Prioritized Sweeping: Reinforcement Learning with Less Data and Less Real Time

Andrew W. Moore

Christopher G. Atkeson

MIT Artificial Intelligence Laboratory

545 Technology Square, Cambridge, MA 02139

Abstract

We present a new algorithm, Prioritized Sweeping, for efficient prediction and control of stochastic Markov systems. Incremental learning methods such as Temporal Differencing and Q-learning have fast real time performance. Classical methods are slower, but more accurate, because they make full use of the observations. Prioritized Sweeping aims for the best of both worlds. It uses all previous experiences both to prioritize important dynamic programming sweeps and to guide the exploration of state-space. We compare Prioritized Sweeping with other reinforcement learning schemes for a number of different stochastic optimal control problems. It successfully solves large state-space real time problems with which other methods have difficulty.

1 Introduction

This paper introduces a memory-based technique, *prioritized sweeping*, which can be used both for Markov prediction and reinforcement learning. Current, model-free, learning algorithms perform well relative to real time. Classical methods such as matrix inversion and dynamic programming perform well relative to the number of observations. Prioritized sweeping seeks to achieve the best of both worlds. Its closest relation from conventional AI is the search scheduling technique of the A^* algorithm (Nilsson 1971). It is a "memory-based" method (Stanfill and Waltz 1986) in that it derives much of its power from explicitly remembering all real-world experiences. Closely related research is being performed by Peng and Williams (1992) into a similar algorithm to prioritized sweeping, which they call Dyna-Q-queue.

We begin by providing a review of the problems and techniques in Markov prediction and control. More thorough reviews may be found in Sutton (1988), Barto *et al.* (1989), Sutton (1990), Kaelbling (1990) and Barto *et al.* (1991).

A discrete, finite Markov system has S *states.* Time passes as a series of discrete clock ticks, and on each tick the state may change. The probability of possible successor states is a function only of the current system state. The entire system can thus be specified by S and a table of transition probabilities.

$$\begin{array}{cccc} q_{11} & q_{12} & \cdots & q_{1S} \\ q_{21} & q_{22} & \cdots & q_{2S} \\ \vdots & \vdots & & \vdots \\ q_{S1} & q_{S2} & \cdots & q_{SS} \end{array} \tag{1}$$

where q_{ij} denotes the probability that, given we are in state i, we will be in state j on the next time step. The table must satisfy $\sum_{j=1}^{S} q_{ij} = 1$ for every i.

Figure 1 shows an example with six states corresponding to the six cells. With the exception of the rightmost states, on each time step the system moves at random to a neighbor. For example, state 1 moves directly to state 3 with probability $\frac{1}{2}$, and thus $q_{13} = \frac{1}{2}$.

The state-space of a Markov system is partitioned into two subsets: the non-terminal states NONTERMS, and the terminal states TERMS. Once a terminal state is entered, it is never left

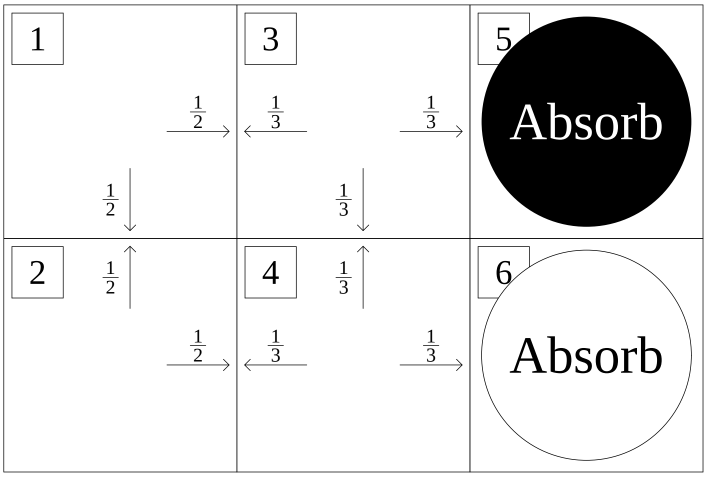


Figure 1: A six state Markov system.

($k \in \mathsf{TERMS} \Rightarrow q_{kk} = 1$). In the example, the two rightmost states are terminal.

A Markov system is defined as *absorbing* if from every non-terminal state it is possible to eventually enter a terminal state. We restrict our attention to absorbing Markov systems.

Let us first consider questions such as "starting in state i, what is the probability of eventual absorption by terminal state k?". Write this value as π_{ik}. All the absorption probabilities for terminal state k can be computed by solving the following set of linear equations. Assume that the non-terminal states are indexed by $1, 2, \ldots, S_{nt}$ where S_{nt} is the number of non-terminals.

$$\begin{array}{ccccccccccc} \pi_{1k} & = & q_{1k} & + & q_{11}\pi_{1k} & + & q_{12}\pi_{2k} & + & \cdots & + & q_{1S_{nt}}\pi_{S_{nt}k} \\ \pi_{2k} & = & q_{2k} & + & q_{21}\pi_{2k} & + & q_{22}\pi_{2k} & + & \cdots & + & q_{2S_{nt}}\pi_{S_{nt}k} \\ \vdots & & \vdots & & & & & & & & \vdots \\ \pi_{S_{nt}k} & = & q_{S_{nt}k} & + & q_{S_{nt}1}\pi_{1k} & + & q_{S_{nt}2}\pi_{2k} & + & \cdots & + & q_{S_{nt}S_{nt}}\pi_{S_{nt}k} \end{array} \tag{2}$$

When the transition probabilities $\{q_{ij}\}$ are known it is thus an easy matter to compute the eventual absorption probabilities. Machine learning can be applied to the case in which the transition probabilities are not known in advance, and all we may do instead is watch a series of state transitions. Such a series is normally arranged into a set of trials—each trial starts in some state and then continues until the system enters a terminal state. In our example, the learner might be shown

$$
\begin{array}{ccccccccccccc}
3 & \rightarrow & 4 & \rightarrow & 3 & \rightarrow & 1 & \rightarrow & 2 & \rightarrow & 4 & \rightarrow & 6 \\
3 & \rightarrow & 5 & & & & & & & & & & \\
1 & \rightarrow & 2 & \rightarrow & 1 & \rightarrow & 3 & \rightarrow & 5 & & & & \\
 & & & & & & \vdots & & & & & &
\end{array} \tag{3}
$$

Learning approaches to this problem have been widely studied. A recent contribution of great relevance is an elegant algorithm called *Temporal Differencing* (Sutton 1988).

1.1 The Temporal Differencing algorithm reviewed

We describe the discrete state-space case of the temporal differencing algorithm. TD can, however, also be applied to systems with continuous state-spaces in which long term probabilities are represented by parametric function approximators such as neural networks (Tesauro 1991).

The prediction process runs in a series of epochs. Each epoch ends when a terminal state is entered. Assume we have passed through states $i_1, i_2, \ldots i_n, i_{n+1}$ so far in the current epoch. n is our age within the epoch and t is our global age. $i_n \rightarrow i_{n+1}$ is the most recently observed transition. Let $\hat{\pi}_{ik}[t]$ be the estimated value of π_{ik} after the system has been running for t state transition observations. Then the TD algorithm for discrete state-spaces updates these estimates according to the following rule:

$$
\begin{array}{l}
\text{for each } i \in \mathsf{NONTERMS}\,\textit{(the set of non-terminal states)} \\
\quad \text{for each } k \in \mathsf{TERMS}\,\textit{(the set of terminal states)} \\
\qquad \hat{\pi}_{ik}[t+1] = \hat{\pi}_{ik}[t] + \alpha\left(\hat{\pi}_{i_{n+1}k}[t] - \hat{\pi}_{i_n k}[t]\right) \sum_{j=1}^{n} \lambda^{j-n} X_i(i_j)
\end{array} \tag{4}
$$

where α is a learning rate parameter $0 < \alpha < 1$, where λ is a memory constant $0 \leq \lambda \leq 1$ and where

$$
X_i(i_j) = \begin{cases} 1 & \text{if } i_j = i \\ 0 & \text{otherwise} \end{cases} \tag{5}
$$

In practice there is a computational trick which requires considerably less computation than the algorithm of Equation (4) but which computes the same values (Sutton 1988). The TD algorithm then requires $O(S_t)$ computation steps per real observation, where S_t is the number of terminal states. Convergence proofs exist for several formulations of the TD algorithm (Sutton 1988; Dayan 1992).

1.2 The classical approach

The classical method proceeds by building a maximum likelihood model of the state transitions. q_{ij} is estimated by

$$\hat{q}_{ij} = \frac{\text{Number of observations } i \rightarrow j}{\text{Number of occasions in state } i} \tag{6}$$

After $t+1$ observations the new absorption probability estimates are computed to satisfy, for each terminal state k, the $S_{nt} \times S_{nt}$ linear system

$$\hat{\pi}_{ik}[t+1] = \hat{q}_{ik} + \sum_{j \in \mathbf{succs}(i) \cap \mathsf{NONTERMS}} \hat{q}_{ij} \hat{\pi}_{jk}[t+1] \tag{7}$$

where $\mathbf{succs}(i)$ is the set of all states which have been observed as immediate successors of i and $\mathsf{NONTERMS}$ is the set of non-terminal states. It is clear that if the $\hat{q}_{ik}$ estimates were correct then the solution of Equation (7) would be the solution of Equation (2).

Notice that the values $\hat{\pi}_{ik}[t+1]$ depend only on the values of $\hat{q}_{ik}$ after $t+1$ observations—they are not defined in terms of the previous absorption probability estimates $\hat{\pi}_{ik}[t]$. However, it is efficient to solve Equation (7) iteratively. Let $\{\rho_{ik}\}$ be a set of intermediate iteration variables containing intermediate estimates of $\hat{\pi}_{ik}[t+1]$. What initial estimates should be used to start the iteration? An excellent answer is to use the previous absorption probability estimates $\hat{\pi}_{ik}[t]$.

The complete algorithm, performed once after every real-world observation, is shown in Figure 2. The transformation on the ρ_{ik}'s can be shown to be a *contraction mapping* as defined in Section 3.1 of Bertsekas and Tsitsiklis (1989), and thus, as the same reference proves, convergence to a solution satisfying Equation (7) is guaranteed. If, according to the estimated transitions, all states can reach a terminal state, then this solution is unique. The inner loop ("for each $k \in \mathsf{TERMS} \cdots$") is referred to as a probability *backup* operation, and requires $O(S_t \mu_{\text{succs}})$ basic operations, where μ_{succs} is the mean number of observed stochastic successors.

1. for each $i \in \mathsf{NONTERMS}$, for each $k \in \mathsf{TERMS}$,

$$\rho_{ik} \texttt{ := } \hat{\pi}_{ik}[t]$$

2. repeat

   2.1 $\Delta_{\max} \texttt{ := } 0$

   2.2 for each $i \in \mathsf{NONTERMS}$

   for each $k \in \mathsf{TERMS}$

$$\rho_{\text{new}} = \hat{q}_{ik} + \sum_{j \in \mathbf{succs}(i)} \hat{q}_{ij} \rho_{jk}$$

$$\Delta \texttt{ := } |\, \rho_{\text{new}} - \rho_{ik} \,|$$

$$\rho_{ik} \texttt{ := } \rho_{\text{new}}$$

$$\Delta_{\max} \texttt{ := } \max(\Delta_{\max}, \Delta)$$

   until $\Delta_{\max} < \epsilon$

3. for each $i \in \mathsf{NONTERMS}$, for each $k \in \mathsf{TERMS}$

$$\hat{\pi}_{ik}[t+1] \texttt{ := } \rho_{ik}$$

Figure 2: Stochastic prediction with full Gauss-Seidel iteration.

Gauss-Seidel is an expensive algorithm, requiring $O(S_{nt})$ backups per real-world observation for the inner loop 2.2 alone. The absorption predictions before the most recent observation, $\hat{\pi}_{ik}[t]$, normally provide an excellent initial approximation, and only a very few iterations are required. However, when an "interesting" observation is encountered, for example a previously never-experienced transition to a terminal state, many iterations, perhaps more than S_{nt}, are needed for convergence.

2 Prioritized Sweeping

Prioritized sweeping is designed to perform the same task as Gauss-Seidel iteration while using careful bookkeeping to concentrate all computational effort on the most "interesting" parts of the system. It operates in a similar computational regime as the Dyna architecture (Sutton 1990), in which a fixed, but non-trivial, amount of computation is allowed between each real-world observation. Peng and Williams (1992) are exploring a closely related approach to prioritized sweeping, developed from Dyna and Q-learning (Watkins 1989).

Prioritized sweeping uses the Δ value from the probability update step 2.2 in the previous algorithm to determine which other updates are likely to be "interesting"—if the step produces a large change in the state's absorption probabilities then it is interesting because it is likely that the absorption probabilities of the predecessors of the state will change given an opportunity. If, on the other hand, the step produces a small change then we will assume that there is less urgency to process the predecessors. The predecessors of a state i are all those states i' which have, at least once in the history of the system, performed a one-step transition $i' \rightarrow i$.

If we have just changed the absorption probabilities of i by Δ, then the maximum possible one-processing-step change in predecessor i' caused by our change in i is $\hat{q}_{i'i}\Delta$. This value is the priority P of the predecessor i', and if i' is not currently on the priority queue it is placed there at priority P. If it *is* already on the queue, but at lower priority, then it is promoted.

After each real-world observation $i \rightarrow j$, the transition probability estimate $\hat{q}_{ij}$ is updated along with the probabilities of transition to all other previously observed successors of i. Then state i is promoted to the top of the priority queue so that its absorption probabilities are updated immediately. Next, we continue to process further states from the top of the queue. Each state that is processed may result in the addition or promotion of its predecessors within the queue. This loop continues for a preset number of processing steps or until the queue empties.

Thus if a real world observation is interesting, all its predecessors and their earlier ancestors quickly find themselves near the top of the priority queue. On the other hand, if the real world observation is unsurprising, then the processing immediately proceeds to other, more important areas of state-space which had been under consideration on the previous time step. These other areas may be different from those in which the system currently finds itself.

Let us look at the formal algorithm in Figure 3. On entry we assume the most recent state transition was from i_{recent}. We drop the $[t]$ suffix from the $\hat{\pi}_{ik}[t]$ notation.

The decision of when we are allowed further processing, at the start of Step 2, could be implemented in many ways. In our subsequent experiments the rule is simply that a maximum of β backups are permitted per real-world observation.

There are many possible priority queue implementations, including a heap (Knuth 1973), which was used in all experiments in this paper. The cost of the algorithm is

$$O\left(\beta S_t(\mu_{\text{succs}} + \mu_{\text{preds}}\mathtt{PQCOST}(S_{nt}))\right) \tag{8}$$

basic operations, where at most β states are processed from the priority queue and $\mathtt{PQCOST}(N)$ is the cost of accessing a priority queue of length N. For the heap implementation this is $\log_2 N$.

States are only added to the queue if their priorities are above a tiny threshold ϵ. This is a value close to the machine floating-point precision. Stopping criteria are fraught with danger, but in this paper we discuss such dangers no further except to note that in our experiments they have caused no problem.

Prioritized sweeping is a heuristic, and in this paper no formal proof of convergence, or convergence rate, is given. We expect to be able to prove convergence using techniques from asynchronous Dynamic Programming (Bertsekas and Tsitsiklis 1989) and variants of the Temporal Differencing analysis of Dayan (1992). Later, this paper gives some empirical experiments in which convergence is relatively fast.

1. Promote state i_{recent} to top of priority queue.

2. While we are allowed further processing and priority queue not empty

   2.1 Remove the top state from the priority queue. Call it i

   2.2 $\Delta_{\max} = 0$

   2.3 for each $k \in \mathsf{TERMS}$

   $$\rho_{\text{new}} = \hat{q}_{ik} + \sum_{j \in \mathbf{succs}(i) \cap \mathsf{NONTERMS}} \hat{q}_{ij} \hat{\pi}_{jk}$$

   $$\Delta := \mid \rho_{\text{new}} - \hat{\pi}_{ik} \mid$$

   $$\hat{\pi}_{ik} := \rho_{\text{new}}$$

   $$\Delta_{\max} := \max(\Delta_{\max}, \Delta)$$

   2.4 for each $i' \in \mathbf{preds}(i)$

   $$P := \hat{q}_{i'i} \Delta_{\max}$$

   If $P > \epsilon$ (a tiny threshold) and if (i' is not on queue or P exceeds the current priority of i') then promote i' to new priority P.

Figure 3: The prioritized sweeping algorithm.

The memory requirements of learning the $S \times S$ transition probability matrix, where S is the number of states, may initially appear prohibitive, especially since we intend to operate with more than 10,000 states. However, we need only allocate memory for the experiences the system actually has, and for a wide class of physical systems there is not enough time in the lifetime of the system to run out of memory.

Similarly, the average number of successors and predecessors of states in the estimated transition matrix can be assumed $<< S$. A simple justification is that few real problems are fully connected, but a deeper reason is that for large S, even if the true transition probability matrix is not sparse, there will never be time to gain enough experience for the estimated transition matrix to not be sparse.

3 A Markov Prediction Experiment

Consider the 500 state Markov system depicted in Figure 4, which is a more complex version of the problem presented in Figure 1. Appendix A gives details of how this problem was randomly generated. The system has sixteen terminal states, depicted by white and black circles. The prediction problem is to estimate, for every non-terminal state, the long-term probability that it will terminate in a black, rather than a white, circle. The data available to the learner is a sequence of observed state transitions.

Temporal differencing, the classical method, and prioritized sweeping were all applied to this problem. Each learner was shown the same sequence. TD used parameters $\lambda = 0.25$ and $\alpha = 0.05$, which gave the best performance of a number of manually optimized experiments. The classical method was required to compute up to date absorption probability estimates after every real-world observation. Prioritized sweeping was allowed five backups per real experience ($\beta = 5$); it thus updated the $\hat{\pi}_{ik}$ estimates for the five highest priority states between each real-world observation. The threshold for ignoring tiny changes, ϵ, was 10^{-5}. Each method was evaluated at a number of stages of learning, by stopping the real time clock and computing the error between the estimated white-absorption probabilities which we denote by $\hat{\pi}_{i,\mathtt{WHITE}}$ and the true values which we denote by

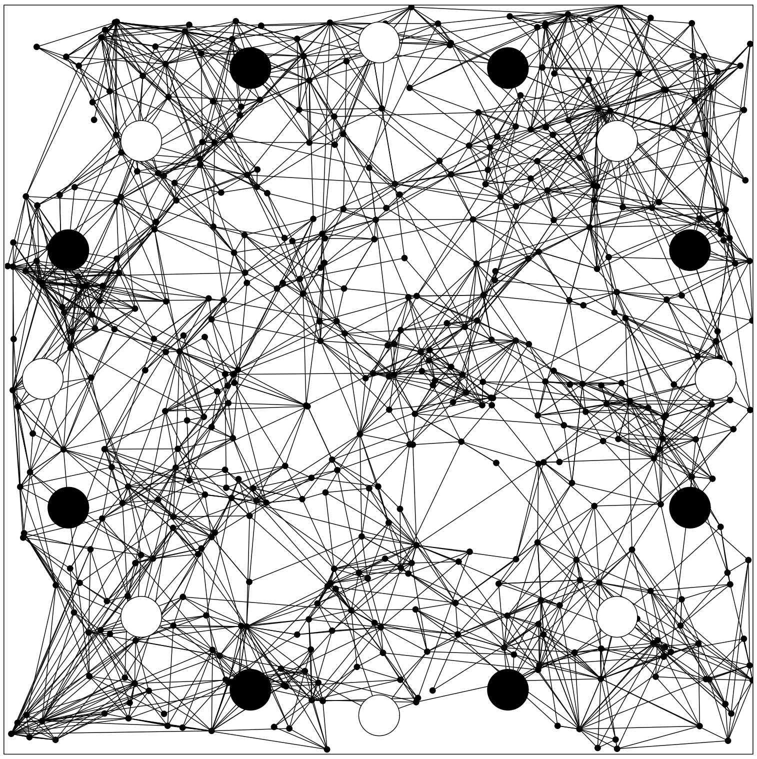


Figure 4: A 500-state Markov system. Each state has, on average, 5 stochastic successors.

$\pi_{i,\mathtt{WHITE}}$. The following RMS error over all states was recorded:

$$\sqrt{\frac{1}{S_{nt}} \sum_{i=0}^{S_{nt}} \left(\pi_{i\mathtt{WHITE}} - \hat{\pi}_{i\mathtt{WHITE}}\right)^2} \tag{9}$$

In Figure 5 we look at the RMS error plotted against the number of observations. After 100,000 experiences all methods are performing well; TD is the weakest but even it manages an RMS error of only 0.1.

In Figure 6 we look at a different measure of performance: plotted against real time. Here we see the great weakness of the classical technique. Performing the Gauss-Seidel algorithm of Figure 2 after each observation gives excellent predictions but is very time consuming, and after 300 seconds there has only been time to process a few thousand observations. After the same amount of time, TD has had time to process almost half a million observations. Prioritized sweeping performs best relative to real time. It takes approximately ten times as long as TD to process each observation but because the data is used more effectively, convergence is superior.

Ten further experiments, each with a different random 500 state problem, were run. These

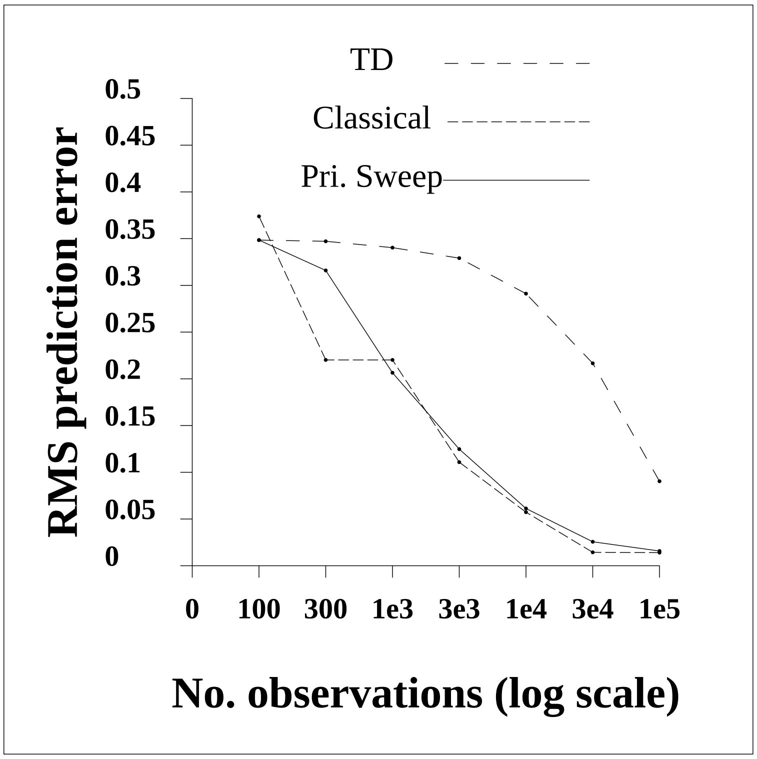


Figure 5: RMS prediction error between true absorption probabilities and predicted values, plotted against number of data-points observed. For prioritized sweeping, $\beta = 5$ and $\epsilon = 10^{-5}$.

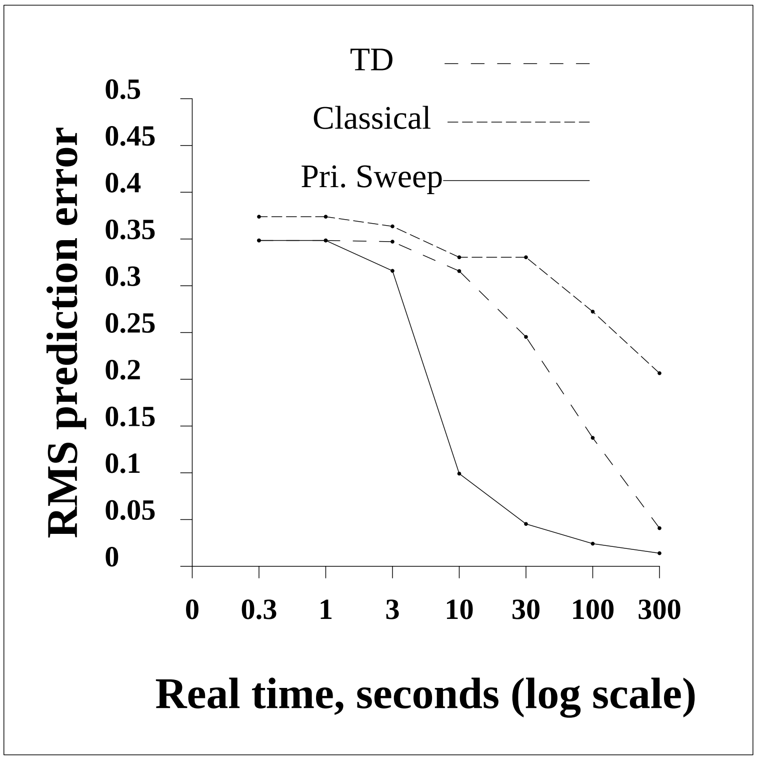


Figure 6: RMS prediction error between true absorption probabilities and predicted values, plotted against real time, in seconds, running the problem on a Sun-4 workstation.

	TD	Classical	Pri. Sweep
After 100,000 observations	0.14 ± 0.077	0.024 ± 0.0063	0.024 ± 0.0061
After 300 seconds	0.079 ± 0.067	0.23 ± 0.038	0.021 ± 0.0080

Table 1: RMS prediction error: mean and standard deviation for ten experiments.

further runs, the final results of which are given in Table 1, indicate that the graphs in Figures 5 and 6 are not atypical.

This example has shown the general theme of this paper. Model-free methods perform well in real time but make weak use of their data. Classical methods make good use of their data but are often impractically slow. Techniques such as prioritized sweeping are interesting because they may be able to achieve both.

There is an important footnote concerning the classical method. If the problem had only required that a prediction be made after all transitions had been observed, then the only real time cost would have been recording the transitions in memory. The absorption probabilities could then have been computed as an individual large computation at the end of the sequence, giving the best possible estimate with a relatively small overall time cost. For the 500-state problem, we estimate the cost as approximately 30 seconds for 100,000 points. Prioritized sweeping could also benefit from only being required to predict after seeing all the data, although with little advantage over the simpler, classical algorithm. Prioritized sweeping is thus most usefully applicable to the class of tasks in which a prediction is required on *every* time step. Furthermore, the remainder of the paper concerns control of Markov decision tasks, in which the maintenance of up to date predictions is particularly beneficial.

4 Learning Control of Markov Decision Tasks

Let us consider a related stochastic prediction problem, which bridges the gap between Markov prediction and control. Suppose the system gets rewarded for entering certain states and punished for entering others. Let the reward of the ith state be r_i. An important quantity is then the *expected discounted reward-to-go* of each state. This is an infinite sum of expected future rewards, with each term supplemented by an exponentially decreasing weighting factor γ^k where γ is called the *discount factor*. The expected discounted reward-to-go is

$$\begin{array}{rcl} J_i & = & \text{(This reward)}+ \\ & \gamma & \text{(Expected reward in 1 time step)}+ \\ & \gamma^2 & \text{(Expected reward in 2 time steps)}+ \\ & & \vdots \\ & \gamma^k & \text{(Expected reward in } k \text{ time steps)}+ \\ & & \vdots \end{array} \tag{10}$$

For each i, J_i can be computed recursively as a function of its immediate successors.

$$\begin{array}{lllllllllll} J_1 & = & r_1 & +\gamma & (q_{11}J_1 & + & q_{12}J_2 & + & \ldots & + & q_{1S}J_S) \\ J_2 & = & r_2 & +\gamma & (q_{21}J_2 & + & q_{22}J_2 & + & \ldots & + & q_{2S}J_S) \\ \vdots & & \vdots & & & & & & & & \vdots \\ J_S & = & r_S & +\gamma & (q_{S1}J_1 & + & q_{S2}J_2 & + & \ldots & + & q_{SS}J_S) \end{array} \tag{11}$$

which is another set of linear equations that may be solved if the transition probabilities q_{ij} are known. If they are not known, but instead a sequence of state transitions and r_i observations is given, then slight modifications of TD, the classical algorithm, and prioritized sweeping can all be used to estimate J_i.

Markov decision tasks

Markov decision tasks are an extension of the Markov model in which, instead of passively watching the state move around randomly, we are able to influence it.

Associated with each state, i, is a finite, discrete set of actions, **actions**(i). On each time step, the controller must choose an action. The probabilities of potential next states depend not only on the current state, but also on the chosen action. We will supplement our example problem with actions:

$$\begin{array}{lll} \textbf{actions}(1) = \{\texttt{RANDOM},\texttt{RIGHT}\} & \textbf{actions}(3) = \{\texttt{RANDOM},\texttt{RIGHT}\} & \textbf{actions}(5) = \{\texttt{STAY}\} \\ \textbf{actions}(2) = \{\texttt{RANDOM},\texttt{RIGHT}\} & \textbf{actions}(4) = \{\texttt{RANDOM},\texttt{RIGHT}\} & \textbf{actions}(6) = \{\texttt{STAY}\} \end{array} \tag{12}$$

where RANDOM causes the same random transitions as before, RIGHT moves, with probability 1, to the cell immediately to the right, and STAY makes us remain in the same state. There is still no escape from states 5 and 6.

We use the notation q_{ij}^{a} for the probability that we move to state j, given that we have commenced in state i and applied action a. Thus, in our example $q_{13}^{\mathtt{RANDOM}} = \frac{1}{2}$ and $q_{13}^{\mathtt{RIGHT}} = 1$.

A *policy* is a mapping from states to actions. For example, Figure 7 shows the policy

$$\begin{array}{lll} 1 \rightarrow \mathtt{RIGHT} & 3 \rightarrow \mathtt{RIGHT} & 5 \rightarrow \mathtt{STAY} \\ 2 \rightarrow \mathtt{RANDOM} & 4 \rightarrow \mathtt{RANDOM} & 6 \rightarrow \mathtt{STAY} \end{array} \qquad (13)$$

If the controller chooses actions according to a fixed policy then it behaves like a Markov system. The expected discounted reward-to-go can then be defined and computed in the same manner as Equation (11).

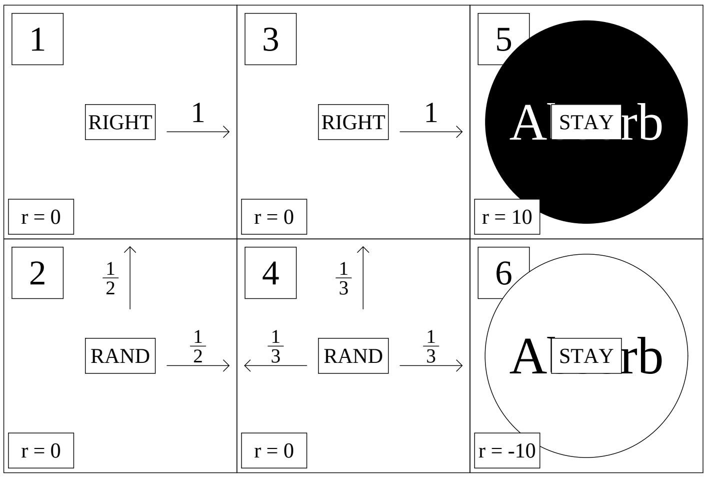


Figure 7: The policy defined by Equation (13). Also shown is a reward function (bottom left of each cell). Large expected reward-to-go involves getting to '5' and avoiding '6'.

If the goal is large reward-to-go, then some policies are better than others. An important result from the theory of Markov decision tasks tells us that there always exists at least one policy which is *optimal* in the following sense. For every state, the expected discounted reward-to-go using an optimal policy is no worse than that from any other policy.

Furthermore, there is a simple algorithm for computing both an optimal policy and the expected discounted reward-to-go of this policy. The algorithm is called *Dynamic Programming* (Bellman 1957). It is based on the following relationship known as *Bellman's optimality equation* which holds between the optimal expected discounted reward-to-go at different states.

$$J_i = \max_{a \in \textbf{actions}(i)} (r_i + \gamma(q^a_{i1}J_1 + q^a_{i2}J_2 + \cdots + q^a_{iS}J_S)) \tag{14}$$

Dynamic programming applied to our example gives the policy shown in Figure 7, which happens to be the unique optimal policy.

A very important question for machine learning has been how to obtain an optimal, or near optimal, policy when the q^a_{ij} values are not known in advance. Instead, a series of actions, state transitions, and rewards is observed. For example:

$$\begin{array}{l}
1(r_1 = 0) \stackrel{\texttt{RANDOM}}{\longrightarrow} 2(r_2 = 0) \stackrel{\texttt{RANDOM}}{\longrightarrow} 4(r_4 = 0) \stackrel{\texttt{RIGHT}}{\longrightarrow} 6(r_6 = 10) \\
2(r_2 = 0) \stackrel{\texttt{RANDOM}}{\longrightarrow} 1(r_1 = 0) \stackrel{\texttt{RIGHT}}{\longrightarrow} 3(r_3 = 0) \stackrel{\texttt{RANDOM}}{\longrightarrow} 5(r_5 = -10) \\
3(r_3 = 0) \stackrel{\texttt{RANDOM}}{\longrightarrow} 5(r_5 = 10) \\
\quad\quad\vdots
\end{array} \tag{15}$$

A critical difference between this problem and the Markov prediction problem of the earlier sections is that the controller now affects which transitions are seen, because it supplies the actions.

The question of learning such systems is studied by the field of *reinforcement learning*, which is also known as "learning control of Markov decision tasks". Early contributions to this field were the checkers player of Samuel (1959) and the BOXES system of Michie and Chambers (1968). Even systems which may at first appear trivially small, such as the two armed bandit problem (Berry and Fristedt 1985) have promoted rich and interesting work in the statistics community.

The technique of gradient descent optimization of neural networks in combination with approximations to the policy and reward-to-go (called the "adaptive heuristic critic") was introduced by Sutton (1984). Kaelbling (1990) introduced several applicable techniques, including the *Interval Estimation algorithm*. Watkins (1989) introduced an important model-free asynchronous Dynamic Programming technique called Q-learning. Sutton (1990) has extended this further with the Dyna architecture. Christiansen *et al.* (1990) applied a planner, closely related to Dynamic Programming, to a tray tilting robot. An excellent review of the entire field may be found in (Barto *et al.* 1991).

4.1 Prioritized sweeping for learning control of Markov decision tasks

The main differences between this case and the previous application of prioritized sweeping are

1. We need to estimate the optimal discounted reward-to-go, J, of each state, rather than the eventual absorption probabilities.

2. Instead of using the absorption probability backup Equation (6), we use Bellman's equation (Bellman 1957; Bertsekas and Tsitsiklis 1989):

$$\hat{J}_i = \max_{a \in \mathbf{actions}(i)} \left(\hat{r}_i^a + \gamma \times \sum_{j \in \mathbf{succs}(i,a)} \hat{q}_{ij}^a \hat{J}_j \right) \tag{16}$$

   where $\hat{J}_i$ is the estimate of the optimal discounted reward starting from state i, γ is the discount factor, **actions**(i) is the set of possible actions in state i, and $\hat{q}_{ij}^a$ is the maximum likelihood estimated probability of moving from state i to state j given that we have applied action a. The estimated immediate reward, $\hat{r}_i^a$, is computed as the mean reward experienced to date during all previous applications of action a in state i.

3. The rate of learning can be affected considerably by the controller's exploration strategy.

The algorithm for prioritized sweeping in conjunction with Bellman's equation is given in Figure 8. The only substantial difference between this algorithm and the prediction case is the state backup step, namely the Bellman's equation application of Step 2.2. Notice also that the predecessors of a state are now a set of state-action pairs.

Let us now consider the question of how best to gain useful experience in a Markov decision task. The formally correct method would be to compute that exploration which maximizes the expected reward received over the robot's remaining life. This computation, which requires a prior probability distribution over the space of Markov decision tasks, is unrealistically expensive. It is computationally exponential in all of (i) the number of time steps for which the system is to remain alive (ii) the number of states in the system, and (iii) the number of actions available (Berry and Fristedt 1985).

An exploration heuristic is thus required. Kaelbling (1990) and Barto *et al.* (1991) both give excellent overviews of the wide range of heuristics which have been proposed.

1. Promote state i_{recent} to top of priority queue.

2. While we are allowed further processing and priority queue not empty

   2.1 Remove the top state from the priority queue. Call it i

   2.2 $\rho_{\text{new}} := \max_{a \in \mathbf{actions}(i)} \left(\hat{r}_i^a + \gamma \times \sum_{j \in \mathbf{succs}(i,a)} \hat{q}_{ij}^a \hat{J}_j \right)$

   2.3 $\Delta_{\max} := \mid \rho_{\text{new}} - \hat{J}_i \mid$

   2.4 $\hat{J}_i := \rho_{\text{new}}$

   2.5 for each $(i', a') \in \mathbf{preds}(i)$

   $P := \hat{q}_{i'i}^{a'} \Delta_{\max}$

   If $P > \epsilon$ (a tiny threshold) and if (if i' not on queue or P exceeds the current priority of i') then promote i' to new priority P.

Figure 8: The prioritized sweeping algorithm for Markov Decision Tasks.

We use the philosophy of *optimism in the face of uncertainty*, a method successfully developed by the Interval Estimation (IE) algorithm of Kaelbling (1990) and by the exploration bonus technique in Dyna (Sutton 1990). The same philosophy is also used by Thrun and Möller (1992).

A slightly different heuristic is used with the prioritized sweeping algorithm. This is because of minor problems of computational expense for IE and the instability of the exploration bonus in large state-spaces.

The slightly different optimistic heuristic is as follows. In the absence of contrary evidence, any action in any state is assumed to lead us directly to a fictional absorbing state of permanent large reward r^{opt}. The amount of evidence to the contrary which is needed to quench our optimism is a system parameter, T_{bored}. If the number of occurrences of a given state-action pair is less than T_{bored}, we assume that we will jump to fictional state with subsequent long term reward $r^{\mathrm{opt}} + \gamma r^{\mathrm{opt}} + \gamma^2 r^{\mathrm{opt}} + \ldots = r^{\mathrm{opt}}/(1-\gamma)$. If the number of occurrences is not less than T_{bored}, then we use the true, non-optimistic, assumption. Thus the optimistic reward-to-go estimate $\hat{J}^{\mathrm{opt}}$ is

$$\hat{J}_i^{\mathrm{opt}} = \max_{a \in \mathbf{actions}(i)} \begin{cases} r^{\mathrm{opt}}/(1-\gamma) & \text{if } n_i^a < T_{\mathrm{bored}} \\ \hat{r}_i^a + \gamma \times \displaystyle\sum_{j \in \mathbf{succs}(i,a)} \hat{q}_{ij}^a \hat{J}_j^{\mathrm{opt}} & \text{otherwise} \end{cases} \tag{17}$$

where n_i^a is the number of times action a has been tried to date in state i. The important feature, identified by Sutton (1990), is the *planning to explore* behavior caused by the appearance of the optimism on both sides of the equation. A related exploration technique was used by (Christiansen *et al.* 1990). Consider the situation in Figure 9. The top left hand corner of state-space only looks attractive if we use an optimistic heuristic. The areas near the frontiers of little experience will have high $\hat{J}^{\mathrm{opt}}$, and in turn the areas near those have nearly as high $\hat{J}^{\mathrm{opt}}$. Therefore, if prioritized sweeping (or any other asynchronous dynamic programming method) does its job, from START we will be encouraged to go north towards the unknown instead of east to the best reward discovered to date.

The system parameter r^{opt} does not require fine tuning. It can be set to a gross overestimate of the largest possible reward, and the system will simply continue exploration until it has sampled all state-action combinations T_{bored} times. However, Section 6 discusses its use as a search-guiding heuristic similar to the heuristic at the heart of $A^\star$ search.

Figure 9: The state-space of a very simple path planning problem.

The T_{bored} parameter, which defines how often we must try a given state-action combination before we cease our optimism, certainly does require forethought by the human programmer. If too small, we might overlook some low probability but highly rewarding stochastic successor. If too high, the system will waste time needlessly resampling already reliable statistics. Thus, the exploration procedure does not have full autonomy. This is, arguably, a necessary weakness of any non-random exploration heuristic. Dyna's exploration bonus contains a similar parameter in the relative size of the exploration bonus to the expected reward, and Interval Estimation has the parameter implicit in the optimistic confidence level.

The selection of an appropriate T_{bored} would be hard to formalize. It should take into account: the expected lifetime of the system, a measure of the importance of not becoming stuck during learning, and perhaps any available prior knowledge of the stochasticity of the system, or known constraints on the reward function. An automatic procedure for computing T_{bored} would require a formal definition of the human programmer's requirements and a prior distribution of possible worlds.

5 Experimental Results

This section begins with some comparative results in the familiar domain of stochastic two dimensional maze worlds. It then examines the β parameter which specifies the amount of computation (number of Bellman equation backups) allowed per real-world observation and also the T_{bored} parameter which defines how much exploration is performed. A number of larger examples are then used to investigate performance for a range of different discrete stochastic reinforcement tasks.

Maze problems

Each state has four actions: one for each direction. Blocked actions do not move. One goal state (the star in subsequent figures) gives 100 units of reward, all others give no reward, and there is a discount factor of 0.99. Trials start in the bottom left corner. The system is reset to the start state whenever the goal state has been visited ten times since the last reset. The reset is outside the learning task: it is not observed as a state transition.

Dyna and prioritized sweeping were both allowed ten Bellman's equation backups per observation ($\beta = 10$). Two versions of Dyna were tested:

1. Dyna-`PI+` is the original Dyna-`PI` of Sutton (1990), supplemented with the exploration bonus ($\epsilon = 0.001$) from the same paper.

2. Dyna-opt is the original Dyna-`PI` supplemented with the same T_{bored} optimistic heuristic that is used by prioritized sweeping.

Table 2 shows the number of observations before convergence. A trial was defined to have converged by a given time if no subsequent sequence of 1000 decisions contained more than 2% suboptimal decisions. The test for optimality was performed by comparison with the control law obtained from full dynamic programming using the true simulation.

We begin with some results for deterministic problems, in the first three rows of Table 2. The first row shows that Dyna-`PI+` converged for all problems except the 4,528 state problem. A smaller exploration bonus than $\epsilon = 0.001$ might have helped the latter problem converge, albeit slowly.

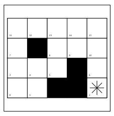
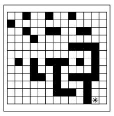
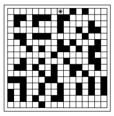
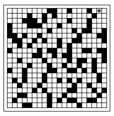
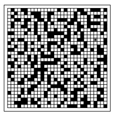
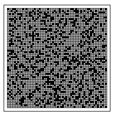
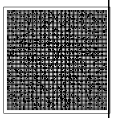
Maze							
	15 state	117 state	178 state	284 state	605 state	2627	4528
Det Dyna-PI+	400	500	10,000	18,000	36,000	195,000	$> 10^6$
Det Dyna-opt	300	900	4,250	12,000	21,000	105,000	245,000
Det PriSweep	150	1,200	3,250	2,800	6,000	29,000	59,000
Stc Q	600	31,000	62,000	310,000	untested	untested	untested
Stc Q-opt	500	$> 10^6$	$> 10^6$	untested	untested	untested	untested
Stc Dyna-PI+	400	4750	12,000	25,000	58,000	240,000	525,000
Stc Dyna-opt	700	5250	7500	14,000	35,000	155,000	310,000
Stc PriSweep	600	3500	5500	11,000	22,000	94,000	200,000

Table 2. Number of observations before 98% of decisions were subsequently optimal. These values have been rounded. For prioritized sweeping (and Dyna, where applicable) $\beta = 10$, $\epsilon = 10^{-3}$ and $r^{\text{opt}} = 200$. The tabulated experiments were all only run once; however, further multiple runs of the optimistic Dyna and prioritized sweeping have revealed little variance in convergence rate. See also Figures 11 and 14.

The other two rows used the optimistic heuristic with $r^{\text{opt}} = 200$ and $T_{\text{bored}} = 1$. The r^{opt} value thus overestimated the best possible reward by a factor of two—this was to see if we would converge without an accurate estimation of the true best possible reward. $T_{\text{bored}} = 1$ meant that as soon as something was tried all optimism was lost. This is a safe strategy in a deterministic environment.

The learning controller was given no clues beyond those implicit in the two parameters r^{opt} and T_{bored}. Thus, to ensure convergence to the optimum, it *had* to sample each state-action pair at least once.

Prioritized sweeping required fewer steps than optimistic Dyna in all mazes but one small one. All learners and runs took between 10—30 seconds per thousand observations running on a Sun-4 workstation. Interestingly, prioritized sweeping usually took about half the real time of Dyna. This is because during much of the exploration there were so few surprises that it did not need to use its full allocation of Bellman's equation processing steps. This effect is even more pronounced if 300 processing steps per observation are allowed instead of ten. For example, in the 4,528 state problem, optimistic Dyna then required 143,000 observations and took three hours. Prioritized sweeping required 21,000 observations and took fifteen minutes.

The lower part of Table 2 shows the results for stochastic problems using the same mazes. Each action had a 50% chance of being corrupted to a random value before it was applied. Thus if "North" was applied the outcome was movement North $\frac{1}{2} + \frac{1}{8} = \frac{5}{8}$ of the time, and each other direction $\frac{1}{8}$ of the time. Prioritized sweeping and optimistic Dyna each used a T_{bored} value of 5. Thus, they sampled every state-action combination five times before losing their optimism. This value was chosen as a reasonable balance between exploration and exploitation, given the authors' knowledge of the stochasticity of the system, and happily it proved to be satisfactory. As we discussed in Section 4.1, the choice of T_{bored} is not automated for any of these experiments.

These stochastic results also include a recent interesting incremental technique called Q-learning (Watkins 1989), which manages to learn without constructing a state transition model. Additionally, we tried Q-learning using the same T_{bored} optimistic heuristic as prioritized sweeping. The initial Q values were set high to encourage better initial exploration than a random walk. Much effort was put into tuning Q for this application. Its performance was, however, worse. In particular, the optimistic heuristic is a disaster for Q-learning which easily gets trapped—this is because Q-learning only pays attention to the current state of the system while the "planning to explore" behavior requires that attention is paid to areas of the state-space which the system is not currently in.

For the stochastic maze results the difference between optimistic Dyna and prioritized sweeping is less pronounced. This is because the large number of predecessors quickly dilute the wave of interesting changes which are propagated back on the priority queue, leading to a queue of many, very similar, priorities. However, prioritized sweeping still required less than half the total real time of either version of Dyna before convergence.

A small, fully connected, example

We also have results for a five state bench-mark problem described by Sato *et al.* (1988) and also used in Barto and Singh (1990). The transition matrix is in Figure 10 and the results are shown in Table 3. A T_{bored} parameter of 20 was used. In fact, $T_{\mathrm{bored}} = 5$ also converged 20 times out of 20, taking on average 120 steps and therefore $T_{\mathrm{bored}} = 20$ was considered a safe safety margin. The two Q-learners were heavily tweaked to find their best performance. The EQ-algorithm (Barto and Singh 1990) is designed to guarantee convergence at all costs—and so its poor comparative performance here is to be expected. Dyna-`PI+` was given what was probably too small an exploration bonus for the problem. The reduced exploration meant faster convergence, but on one occasion some misleading early transitions caused it to get stuck with a suboptimal policy.

The system parameters for prioritized sweeping

We now look at two results to give insight into two important parameters of prioritized sweeping. Firstly we consider its performance relative to the number of backups per observation. This experiment used the stochastic, 605 state example from Table 2 and the results are graphed in Figure 11. Using one operation is almost equivalent to optimistic Q-learning which does not converge. Even using only two backups gives reasonable performance, and performance improves as the number of

Q	Q-opt	Dyna-`PI+`	Dyna-opt	PriSweep	EQ
$2,105 \pm 520$	$2,078 \pm 430$	252 ± 35 (one failure)	470 ± 21	472 ± 22	$7,105 \pm 662$

Table 3. The mean number of observations before $> 98\%$ of subsequent decisions were optimal. Each learner was run twenty times and in all cases, bar one, there was eventual convergence to optimal performance. Also shown is the standard deviation of the twenty trials. The discount factor was $\gamma = 0.8$. For the optimistic methods $r^{\text{opt}} = 10$ and $T_{\text{bored}} = 20$. For prioritized sweeping and Dyna $\beta = 10$, and for prioritized sweeping $\epsilon = 10^{-3}$.

	R	n=0	n=1	n=2	n=3	n=4
sa=0,0	-0.4	10	20	20	20	30
sa=0,1	0	20	20	20	20	20
sa=0,2	1.2	10	10	10	30	40
sa=1,0	-2.5	10	10	20	30	30
sa=1,1	0.2	10	10	60	10	10
sa=1,2	-1.3	10	50	20	10	10
sa=2,0	1.5	10	40	20	20	10
sa=2,1	0.8	30	20	20	20	10
sa=2,2	-1.4	30	20	10	10	30
sa=3,0	-1.2	20	50	10	10	10
sa=3,1	-0.7	30	10	10	40	10
sa=3,2	0.1	20	30	30	10	10
sa=4,0	2.4	20	20	20	20	20
sa=4,1	-1.7	20	30	20	10	20
sa=4,2	1	10	40	20	10	20

Figure 10: Transition probabilities ($\times 100$) and expected rewards of a five state, three action, Markov control problem.

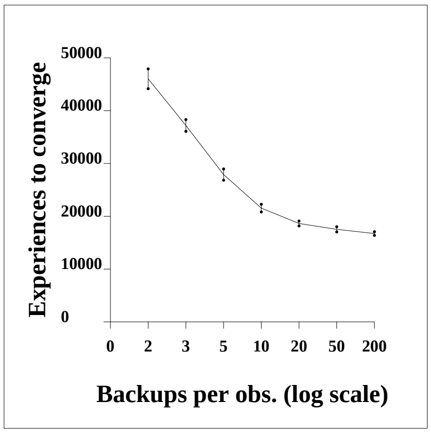


Figure 11: Number of experiences needed for prioritized sweeping to converge, plotted against number of backups per observation (β). This used the 605 state stochastic maze from Table 2 ($\gamma = 0.99$, $r^{\text{opt}} = 200$, $T_{\text{bored}} = 5$, $\epsilon = 10^{-3}$). The error bars show the standard deviations from ten runs with different random seeds.

backups increases. Beyond fifty backups, the priority queue usually gets exhausted on each time step, and there is little further improvement.

The other parameter is T_{bored}. We use a test case in which inadequate exploration is particularly dangerous. The maze in Figure 12 has two reward states. The lesser reward of 50 comes from the state in the bottom right. The greater reward of 100 is from the more inaccessible state near the top right. Trials always begin from the bottom left and the world is stochastic in the same manner as the earlier examples. Trials are reset when either goal state is encountered ten times. If T_{bored} is set too low and if there is bad luck while attempting to explore near the large reward state then the controller will lose interest, never return, and very likely spend the rest of its days traveling to the inferior reward. Each value of T_{bored} was run ten times and we recorded the percentage of runs which had converged correctly by 50,000 observations. Figure 13 graphs the results. For this problem $T_{\mathrm{bored}} = 5$ (which was checked a further 30 times) appears sufficient to ensure that we do not become stuck.

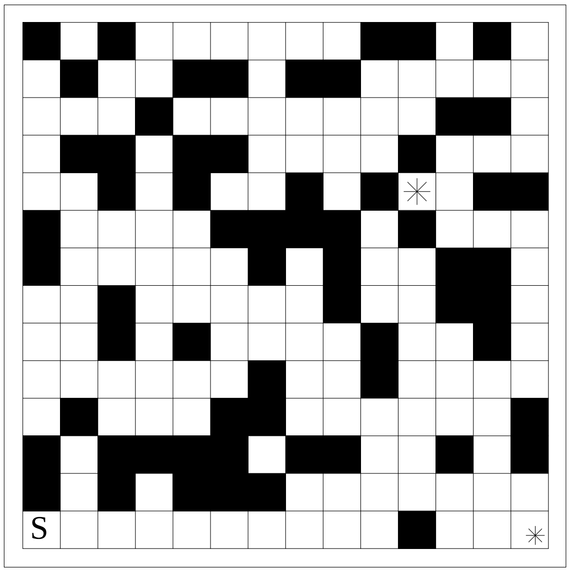


Figure 12: A misleading maze. A small reward in the bottom right tempts us away from a larger reward.

Figure 14 shows the number of experiences needed for convergence as a function of T_{bored} for the same set of experiments.

Other tasks

We begin with a task with a 3-d state-space quantized into 14,400 potential discrete states: guiding a rod through a planar maze by translation and rotation. There are four actions: move forwards one unit along the rod's length, move backwards one unit, rotate left one unit and rotate right one

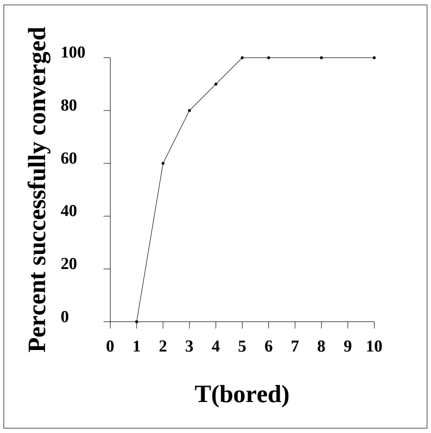


Figure 13: The frequency of correct convergence versus T_{bored} for the misleading maze ($\gamma = 0.99$, $r^{\text{opt}} = 200$, $\beta = 10$, $\epsilon = 10^{-3}$).

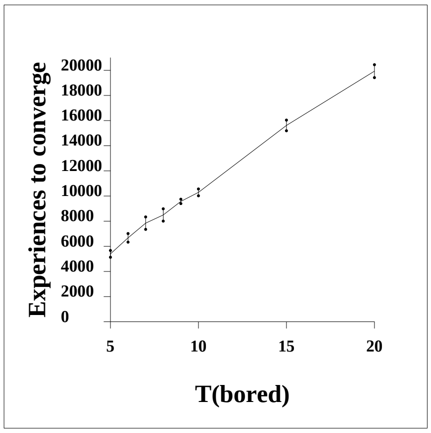


Figure 14: The mean and standard deviation number of experiences before convergence for ten independent experiments, as a function of T_{bored} for the misleading maze. Parameter values are as in Figure 13.

unit. In fact, the action takes us to the nearest quantized state after having applied the action. There are 20×20 position quantizations and 36 angle quantizations producing 14,400 states, though many are unreachable from the start. The distance unit is 1/20th the width of the workspace and the angular unit is 10 degrees. The problem is deterministic but requires a long, very specific, sequence of moves to get to the goal. Figure 15 shows the problem, obstacles and shortest solution for our experiments.

Q, Dyna-`PI+`, Optimistic Dyna and prioritized sweeping were all tested. The results are in Table 4.

Q and Dyna-`PI+` did not even travel a quarter of the way to the goal, let alone discover an optimal path, within 200,000 experiences. It is possible that a very well-chosen exploration bonus would have helped Dyna-`PI+` but in the four different experiments we tried, no value produced stable exploration.

Optimistic Dyna and prioritized sweeping both eventually converged, with the latter requiring a third the experiences and a fifth the real time.

When 2000 backups per experience were permitted, instead of 100, then both optimistic Dyna

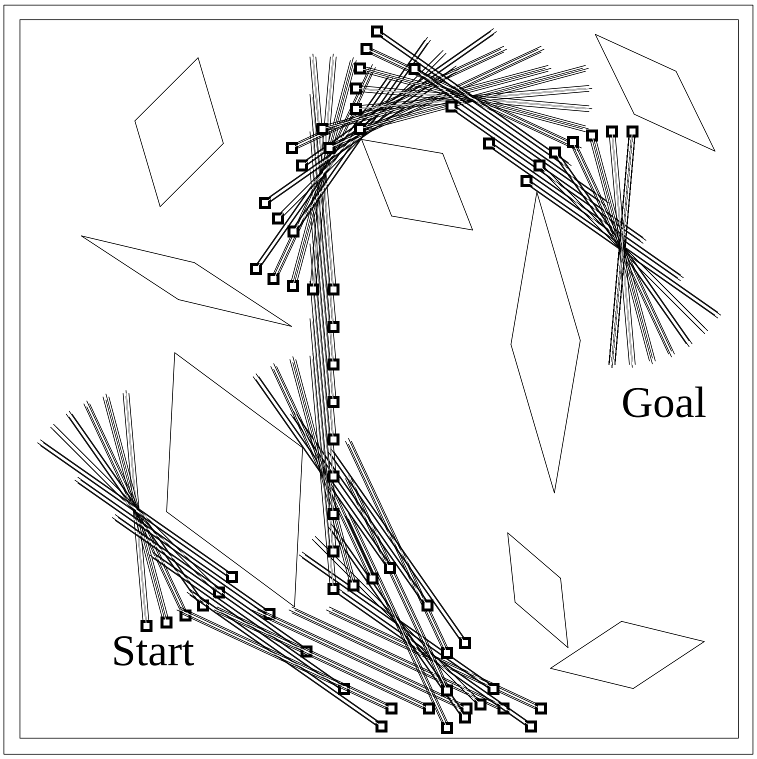


Figure 15: A three-DOF problem and the shortest solution path.

	Experiences to converge	Real time to converge
Q	never	
Dyna-PI+	never	
Optimistic Dyna	55,000	1500 secs
Prioritized Sweeping	14,000	330 secs

Table 4. Performance on the deterministic rod-in-maze task. Both Dynas and prioritized sweeping were allowed 100 backups per experience ($\gamma = 0.99, r^{\mathrm{opt}} = 200, \beta = 100, T_{\mathrm{bored}} = 1, \epsilon = 10^{-3}$).

and prioritized sweeping required fewer experiences to converge. Optimistic Dyna took 21,000 experiences instead of 55,000 but took 2,900 seconds—almost twice the real time. Prioritized sweeping took 13,500 instead of 14,000 experiences—very little improvement, but it used no extra time. This indicates that for prioritized sweeping, 100 backups per observation is sufficient to make almost complete use of its observations, so that all the long term reward ($\hat{J}_i$) estimates are very close to the estimates which would be globally consistent with the transition probability estimates ($\hat{q}_{ij}^a$). Thus, we conjecture that even full dynamic programming after each experience (which would take days of real time) would do little better.

We also consider a more complex extension of the maze world, invented by Singh (1991), which consists of a maze and extra state information dependent on where you have visited so far in the maze. We use the example in Figure 16. There are 263 cells, but there are also four binary flags appended to the state, producing a total of $263 \times 16 = 4208$ states. The flags, named A, B, C and X, are set whenever the cell containing the corresponding letter is passed through. All flags are cleared when the start state (in the bottom left hand corner) is entered. A reward is given when the goal state (top right) is entered, only if flags A, B and C are set. Flag X provides further interest. If X is clear, the reward is 100 units. If X is set, the reward is only 50 units. This task does not specify which order A, B and C are to be visited. The controller must find the optimal path.

Prioritized sweeping was tried with both the deterministic and stochastic maze dynamics ($\gamma = 0.99, r^{\rm opt} = 200, \beta = 10, \epsilon = 10^{-3}$). In the deterministic case $T_{\rm bored} = 1$. In the stochastic case $T_{\rm bored} = 5$. In both cases it found the globally optimal path through the three good flags to the goal, avoiding flag X. The deterministic case took 19,000 observations and twenty minutes of real time. The stochastic case required 120,000 observations and two hours of real time.

In these experiments, no information regarding the special structure of the problem was available to the learner. For example, knowledge of the cell at coordinates $(7, 1)$ with flag A set had no bearing on knowledge of the cell at coordinates $(7, 1)$ with A clear. If we told the learner that cell transitions are independent of flag settings then the convergence rate would be increased considerably. A far more interesting possibility is the automatic discovery of such structure by inductive inference on

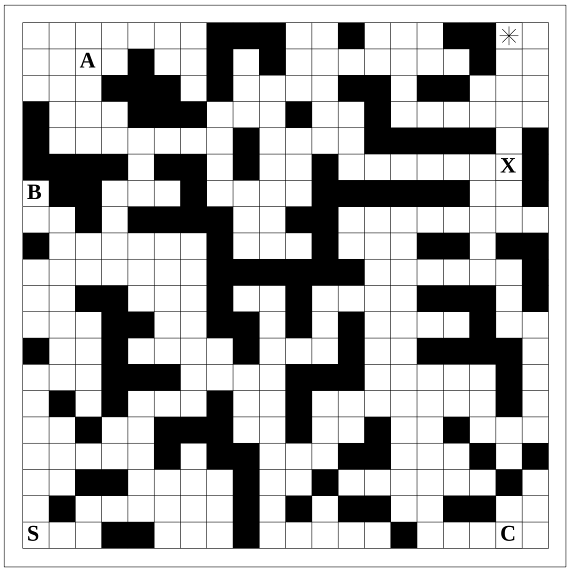


Figure 16: A maze with extra state in the form of four binary flags.

the structure of the learned state transition matrix. See Singh (1991) for current interesting work in that direction.

The third experiment is the familiar pole-balancing problem of Michie and Chambers (1968). There is no place here to discuss the enormous number of techniques which have been applied to this problem along with an equally enormous variation in details of the task formulation. The state-space of the cart is quantized at three equal levels for cart position, cart velocity, and pole angular speed. It is quantized at six equal levels for pole angle. The simulation used four real-valued state variables, yet the learner was only allowed to base its control decisions on the current quantized state. There are two actions: thrust left $10N$ and thrust right $10N$. The problem is interesting because it involves *hidden state*—the controller believes the system is Markov when in fact it is not. This is because there are many possible values for the real-valued state variables in each discretized box, and successor boxes are partially determined by these real values, which are not given to the controller. The task is defined by a reward of 100 units for every state except one absorbing state corresponding to a crash, which receives zero reward. Crashes occur if the pole angle or cart position exceed their limits. A discount factor of $\gamma = 0.999$ is used and trials start in random survivable configurations. Other parameters are ($r^{\rm opt} = 200, \beta = 100, T_{\rm bored} = 1, \epsilon = 10^{-3}$).

If the simulation contains no noise, or a very small amount (0.1% added to the simulated thrust), prioritized sweeping very quickly (usually in under 1000 observations and 15 crashes) develops a policy which provides stability for approximately 100,000 cycles. With a small amount of noise (1%), stable runs of approximately 20,000 time steps are discovered after, on average, 30 crashes.

6 Heuristics to Guide Search

In all experiments to date, the optimistic estimate of the best available one-step reward, r^{opt}, has been set to an overestimate of the best reward which is actually available. However, if the human programmer knows in advance what is the best possible reward-to-go from any given state, then the resultant, more realistic, optimism does not need to experience all state-action pairs.

For example, consider the maze world. If the robot is told the location of the goal state (in all previous experiments it was not given this information), but is not told which states are blocked, then it can nevertheless compute what would be the best possible reward-to-go from a state. It could not be greater than the reward obtained from the shortest possible path to the goal. The length of the path, l, can be computed easily with the Manhattan distance metric and then the best possible reward-to-go is

$$0\gamma^1 + 0\gamma^2 + \ldots + 0\gamma^{l-1} + r^{\mathrm{opt}}\gamma^l + r^{\mathrm{opt}}\gamma^{l+1} + \ldots = \frac{r^{\mathrm{opt}}\gamma^l}{1-\gamma} \tag{18}$$

When this optimistic heuristic is used, initial exploration is biased towards the goal, and once a path is discovered then many of the unexplored areas may be ignored. Ignoring occurs when even the most optimistic reward-to-go of a state is no greater than that of the already obtained path.

For example, Figure 17 shows the areas explored using a Manhattan heuristic when finding the optimal path from the start state at the bottom leftmost cell to the goal state at the center of the maze. The maze has 8525 states of which only 1722 needed to be explored.

For some tasks we may be satisfied to cease exploration when we have obtained a solution known to be, say, within 50% of the optimal solution. This can be achieved by using a heuristic which lies: it tells us that the best possible reward-to-go is that of a path which is twice the length of the true shortest possible path.

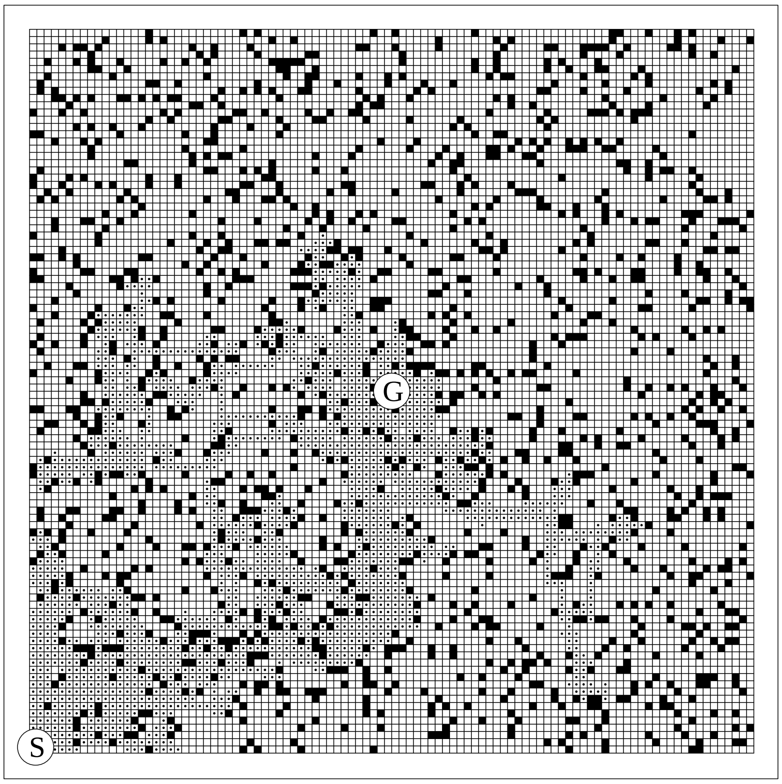


Figure 17: Dotted states are all those visited when the Manhattan heuristic was used to derive r^{opt} ($\gamma = 0.99, \beta = 10, T_{\mathrm{bored}} = 1, \epsilon = 10^{-3}$).

7 Discussion

Generalization of the state transition model

This paper has been concerned with discrete state systems in which no prior assumptions are made about the structure of the state-space. Despite the weakness of the assumptions, we can successfully learn large stochastic tasks. However, very many problems do have extra known structure in the state-space, and it is important to consider how this knowledge can be used. By far the most common knowledge is smoothness—given two states which are in some way similar, in general their transition probabilities will be similar.

TD can also be applied to highly smooth problems using a parametric function approximator such as a neural network. This technique has recently been used successfully on a large complex problem, Backgammon, by Tesauro (1991). The discrete version of prioritized sweeping given in this paper could not be applied directly to Backgammon because the game has 10^{23} states, which is unmanageably large by a factor of at least 10^{10}. However, a method which quantized the space of board positions, or used a more sophisticated smoothing mechanism, might conceivably be able to compute a near-optimal strategy.

We are currently developing memory-based algorithms which take advantage of local smoothness assumptions. In these investigations, state transition models are learned by memory-based function approximators (Moore and Atkeson 1992). Prioritized sweeping takes place over non-uniform tessellations of state-space, partitioned by variable resolution *k*d-trees (Moore 1991). We are also investigating the role of locally linear control rules and reward functions in such partitionings, in which instead of using Bellman's Equation (16) directly, we use local linear quadratic regulators (LQR) (see, for example, Sage and White (1977)). It is worth remembering that, if the system is sufficiently linear, LQR is an extremely powerful technique. In a pole balancer experiment in which we used local weighted regression to identify a local linear model, LQR was able to create a stable controller based on only 31 state transitions!

Other current investigations which attempt to perform generalization in conjunction with reinforcement learning are Mahadevan and Connell (1990) which investigates clustering parts of the policy, Chapman and Kaelbling (1990) which investigates automatic detection of locally relevant state variables, and Singh (1991) which considers how to automatically discover the structure in tasks such as the multiple-flags example of Figure 16.

7.1 Related work

The Dyna-Q-queue algorithm of Peng and Williams

Peng and Williams (1992) have concurrently been developing a closely related algorithm which they call Dyna-Q-queue. This conceptually similar idea was discovered independently. Where prioritized sweeping provides efficient data processing for methods which learn the state transition model, Dyna-Q-queue performs the same role for Q-learning (Watkins 1989), an algorithm which avoids building an explicit state-transition model. Dyna-Q-queue is also more careful about what it allows onto the priority queue: it only allows predecessors which have a predicted change ("interestingness" value) greater than a significant threshold δ, whereas prioritized sweeping allows everything above a minuscule change ($\epsilon = 10^{-5}$ times the maximum reward) onto the queue. The initial experiments in Peng and Williams (1992) consist of sparse, deterministic maze worlds of several hundred cells. Performance, measured by total number of Bellman's equation processing steps before convergence, is greatly improved over conventional Dyna-Q (Sutton 1990).

Other related work

Sutton (1990) identifies reinforcement learning with asynchronous dynamic programming and introduces the same computational regime as that used for prioritized sweeping. The notion of using an optimistic heuristic to guide search goes back to the $A^\star$ tree search algorithm Nilsson (1971), which also motivated another aspect of prioritized sweeping: it too schedules nodes to be expanded according to an (albeit different) priority measure. More recently Korf (1990) gives a combination of $A^\star$ and Dynamic Programming in the LRTA* algorithm. LRTA* is, however, very different from prioritized sweeping: it concentrates all search effort in a finite-horizon set of states beyond the current actual system state. Finally, Lin (1991) has investigated a simple technique which replays, backwards, the memorized sequence of experiences which the controller has recently had. Under some circumstances this may produce some of the beneficial effects of prioritized sweeping.

8 Conclusion

Our investigation shows that prioritized sweeping can solve large state-space real time problems with which other methods have difficulty. Other benefits of the memory-based approach, described in Moore and Atkeson (1992), allow us to control forgetting in potentially changeable environments and to automatically scale state variables. Prioritized sweeping is heavily based on learning a world model and we conclude with a few words on this topic.

If a model of the world is not known to the human programmer in advance then an adaptive system is required, and there are two alternatives:

Learn a model and from this develop a control rule.	Learn a control rule without building a model.

Dyna and prioritized sweeping fall into the first category. Temporal differences and Q-learning fall into the second. Two motivations for *not* learning a model are (i) the interesting fact that the methods do, nevertheless, learn, and (ii) the possibility that this more accurately simulates some kinds of biological learning (Sutton and Barto 1990). However, a third advantage which is sometimes touted—that there are computational benefits in not learning a model—is, in our view, dubious. A common argument is that with the real world available to be sensed directly, why should we bother with less reliable, learned internal representations? The counterargument is that even systems acting in real time can, for every one real experience, sample millions of mental experiences from which to make decisions and improve control rules.

Consider a more colorful example. Suppose the anti-model argument was applied by a new arrival at a university campus: "I don't need a map of the university—the university is its own map." If the new arrival truly mistrusts the university cartographers then there might be an argument for one full exploration of the campus in order to create their own map. However, once this map has been produced, the amount of time saved overall by pausing to consult the map before traveling to each new location—rather than exhaustive or random search in the real world—is undeniably enormous.

It is justified to complain about the indiscriminate use of combinatorial search or matrix inversion prior to each supposedly real time decision. However, models need not be used in such an extravagant fashion. The prioritized sweeping algorithm is just one example of a class of algorithms which can easily operate in real time and also derive great power from a model.

Acknowledgements

Many thanks to Mary Lee, Rich Sutton and the reviewers of the paper for some very valuable comments and suggestions. Thanks also to Stefan Schaal and Satinder Singh for useful comments on an early draft. Andrew W. Moore is supported by a Postdoctoral Fellowship from SERC/NATO.

Support was also provided under Air Force Office of Scientific Research grant AFOSR-89-0500, an Alfred P. Sloan Fellowship, the W. M. Keck Foundation Associate Professorship in Biomedical Engineering, Siemens Corporation, and a National Science Foundation Presidential Young Investigator Award to Christopher G. Atkeson.

Appendix A. The random generation of a stochastic problem

Here is an algorithm to generate stochastic systems such as Figure 4 in Section 3. The parameters are: S_{nt}, the number of non-terminal states; S_t , the number of terminal states and $\mu_{\rm succs}$, the mean number of successors.

All states have a position within the unit square. The terminal states are generated on an equispaced circle, diameter 0.9, alternating between black and white. Non-terminal states are each positioned in a uniformly random location within the square. Then the successors of each non-terminal are selected. The number of successors is chosen randomly as $1 + X$ where X is a random variable drawn from the exponential distribution with mean $\mu_{\rm succs} - 1$.

The choice of successors is affected by locality within the unit square. This provides a more interesting system than allowing successors to be entirely random. It was empirically noted that entirely random successors cause the long-term absorption probabilities to be very similar across most of the set of states. Locality leads to a more varied distribution.

The successors are chosen according to a simple algorithm in which they are drawn from within a slowly growing circle centered on the parent state.

If the parent state is i, and there are N_i successors, then the jth transition probability is computed by $X_j / \sum_{k=1}^{N_i} X_k$ where $\{X_1, \ldots, X_{N_i}\}$ are independent random variables, uniformly distributed in the unit interval.

Once the system has been generated, a check is performed that the system is absorbing—all non-terminals can eventually reach at least one terminal state. If not, entirely new systems are randomly generated until an absorbing Markov system is obtained.

References

Barto, A. G. and Singh, S. P. (1990). On the Computational Economics of Reinforcement Learning. In Touretzky, D. S., editor, *Connectionist Models: Proceedings of the 1990 Summer School*. Morgan Kaufmann.

Barto, A. G., Sutton, R. S., and Watkins, C. J. C. H. (1989). Learning and Sequential Decision Making. COINS Technical Report 89-95, University of Massachusetts at Amherst.

Barto, A. G., Bradtke, S. J., and Singh, S. P. (1991). Real-time Learning and Control using Asynchronous Dynamic Programming. Technical Report 91-57, University of Massachusetts at Amherst.

Bellman, R. E. (1957). *Dynamic Programming*. Princeton University Press, Princeton, NJ.

Berry, D. A. and Fristedt, B. (1985). *Bandit Problems: Sequential Allocation of Experiments*. Chapman and Hall.

Bertsekas, D. P. and Tsitsiklis, J. N. (1989). *Parallel and Distributed Computation*. Prentice Hall.

Chapman, D. and Kaelbling, L. P. (1990). Learning from Delayed Reinforcement In a Complex Domain. Technical Report No. TR-90-11, Teleos Research.

Christiansen, A. D., Mason, M. T., and Mitchell, T. M. (1990). Learning Reliable Manipulation Strategies without Initial Physical Models. In *IEEE Conference on Robotics and Automation*, pages 1224–1230.

Dayan, P. (1992). The Convergence of TD(λ) for General λ. *Machine Learning*, 8(3).

Kaelbling, L. P. (1990). Learning in Embedded Systems. PhD. Thesis; Technical Report No. TR-90-04, Stanford University, Department of Computer Science.

Knuth, D. E. (1973). *Sorting and Searching.* Addison Wesley.

Korf, R. E. (1990). Real-Time Heuristic Search. *Artifical Intelligence*, 42.

Lin, L. J. (1991). Programming Robots using Reinforcement Learning and Teaching. In *Proceedings of the Ninth International conference on Artificial Intelligence (AAAI-91).* MIT Press.

Mahadevan, S. and Connell, J. (1990). Automatic Programming of Behavior-based Robots using Reinforcement Learning. Technical Report, IBM T. J. Watson Research Center, NY 10598.

Michie, D. and Chambers, R. A. (1968). BOXES: An Experiment in Adaptive Control. In Dale, E. and Michie, D., editors, *Machine Intelligence 2.* Oliver and Boyd.

Moore, A. W. and Atkeson, C. G. (1992). Memory-based Function Approximators for Learning Control. In preparation.

Moore, A. W. (1991). Variable Resolution Dynamic Programming: Efficiently Learning Action Maps in Multivariate Real-valued State-spaces. In Birnbaum, L. and Collins, G., editors, *Machine Learning: Proceedings of the Eighth International Workshop.* Morgan Kaufman.

Nilsson, N. J. (1971). *Problem-solving Methods in Artificial Intelligence.* McGraw Hill.

Peng, J. and Williams, R. J. (1992). Efficient Search Control in Dyna. College of Computer Science, Northeastern University.

Sage, A. P. and White, C. C. (1977). *Optimum Systems Control.* Prentice Hall.

Samuel, A. L. (1959). Some Studies in Machine Learning using the Game of Checkers. *IBM Journal on Research and Development*, 3. Reprinted in Feigenbaum, E. A. and Feldman, J., editors, *Computers and Thought*, McGraw-Hill, 1963.

Sato, M., Abe, K., and Takeda, H. (1988). Learning Control of Finite Markov Chains with an Explicit Trade-off Between Estimation and Control. *IEEE Trans. on Systems, Man, and Cybernetics*, 18(5):667–684.

Singh, S. P. (1991). Transfer of learning across compositions of sequential tasks. In Birnbaum, L. and Collins, G., editors, *Machine Learning: Proceedings of the Eighth International Workshop.* Morgan Kaufman.

Stanfill, C. and Waltz, D. (1986). Towards Memory-Based Reasoning. *Communications of the ACM*, 29(12):1213–1228, December.

Sutton, R. S. and Barto, A. G. (1990). Time-Derivative Models of Pavlovian Reinforcement. In Gabriel, M. and Moore, J., editors, *Learning and Computational Neuroscience: Foundations of Adaptive Networks*, pages 497–537. MIT Press.

Sutton, R. S. (1984). Temporal Credit Assignment in Reinforcement Learning. Phd. thesis, University of Massachusetts, Amherst.

Sutton, R. S. (1988). Learning to Predict by the Methods of Temporal Differences. *Machine Learning*, 3:9–44.

Sutton, R. S. (1990). Integrated Architecture for Learning, Planning, and Reacting Based on

Approximating Dynamic Programming. In *Proceedings of the 7th International Conference on Machine Learning*. Morgan Kaufman.

Tesauro, G. J. (1991). Practical Issues in Temporal Difference Learning. Report RC 17223 (76307), IBM T. J. Watson Research Center, NY.

Thrun, S. B. and Möller, K. (1992). Active Exploration in Dynamic Environments. In Moody, J. E., Hanson, S. J., and Lippman, R. P., editors, *Advances in Neural Information Processing Systems 4*. Morgan Kaufmann.

Watkins, C. J. C. H. (1989). Learning from Delayed Rewards. PhD. Thesis, King's College, University of Cambridge.