A Simulation-Based Learning Automata Framework for Solving Semi-Markov Decision Problems Under Long-Run Average Reward

Abhijit Gosavi
Department of Industrial Engineering
State University of New York at Buffalo,
Buffalo, NY 14260
gosavi@eng.buffalo.edu
Tapas K. Das
Department of Industrial and Management Systems Engineering
University of South Florida, Tampa, FL 33620
das@eng.usf.edu
Sudeep Sarkar
Department of Computer Science
University of South Florida, Tampa, FL 33620
sarkar@bigpine.csee.usf.edu

October 7, 2002

Abstract. Many problems of sequential decision-making under uncertainty, whose underlying probabilistic structure has a Markov chain, can be set up as Markov decision problems (MDPs). However, when their underlying transition mechanism cannot be characterized by the Markov chain alone, the problems may be set up as *semi*-Markov decision problems (SMDPs). The framework of dynamic programming has been used extensively in the literature to solve such problems. An alternative framework that exists in the literature is that of the learning automata (LA). This framework can be combined with simulation to develop convergent LA algorithms for solving MDPs under long-run cost (or reward). A very attractive feature of this framework is that it avoids a major stumbling block of dynamic programming — that of having to compute the one-step transition probability matrices of the Markov chain for every possible action of the decision-making process. In this paper, we extend this framework to the more general SMDP. We also present numerical results on a case study from the domain of preventive maintenance in which the decision-making problem is modeled as an SMDP. An algorithm based on LA theory is employed, which may be implemented in a simulator as a solution method. It produces satisfactory results in all the numerical examples studied.

1 Introduction

A large class of problems of sequential decision-making under uncertainty, whose underlying probability structure has a controlled Markov chain, can be formulated as Markov decision problems (MDPs). The objective underlying the study of these problems is to find the optimal action (control) in each state of the Markov chain. Optimality of actions is usually measured with respect to some performance metric such as the total discounted reward over an infinite time horizon, or the average reward over an infinite time horizon (also called the long-run average reward). In such a characterization (i.e., MDP), the time spent in the transition from one state to another is either unity or else it is ignored (and assumed to be unity) in the calculation of the performance metric. However, problems in which we cannot ignore the transition times (which may be deterministic or random variables) can often be solved using the semi-Markov decision model. This paper is centered on a simulation-based technique related to the semi-Markov model.

The framework of dynamic programming (DP) has been used extensively in the literature to solve Markov and semi-Markov decision problems (see Puterman, 1994). A stochastic-approximation method called reinforcement learning, which is also based on the framework of DP (see Bertsekas and Tsitsiklis, 1996 and Sutton and Barto, 1998 for textbook treatments), has also appeared in the literature in recent years.

The theory of learning automata (LA) (see Narendra and Thatachar, 1989) forms an alternative framework for solving MDPs — a framework that is very distinct from that of DP. It is rooted in the principles of game theory. The LA framework, which is simulation-based, can be used to develop convergent algorithms to solve MDPs under long-run cost (or reward) (see Wheeler and Narendra, 1986). A very attractive feature of this framework, unlike traditional DP, is its ability to generate near-optimal solutions without having to compute the one-step transition probability matrices of the decision-making problem. In this paper, we present an extension of the LA framework to the more general semi-Markov decision problem (SMDP). We also conduct numerical tests on a decision-making problem given in a paper by Das and Sarkar (1999). Their paper contains results for the optimal policy for preventive maintenance (PM) of a production-inventory system. They obtain the optimal policy via the harder route, which involves setting up the analytical expressions for the one-step transition probabilities and the steady-state probabilities of the semi-Markov process. (Such analytical techniques are feasible only for small problems. For real-world large-sized problems, we have to rely on other techniques such as the one presented here.)

We model the PM optimization problem as an SMDP as in their paper, but use an algorithm based on LA theory as a solution method. We refer to the algorithm as the MCAT (short for Markov Chain Automata Theory) algorithm. The algorithm is shown to produce optimal results in all the numerical examples studied. It was encoded in ARENA (ARENA is marketed by Rockwell Software) and C. The computational savings obtained from the LA approach are substantial because the LA approach does not compute the transition probabilities and the steady-state probabilities.

The LA approach is a simulation-based optimization technique and as such requires a simulator of the system. The optimal policy, using the LA approach, is "learned" from *one simulation run* of the system. The LA simulation run is not anything like what is understood by a standard simulation replication, in which the policy is fixed at the start, once and for all, and the system is simulated over a fixed time horizon to estimate the performance metric. Rather, in an LA simulation run, the simulation starts with a random policy, but the policy changes at every decision epoch, and the run ends with the system having "learned" the optimal (deterministic) policy. It may be noted here that to simulate the system, it is not necessary to generate the transition probabilities; all one needs is the knowledge of the distributions of the random variables that govern the system behavior.

There is a vast body of literature on simulation-based optimization techniques. Lately, research in simulation-based optimization has been drawing a great deal of attention perhaps because analytical approaches do not fare well on many large, real-world, stochastic problems. The response surface method (RSM) was one of the first simulation-based optimization approaches to have appeared in the literature. Employment of this approach on our problem would require the simulation of several policies followed by the use of a function-approximation technique, such as regression, to approximate the cost function over the policy space (see Law and Kelton, 2000). RSM will be time consuming for our problem but is nevertheless a robust procedure — which is why it is still used widely.

Gradient-based techniques for simulation optimization compute the gradient of the cost function and use a gradient descent approach. Initial gradient-based methods hinged on the technique of finite-differences (see Azadivar and Talavage, 1980). With the *simultaneous* perturbation technique, Spall (1992) has shown that the gradient-based approach can be made much faster and can be performed with a modest number of function evaluations. The number of function evaluations is an important issue in simulation-based optimization because each function evaluation can take up a considerable amount of computer time. The literature speaks of several other approaches that can be combined with simulation — especially when the problem is one of finding the optimal values of *a set of parameters* to minimize a cost function of the parameters in question. This problem is often called a problem of *parametric* optimization as opposed to *control* optimization. In the latter, one seeks to find *the optimal action in each state* visited by the relevant stochastic process. Although the problem considered in this paper is one of control optimization, such problems can be often solved by using a parametric-optimization approach especially when the optimal policy has a *threshold* nature. Therefore, some discussion on the more recent approaches, advocated by researchers for solving the parametric simulation-optimization problem, is in order.

Metaheuristics such as tabu search (see Glover, 1990), simulated annealing (see Homem-de-Mello, 2001), and genetic algorithms (see Boesel et al., 2000) have been combined with simulation to solve problems of discrete stochastic optimization, and a varying degree of success has been reported. A genetic algorithm has been used in the optimization module of a commercial software (see Harrell et al., 2000). Random search methods proposed by Andradóttir (1995, 1996) have been shown to be convergent. Ranking and selection (R & S) procedures (see Goldsman et al., 1991, Nelson, 1992 and Goldsman and Nelson, 1994) are statistically proven methods for comparing alternative solutions (systems or configurations) and selecting the best with a high probability. Jacobson and Schruben (1999) present a methodology to compute the gradient of the objective function and Hessian estimators using harmonic analysis. They use Taylor series to approximate the objective function and show that these estimates can be made from just one replication of the system. In this respect their work bears resemblance to MCAT; however, the problem they attack is one of parametric optimization. We have more to say on these topics (see Section 5.2) in the context of the case study used in this paper. For some other references to these methods, see Rubinstein and Shapiro (1983), Glasserman (1991), and Bonnans and Shapiro (2000). A comprehensive source is Andradóttir (1998).

This paper, to the best of our knowledge, is the first to use a learning automata approach to solve an SMDP. Previous computational work on the use of LA to problems of control optimization was limited to small MDPs with a handful of states (see Wheeler and Narendra, 1986).

The rest of this paper is organized as follows. Section 2 contains an overview of the existing literature on SMDPs and automata theory. Section 3 describes the SMDP. Section 4 presents an overview of the automata framework. Section 5 contains the details of the LA algorithm used in this paper. Section 6 is centered on the preventive maintenance case study. It also discusses results from our application of the LA algorithm. Section 7 concludes this paper.

2 Overview of Related Literature

The existing literature reveals that Markovian and semi-Markovian models have been studied extensively for decision-making purposes and have been used in the context of dynamic programming. For a detailed account on MDPs / SMDPs, the reader is referred to Puterman (1994). Applications of these models can be found in a multitude of stochastic optimization problems, some examples of which are: finding optimal inspection policies in a quality control problem (Cassandras and Han, 1992), optimal control of a queuing problem with two customer types (Shioyama, 1991), optimal maintenance in random environments (Ozekici, 1995), part selection in an FMS (Seidmann and Schweitzer, 1983), optimal maintenance of a production-inventory system (Das and Sarkar, 1999), and a stochastic economic lot-size scheduling problem with two part types (Bai et al., 1996). Sennott (1999) discusses examples of decision-making in stochastic systems, especially those in queuing systems.

The theory of learning in controlled Markov Chains, in the context of game theory, has received an excellent treatment in Narendra and Thatachar (1989). The first significant result, which may be applied to MDPs, appeared in Wheeler and Narendra (1986). Some other comprehensive references to research related to learning algorithms, using the paradigm of game theory, are Lakshmivarahan (1981) and Lakshmivarahan and Narendra (1981). Das and Sarkar (1999) derive the optimal policy for the preventive maintenance problem using a semi-Markov model, and hence we use their problem to benchmark the performance of the MCAT algorithm.

3 Semi-Markov Decision Problems

An SMDP, which has an underlying Markov chain (MC), can be characterized by the following five elements: 1) the state space of the MC, 2) the action (decision or control) space, 3) the transition cost (or reward) matrix, 4) the transition probability matrix, 5) the transition time matrix. All the states in the MC are not necessarily decision-making states. This does not mean that we are dealing with an uncontrolled MC here. Rather it means that there are some states in the MC in which there is no decision making involved and only one action is taken.

The two stochastic processes that accompany the underlying MC are (i) the natural process (NP) and (ii) the decision-making process (DMP). The NP tracks all the state changes in the MC while the DMP tracks only the decision-making states. Obviously, the NP and the DMP coincide at the decision-making instants. Any given transition in the SMDP is characterized by a current state, the next state, and the action that caused it. The transition probability refers to the probability that the DMP goes from the given current state to the given next state, under the given action. Similarly, the transition cost and the transition time denote the cost incurred and the time spent, respectively, in any given transition. The time spent in a transition for an SMDP is usually a random variable drawn from a general distribution. Next, we quickly formalize the above discussion.

Let X_m and T_m denote the system state and time respectively at the mth decision-making instant. We define two stochastic processes: $\mathbf{X} = \{X_m : m \in \mathcal{N}\}$ and $\mathbf{T} = \{T_m : m \in \mathcal{N}\}$ where $\mathcal{N}$ is the set of natural numbers. We also define a joint process $(\mathbf{X}, \mathbf{T}) = \{(X_m, T_m) : m \in \mathcal{N}\}$, which is the DMP underlying the Markov chain, $\mathbf{X}$. If it can be shown that

$$P[X_{m+1} = j, T_{m+1} - T_m \leq t | X_0, ..., X_m; T_0, ..., T_m] = P[X_{m+1} = j, T_{m+1} - T_m \leq t | X_m, T_m],$$

then the DMP $(\mathbf{X}, \mathbf{T})$ is a Markov renewal process. If Y_t denotes the state of the system at any time t, then in light of the (X, T) process being a Markov renewal process, the stochastic process $\mathbf{Y} = \{Y_t : t > 0\}$ will be a semi-Markov decision process. The process, $\mathbf{Y}$, is also the NP of the MC. The average reward of an SMDP, starting at state i and continuing with policy π for an infinite time period can be given (using the renewal reward theorem; cf.

Puterman, 1994 and Ross, 1997) as

$$\rho^{\pi}(i) = \frac{\lim_{N\to\infty} \frac{1}{N} E_i^{\pi}\left[\sum_{k=1}^{N} g(i_k, a_k, i_{k+1})\right]}{\lim_{N\to\infty} \frac{1}{N} E_i^{\pi}\left[\sum_{k=1}^{N} t(i_k, a_k, i_{k+1})\right]},$$

where i_k and a_k denote the state and action, respectively, at the kth decision-making epoch, $g(i_k, a_k, i_{k+1})$ denotes the immediate reward earned in the transition from state i_k to state i_{k+1} under the action a_k taken in state i_k, $t(i_k, a_k, i_{k+1})$ denotes the time taken in the same transition, and $E_i^{\pi}[.]$ denotes the expectation operator under policy π when the initial state is i. If the Markov chain $\mathbf{X}$ has a unichain transition probability matrix (see Puterman, 1994 for a detailed analysis and definition of a unichain transition probability matrix), the expected average cost does not vary with the initial state for any stationary policy.

In the next section, we give a brief outline of the theory of learning automata for MDPs.

4 A Simulation-Based Learning Automata Framework

The idea underlying any learning algorithm for MDPs / SMDPs is simple and intuitively appealing (see Figure 1). The learning agent (or decision-maker) starts with a *random* policy. As it implements the policy (either in real time or in simulation), it obtains a response from the system (environment). This response is used as feedback to update the policy. Depending on what feedback the actions of a policy generate, the actions are deemed good or bad. Over time, with trial and error, the system learns the optimal action in each state. The feedback or response is actually the immediate reward or cost generated by an action. We next discuss the framework in more formal terms.

Let $\mathcal{A}_i$ denote the set of actions available at state i. Hence the union of $\mathcal{A}_i$ over i gives the action space. With every state-action pair, we associate a probability $P(i, a)$ of taking

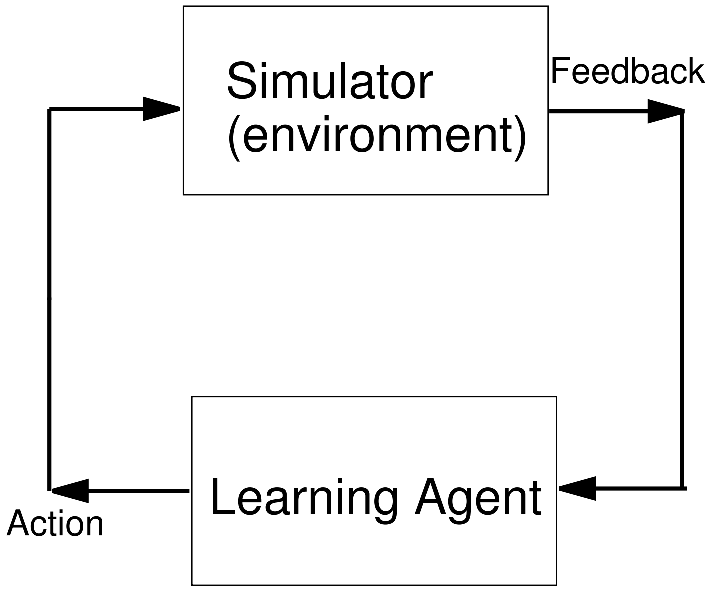

Figure 1: Mechanism of Learning Using the Simulation-Based MCAT Algorithm

action a in state i. Clearly $\sum_{a \in \mathcal{A}_i} P(i,a) = 1$. At the start of the learning process, the policy is random and each action is equally likely. Hence in the beginning, $P(i,a) = 1/r_i$ where $r_i = |\mathcal{A}_i|$ is the number of possible actions in state i. If the performance of an action is good (in terms of the costs or rewards), the probability of that action is increased and if the performance is poor, the probability is reduced. This is called *updating* of the state-action probabilities. Of course, the updating scheme must always ensure that the sum of the probabilities of the different actions for any given state is always 1.

The feedback mechanism works as follows. Whenever a state i is revisited, the average reward earned in the system since its last visit to that state is used as a measure of the response. The average reward is calculated by the total reward earned since the last visit to that state divided by the number of decision-making state transitions since the last visit to that state. Thus, if state i is visited, the response $\psi(i)$ in the MDP is

$$\psi(i) = \frac{R(i)}{N(i)},$$

where $R(i)$ is the total reward earned since the last visit to state i and $N(i)$ is the number of transitions that have taken place in the DMP since the last visit to i. (The SMDP case is discussed in the next section.) The response is further normalized to ensure that it lies between 0 and 1 — since it is used to update the probabilities, which must lie in the same range. The normalized response is called feedback. The normalization is done using: $\beta(i) = \frac{\psi(i)-\psi_{min}}{\psi_{max}-\psi_{min}}$, where ψ_{min} denotes the minimum response possible in the system and ψ_{max} denotes the maximum response possible in the system. The feedback is used to update the probabilities. The conversion of the response to feedback is a research topic by itself, and for further reading, see Narendra and Thatachar (1989).

Figure 2 depicts a 3-state two-action SMDP and shows how the MCAT algorithm works in a simulator. Here $\mathcal{A}_1 = \mathcal{A}_2 = \mathcal{A}_3$. We next discuss a learning scheme for updating the action probabilities.

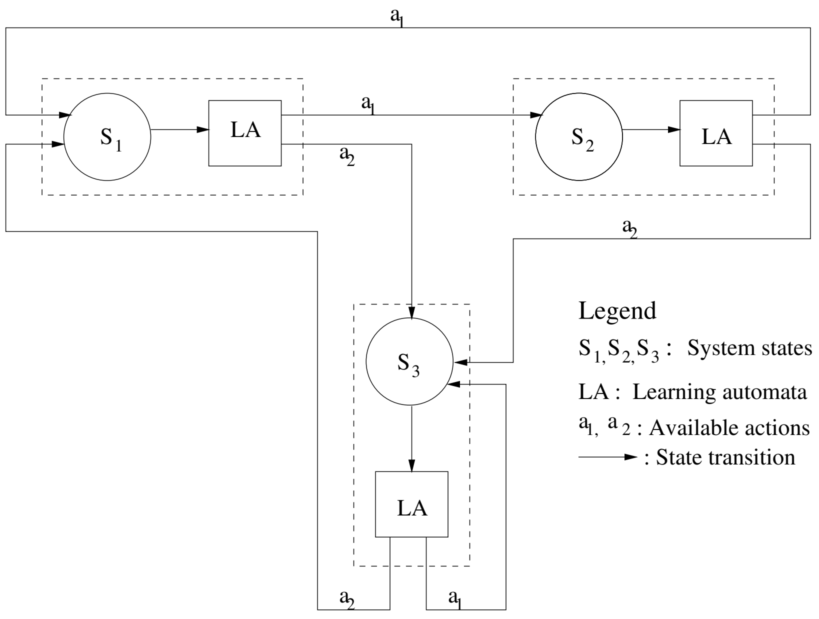

Figure 2: Schematic showing how the learning automata updates its knowledge base and selects actions causing the Markov chain to jump from one state to another.

4.1 Learning Scheme

The scheme for updating the probabilities using feedback is also a topic of considerable research and several schemes have been suggested in literature. The scheme that gave us the best results is known as the Reward-Inaction scheme (see Narendra and Thatachar, 1989). In what follows, we describe this scheme.

When the system visits a new state i, the probabilities of all the actions in state i are first updated. For this, $\beta(i)$ is calculated first, and then the probabilities are updated as shown in equation (1). Let $A(i)$ denote the action selected in state $i \in \mathcal{S}$ in its *last* visit to i, and let η denote the "learning rate." Update $P(i, u_i)$ as shown next:

$$P(i, u_i) \leftarrow P(i, u_i) + \eta\beta(i)I(A(i) = u_i) - \eta\beta(i)P(i, u_i), \tag{1}$$

where u_i denotes the action whose probability of getting selected in state i is to be updated, and $I(.)$ equals 1 if the condition inside the round brackets is satisfied and 0 otherwise.

The reason why the above scheme is so named can be explained as follows. An action that results in a good reward automatically has a high value for β, and thus significantly increases the probability of that action . On the other hand, an action that results in a poor reward has a low value for β, and thus does not significantly change the probabilities (see equation (1)). Thus a reward is acted upon, but a poor response is somewhat ignored (inaction). As the probabilities are updated, usually one action starts predominating. In other words, the probability of the predominant action starts converging to 1, while the probabilities of the other actions start converging to 0. If there are k optimal actions, then the probability of each of the optimal actions starts converging to $1/k$. That such a mechanism will converge to the optimal solution has been established by Wheeler and Narendra (1986).

A straightforward extension of this MDP framework to the SMDP case is possible. The response term, $\psi(i)$, for a given state i, requires a simple modification. The response term actually denotes the average reward earned since the last visit to the state. So instead of dividing the total earned reward by the *number of transitions* (since the last visit) in the DMP of the SMDP, we divide the total reward by the total *time* spent in the NP of the SMDP since the last visit. Thus response for the SMDP may thus be calculated as:

$$\psi(i) = \frac{R(i)}{T(i)}, \tag{2}$$

where $T(i)$ denotes the time spent in the NP of the MC since the last visit to the state i. All other steps of the updating process for an SMDP will be identical to those followed for an MDP. Next, we present a step-by-step description of the MCAT algorithm, and in Section 4.3 we illustrate the use of the MCAT algorithm on a simple 3-state SMDP.

4.2 The MCAT Algorithm

A step-by-step description of the MCAT algorithm is given in Figure 3. The algorithm presented in this paper is for a PM case study where there are 2 possible actions in each state. The terms ψ_{min} and ψ_{max} denote the minimum and maximum values, respectively, of the average reward from the system over the policy space. Knowledge of ψ_{max} and ψ_{min} can be obtained easily from the values of the cost elements in the problem and some elementary extreme-case analysis. (See Section 5.1 for more on this.)

There are a number of ways to terminate the algorithm. One way is to wait until all probabilities have approached 0 or 1. The other method is to run the algorithm for a finite number of iterations and then choose the most probable action in each state.

4.3 An Illustrative 3-state Example

In this section, we show how MCAT works on a simple 3-state SMDP. For the sake of simplicity, we shall first assume that each state has two possible actions. In the next section,

MCAT ALGORITHM:

A. Let $\mathcal{S}$ denote the set of decision-making states in the SMDP. Set iteration count $m = 0$. Initialize action probabilities $P(i, a) = \frac{1}{r_i}$ for all $i \in \mathcal{S}$ and $a \in \mathcal{A}_i$. Set the cumulative reward $C_r(i) = 0$ and the cumulative time $C_t(i) = 0$ for all $i \in \mathcal{S}$. Also initialize to 0 the total reward earned in system from the start, $R_{current}$, and the simulation clock, $T_{current}$. Initialize s_{min} and s_{max} as given in the text. Prefix u_i to any action that is allowed in state i. Start system simulation.

B. While $m <$ MAX_STEPS do

If the system state at iteration m is $i \in \mathcal{S}$,

1. If state i has been visited for the first time, go to step 3; else, in the following order, set:
   $R(i) \leftarrow R_{current} - C_r(i)$, $T(i) \leftarrow T_{current} - C_t(i)$,
   $\psi(i) \leftarrow \frac{R(i)}{T(i)}$ and finally $\beta(i) \leftarrow \frac{\psi(i) - \psi_{min}}{\psi_{max} - \psi_{min}}$.
2. Update $P(i, u_i)$ using equation (1).
3. With probability $P(i, a)$, select an action $a \in \mathcal{A}_i$.
4. Set $A(i) \leftarrow a$, $C_r(i) \leftarrow R_{current}$, and $C_t(i) \leftarrow T_{current}$.
5. Simulate the chosen action a. Let the system state at the next decision epoch be j. Also let $t(i, a, j)$ be the transition time, and $g(i, a, j)$ be the immediate reward earned in the transition resulting from selecting action a in state i.
6. Set $R_{current} \leftarrow R_{current} + g(i, a, j)$ and $T_{current} \leftarrow T_{current} + t(i, a, j)$.
7. Set current state i to new state j, and $m \leftarrow m + 1$.

Figure 3: Steps in the MCAT Algorithm

we shall describe how MCAT is combined with a simulator, and then in the following section, we show the details of how the action probabilities are updated.

4.3.1 MCAT and the simulator

We will not require the values of the transition probabilities of the underlying Markov chain; a simulator of the system will be sufficient. Using the simulator, we can generate a trajectory of states, and when the simulator visits a given state, certain quantities related to that state will be updated.

Let us assume that the system starts out in state i. With a probability of $P(i,a)$, where $a \in \mathcal{A}_i$, action a is chosen. The updating required is performed within the simulator, and the action chosen is simulated. Let the next state be j. Again, an action is chosen, the quantities in question are updated, and the selected action is simulated. This continues until, in each state, the probability of one action(s) starts dominating the probabilities of the other actions. We next illustrate the updating process with numerical values.

4.3.2 MCAT and the updating

Let us assume that the first few states of the trajectory generated by the simulator turn out to be $(2,1,3,1)$. We begin by setting $C_r(i)=0$ and $C_t(i)=0, \forall i$. We also set $R_{current}$ and $T_{current}$ to 0. Also, we shall assume that $u_i = 1, \forall i$, $\psi_{max}=10$, $\psi_{min}=-5$, and $\eta = 0.1$. Also,

$$P(1,1) = P(1,2) = P(2,1) = P(2,2) = P(3,1) = P(3,2) = 0.5.$$

The updating calculations performed in each state are listed in Table 1. The updating is continued in this fashion for a large number of iterations. The most likely action (the action with the largest probability) in each state is considered to be the best action for that state.

Table 1: Sample calculations from a 3-state SMDP. Iteration 4 updates probability $P(1,1)$ from 0.5 to 0.476, via equation (1), using $R(1) = 3.5$, $T(1) = 1.64$, $\psi(1) = 2.134$, and $\beta(1) = 0.476$.

Iteration	$\{$Current State (i), Next State $(j)\}$	Action, a, in i $a = A(i)$	$\{C_r(i), C_t(i)\}$	$\{g(i,a,j), t(i,a,j)\}$	$\{R_{current}, T_{current}\}$
1	$\{2,1\}$	1	$\{0,0\}$	$\{4.5,2.34\}$	$\{4.5,2.34\}$
2	$\{1,3\}$	2	$\{4.5,2.34\}$	$\{3.5,0.11\}$	$\{8,2.45\}$
3	$\{3,1\}$	2	$\{8,2.45\}$	$\{-1,1.55\}$	$\{7,4\}$
4	$\{1,2\}$	1	$\{7,4\}$	$\{2.4,1.23\}$	$\{9.4,5.23\}$

We next discuss convergence of the MCAT for SMDPs. Since the convergence result is based on game-theoretic concepts, we will present some key game-theoretic ideas to relate the SMDP to a stochastic game.

4.4 Semi-Markov Convergence

An SMDP can be thought of as a non-zero sum game with multiple players and a common payoff. We associate one player (automaton) with each state in the Markov chain. Then running the system that has N decision-making states with a fixed policy is equivalent to playing a game with N players. The common payoff to all the players is defined in the SMDP by the long-run average reward associated with the policy. Our task then is to find the equilibrium point of the game — where the payoff is maximized. We are measuring the payoff in terms of the feedback from the system, i.e., β. Nash (1950) proved that all finite games have at least one pure or mixed strategy equilibrium.

The convergence of the MCAT algorithm has been established via a game-theoretic connection (see Wheeler and Narendra, 1986) in a very general setting that includes the MDP case. We need to define two terms before stating the result.

Let us define $\beta^k(i)$ to be feedback received by the automaton in the ith state at the kth iteration of the algorithm, and $M^k(i)$ to be its *expected* value. The feedback is associated with given values for the vector $\vec{P^k} = (P^k(i,1), P^k(i,2), \ldots, P^k(i,r_i))$, where $P^k(i,a)$ denotes the probability of selecting action a in state i in the kth iteration of the algorithm. A mathematical definition for $M^k(i)$ is:

$$M^k(i) \equiv E[\beta^k(i)|\vec{P^k}].$$

We next define $d(\pi)$ to be the expected feedback received by the MDP or SMDP when a policy π is followed. A policy is a rule that associates a unique action with each state. Clearly, in the SMDP (like in the MDP), there is a single performance metric to be optimized (the long-run average reward in this case). It can be shown that

$$\lim_{k\to\infty} M^k(i)$$

exists (Narendra and Thatachar, 1989) and is the same for all values of i. We denote this limit by M^k. Then Wheeler and Narendra's result can be stated as:

Theorem: Let $\mathcal{G}$ be a common-payoff game among N players ($i = 1, 2, \ldots, N$); the ith player has r_i actions. Assume that all the players use the Reward-Inaction learning scheme. Further, let $\mathcal{G}$ have a unique equilibrium π^*, whose expected feedback is $d(\pi^*)$. Then for any $\epsilon > 0$, there exists an η^*, $0 < \eta^* < 1$, such that for any positive $\eta < \eta^*$:

$$\lim_{k\to\infty} E[M^k] > d(\pi^*) - \epsilon.$$

What this result says is that by a suitable choice of the learning rate (or step size), η, in the Reward-Inaction scheme, one can obtain a feedback that is arbitrarily close to that associated with the *optimal* policy. In other words, the Reward-Inaction schemes are ϵ-optimal.

The above-stated result holds for any scenario (as long as the conditions of the learning algorithm are satisfied) in which one seeks to *maximize* the feedback. According to the normalization scheme, a feedback equaling 1 is the best, as it is associated with ψ_{max}, and that associated with 0 is the worst — hence the term 'maximize'. Therefore, for the SMDP, this result will hold as long as the feedback is calculated (properly) — keeping track of the time spent in earning any given reward. For the SMDP, one is interested in maximizing the average reward calculated per unit *time* over an infinitely long time horizon. (This differs from the MDP case, where the performance metric is calculated on a unit-*step* basis). And therefore, we must divide the incremental reward $R(i)$ by the incremental time $T(i)$. This is how we justify equation (2). Our equation captures the performance metric we are interested in.

Of course, the result will hold regardless of how we define the response, as long as it is normalized, but the policy learned will only be as good as our definition of the performance metric. In other words, all that the result guarantees us is that the policy learned will optimize the performance metric *chosen*. Because of the generality of its conditions, the result actually holds even when the system does *not* have an underlying Markov chain (see Wheeler and Narendra, 1986). Furthermore, it is likely to have applications to problems modeled with *partially observable* Markov chains.

5 Case Study

To demonstrate empirically that the MCAT algorithm generates a near-optimal policy on SMDPs from real-life problems, we choose a case study from preventive-maintenance literature (Das and Sarkar, 1999). This case study is especially suitable because the optimal policies are available in Das and Sarkar (1999) on some experiments; and hence it can serve as a nice vehicle for benchmarking MCAT. We now describe the problem in some detail.

Consider a production-inventory system as shown in Figure 4. It produces a single product type to satisfy an external demand process. The system is prone to failures; the time between

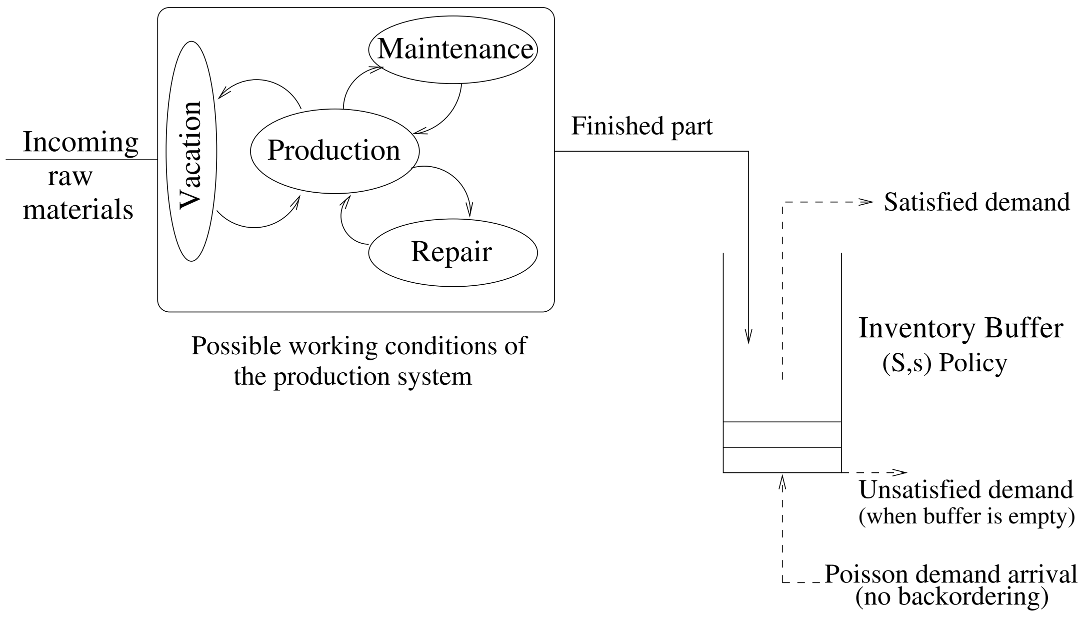

Figure 4: A Production-Inventory System

failures, a random variable, is *not* exponentially distributed. Hence preventive maintenance (see Lewis, 1994) 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. A finished-product inventory buffer of the product is maintained 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 inventory reaches $\mathcal{Z}_1$ and the production resumes when the inventory drops to $\mathcal{Z}_2$. During its vacation, i.e., when production stops due to the inventory reaching its upper limit $\mathcal{Z}_1$, the system is assumed not to age or fail. Preventive maintenance decisions are made only at the completion of a part production. The problem of finding the optimal PM decisions can be cast as an SMDP. We next discuss the SMDP formulation.

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. 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 to the decision maker: (i) to produce one more part and (ii) to schedule a maintenance. Clearly, when the decision maker is faced with the decision of choosing between one more production cycle and a maintenance, it has to make the decision based on the age of the machine and the number of parts in the buffer (w). A measure of the age is the number of parts produced since the last repair or maintenance. This will be denoted by c (also called the *production count*). The decision-making state space of the system may hence be denoted by (w, c). For more details on the semi-Markov model and the method used to obtain the optimal policies, the reader is referred to Das and Sarkar (1999).

5.1 Numerical Results

We present results from numerical examples of two types for which 1) theoretical results via exact Markov chain analysis are available and 2) optimal solutions are not available. It is usually the case that a few modeling complications or an increase in the size of the state space for stochastic systems can cause analytical methods to break down; then one has to turn to simulation for modeling the system. It is not surprising that the Markov chain approach of Das and Sarkar (1999) breaks down if the buffer limits $(\mathcal{Z}_1, \mathcal{Z}_2)$ are increased from $(3, 2)$ (an analytically-tractable size) to vectors of larger entries. It is for these cases (type 2) that we turn to a popular heuristic based on renewal theory from the domain of PM. The idea is again to benchmark MCAT's performance against some standard — the optimal solution or a heuristic solution. The renewal-theory heuristic that we have used is known as the age-replacement (AR) heuristic. Details related to the heuristic have been placed in the appendix.

The demand arrival process (with batch size of 1) is Poisson (γ). The time between machine failures, the production time, and the machine repair time are gamma distributed with parameters $(k,\ \mu)$ (d,λ), $(r,\ \delta)$, and respectively. The time for preventive maintenance has a uniform distribution with parameters (a, b). For each of the fourteen parameter sets (shown in Table 2), 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). The results (average reward) for the single-product inventory system obtained from the theoretical model / heuristic model, and those obtained from the LA algorithm are summarized in Table 3. In the first nine cases, the LA results are compared to the optimal results and in the last five cases, the comparisons are with AR.

The first nine cases have a smaller state space compared to the last five cases. Exact expressions for the transition probabilities (in the first nine cases) were derived by Das and Sarkar (1999) for the buffer limits $(\mathcal{Z}_1, \mathcal{Z}_2) = (3, 2)$. However, the expressions were derived for these numbers specifically; new values would require a re-determination of the expressions. Furthermore, a unit increase in the upper buffer limit needs at least 25 *additional* expressions. (The number 25 is the minimum value of c at which the machine fails almost surely.) As such, we do not have the optimal results for the last five cases, where $\mathcal{Z}_1 > 4$.

An approximation for the value of ψ_{min} can be obtained from the worst-case scenario, in which no maintenance is performed. In such a situation, ψ_{min} can be approximated by the quotient of the negative reward (cost) of one repair and the mean time between failure. Of course, ψ_{max} is set to some value higher than ψ_{min}. In our study, we know the optimal solution in some cases and hence the value of ψ_{max} for those cases. When the optimal solution cannot be obtained, a heuristic (such as AR) is often used, and then ψ_{max} can be set to be a multiple of the average reward obtained from the heuristic.

A statistical test comparing two means — the average reward obtained from the policy generated by the MCAT algorithm and that obtained from the theoretical results of Das and Sarkar — reveals no significant difference at a 95 % confidence level for the first nine cases. A similar test *does* show a significant difference, in the last five cases (between the MCAT and AR values). This shows that MCAT outperforms AR, in the experiments performed here.

Table 2: Input parameters for fourteen sample numerical examples

System	Time Bet. Demands (γ)	Time Bet. Failure (k, μ)	Time bet. Product. (d, λ)	Maint. Time (a, b)	Repair Time (r, δ)	Inventory Limits ($\mathcal{Z}_1, \mathcal{Z}_2$)
1	1/10	(8, 0.08)	(8, 0.8)	(5, 20)	(2, 0.01)	(3,2)
2	1/10	(8, 0.008)	(8, 0.8)	(5, 20)	(2, 0.01)	(3,2)
3	1/7	(8, 0.08)	(8, 0.8)	(5, 20)	(2, 0.01)	(3,2)
4	1/15	(8, 0.08)	(8, 0.8)	(5, 20)	(2, 0.01)	(3,2)
5	1/15	(8, 0.08)	(8, 0.8)	(25, 40)	(2,0.01)	(3,2)
6	1/15	(8, 0.08)	(8, 0.8)	(5, 20)	(2, 0.02)	(3,2)
7	1/15	(8, 0.08)	(8, 0.8)	(5, 20)	(4, 0.02)	(3,2)
8	1/15	(8, 0.01)	(8, 0.8)	(5, 20)	(4,0.02)	(3,2)
9	1/20	(8, 0.04)	(8, 0.4)	(5, 20)	(4, 0.02)	(3,2)
10	1/15	(8, 0.08)	(8, 0.8)	(5, 20)	(2, 0.01)	(5,4)
11	1/15	(8, 0.08)	(8, 0.8)	(5, 20)	(2, 0.01)	(6,5)
12	1/15	(8, 0.08)	(8, 0.8)	(5, 20)	(2, 0.01)	(7,6)
13	1/15	(8, 0.08)	(8, 0.8)	(5, 20)	(2, 0.01)	(8,7)
14	1/15	(8, 0.08)	(8, 0.8)	(5, 20)	(2, 0.01)	(9,8)

We found that the learning rates do affect the solution that the algorithm generates. The learning rate was set to 0.1 during our calculations. We found that lowering the learning rate below 0.1 made the learning process slower, although convergence to the optimum was still achieved. However when the learning rate was increased above 0.1, the algorithm generated a sub-optimal solution. The reason for this can be traced to the convergence result which states that the learning rate has to be *sufficiently* small.

5.2 Other Simulation Optimization Methods

The PM case study can also be cast as a stochastic integer program. But there are two difficulties with this. Firstly, it is not easy to find the objective function in closed form and secondly, the solution space will be enormous for a pointwise evaluation of the search space

Table 3: Results from the MCAT algorithm, which were averaged over 30 runs, where each simulation run lasted for 1.0 million time units.

$C_d = 1, C_r = 5, C_m = 2$		
System	Optimal/AR Average Reward ($ per unit time)	MCAT Average Reward ($ per unit time)
1	0.034	0.034
2	0.076	0.075
3	0.035	0.035
4	0.028	0.028
5	0.025	0.025
6	0.031	0.030
7	0.028	0.028
8	0.057	0.057
9	0.020	0.020
10	0.01531	0.02951
11	0.01545	0.02993
12	0.01578	0.02957
13	0.01607	0.03012
14	0.01608	0.03063

via simulation. A stochastic integer formulation is shown below.

$$\text{Maximize } h(x_0, x_1, \ldots, x_{\mathcal{Z}_1}) \text{ where } x_i \in \{0, 1, 2, \ldots, \theta\}.$$

Here θ denotes the smallest production count at which the machine fails with probability approaching 1, and $h(.)$ denotes the long-run average reward earned by following a policy which maintains the machine when there are i units in the buffer, and the production count equals x_i. (Recall that $\mathcal{Z}_1$ is the upper limit for the buffer.) Hence it is clear that the total number of policies is equal to $(\theta + 1)^{(\mathcal{Z}_1+1)}$. For most of the cases considered in this paper, $\theta = 25$ (see Das and Sarkar, 1999), and so the total number of alternative configurations (each configuration is associated with a policy) is 26^4. Therefore a separate evaluation of each configuration will be computationally burdensome.

Ranking and selection procedures are recommended for systems with up to 20 or 30 configurations; however recently they have been used on larger problems (see Boesel et al., 2000). Boesel et al. (2000) have designed software that uses ranking and selection procedures in combination with a genetic-algorithm search that has a remarkable *statistical error control.* In particular, the algorithm search delivers a statistical guarantee of the solution generated after the search method ends. An interesting feature of the MCAT algorithm and the genetic-algorithm procedure, as described in Boesel et al. (2000), is that both use a somewhat similar philosophy. Both use selection probabilities that favor the seemingly better "solutions" while retaining the probabilities of the "solutions" that seem to be giving poorer feedback. However, the analogy ends here because a genetic-algorithm search is designed for solving a problem of parametric optimization whereas the MCAT search is useful in control optimization.

6 Conclusions

This paper presented an application of simulation-based optimization to a hard problem in manufacturing. The MCAT algorithm is representative of a new paradigm in simulation-based optimization. For decision-making problems, it forms an alternative both to traditional response surface and gradient-based methods, and to reinforcement learning. The extension of the MCAT framework from the MDP to the SMDP, which is a contribution of this paper, should make it more amenable to real-life problems, which more frequently tend to be SMDPs than MDPs. Also, the theory of learning automata remains largely unknown in the simulation-optimization community, and one objective of this paper was to show how this useful framework may be utilized in solving complex problems. We hope that this will lead to more applications along similar or related lines to difficult problems of stochastic optimization. We are currently working on the use of Bayesian networks to model the action probabilities. We are also exploring the possibility of using the framework on the more difficult, partially observable version of the SMDP. Other possible avenues for further research in this area include the study of this framework for discounted rewards.

Acknowledgements. The authors would like to thank the simulation area editor Professor Dave Goldsman and an anonymous reviewer for carefully reading the manuscript and making various suggestions that have improved the quality of this work. This research was partly funded by National Science Foundation via grant ITR/AP 0114007 to the first author and grant DMI-0113946 to the second and the third author.

A Appendix

The AR heuristic, which was used for benchmarking MCAT's performance on large-sized problems, is developed via the powerful renewal-reward model.

With an analytical expression for the cost of maintaining the machine, when its age is τ, one can use non-linear programming to find the optimal value of the age, τ, that minimizes the average cost. See Ross (1997) for details of this model. The average cost using the renewal-reward theorem can be written as EC/CT, if EC is the expected renewal cost and CT is the expected renewal time. Let X denote the time between failures. Then,

$$EC = (1 - F(\tau))C_m + F(\tau)C_r,$$

where τ is the age of maintenance, $F(x)$ is the cumulative distribution function of the random variable X, C_m is the cost of one maintenance, and C_r is the cost of one repair. Also,

$$CT = \tau_r F(\tau) + \int_0^\tau x f(x)dx + (\tau + \tau_m)(1 - F(\tau)),$$

where $f(x)$ denotes the probability density function of the random variable X, and τ_m and τ_r denote the mean time for maintenance and failure, respectively.

References.

S. Andradóttir. 1995. A method for stochastic approximation with varying bounds. *Operations Research.* **41**. 1946-1961.

S. Andradóttir. 1996. A global search method for discrete stochastic approximation. *SIAM Journal on Optimization* **6**. 513-530.

S. Andradóttir. 1998. Simulation Optimization. In *Handbook of Simulation* (edited by Jerry Banks). Chapter 9, John Wiley and Sons, New York, NY.

F. Azadivar and J.J.Talavage. 1980. Optimization of stochastic simulation models. *Mathematics and Computers in Simulation.* **22**. 231-241.

S.X. Bai, J.H. Burhanpurwala, M. ElHafsi, and Y.K. Tsai. 1996. Hierarchical production control for a flow shop with dynamic setup changes and random machine breakdowns. *OR Spektrum.* **18**. 81-96.

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

J. Boesel, B.L. Nelson, and N. Ishii. 1999. A Framework for Simulation-Optimization Software. *To appear in IIE Transactions.*

J.F. Bonnans, and A. Shapiro. 2000. *Perturbation Analysis of Optimization Problems.* Springer Verlag. New York, NY.

C.G. Cassandras and Y. Han. 1992. Optimal Inspection Policies for a Manufacturing Station. *European Journal of Operational Research.* **63**. 35-53.

T. K. Das and S. Sarkar. 1999. Optimal Preventive Maintenance in a Production Inventory System. *IIE Transactions.* **31.** 537-551.

D. Goldsman, B.L. Nelson, and B. Schmeiser. 1991. Methods for selecting the best system.

In *Proceedings of the 1991 Winter Simulation Conference.* (B. L. Nelson, W.D. Kelton, and G.M. Clark, eds.) 177-186.

D. Goldsman and B.L. Nelson. 1994. Ranking, selection, and multiple comparisons in computer simulation. In *Proceedings of the 1994 Winter Simulation Conference.* (J.D. Tew, S. Manivanan, D.A. Sadowski, A.F. Seila, eds.) 192-199.

P. Glasserman. 1991. *Gradient Estimation Via Perturbation Analysis.* Kluwer Academic Press, Boston.

F. Glover. 1990. Tabu Search: A Tutorial. *Interfaces.* **20(4)**. 74-94.

C. Harrell, B.K. Ghosh, and R. Bowden. 2000. *Simulation Using Promodel.* McGraw Hill Higher Education, Boston.

T. Homem-de-Mello. 2001. On the Convergence of Simulated Annealing for Discrete Stochastic Optimization. *Working Paper, Department of Industrial, Welding and Systems Engineering, Ohio State University, Ohio.*

S. H. Jacobson and L. W. Schruben. 1999. A harmonic analysis approach to simulation sensitivity analysis. *IIE Transactions.* **31(3)**. 231-243.

S. Lakshmivarahan. 1981. *Learning Algorithms: Theory and Applications.* Springer-Verlag, New York, NY.

S. Lakshmivarahan and K.S. Narendra. 1981. Learning Algorithms for two-person zero sum stochastic games with incomplete information. *Mathematic of Operations Research.* **6**. 379-386.

A.M. Law and W.D. Kelton. 2000. *Simulation Modeling and Analysis.* McGraw Hill, New York, NY.

E.E. Lewis. 1994. *Introduction to Reliability Engineering.* John Wiley and Sons, New York, NY.

K.S. Narendra and M.A.L. Thatachar. 1989. *Learning Automata.* Prentice Hall, Englewood Cliffs, NJ.

J.F. Nash. 1950. Equilibrium Points in N-Person Games. *Proceedings, Nat. Acad. of Science, USA*. 36. 48-49.

B.L. Nelson. 1992. Designing efficient simulation experiments. In *Proceedings of the 1992 Winter Simulation Conference.* (J.J. Swain, D. Goldsman, R.C. Crain and J.R. Wilson eds.) 126-132.

S. Ozekici. 1995. Optimal Maintenance policies in random environments. *European Journal of Operational Research.* **82**. 283-294.

M. L. Puterman. 1994. *Markov Decision Processes.* John Wiley and Sons Inc., New York, NY.

S. M. Ross. 1997. *Introduction to Probability Models.* Academic Press, San Diego, CA.

R. Y. Rubinstein and A. Shapiro. 1983. *Sensitivity Analysis and Stochastic Optimization by the Score Function Method.* John Wiley and Sons, Inc., New York, NY.

A. Seidmann and P.J. Schweitzer. 1983. Part Selection Policy for a Flexible Manufacturing Cell Feeding Several Production Lines. *IIE Transactions.* **16(4)**. 355-362.

L.I. Sennott. 1999. *Stochastic Dynamic Programming and the Control of Queueing Systems.* John Wiley and Sons, Inc., New York, NY.

T. Shiyoyama. 1991. Optimal Control of a Queueing Network System with Two Types of Customers. *European Journal of Operational Research.* **52**. 367-372.

J.C. Spall. 1992. Multivariate Stochastic Approximation Using a Simultaneous Perturbation Gradient Approximation. *IEEE Transactions on Automatic Control.* **37**. 332-341.

R. Sutton and A.G. Barto. 1998. *Reinforcement Learning.* MIT Press. Cambridge, MA.

R. Wheeler and K. Narendra. 1986. Decentralized Learning in Finite Markov Chains. *IEEE Transactions on Automatic Control.* **31(6)**. 373-376.