Self-Improving Factory Simulation using Continuous-time Average-Reward Reinforcement Learning

Sridhar Mahadevan and Nicholas Marchalleck
Department of Computer Science and Engineering
University of South Florida
Tampa, Florida 33620
(mahadeva,marchall@csee.usf.edu)

Tapas K. Das and Abhijit Gosavi
Department of Industrial Engineering
University of South Florida
Tampa, FL 33620
(das,gosavi@eng.usf.edu)

Abstract

Many factory optimization problems, from inventory control to scheduling and reliability, can be formulated as continuous-time Markov decision processes. A primary goal in such problems is to find a gain-optimal policy that minimizes the long-run average cost. This paper describes a new average-reward algorithm called SMART for finding gain-optimal policies in continuous time semi-Markov decision processes. The paper presents a detailed experimental study of SMART on a large unreliable production inventory problem. SMART outperforms two well-known reliability heuristics from industrial engineering. A key feature of this study is the integration of the reinforcement learning algorithm directly into two commercial discrete-event simulation packages, ARENA and CSIM, paving the way for this approach to be applied to many other factory optimization problems for which there already exist simulation models.

1 Introduction

Many problems in industrial design and manufacturing, such as scheduling, queueing, inventory control, and reliability engineering, can be formulated as continuous-time sequential decision tasks. These problems can be formally characterized as continuous-time semi-Markov decision processes (SMDP's), where the goal is to find an optimal policy that minimizes the long-term average cost [11]. Since dynamic programming methods, such as policy or value iteration, cannot be applied to large SMDP's, a standard approach to studying these problems is through the use of discrete-event simulation [8]. Although simulation is valuable in evaluating the expected long-run performance of a fixed policy, it is of limited use in finding good or optimal policies.

Reinforcement learning (RL) is an ideal approach to solving large SMDP's, since it can be easily combined with discrete-event simulation models. However, work in average-reward reinforcement learning has so far been limited to small discrete-time Markov decision processes (MDP's) [9, 12, 14, 16]. This paper introduces a new model-free average-reward algorithm for continuous-time SMDP's called SMART. Second, it presents the first large scale experimental test of an average-reward RL algorithm on a realistic multi-product unreliable production inventory system (with a state space in excess of 10^{15} states). Finally, it illustrates how RL can be integrated into widely available discrete-event simulation packages (in particular, CSIM and ARENA [1]).

There has been previous work on continuous-time SMDP's in the RL literature. In particular, Bradtke and Duff [2] describe how the well-known discounted RL algorithms, including TD(λ) [15] and Q-learning [17], can be extended to SMDP's. Crites and Barto [3] describe an impressive application of Q-learning to a large scale SMDP problem of controlling a group of elevators. Both these papers primarily focus on an optimality framework, where the goal of the decision maker is to maximize the discounted sum of rewards.

The rest of the paper is organized as follows. Section 2 briefly introduces the semi-Markov decision process (SMDP) framework under the average-reward paradigm. Section 3 describes the SMART algo-

[1]ARENA is marketed by Systems Modeling Corporation. CSIM is marketed by Mesquite Software.

rithm. Section 4 describes an unreliable multi-product production inventory system, and also discusses how SMART was integrated into a discrete-event simulation system. Section 5 presents experimental results of using SMART to optimize the production inventory system. Finally, Section 6 summarizes the paper, and discusses some directions for future work.

2 Semi-Markov Decision Problems

Semi-Markov decision processes extend the usual discrete-time MDP model in several respects. The most obvious distinction is one of time, which is continuous. However, decisions are only allowed at discrete points in time (or *events*). In between decisions, the state of the system may change continually, unlike MDP's where state changes are solely due to actions. The reward structure is also different: taking an action in a state results in both an immediate lump-sum reward, as well as an accumulated reward generated at some fixed rate until the next decision epoch.

We briefly introduce the Semi-Markov Decision Process (SMDP) model here [1, 11]. An SMDP is defined as a five tuple (S, A, P, R, F), where S is a finite set of states, A is a set of actions, P is a set of state and action dependent transition probabilities, R is a reward function, and F is a function giving probability of transition times for each state-action pair. $P_{xy}(a)$ denotes the probability that action a will cause the system to transition from state $x \in S$ to $y \in S$. This transition probability function describes the transitions at decision epochs only. It is worth pointing out that the trajectory of system states between decision epochs in general has no effect on the state transition matrix, and as such gives the decision maker no information. The SMDP represents snapshots of the system at decision points, whereas the so-called *natural process* describes the evolution of states over all time.

F is a function where $F(t \mid s, a)$ is the probability that the next decision epoch occurs within t time units, after the agent chooses action a in state s at a decision epoch. From F, and P, we can compute Q by

$$Q(t, y \mid x, a) = P(y \mid x, a)F(t \mid x, a)$$

where Q denotes the probability that the system will be in state y for the next decision epoch, at or before t time units after choosing action a in state s, at the last decision epoch. Q can be used to calculate the expected transition time between decision epochs.

In general, the reward function for SMDP's is more complex than in the MDP model. In addition to the fixed reward $k(x, a)$, accrued due to an action taken at a decision epoch, an additional reward may be accumulated at *rate* $c(y, x, a)$ for the time the natural process remains in state y between the decision epochs. Note that the natural process may change state several times between decision epochs, and therefore, the rate at which the rewards are accumulated between decision epochs may vary.

Figure 1 illustrates the SMDP model using a simple machine maintenance problem. Here, a machine makes only one part, and will fail at some unknown point in time (characterized by a failure gamma probability distribution). The state of the machine is represented by the number of parts produced. The machine can be maintained before it fails, which will restore it to its original condition. Failure times are generally long, and repairs are expensive. Maintenance times are shorter, and less expensive. The tradeoff here is to decide when to maintain the machine: do it too late and risk repair, or do it too early and pay excessive maintenance cost. The production inventory problem, described later in the paper, is a much more complex and realistic version of the maintenance problem, where the machine may make multiple products, each product fills a buffer, which is depleted by different demand rates.

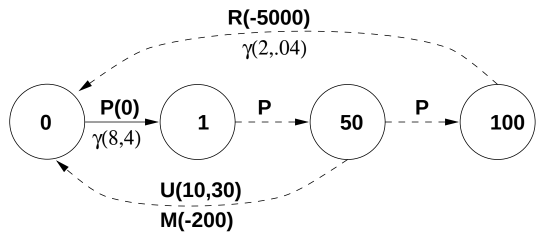

Figure 1: A simple machine maintenance problem to illustrate SMDP's. The decision maker has to choose between two actions: produce (P) and maintain (M). In case of a failure, the only possible action is a repair (R). The numbers inside the brackets indicate lump-sum rewards. Maintenance times are uniformly distributed, whereas production and failure times are gamma distributed random variables. Reward rates are not shown.

2.1 Value Functions for Average Reward SMDPs

Formally, the expected reward between two decision epochs, given that the system in state x, and a is chosen at the first decision epoch, may be expressed as

$$r(x,a) = k(x,a) + E_x^a\left\{\int_0^\tau c(W_t,x,a)dt\right\} \quad (1)$$

where τ is the transition time to the second decision epoch, and W_t denotes the state of the natural process. Now, starting from state x at time 0, and using a policy π for n decision epochs until time t, the expected total reward can be expressed as

$$V_t^\pi(x) = E_x^\pi\left\{\int_0^t c(W_u,x_{n_u},a_{n_u})du + \sum_{q=0}^{n_t-1} k(x_n,a_n)\right\} \quad (2)$$

where $k(x_n,a_n)$ is the fixed reward earned at the nth decision epoch, and $c(y,x_n,a_n)$ is the rate at which reward is accumulated from the nth to the $(n+1)$th decision epoch. The average reward ρ^π for a policy π can be defined by taking the limit inferior of the ratio of the expected total reward up until the nth decision epoch to the expected total time until the nth epoch. So the average reward of a policy $\rho^\pi(x)$ can be expressed as the ratio

$$\lim_{n\to\infty} \frac{E_x^\pi\left\{\sum_{i=0}^n\left[k(x_i,a_i) + \int_{\sigma_i}^{\sigma_{i+1}} c(W_t,x_i,a_i)dt\right]\right\}}{E_a^\pi\left\{\sum_{i=0}^n \tau_i\right\}} \quad (3)$$

For each transition, the expected transition time is defined as:

$$y(x,a) = E_s^a\tau = \int_0^\infty t\sum_{z\in S} Q(dt,z \mid x,a) \quad (4)$$

for each $x \in S$, and $a \in A$.

The Bellman optimality equation for *unichain* average reward SMDP's is analogous to that for the discrete-time model, and can be written as

$$V^*(x) = \max_a\left(r(x,a) - \rho y(x,a) + \sum_{z\in S} P_{xz}(a)V^*(z)\right) \quad (5)$$

In unichain SMDP's, the average reward is constant across states. Many real-world SMDP's, including the elevator task [3] and the production inventory task described below, are unichain.

3 SMART: An Average-Reward Algorithm

We now describe a new model-free average-reward algorithm (SMART) for finding gain-optimal policies for SMDP's. The derivation of this algorithm from the Bellman equation for SMDP's (Equation 5) is fairly straightforward. First, we reformulate Equation 5 using the concept of action-values. The average-adjusted sum of rewards received for the non-stationary policy of doing action a once, and then subsequently following a stationary policy π can be defined as

$$R^\pi(x,a) = r(x,a) - \rho^\pi y(x,a) + \sum_{z\in S} P_{xz}(a) \max_b R^\pi(z,b) \quad (6)$$

The temporal difference between the action-values of the current state and the actual next state is used to update the action values. In this case, the expected transition time is replaced by the actual transition time, and the expected reward is replaced by the actual immediate reward. Therefore, the action values are updated as follows:[2]

$$R(x,a) \xleftarrow{\alpha_n} \left(r_{imm}(x,a) - \rho\tau + \max_b R(z,b)\right) \quad (7)$$

where $r_{imm}(x,a)$ is the actual cumulative reward earned between decision epochs due to taking action a in state x, z is the successor state, ρ is the average reward, and α_n is a learning rate parameter. Note that ρ is actually the reward rate, and it is estimated by taking the ratio of the total reward so far, to the total simulation time.

$$\rho = \frac{\sum_{i=0}^n r_{imm}(x_i,a_i)}{\sum_{i=0}^n \tau_i} \quad (8)$$

where $r_{imm}(x_n,a_n)$ is the total reward accumulated between the nth, and $(n+1)$th decision epochs, and τ_n is the corresponding transition time. Details of the algorithm are given in Figure 2. The learning rate α_n and the exploration rate p_n are both decayed slowly to 0 (we used a Moody-Darken search-then-converge procedure).

3.1 Value Function Approximation in SMDP's

In most interesting SMDP's, such as the elevator problem [3], or the production inventory problem described below, the number of states is usually so large as to

[2] We use the notation $u \xleftarrow{\alpha} v$ as an abbreviation for the stochastic approximation update rule $u \leftarrow (1-\alpha)u + \alpha v$.

1. Set decision epoch $n = 0$, and initialize action values $R_n(x,a) = 0$. Choose the current state x arbitrarily. Set the total reward c_n and total time t_n to 0.
2. While $n <$ MAX_STEPS do
   (a) With high probability p_n, choose an action a that maximizes $R_n(x,a)$, otherwise choose a random action.
   (b) Perform action a. Let the state at the next decision epoch be z, the transition time be τ, and r_{imm} be the cumulative reward earned in this epoch as a result of taking action a in state x.
   (c) Update $R_n(x,a)$ using :
   $$R_{n+1}(x,a) \stackrel{\alpha_n}{\leftarrow} \left(r_{imm} - \rho_n \tau + \max_b R_n(z,b) \right)$$
   (d) In case a nonrandom action was chosen in step 2(a)
      - Update total reward $c_n \longleftarrow c_n + r_{imm}$
      - Update total time $t_n \longleftarrow t_n + \tau$
      - Update average reward $\rho_n \longleftarrow \frac{c_n}{t_n}$
   (e) Set current state x to new state z, and $n \leftarrow n+1$.

Figure 2: The SMART algorithm.

rule out tabular representation of the action values. In particular, the state space in our problem is a 10-dimensional integer space. Following Crites and Barto [3], we also used a feed-forward net to represent the action value function. Equation 7 used in SMART is replaced by a step which involves updating the the weights of the net. So after each action choice, in step 2(c) of the algorithm, the weights of the corresponding action net is updated according to:

$$\Delta\phi = \alpha_n \epsilon(x,z,a,r_{imm},\phi) \nabla_\phi R_n(x,a,\phi) \qquad (9)$$

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

$$\left[r_{imm}(x,a) - \rho_n\tau + \max_b R_n(z,b,\phi) - R_n(x,a,\phi) \right]$$

and ϕ is the vector of weights of the net, α_n is the learning rate, and $\nabla_\phi R_n(x,a,\phi)$ is the vector of partial derivatives of $R_n(x,a,\phi)$ with respect to each component of ϕ.

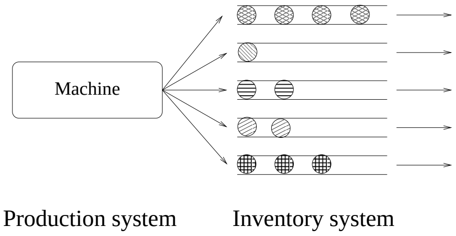

Figure 3: A production-inventory system with five product types.

4 A Production Inventory Task

This section presents the results of simulation experiments applying the SMART technique to the problem of optimal preventive maintenance in a discrete part production inventory system. Das and Sarkar [4] present a mathematical model of such a system for the case where only a single product is produced. Here we consider a similar system except that we add the extra complexity of multiple products, and multiple buffers.

The production system we consider is a discrete part production inventory system with a single machine capable of producing multiple product types to satisfy external demands (see Figure 3). The system also consists of inventories, or product buffers, one for each product type, that store the appropriate products as their production cycle is completed. The buffers for each product type have differing, finite sizes. When the buffer for the product type currently being produced becomes full, the machine changes production mode, and begins producing another product type, chosen according to a fixed product-switching policy. [3] If the buffers for all product types are full, the machine is said to go on vacation. During this period the machine does not perform any productions, and so does not age. Once the buffer for any product type falls below its maximum size, the machine again begins producing, commencing with that product type. A demand for a product type is satisfied if its buffer is not empty, otherwise the demand is lost.

Failures usually cause long interruptions in the production process, depending on the repair time, and are expensive due to the monetary costs associated with

[3] We used a *round-robin* policy of going to the next product. We are currently also experimenting with learning the best switching policy.

repair. Failures that occur at times when buffer levels are low, or buffers are empty, tend to be extremely costly due to the lost demand.

Failures may be avoided by performing maintenance on the machine. We assume that after a production is completed, the machine can either be maintained, or it can begin another production cycle. So, machine maintenance also interrupts production. In addition, maintenances have an associated cost. Fortunately in general, maintenance costs are usually less than repair costs, and we can assume that they take less time because they are planned. In addition, because maintenances are planned, it seems logical to perform them when the buffer levels are high. The idea is that even though production is interrupted, the service level (% of demands satisfied) can be kept high if the buffers contain enough products to satisfy demand.

The cost metric we chose is as follows. We use a reward rate (reward in $ incurred in unit time) performance measure defined as:

$$\rho_{rate} = \frac{R - C_r - C_m}{T} \tag{10}$$

where R is the total revenue from satisfied demand of all products, C_r is the total cost of repairs, C_m is the total cost of maintenances, and T is the total time.

4.1 The Simulation Model

The system we consider consists of a machine capable of producing 5 different products. The main parameters for the system are estimated as follows:

- Demand arrival process for each product i is Poisson (γ_i)
- Production time for each product has a Gamma (d_i, λ_i) distribution
- Time between failures has a Gamma (k, μ) distribution
- Time required for maintenance has a Uniform (a, b) distribution
- Time for repair has a Gamma (r, δ) distribution

The system is simulated as a discrete-event system where the significant events are: demand arrival, failure, production completion, repair completion, and maintenance completion. An on-vacation event occurs when all buffers reach their maximum level. When any buffer falls below its maximum level, an off-vacation event occurs, causing a decision epoch. Decision epochs also occur immediately following production completion, repair completion, and maintenance completion events. At these times, the agent may choose one of two actions: begin the next production cycle, or perform a maintenance.

4.2 State Space

The state of the system at any point in time is a 10-dimensional vector of integers $< P_1, B_1, \ldots, P_5, B_5 >$, where P_i is the number of products of type i produced since the last renewal (repair or maintenance), and B_i is the level of buffer for product i. The buffers have a limited finite size, but the number of production cycles that can possibly be completed for each product depends on the time between failures, and the time for production of each product. Due to the uncertainty involved in both these factors, it is difficult to determine the exact size of the state space. Instead we observe the number of production runs completed for each product in a system which produces, without maintenance, until failure, and obtain a rough estimate. For the systems outlined in Tables 1 and 2, the number of production runs until failure for each product type is $\approx 10^2$. The maximum buffer sizes are ≈ 10. . Therefore the size of the state space may be estimated as $\approx (10^2)^5 \cdot 10^5 = 10^{15}$.

4.3 Calculating Reinforcements

The costs incurred between decision epochs depends on the action taken, and the demands satisfied while that decision is carried out. For example, if the agent decides to perform a maintenance, then the cost incurred is equal to the total rewards from satisfied demands during the maintenance, minus the cost of maintenance. If the agent chooses to continue producing, and begins the next production cycle, then the costs incurred equals the the total rewards from satisfied demands until the next epoch, less the cost of a repair should a failure occur.

4.4 Network Architecture

The size of the state space of this system is large enough to rule out use of a lookup table. Instead, two multi-layer feedforward neural nets are used to estimate the action values for each action ("produce" or "maintain"). We experimented with a number of different strategies for encoding the state to the neural net. The approach that produced the best results

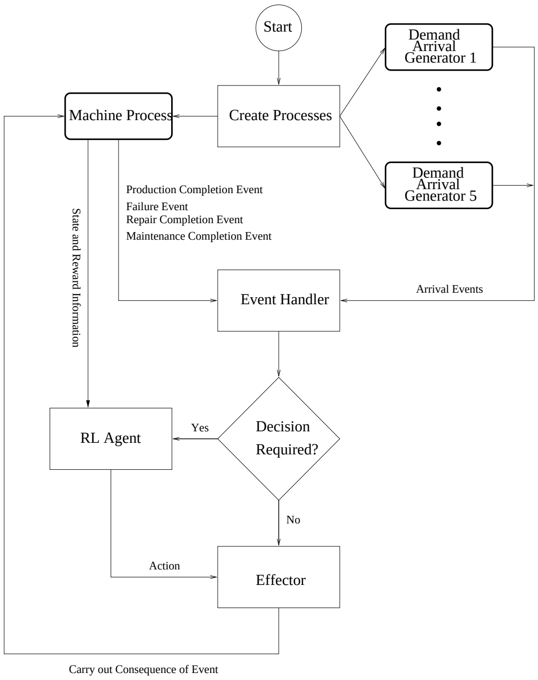

Figure 4: SMART is implemented using CSIM, a commercial discrete-event simulator.

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 to used 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.

4.5 Integrating SMART with CSIM and ARENA

We have implemented SMART using two commercial discrete-event simulators, ARENA and CSIM. ARENA is a graphical simulation language based on the well-known SIMAN modeling language [10]. The attraction of ARENA is that it allows a nice animation of the simulation, which is useful for understanding and demonstration purposes. However, the results described below were obtained with a simulation model implemented using CSIM, a C++ library of routines for creating process-oriented discrete-event models (see Figure 4). The key factor in why CSIM was chosen to implement the model was because of its efficiency. The RL Agent is simply a procedure call from an event handler. When an event pertaining to the machine occurs (production completion, maintenance completion, repair completion, or off-vacation), action choice is required, and the RL code is invoked.

5 Experimental Results

We now provide detailed experimental results on using SMART with CSIM to optimize the production inventory system described above. Since we know of no optimal solution to this problem, we compare the results produced by SMART to two well-known heuristic procedures, COR and AR. A brief description of the heuristics follows.

Coefficient of Operational Readiness (COR): The *coefficient of operational readiness* (COR) heuristic [7] attempts to maximize the uptime of the machine. The coefficient of operational readiness is given by:

$$COR = \frac{\text{uptime}}{\text{uptime} + \text{downtime}} \tag{11}$$

where *uptime* is defined as the total time spent producing, and *downtime* is the time spent performing repairs and maintenances. The idea behind the heuristic is to determine a time interval between preventive maintenances τ_m that will reduce the number of repairs, effectively reducing the total down time, and maximizing the coefficient of operational readiness. So we wish to determine

$$COR^* = \frac{\tau_m^*}{\tau_m^* + t_{maint} + kt_{repair}} \tag{12}$$

where COR^* is the optimal coefficient of operational readiness, τ_m^* is the optimum maintenance interval, k is an estimate[4] of the number of failures that occur during the interval τ_m, and t_{maint}, and t_{repair} are the average time for maintenances, and repairs respectively. COR^* can be determined by using a simple gradient descent algorithm to search for the optimum maintenance interval τ_m^*.

Age Replacement (AR): This heuristic has been studied in the context of machine maintenance by Schouten [5]. Here, the machine is maintained when it reaches age T. If the machine fails before time T, it is repaired. Both repair and maintenance renew the machine (i.e., reset its age to zero). Let C_m and C_r

[4] It can be shown that the average number of failures does not exceed $k = \frac{F(\tau_m)}{1-F(\tau_m)}$, where $F(\tau_m)$ is the value of the cumulative distribution function at τ_m of the random variable, time between failures [7].

denote the cost of maintenance and repair respectively. The renewal cycle time CT is given as

$$CT = \int_0^T (x+t_{repair}) f(x) dx + \int_T^\infty (T+t_{maint}) f(x) dx,$$

where $f(x)$ is the probability density function of the time to machine failure. Also the expected cost of a renewal cycle (EC) is given as

$$EC = P(X > T)C_m + P(X < T)C_r.$$

The average cost, using the renewal reward theorem, is $\rho = EC/CT$. Maintenance age T is selected such that ρ is minimized.

For the experiments described in this paper, we used 10 different systems which conform to the model described above, by varying the input parameters (see Table 1). For all systems considered, the parameters concerning the five different product types are as shown in Table 2.

Table 1: Production inventory system parameters. In all these systems, the repair cost was fixed at 5000.

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 2: Production inventory product parameters

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

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 net were fixed, and the simulation was rerun, using the nets to choose actions. That is, there was no exploration, and no learning. At each decision epoch, the agent simply chose the action with the maximum R-value. The results 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.

Figures 5 show the learning curve for the first system, as well as a plot showing the performance of the fixed SMART agent (no learning) versus the AR and COR heuristics.

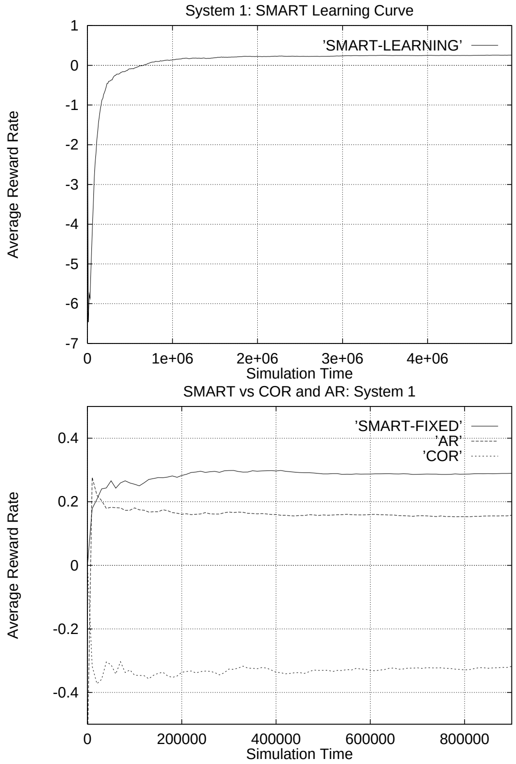

Figure 5: Results for production-inventory system 1. The graph on the top shows the SMART learning curve. The graph on the bottom shows the median-averaged cost rate, over 30 runs, incurred by the (fixed) policy learned by SMART, compared to the COR and AR heuristics.

Table 3 compares the average cost rate incurred by the policy learned by SMART versus that for the COR and AR heuristics, for all 10 systems. In all cases, SMART produced significantly [5] better results than both heuristics.

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

Table 3: Comparison of the average reward rate incurred by SMART vs. the two heuristics for the production inventory problem.

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

5.1 Interpreting the Results of Learning

In order to understand what the SMART algorithm has actually learned, we analyzed the results in greater detail. We summarize the main findings of our study here. Figure 6 compares the sensitivity of SMART vs. AR and COR, as the cost of maintenance is increased from a low value of 500 to a high value of 1200 (for the 10 systems in the above table). Note that the COR heuristic is essentially insensitive to maintenance cost. The total number of failures and maintenance actions, as well as the total vacation time (when the product buffers are full), for COR is essentially flat. This insensitivity clearly reveals the reason for the lackluster performance of the COR heuristic (e.g. as shown in the bottom graph in Figure 5).

Note that AR demonstrates a linear dependence on maintenance cost. As the cost of maintenance is increased, the number of maintenance actions performed by AR linearly decreases, whereas the number of failures and the vacation time correspondingly increases. Now, compare these with the situation for SMART, which demonstrates a somewhat nonlinear dependence on maintenance cost. The number of maintenance actions performed by SMART exhibits a nonlinear decrease as maintenance cost is increased. Similarly, the number of failures and vacation time increases nonlinearly as maintenance cost is increased. At the highest maintenance cost, SMART incurs almost 50% percent more failures than AR, whereas at the lowest maintenance cost, the number of failures incurred by SMART and AR are almost identical.

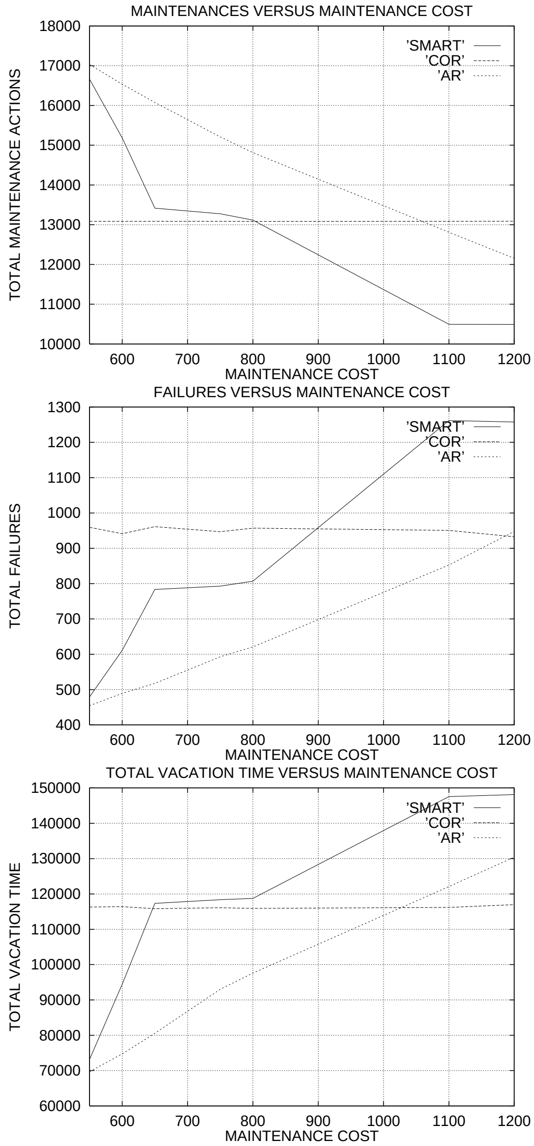

Figure 6: These graphs compare the sensitivity of AR, COR, and SMART to the maintenance cost. All plots are averaged over 30 runs of 1 million decision epochs. The graphs show that COR is insensitive to maintenance cost. AR exhibits a linear dependence, whereas SMART exhibits a nonlinear dependence. This suggests that SMART is more flexible than both the AR and COR heuristics in finding appropriate maintenance schedules as the costs are varied. Note that when maintenance costs are high, the policy learned by SMART causes more failures because it cuts back on maintenance. Nevertheless, the average reward rate of SMART remains higher than COR or AR over all 10 systems.

6 Summary and Future Work

This paper describes a new average reward reinforcement learning technique (SMART) that finds gain-optimal policies in continuous-time semi-Markov decision problems. SMART was implemented and tested on two commercial discrete-event simulators, ARENA and CSIM. The effectiveness of the SMART algorithm was demonstrated by applying it to a large scale unreliable production inventory system. To our knowledge, this is the first large-scale study of average-reward reinforcement learning.

This work is part of a major cross-disciplinary study of industrial process optimization using reinforcement learning. This paper only discussed a production system with a single machine, but real factories are much more complex systems consisting of numerous interrelated subsystems of machines [6]. In these cases, 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 [3] 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 [13] 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.

Acknowledgements

This research is supported in part by an NSF CAREER Award Grant No. IRI-9501852 (to Sridhar Mahadevan).

References

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

[2] 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.

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

[4] T. Das and S. Sarkar. Optimal preventive maintenance in a production inventory system. Submitted.

[5] F. Van der Duyn Schouten and S. Vanneste. Maintenance optimization of a production system with buffer capacity. *European Journal of Operational Research*, 82:323–338, 1995.

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

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

[8] A. Law and W. Kelton. *Simulation Modeling and Analysis*. McGraw-Hill, New York, USA, 1991.

[9] S. Mahadevan. Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine Learning*, 22:159–196, 1996.

[10] D. Pegden, R. Sadowski, and R. Shannon. *Introduction to Simulation using SIMAN*. McGraw Hill, New York, USA, 1995.

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

[12] A. Schwartz. A reinforcement learning method for maximizing undiscounted rewards. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 298–305. Morgan Kaufmann, 1993.

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

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

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

[16] P. Tadepalli and D. Ok. Auto-exploratory average-reward reinforcement learning. In *Proceedings of the Thirteenth AAAI*, pages 881–887. AAAI/MIT Press, 1996.

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