machine (e.g., on vacation,

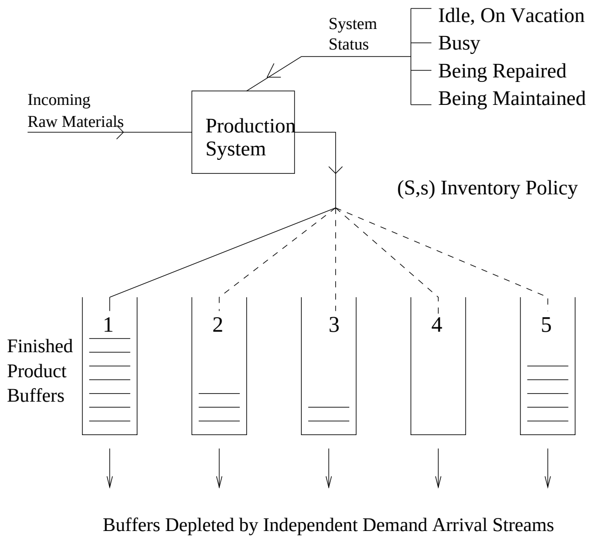

Figure 3: A production-inventory system with five product types.

working, under maintenance, and under repair). $|\Re^+|$ represents the state space of the system age. Hence, for a problem with maximum buffer sizes (S_i for $i = 1, \cdots, 5$) of 30, 20, 15, 15, and 10, the state space is approximately $7 \times 10^7 \times |\Re^+|$. If $|\Re^+|$ is approximated by 10^3 intervals, then the state space is nearly 10^{11}.

4.2.1 Input Parameters

Ten different numerical variants of the five product problem, that are given in Table 3, were considered to study SMART performance. The variations among the parameter values are highlighted in the table containing the parameters. For all of the 10 systems, product related parameters were kept the same (Table 4).

4.2.2 SMART Implementation

The production inventory system was simulated, using a discrete event simulator (CSIM), where the significant events were: demand arrival, failure, production completion, repair completion, and maintenance completion. Except for demand arrival, all the remaining event epochs were considered to be decision epochs.

The size of the state space of this system is large enough to rule out explicit storage of the action values for each action-state pair. Hence, two multi-layer feedforward neural nets were used to approximate the action values for each action ("resume production/vacation" or "start maintenance"). Baird [3] showed that, instead of gradient descent based neural network, a residual gradient method should be used for RL applications. However, for our problem, gradient descent based method converged for all the cases studied. We experimented with a number of different strategies for encoding the state to the neural net. The approach that produced the best results used a "thermometer" encoding with 41 input units. The status of each buffer was encoded using 4 input units, where 3 units were successively turned on in equally sized steps, and the last one was used to handle the "excess" value. Similarly, the "age" of the machine (actual simulation time since last renewal) was encoded using 21 input units. Here, the age was discretized in steps of 30, with the last unit being turned on proportional to the excess value beyond 600. The constants used in the Darken-Chang-Moody decaying scheme for the learning and exploration rates were chosen as $\alpha_0 = p_0 = 0.1$, and $\alpha_\tau = p_\tau = 10^{11}$.

4.2.3 Heuristic Methods

Two different heuristic algorithms were developed in order to benchmark the performance of the SMART algorithm on the five product inventory problem. We describe the heuristics next.

COR Heuristic

The COR (coefficient of operational readiness) heuristic, which was adopted from Gertsbakh [12], is as follows. Let T be the operational time after which the system is preventively maintained. Then the expected number of failures (k) occurring over the interval T of operation time can be shown to not exceed $\frac{F(T)}{(1-F(T))}$, where $F(\cdot)$ is the cumulative probability distribution of the time to failure of the machine. Define a maintenance cycle time as $CT = T + kT_r + T_m$, where T_r and T_m are the mean time required for repair and maintenance respectively. Then the operational availability of the machine in a maintenance cycle is given as COR $= \frac{T}{CT}$. The value of T that maximizes COR is selected. An assumption made in this heuristic is that repairs carried out after failures are of minimal type (i.e., repairs do not change the age of the system). Only after maintenance (done after the system has operated for a time T) the system age is reset to zero.

Age Replacement (AR) Heuristic

The age replacement (AR) heuristic, that we have developed based on the renewal reward theorem, can be given as follows. The system is maintained when it reaches age T. If a failure occurs before time T, it is repaired. Both repair and maintenance renew the machine (i.e., resets its age to zero). Let C_m and C_r denote the cost of maintenance and repair respectively. Then the renewal cycle time CT is given as

$$CT = \int_0^T (x + T_r) f(x) dx + \int_T^\infty (T + T_m) f(x) dx,$$

where $f(x)$ is the probability density function of the time to machine failure (X). The expected cost (EC) of a renewal cycle is given as $EC = P(X > T)C_m + P(X < T)C_r$. Then the average cost (ρ), as per renewal reward theorem, is given as: $\rho = \frac{EC}{CT}$. Maintenance age T is selected such that ρ is minimized. For the numerical example problems, a gradient unconstrained optimization algorithm (available within MATLAB software) was used to obtain the maintenance interval (T).

System	Time Bet. Failure (κ, μ)	Repair Time (r, δ)	Maint. Time (a, b)	Cost of Maint.
1	(6, 0.02)	(2, 0.04)	(20, 40)	500
2	(6, 0.02)	(2, 0.04)	**(10, 30)**	500
3	(6, 0.02)	(2, 0.04)	(10, 30)	**550**
4	(6, 0.02)	(2, 0.04)	(10, 30)	**600**
5	(6, 0.02)	(2, 0.04)	(10, 30)	**650**
6	(6, 0.02)	(2, 0.04)	(10, 30)	**750**
7	(6, 0.02)	(2, 0.04)	(10, 30)	**800**
8	(6, 0.02)	(2, 0.04)	(10, 30)	**900**
9	(6, 0.02)	(2, 0.04)	(10, 30)	**1100**
10	(6, 0.02)	(2, 0.04)	(10, 30)	**1200**

Table 3: Production inventory system parameters. In all these systems, the repair cost was fixed at \$5000.

Product	Prod. Time (d, λ)	Demand Arrival (γ)	Buffer Size	Unit Reward
1	(8,8/1)	1/6	30	9
2	(8,8/2)	1/9	20	7
3	(8,8/3)	1/21	15	16
4	(8,8/4)	1/26	15	20
5	(8, 8/5)	1/30	10	25

Table 4: Production inventory product parameters

4.2.4 Results and Analysis

Tables 3 and 4 give the input parameters for the ten numerical example problems. SMART was trained on all systems for five million decision epochs. In most cases, the learning converged in much less time. After training, the weights of the neural nets were fixed, and the simulation was rerun using the nets (with fixed weights) to choose actions without any exploration or learning. The results (average rewards) were averaged over thirty runs, each of which ran for 2.5×10^6 time units of simulation time, or, approximately 1 million decision epochs.

Figure 4 shows the learning curve for the system #1 of Table 3, and also a plot showing the performance of the SMART agent (after learning) versus the AR and COR heuristics. Table 5 compares the average reward rate accrued by the policy learned by SMART versus that for the COR and AR heuristics for all 10 systems. A negative average reward implies cost. In all cases, SMART produced significantly [1] better results than both heuristics.

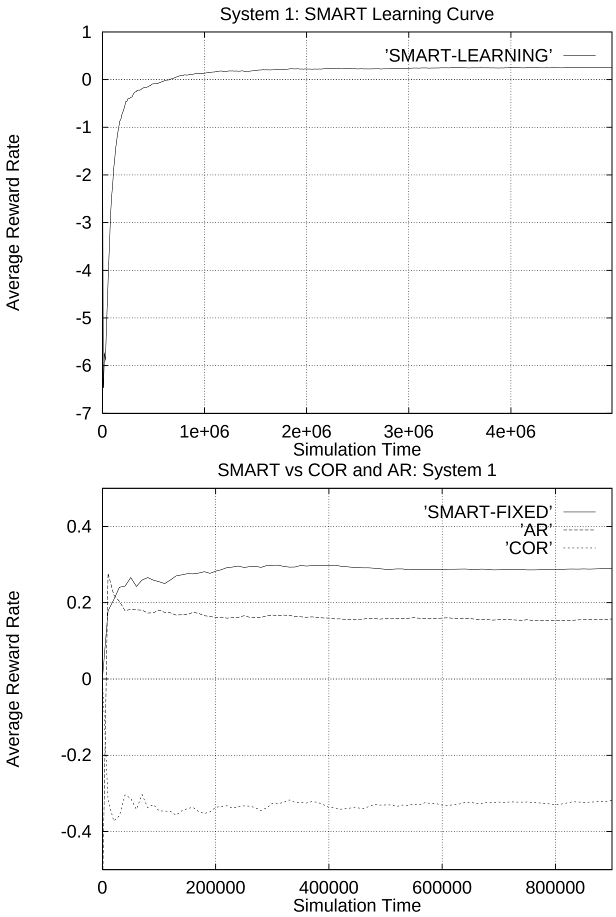

Figure 4: Results for production-inventory system 1. The graph on the top shows the SMART learning curve. The graph on the bottom shows the median-averaged cost rate, over 30 runs, incurred by the (fixed) policy learned by SMART, compared to the COR and AR heuristics.

[1]The differences are all significant at the 95% confidence level using a Wilcoxon test.

System	COR	AR	SMART
1	-0.31	0.16	0.28
2	0.09	0.22	0.35
3	-0.2	-0.06	-0.03
4	-0.45	-0.35	-0.26
5	-0.7	-0.61	-0.45
6	-1.2	-1.14	-0.95
7	-1.5	-1.37	-1.2
8	-2.0	-1.9	-1.7
9	-3.0	-2.7	-2.5
10	-3.7	-3.5	-2.9

Table 5: Comparison of the average reward rate incurred by SMART vs. AR and COR heuristics for the five product production inventory problem.

4.2.5 Sensitivity Analysis

Since there are several factors that seem to affect the maintenance decision, it is imperative for the maintenance planner to know what factors are most critical to influencing the average reward. To address this issue, we designed a simple fractional factorial experiment (Taguchi's L8 array) [33] for analyzing the variances resulting from the key factors affecting the average reward. We studied four factors, namely maintenance time, fixed cost of maintenance, repair time, and fixed cost of repair. The L8 experiment allows up to four main factors and three interactions to be tested. We only tested the above four main factors and used the other columns to estimate the error. The two levels of the four factors that were considered are given in Table 6. Other parameters of the problem were kept the same as for system 1 of Table 3. At 90 per cent confidence level, only the fixed cost of maintenance was found to be significant in affecting the response (average reward). The analysis of variance (ANOVA) is shown in Table 7.

As a follow up of the variance analysis, we studied the sensitivities of the SMART, AR, and COR algorithms with respect to the fixed cost of maintenance. The sensitivities were measured via factors, namely 1) total number of maintenance actions, 2) total number of

Factor	Level 1	Level 2
Fixed Cost of Maintenance	\$500	\$750
Maintenance Time Parameters	(20-40)	(10-30)
Fixed Cost of Repair	\$5000	\$3000
Repair Time Parameters	(2,0.04)	(2,0.02)

Table 6: Levels of the factors used in variance analysis using L8 array

Factors	Sums of Squares	$F_{calculated}$	Decision
Maintenance Cost	1.5506	9.5071	Significant
Maintenance Time	0.0212	0.1301	Not Significant
Repair Cost	0.7812	4.7902	Not Significant
Repair Time	0.0001	0.0005	Not Significant
Error	0.1631		

Table 7: ANOVA: Analysis is performed at 90 percent confidence where $F_{critical}$ is 5.5383

failures, and 3) total vacation time of the system. For system 2 of Table 3, maintenance cost was varied from a low value of \$500 to a high value of \$1200 (systems 2 through 10 of Table 3). Figure 5 shows the sensitivity plots. The COR heuristic algorithm does not consider cost at all, hence its performance did not vary with maintenance cost. This insensitivity is the primary reason for the lackluster performance of the COR heuristic (as shown in the bottom graph in Figure 4). The AR algorithm shows a linear trend. The SMART algorithm shows an adaptive trend, as expected, which partly explains its superior performance (shown in Figure 4).

5 Concluding Remarks

This paper presents an average reward reinforcement learning algorithm (SMART) that finds near optimal policies for semi-Markov decision problems. An earlier version of this algorithm with some preliminary results originally appeared in Mahadevan et al. [19]. SMART was

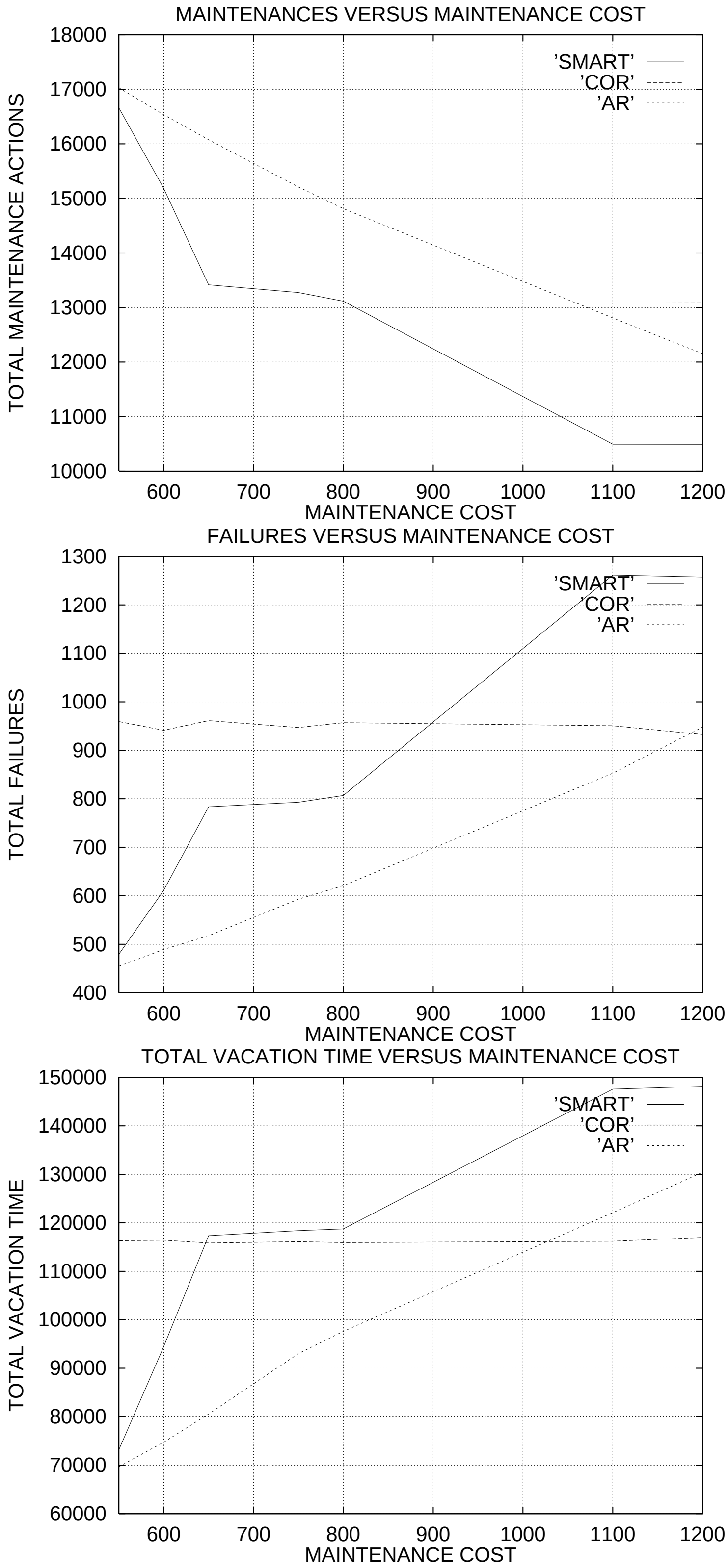

Figure 5: Sensitivity of AR, COR, and SMART to the maintenance cost

implemented and tested on a commercial discrete-event simulator, CSIM. The effectiveness of the SMART algorithm was demonstrated by first comparing its results with optimal results (for a small problem), and then applying it to a large scale problem and comparing its results with those obtained from two good heuristic methods. The results obtained are quite encouraging, and, to our knowledge, this study is the first large-scale implementation of average-reward reinforcement learning. This work is part of a major ongoing cross-disciplinary study of industrial process optimization using reinforcement learning. For more complex systems consisting of numerous inter-related subsystems of machines [11], instead of having a single agent governing the whole system, it may be more appropriate to design a hierarchical control system where each subsystem is controlled using separate agents. The elevator problem [8] is a simplified example of such a multi-agent system, where the agents are homogeneous and control identical subsystems. Global optimization in a system consisting of heterogeneous agents poses a significantly challenging problem. Sethi and Zhang [25] present an in-depth theoretical study of hierarchical optimization for complex factories. We are currently exploring such hierarchical extensions of SMART for more complex factory processes, such as jobshops and flowshops. An application of SMART on a transfer line can be found in Mahadevan and Theocharous [20]. Our recent work also includes development of a TD(λ) extension of the SMART algorithm (named λ-SMART) and its implementation on a dynamic seat allocation problem faced by the airline industries (Gosavi et al. [13]).

Acknowledgements

This research is supported in part by NSF Grant # DMI-9500289 (to Tapas K. Das) and NSF CAREER Award Grant # IRI-9501852 (to Sridhar Mahadevan).

References

[1] J. Abounadi. Personal Communication via email.

[2] J. Abounadi. Stochastic approximation for non-expansive maps: Application to q-learning algorithms. Unpublished Ph.D. Thesis, MIT, Department of Electrical Engineering and Computer Science, February, 1998.

[3] L. C. Baird. Residual algorithms: Reinforcement learning with function approximation. In *Proceeding of the Twelfth International Conference on Machine Learning*. 1995.

[4] R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[5] D. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

[6] D. Blackwell. Discrete dynamic programming. *Ann. Math. Stat.*, 33:226–235, 1965.

[7] 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, 1995.

[8] R. Crites and A. Barto. Improving elevator performance using reinforcement learning. In *Neural Information Processing Systems (NIPS)*. 1996.

[9] 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.

[10] T. Das and S. Sarkar. Optimal preventive maintenance in a production inventory system. Revision submitted.

[11] S. Gershwin. *Manufacturing Systems Engineering.* Prentice Hall, 1994.

[12] I.B. Gertsbakh. *Models of Preventive Maintenance.* North-Holland, Amsterdam, Netherlands, 1976.

[13] A. Gosavi, N. Bandla, and T. K. Das. Dynamic airline seat allocation among different fare classes. Working paper, Department of Industrial & Management Systems Engineering, University of South Florida, Tampa, Fl 33620.

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

[15] L.P. Kaelbling, M.L. Littman, and A.W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4, 1996.

[16] S. Karlin. The structure of dynamic programming models. *Naval Res. Log. Quart.*, 2:285–294, 1955.

[17] A. M. Law and W. D. Kelton. *Simulation Modeling and Analysis.* McGraw Hill, Inc., New York, NY, 1991.

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

[19] S. Mahadevan, N. Marchalleck, T. K. Das, and A. Gosavi. Self-improving factory simulation using continuous-time average-reward reinforcement learning. In *Proceeding of the Thirteenth International Conference on Machine Learning, pp. 202-210, Morgan Kaufman Publ.* 1996.

[20] S. Mahadevan and G. theocharous. Optimizing production manufacturing using reinforcement learning. In *Proceedings of the Eleventh International FLAIRS Conference, AAAI Press, pp. 372-377.* 1998.

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

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

[23] S. M. Ross. *Stochastic Processes.* John Wiley Sons, New York, NY, 1983.

[24] B. Van Roy, D. P. Bertsekas, Y. Lee, and J. N. Tsitsikilis. *A Neuro-Dynamic Programming Approach to Retailer Inventory Management.* MIT Technical Report, Massachusetts Institute of Technology, 1997.

[25] S. Sethi and Q. Zhang. *Hierarchical Decision Making in Stochastic Manufacturing Systems.* Birkhauser, 1994.

[26] S. Singh and D. Bertsekas. Reinforcement learning for dynamic channel allocation in cellular telephone systems. In *Neural Information Processing Systems (NIPS).* 1996.

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

[28] R.S. Sutton. *Temporal Credit Assignment in Reinforcement Learning.* PhD thesis, University of Massachusetts, Amherst, MA, May 1984.

[29] R.S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[30] R.S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the 7th International Workshop on Machine Learning*, pages 216–224. Morgan Kaufmann, San Mateo, CA, 1990.

[31] R.S. Sutton. DYNA, an integrated architecture for learning, planning, and reacting. In *Working Notes of the AAAI Spring Symposium on Integrated Intelligent Architectures.* 1991.

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

[33] G. Taguchi. *System of Experimental Design, Volume 1.* Kraus International Publications, 1994.

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

[35] R. Wheeler and K. Narenda. Decentralized learning in finite markov chains. *IEEE Transactions on Automatic Control*, 31(6):373–376, 1986.

[36] D. J. White. Dynamic programming, markov chains, and the method of successive approximations. *J. Math. Anal. Appl.*, 6:373–376, 1963.

[37] W. Zhang and T. Dietterich. A reinforcement learning approach to job-shop scheduling. In *Proceedings of the 13th IJCAI.* 1995.