Designing Agent Controllers using Discrete-Event Markov Models

Sridhar Mahadevan and **Nikfar Khaleeli*** and **Nicholas Marchalleck†**
Department of Computer Science
Michigan State University
East Lansing, MI 48864
(mahadeva@cps.msu.edu)

Abstract

This paper describes the use of *discrete-event* Markov decision process models to design robust agent controllers in complex stochastic domains. Unlike discrete-time models, where actions are assumed to take unit time, discrete-event models allow state transitions to take random time. Discrete-event models also provide a convenient form of temporal abstraction: the agent observes and controls the system only at decision epochs. The paper summarizes case studies using discrete-event models in two challenging application domains: mobile robot navigation and manufacturing. The two domains also serve to highlight the difference between probabilistic models, where dynamic programming can be applied, and simulation models, where reinforcement learning methods are more appropriate.

Introduction

There are many domains where agents need to synthesize plans that optimize a time-based cost function. Consider the following examples. A delivery robot navigates around an indoor office environment, making periodic trips between different service locations. The goal of the robot is to plan routes to its destinations that minimize the travel time. The robot has to take into consideration the time taken by the various actions (moving forward, turning), and the variation in these times in different locations (for example, going through a door or an intersection or a cluttered corridor usually takes more time). As a second example, consider a manufacturing system that produces discrete parts. The parts are stored in product buffers, or inventories. Each part takes a different amount of time to manufacture. The machine making the parts is unreliable, and prone to periodic breakdown, unless it is maintained regularly. The problem is to design a maintenance schedule, which will minimize breakdowns, and maximize satisfied demand for the various parts.

These two examples, although arising from very different domains, have a lot in common. The two domains are both *stochastic*, in that there is significant uncertainty in perception (for example, the robot may not know where it is), actions are *non-deterministic* (for example, the size of the buffer when a production action is completed depends on demand arrivals in between), and the number of states may be very large (for example, the manufacturing domain involves on the order of 10^{15} states). The goal of the agent in both domains is to compute (or learn) a *policy* mapping (sensed) states to actions, preferably one that optimizes a cost-based optimality measure (for example, *discounted* sum of rewards, or the *average* reward per step).

This paper describes a general framework based on event-based Markov decision processes (Puterman 1994) to address temporal stochastic planning in the robotics and manufacturing domains. Event-based MDP models extend the usual discrete-time MDP models in two key ways: actions are modeled as temporally extended, with some underlying distribution; the agent observes the environment only at certain discrete points, where actions are necessary. In between decision epochs, the state of the system may be changing in some complex manner, due to other agents or processes. However, by only observing the system at decision epochs, the agent can nonetheless still devise a robust plan that optimizes a continuous cost function over the state of the *natural process* (i.e. the evolution of states over all time).

The paper summarizes experimental results from two ongoing research projects in the area of mobile robot navigation and stochastic manufacturing. The two domains form a nice contrast between the use of probabilistic event-based models, which allow the use of dynamic programming (Puterman 1994), versus the use of simulation models (Pegden, Sadowski, & Shan-

*Now at Windriver Systems.
†Now at Cybear Corporation.

non 1995), where approximation-based reinforcement learning methods are more appropriate.

Discrete-Event Models

Decision-making in many domains can be abstractly viewed as follows. At each step, the agent perceives (perhaps imperfectly) the underlying environment as being in one of a (possibly very large, but finite) set of states. The agent can choose one of a set of finite actions in a given state, and carry it out. The action modifies the environment in some way (or transports the agent around), thereby modifying the perceived state into a new state. Much recent work in autonomous agents, including reinforcement learning (Mahadevan & Kaelbling 1996), decision-theoretic planning (Boutilier, Dearden, & Goldszmidt 1995), and robot navigation (Simmons & Koenig 1995), has adopted this framework.

In this paper, we extend this discrete-time framework to a discrete-event framework, thereby generalizing it in two ways. Time is explicitly modeled as a continuous variable, but the agent observes the environment and makes decisions only at certain discrete points (or decision epochs). In between these epochs, the state of the system can be changing in some complex way, but these changes may not provide the agent with any additional information. Furthermore, actions take non-constant time periods, modeled by some arbitrary time distribution.

Formally, the evolution of the environment at decision epochs can be modeled as a semi-Markov decision process (SMDP) (Puterman 1994). It is termed a semi-Markov model, because the transition depends not only on the current state and action, but also on how long the system has been in the current state. 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 specifies the probability distribution of transition times for each state-action pair. $P(y \mid x, a)$ 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.

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.

Value Functions for SMDP

The optimality metric used to rank policies is usually either a discounted model, or an average-reward model. We will use both in this paper. The continuous time discounting model is defined as follows. A reward of 1 at the current time will be worth $e^{-\beta t}$ at a time t units in the future. Thus, the expected reward between two decision epochs, given that the agent was in state x, and chose a at the first decision epoch, may be expressed as the sum of the lump-sum cost and the expected discounted rate cost over the transition time

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

where τ is the transition time to the second decision epoch, and W_t denotes the state of the natural process. Here, E_x^a represents the expectation with respect to the transition time distribution $F(t|x,a)$.

Let $\pi : S \rightarrow A$ represent a stationary (for example, time-invariant) policy from states to actions. The value of a state s under a stationary policy π can be expressed as sum of the expected immediate reward until the next decision epoch and the expected discounted value of the resulting new state.

$$V_\beta^\pi(x) = r(x,a) + \sum_{y \in S} \int_0^\infty e^{-\beta t} Q(dt, y|x,a) V_\beta^\pi(y) \quad (2)$$

where $Q(dt, y|x, a)$ is the joint probability distribution as before. Note that by defining a discounted transition function

$$m(y|s,a) = \int_0^\infty e^{-\beta t} Q(dt, y|x,a)$$

we can express the discounted value function in more conventional form as simply

$$V_\beta^\pi(x) = r(x,a) + \sum_{y \in S} m(y|s,a) V_\beta^\pi(y) \quad (3)$$

In the average-reward paradigm, the discount factor $\beta = 0$. Also, the average reward ρ^π for a policy π can be defined by taking the limit of the ratio of the expected total reward up until the nth decision epoch to the expected total time until the nth epoch. The Bellman optimality equation for average reward SMDP's is written as

$$V^*(x) = \max_a \left(r(x,a) - \rho y(x,a) + \sum_{y \in S} P_{xy}(a) V^*(y) \right) \quad (4)$$

where the expected transition time is defined as:

$$y(x,a) = E_s^a \tau = \int_0^\infty t \sum_{y \in S} Q(dt, y \mid x, a) \tag{5}$$

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

Temporal Modeling of Actions

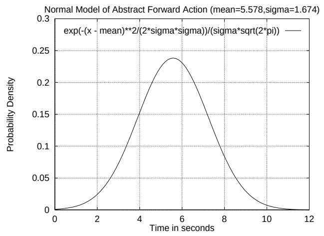

Figure 1: A learned temporal model of an abstract robot planning action, specifying the probability of the time taken to complete the action. The figure shows a normal distribution based temporal model, which was learned from actual execution using the robot.

In the mobile robot navigation task, states represent locations of the robot in an office environment (discretized at 1 meter resolution). Figure 1 shows a generic normal distribution model that is used for modeling an abstract action of moving forward 1 meter. Here, the transition time for an action is modeled as taking a random time that is normally distributed. We can instantiate this model for different actions by specifying different parameters of the distribution.

Loc.	Action	Mean	Std. Dev.
I3_0_S	go_forward	5.578	1.674
I4_0_E	go_forward	12.475	3.968
I4_0_W	turn_right	3.543	0.396
I5_0_E	turn_left	3.26	0.0439

Table 1: Normal temporal models of different abstract planning actions.

The parameters depend on what action is performed, and where it is performed (see Table 1). For example, a forward action taken in the middle of a corridor takes about 3 seconds (depending on the speed of the robot). The same forward action takes twice as long near intersections, since the robot usually slows down there. Turns take roughly the same time everywhere. Busy corridors are modeled by increasing the variance on the transition time (for example, anywhere from 10 to 100 seconds). The transition times can be learned from sample execution traces through actual corridors (see (Khaleeli 1997) for details).

The manufacturing problem we consider is a discrete part production inventory system with a single machine capable of producing multiple product types to satisfy external demands. 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 machine fails occasionally, which can cause long interruptions in the production process, depending on the repair time, and are expensive due to the monetary costs associated with repair. Failures may be avoided by performing maintenance on the machine.

Figure 2 illustrates the SMDP model for a very simplified version of this problem, where the 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.

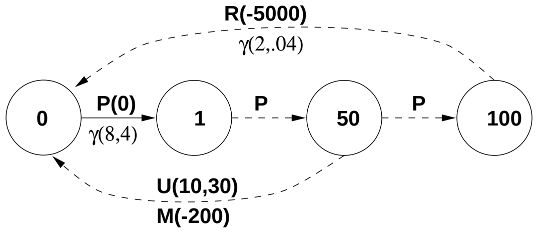

Figure 2: An SMDP model of a machine maintenance problem. States correspond to number of parts produced. Actions are repairs (R) and productions (P), whose times are γ distributed, and maintenances (M), whose times are uniformly distributed (U). Repairs carry a penalty of 5000, maintenances cost 500, and production actions have reward 0.

Planning using Event-based Models

Given a discrete-event model, the goal of the agent is to compute optimal plans that minimize a time-based cost function. Usually, in realistic domains, it is not possible to compute a provably optimal policy, but only an approximation. There are two general approaches, based on whether the agent has available a *transition* model or a *simulation* model.

Dynamic Programming

A transition model of the abstract action of going forward 1 meter is given in Figure 3. Here, the temporal model is a uniform distribution, allowing us to analytically compute the discounted transition function $m(y|x,a)$ as

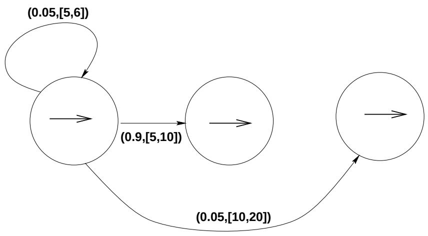

Figure 3: An SMDP model of an abstract "move-forward" action for a mobile robot. Arcs are labeled with the transition probability and transition time distribution.

$$\begin{aligned} m(y|x,a) &= \int_0^\infty e^{-\beta t} Q(dt, y|x,a) \\ &= P_{xy}(a) \int_0^\infty e^{-\beta t} F(dt, y|x,a) \\ &= \frac{P_{xy}(a)(e^{-\beta c} - e^{-\beta d})}{\beta(d-c)} \end{aligned}$$

The optimal value function over all states can be computed using the value iteration algorithm for discounted SMDP's (Puterman 1994):

$$V_k(s) = \max_a ((r(s,a) + m(s|s',a)V_{k-1}(s'))$$

This iteration is repeated over all states at each step until some stopping criterion is reached (for example, the values at successive iterates are close enough). We use the following improved stopping condition

$$sp|V_k(s) - V_{k-1}(s)| \leq \epsilon \left(\frac{1 - e^{-\beta}}{e^{-\beta}} \right)$$

where β is the continuous-time discount factor. sp is the span semi-norm function (Puterman 1994).

Partially-observable SMDP

A partially-observable semi-Markov decision process (POSMDP) extends the SMDP model by incorporating an observational model. This is necessary since robots cannot perceive the true underlying states of the environment, but can only make observations (which can be viewed as a probabilistic function of the true state). The partially observable nature of the model results from the fact that size of S is usually much larger than that of the observations O. For example, in our experiments, $|S| \approx 1200$, but $|O| = 256$. The observational model includes an observation function $\omega : S \rightarrow \Pi(O)$, which is a mapping from states to probability distributions on the space of observations.

As the robot moves about its environment, it generates a stream of observations which can be used to update its belief state. These updates occur at the end of each decision epoch, which is usually when an action (for example, move forward 1 meter) has terminated. At any decision epoch t, if the true state of the robot is s_t, an observation o is generated with probability $P(o \mid s_t)$.

The process of *state estimation* refers to an algorithm for maintaining a belief state or probability distribution on the underlying states, as the robot moves about in the environment. Note that the state estimation process depends on the prior belief state, the action that was just taken, and the observation that was generated. In our case, we will also use the transition time from one state to the next in the state estimation process. To simplify the presentation, we will explain the process of state estimation first for observations, and then for actions and time separately (although one can derive a joint update equation, as done in (Cassandra, Kaelbling, & Kurien 1996)).

We denote b to represent the belief state, and $b(s)$ to denote the probability of some state s as being the true state. Given an observation $o \in O$ at some decision epoch, the posterior belief state can be computed from the prior belief state as follows:

$$b'(s) = P(s \mid o) = \frac{P(o|s)b(s)}{P(o)}$$

where $b(s)$ is the prior probability of state s, and $b'(s)$ is the posterior probability, given observation o. The observation function ω is modeled as a table, and is computed from the topological map. Note that the denominator above is a normalization term that does not depend on the state.

In the discrete-time case, if the action performed was a, the belief state at the end of performing a can be estimated as

$$b'(s') = \frac{1}{scale} \sum_{s \in S | a \in A(s)} P(s' \mid s, a) b(s)$$

where *scale* is a normalization constant that ensures $\sum_{s \in S} b'(s) = 1$.

For SMDP's, the state estimation process for updating the belief state, given an action a was performed, which was observed to take time t, is more complex. Here, the time elapsed since the action was taken can be used as an observational input. The state estimation procedure for POSMDP's can be written as

$$b'(s') = \sum_{s \in S} \frac{F(dt|s,a,s')P(s'|a,s)b(s)}{\sum_{s \in S} F(dt|s,a)b(s)}$$

where $F(dt \mid \cdots)$ is the probability density function of the temporal distributional model of taking an action in some particular state.

Reinforcement Learning

In the manufacturing problem, it is not possible to analytically determine the transition model, because the underlying distributions are complex (for example, gamma model). Also, the number of states is very large, which makes it difficult to represent the model. A more desirable approach is to use a simulation-based dynamic programming (or reinforcement learning) algorithm. We have developed a new average-reward algorithm called SMART (for Semi-Markov Average Reward Technique) (Mahadevan *et al.* 1997). This algorithm is based on representing the value function using action values $R(x, a)$, which is the utility of doing action a in state x. These action values can be learned by running a simulation model of the manufacturing domain, and using a feedforward neural net to approximate the action values.

The derivation of the SMART algorithm from the Bellman equation for SMDP's (Equation 4) is fairly straightforward. The average-adjusted sum of rewards $R^\pi(x, a)$ received for the non-stationary policy of doing action a once, and then subsequently following a stationary policy π can be defined as

$$r(x,a) - \rho^\pi y(x,a) + \sum_z P(z \mid x,a) \max_b R^\pi(z,b) \quad (6)$$

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:[1]

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

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.

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

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

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. 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).

Value Function Approximation

In most interesting problems, such as the elevator control (Crites & Barto 1996), or the production inventory problem described below, the number of states is quite large, and rules out tabular representation of the action values. In particular, the state space in our problem is a 10-dimensional integer space. One standard approach, which we followed, is to use a feedforward net to represent the action value function (Crites & Barto 1996). 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 according to:

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

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 $\bigtriangledown_\phi R_n(x, a, \phi)$ is the vector of partial derivatives of $R_n(x, a, \phi)$ with respect to each component of ϕ.

Experimental Results

We have undertaken a detailed experimental study of both the robotics task, and the manufacturing task. This work is described in our recent papers and theses (Khaleeli 1997; Mahadevan *et al.* 1997; Marchalleck 1997). Here, we briefly summarize the results obtained.

PAVLOV: A Mobile Robot

Recently, there has been much research on the problem of designing robust robot navigation algorithms based on the framework of Partially-Observable Markov Decision Processes (POMDP's) (Cassandra, Kaelbling, & Kurien 1996; Nourbakhsh, Powers, & Birchfield 1995; Simmons & Koenig 1995). We have extended this work to event-based models, and implemented a novel mobile robot navigation architecture based on partially-observable semi-Markov decision processes (POSMDP). The POSMDP model allows the robot to explicitly model the transition time of different actions

Figure 4: The mobile robot PAVLOV uses discrete-event models to navigate around an indoor office environment.

in various locations in the environment (for example, passing through intersections takes more time than going through corridors).

The principal advantage of using event-based models in navigation is that the standard value iteration algorithm can itself take into account stochastic temporal constraints. For example, if a corridor is crowded, and the transition times of actions have a high variance, the policy computed by value iteration can choose a shorter time path and avoid going through the cluttered corridor.

The POSMDP approach was implemented on a real robot called PAVLOV (see Figure 4). Unlike previous approaches, the observation model (doors, openings, walls etc.) was learned by training a neural net. The robot was tested over the 3rd floor of the engineering building at the University of South Florida over a period of several months, over a total distance of several tens of kilometers (Khaleeli 1997). Transition times were learned by PAVLOV from actual execution traces, by using the Viterbi algorithm to determine the most likely sequence of states visited, and generalizing the observed transition times over multiple runs.

Figure 5 shows an odometric plot of PAVLOV traversing a cluttered corridor. In this run, the robot was asked to go from node 1 to node 5. The route taken by the robot is via node 4. Since the corridor (shown in Figure 6) was cluttered, the robot was repeatedly forced to go around obstacles in this run on the way to node 5 and back. The run was repeated several times to get sample transition times. The sample transition times are fitted to a distributional model (chosen from one of three possible profiles: normal, exponential, and uniform). The learned models are subsequently used

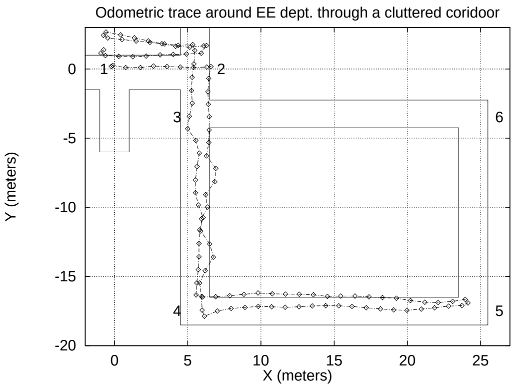

Figure 5: Odometric plot of PAVLOV showing it traversing a cluttered corridor.

Figure 6: This picture shows PAVLOV traversing a cluttered corridor.

in route planning, and as shown in Figure 7, PAVLOV is now able to plan around the cluttered corridor.

Self-Improving Factory Simulation

Many problems in industrial design and manufacturing, such as scheduling, queueing, inventory control, and reliability engineering, can be formulated as continuous-time SMDP's. Reinforcement learning (RL) is an ideal approach to solving large SMDP's, since it can be easily combined with discrete-event simulation models. We now demonstrate that the SMART algorithm (described above) can outperform two standard maintenance heuristics (age replacement (AR) and coefficient of operational readiness (COR)) on a realistic multi-product unreliable production inventory system (with a state space in excess of 10^{15} states). We have implemented SMART using two widely available discrete-event simulation packages (in particular, CSIM and ARENA) (Marchalleck 1997). This approach can undoubtedly also be applied to many other factory optimization problems, for which there already

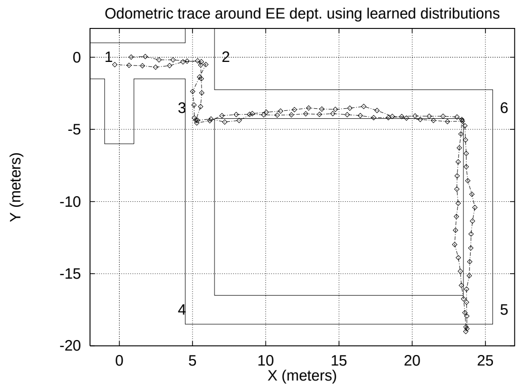

Figure 7: This plot shows PAVLOV going from node 1 to node 5, while avoiding the cluttered corridor leading to node 4.

exist simulation models.

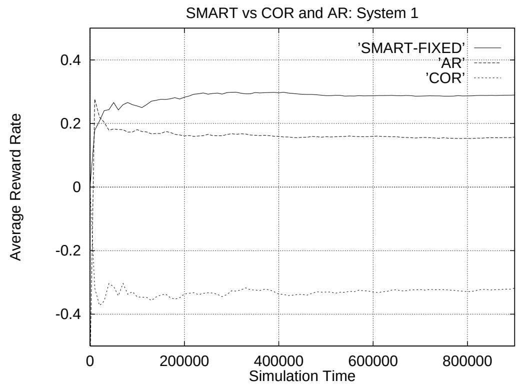

Figure 8: This graph shows the median-averaged cost rate, over 30 runs, incurred by the (fixed) policy learned by SMART, compared to the COR and AR heuristics.

Figures 8 compares the average reward of the maintenance policy learned by SMART, versus that for the two heuristic maintenance policies, AR and COR. Table 2 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[2] better results than both heuristics.

We undertook a further study to understand what the SMART algorithm has actually learned. A design of experiments study revealed maintenance cost to be the most significant factor in the overall system response (average reward of maintenance policy). Figure 9 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. 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 (for example, as shown in the bottom graph in Figure 8).

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

Table 2: 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

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 9 compares the fixed maintenance policy of the two heuristics, with that learned by the SMART algorithm. The plot shown is averaged over 30 fixed runs of 1 million decision epochs. The graph shows 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.

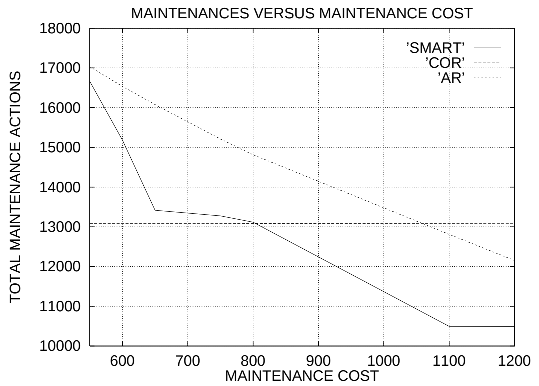

Figure 9: This figure compare the sensitivity of two maintenance heuristics AR and COR with the maintenance policy learned by the SMART algorithm.

Discussion and Future Work

This paper describes an enriched event-based probabilistic decision-making model called a semi-Markov decision process (SMDP). This model enables modeling actions that take non-constant time, as well as allows a form of temporal abstraction since decisions are only required at discrete points. Some experimental results in using these models in two complex application domains, robotics and manufacturing, are provided.

Much more work needs to be done in investigating discrete-event models as a means of controlling agents. One key direction is to use hierarchical abstraction to reduce the number of states in the model. In both the robotics and manufacturing domains, hierarchical abstraction of the underlying event-based model would be very useful in scaling the dynamic programming or reinforcement learning algorithms to solve larger problems. For example, real factories are much more complex systems consisting of numerous inter-related subsystems of machines (Gershwin 1994). 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 work of Crites and Barto on an event-based Q-learning algorithm for scheduling a team of elevators (Crites & Barto 1996) is a nice 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.

Acknowledgements

This research is supported by an NSF CAREER Award Grant No. IRI-9501852. We thank Billy Raulerson for writing some of the navigational software on PAVLOV, and Lynn Ryan for her detailed comments on a draft of this paper.

References

Boutilier, C.; Dearden, R.; and Goldszmidt, M. 1995. Exploiting structure in policy construction. In *Proceedings of the Fourteenth IJCAI.*

Cassandra, T.; Kaelbling, L.; and Kurien, J. 1996. Acting under uncertainty: Discrete bayesian models for mobile-robot navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 963–972.

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

Gershwin, S. 1994. *Manufacturing Systems Engineering.* Prentice Hall.

Khaleeli, N. 1997. A robust robot navigation architecture using partially observable semi-markov decision processes. Master's thesis, University of South Florida, Tampa, FL.

Mahadevan, S., and Kaelbling, L. P. 1996. The NSF workshop on reinforcement learning: Summary and observations. *AI Magazine* 17(4):89–97.

Mahadevan, S.; Marchalleck, N.; Das, T.; and Gosavi, A. 1997. Self-improving factory simulation using continuous-time average-reward reinforcement learning. In *Proceedings of the Fourteenth International Conference on Machine Learning.* Morgan Kaufmann.

Marchalleck, N. 1997. Improving simulation technology using reinforcement learning. Master's thesis, University of South Florida, Tampa, FL.

Nourbakhsh, I.; Powers, R.; and Birchfield, S. 1995. Dervish: An office-navigating robot. *AI Magazine* 16(2):53–60.

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

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

Simmons, R., and Koenig, S. 1995. Probabilistic robot navigation in partially observable environments. In *Proceedings of the IJCAI*, 1080–1087.