Planning with Predictive State Representations

Michael R. James
University of Michigan
mrjames@umich.edu

Satinder Singh
University of Michigan
baveja@umich.edu

Michael L. Littman
Rutgers University
mlittman@cs.rutgers.edu

Abstract

Predictive state representation (PSR) models for controlled dynamical systems have recently been proposed as an alternative to traditional models such as partially observable Markov decision processes (POMDPs). In this paper we develop and evaluate two general planning algorithms for PSR models. First, we show how planning algorithms for POMDPs that exploit the piecewise linear property of value functions for finite-horizon problems can be extended to PSRs. This requires an interesting replacement of the role of hidden nominal-states in POMDPs with linearly independent predictions in PSRs. Second, we show how traditional reinforcement learning algorithms such as Q-learning can be extended to PSR models. We empirically evaluate both our algorithms on a standard set of test POMDP problems.

1 Introduction

Planning in stochastic dynamical systems involves using a model of the system to compute near-optimal policies (mappings from system state to actions). Predictive state representations (PSRs) are a recently developed [7] model for controlled dynamical systems. A wide range of applications can be viewed as controlling a dynamical system. Thus far, research on PSRs has mostly concentrated on learning PSR models from data gathered through interaction with a real system (though see an extended abstract by [6] on policy iteration for PSRs). It has been speculated [9] that PSR models will be easier to learn than other stochastic models such as POMDPs, due to PSRs being based only on observable quantities, while other models are based on hidden or unobservable nominal-states. Given this advantage, it is important to develop algorithms for planning in PSR models. In this paper we present two such algorithms.

Our first PSR planning algorithm is an extension of the POMDP value iteration algorithm called incremental pruning [13]. The incremental pruning (IP) algorithm calculates a series of value functions that, in the limit, approach the optimal value function. The optimal policy is easily calculated from the optimal value function. When applied to POMDPs, POMDP-IP depends on the fact that these value functions are piecewise linear functions of probability distributions over underlying POMDP nominal-states. However, in PSRs there is no notion of nominal-states, and so a replacement must be found that both allows the value function to be calculated correctly, and ensures that the value function remains piecewise linear over this replacement. In addition, the chosen replacement creates the potential for inefficiency when calculating the series of value functions. We develop an approach to compensate for this inefficiency in the PSR-IP algorithm.

Our second algorithm is an application of reinforcement learning to PSRs. We use Q-learning on a known PSR by making use of the PSR's current representation of state. The current representation of state is continuous, so some form of function approximation must be used. This algorithm learns an (approximately) optimal value function through interaction with the dynamical system.

2 PSRs

We consider finite, discrete-time controlled dynamical systems, henceforth dynamical systems, that accept actions from a discrete set $\mathcal{A}$, produce observations from a discrete set $\mathcal{O}$, and produce rewards from a discrete set $\mathcal{R}$. In this section we present the formalism of PSR models from [7] and then extend it to deal explicitly with rewards (which were ignored in the original PSR definition or equivalently treated implicitly as part of the observation).

PSRs are based on the notion of tests. A test t is a finite sequence of alternating actions and observation, i.e., $t \in \{\mathcal{A} \times \mathcal{O}\}^*$. For a test $t = a^1 o^1 \cdots a^k o^k$, its prediction given some history $h = a_1 o_1 \ldots a_n o_n$, denoted $p(t|h)$, is the conditional probability of seeing test t's observation sequence if test t's action sequence is executed from history h: $p(t|h) = prob(o_{n+1} =$

$o^1 \dots o_{n+k} = o^k | h, a_{n+1} = a^1 \dots a_{n+k} = a^k)$.

Littman et. al. [7] show that for any dynamical system, there exists a set of tests $Q = \{q_1 \dots q_n\}$, called the core tests, whose predictions are a sufficient statistic of history. In particular, they showed that for any test t,

$$p(t|h) = p(Q|h)^T m_t$$

for some weight vector m_t and where $p(Q|h) = [p(q_1|h) \dots p(q_n|h)]$. The vector of predictions for the core tests, $p(Q|h)$, is the state representation of PSRs. The state vector in a PSR is called a *prediction vector* to emphasize its defining characteristic. Thus, state in a PSR model of a system is expressed entirely in terms of observable quantities; this sets it apart from hidden-state based models such as POMDPs and is the root cause of the excitement about PSRs.

The prediction vector must be updated as actions are taken and observations received. For core test $q^i \in Q$:

$$p(q^i|hao) = \frac{p(aoq^i|h)}{p(ao|h)} = \frac{p^T(Q|h)m_{aoq^i}}{p^T(Q|h)m_{ao}} \qquad (1)$$

We can more easily write the prediction vector update by defining $(|Q| \times |Q|)$ matrices M_{ao} for every $a \in \mathcal{A}, o \in \mathcal{O}$, where column i of M_{ao} is the vector m_{aoq^i}. Using this notation, the prediction vector update can be written

$$p(Q|hao) = \left(\frac{p^T(Q|h)M_{ao}}{p^T(Q|h)m_{ao}}\right)^T \qquad (2)$$

Thus, a linear-PSR (henceforth just PSR) is specified by the set of core tests Q; the **model parameters**: m_{ao} and M_{ao} for all $a \in \mathcal{A}$ and $o \in \mathcal{O}$; and an initial prediction vector $p(Q|\phi)$, where ϕ is the null history. The next step is to incorporate reward.

2.1 Modeling reward in PSRs

Our approach to adding reward to PSRs is to treat reward as an additional observation. The term **result** is used to denote a (reward, observation) pair (r, o). Thus, tests take the form of $t = a^1(r^1o^1) \dots a^k(r^ko^k)$ and histories take the form $h = a_1(r_1o_1) \dots a_n(r_no_n)$. For the remainder of this paper, we assume that all PSRs include both observations and rewards. Given an n-dimensional PSR for actions $a \in \mathcal{A}$, observations $o \in \mathcal{O}$, rewards $r \in \mathcal{R}$, let the prediction vector at time t be $p(Q|h_t)$. For action a, result (r, o) has probability $p^T(Q|h_t)m_{a,(r,o)}$, for the parameter vector $m_{a,(r,o)}$ (this parameter vector is part of the model and is thus known). Therefore, we can calculate the probability of reward r and observation o separately by:

$$\begin{aligned} prob(r|h_t, a) &= \sum_{o \in \mathcal{O}} p^T(Q|h_t)m_{a,(r,o)} \\ &= p^T(Q|h_t) \sum_{o \in \mathcal{O}} m_{a,(r,o)}, \end{aligned}$$

and

$$\begin{aligned} prob(o|h_t, a) &= \sum_{r \in \mathcal{R}} p^T(Q|h_t)m_{a,(r,o)} \\ &= p^T(Q|h_t) \sum_{r \in \mathcal{R}} m_{a,(r,o)} \end{aligned}$$

Furthermore, we can calculate the expected immediate reward $R(h_t, a)$ for action a at history h_t[1]:

$$\begin{aligned} R(h_t, a) &= \sum_{r \in \mathcal{R}} r\, prob(r|h_t, a) \\ &= \sum_{r \in \mathcal{R}} \left(rp^T(Q|h_t) \sum_{o \in \mathcal{O}} m_{a,(r,o)} \right) \\ &= p^T(Q|h_t) \sum_{r \in \mathcal{R}} \left(r \sum_{o \in \mathcal{O}} m_{a,(r,o)} \right) \\ &= p^T(Q|h_t)n_a \qquad (3) \end{aligned}$$

where Equation 3 defines the $(n \times 1)$ reward-parameter vector n_a (for all $a \in \mathcal{A}$) as a linear combination of the parameter vectors. Thus, given an action, the expected immediate reward is a linear function of the prediction vector. This is somewhat surprising because it suggests that there is a scalar reward for every core test outcome and that the current expected reward is the expected reward over core test outcomes.

In [7], it was shown that PSRs can model any dynamical system that can be expressed as a POMDP, a common model for controlled dynamical systems with partial observability. That result extends straightforwardly to PSRs with reward. Next we define POMDP-based models of dynamical systems.

3 POMDPs

A POMDP [4] is defined by a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{R}, T, O, R, b_0 >$, which includes the sets $\mathcal{A}, \mathcal{O}$, and $\mathcal{R}$ defined above, along with the set $\mathcal{S}$ which contains n unobservable nominal-states. In this paper, we will express the transition (T), observation (O), and reward functions (R) in vector notation. The set T consists of $(n \times n)$ transition matrices T^a, for every

[1] here we assume that the rewards are labeled with their value, i.e., reward r has value r

$a \in \mathcal{A}$. The entry $T^a_{i,j}$ is the probability of a transition from nominal-state i to nominal-state j when action a is selected. The set O consists of $(n \times n)$ diagonal matrices $O^{a,o}$ for every $a \in \mathcal{A}$ and $o \in \mathcal{O}$. The entry $O^{a,o}_{i,i}$ is the probability of observation o occurring when in nominal-state i and action a is selected.

The state in a POMDP is a belief state b, i.e., a probability distribution over nominal states where b_i is the probability that the nominal state is i. The belief state is updated as follows: upon taking action a in current belief state b and observing o, the new belief state b' is

$$b' = \frac{b^T T^a O^{a,o}}{b^T T^a O^{a,o} 1_n}$$

where 1_n is the $(n \times 1)$ vector of all 1s. An initial belief state is specified in the definition of a POMDP.

For a POMDP with n nominal-states, rewards are defined by a set R of $(n \times 1)$ vectors r^a for every $a \in \mathcal{A}$. The i^{th} entry in r^a is the reward received in nominal-state i when taking action a. Thus, the expected reward when taking action a in belief state b is simply

$$R(b, a) = b^T r^a$$

With these definitions of PSRs and POMDPs in place, we now address the planning problem for PSRs.

4 Value Iteration

In this section, we define a planning algorithm for PSRs by extending a POMDP value iteration algorithm. Value iteration algorithms proceed in stages. The value function at stage i is denoted V_i. A dynamic programming update transforms V_i to V_{i+1}, taking into account the one-step system dynamics and one-step rewards. In the limit as $i \to \infty$, the value function V_i will approach the optimal value function. Typically, value iteration is concluded when the difference between two successive value function approximations is sufficiently small.

It is easily shown that for all i value function V_i is a piecewise linear function over belief states. Each linear facet of V_i corresponds to a policy tree (see Littman [8] for details). Each policy tree ρ has an associated vector w_ρ as well as a policy consisting of an initial action and, for each observation, another (one step shorter) policy tree. The vector w_ρ is a linear function over belief states such that $b^T w_\rho$ gives the expected discounted reward obtained by following ρ's policy from belief state b. Without loss of generality, assume a policy tree ρ specifies a single initial action a, plus a new policy tree ρ^o for every observation o. The value for policy tree ρ at belief state b is

$$\begin{aligned} V_\rho(b) &= R(b,a) + \gamma \sum_{o \in \mathcal{O}} prob(o|b,a) V_{\rho^o}(b') \\ &= b^T r^a + \gamma \sum_{o \in \mathcal{O}} prob(o|b,a) \frac{b^T T^a O^{a,o}}{prob(o|b,a)} w_{\rho^o} \\ &= b^T r^a + \gamma \sum_{o \in \mathcal{O}} b^T T^a O^{a,o} w^T_{\rho^o} \\ &= b^T w_\rho \end{aligned} \tag{4}$$

The matrices T^a and $O^{a,o}$ and the vectors r^a have entries corresponding to each nominal-state, thus implicit in these equations are summations over nominal-states. We point out here that nominal-states serve as the basis on which these equations are built.

The value function V_i is represented using a set S_i of vectors w_ρ, one corresponding to each policy tree ρ. The value function itself is the the upper surface over all vectors $w_\rho \in S_i$. A single stage of value iteration can be viewed as transforming the set S_i to S_{i+1}. Note that, for efficiency, all sets S should have minimal size.

There has been much work in the development of efficient value iteration algorithms. The incremental pruning algorithm [13] has emerged as one of the fastest. We present this algorithm next, with details relevant to extending this algorithm to work on PSRs.

4.1 Incremental Pruning on POMDPs

The IP algorithm is best described as a method for transforming the set S_i to S_{i+1} via a series of intermediate sets. We present the basic POMDP-IP algorithm as defined in [3]. For vector notation, vector sums are componentwise, and we define the cross sum of two sets of vectors: $A \oplus B = \{\alpha + \beta | \alpha \in A, \beta \in B\}$. Given a set S_i, there are two intermediate sets used to calculate S_{i+1}. They are

$$S^a_o = purge(\{\tau(\alpha, a, o) | \forall \alpha \in S_i\}) \tag{5}$$

$$S^a = purge\left(\bigoplus_o S^a_o\right) \tag{6}$$

$$S_{i+1} = purge\left(\bigcup_a S^a\right) \tag{7}$$

where $\tau(\alpha, a, o)$ is the $|S|$-vector given by

$$\begin{aligned} &\tau(\alpha, a, o)(s) = \\ &\quad (1/|\mathcal{O}|) r^a(s) + \gamma \sum_{s'} \alpha(s') prob(o|s', a) prob(s'|s, a) \end{aligned} \tag{8}$$

where the purge routine (also called filtering) takes a set of vectors and returns only those vectors necessary

to represent the upper surface of the set. Here, we use Lark's algorithm [8] to purge. This involves solving a linear program for each vector, giving a belief state for which that vector is optimal, or returning null if that vector is not optimal at any belief state.

Recall that a single stage of value iteration algorithms (including IP) transforms the set S_i to the set S_{i+1}. For IP, this transformation is accomplished in equations 5, 6, and 7. The sets in Equations 5 and 7 are constructed in a straightforward manner, while Equation 6 makes use of the fact that

$$purge(A \oplus B \oplus C) = purge(purge(A \oplus B) \oplus C) \quad (9)$$

This incremental construction of S^a is the key to the performance benefits of incremental pruning, as well as providing its name. We now present the details of extending the IP algorithm to work on PSRs.

4.2 Incremental Pruning on PSRs

The problem of extending IP to PSRs involved two major questions. In POMDP-IP, the value function was a function of belief state, which used the underlying nominal-states. However, there are no underlying nominal-states in PSRs, so the first question was whether the value function could be expressed as a function of some PSR-based replacement. The second question concerns the fact that IP algorithm depends on the value function being piecewise linear. Given a PSR-based replacement, it was unknown whether the value function would retain this essential property.

To answer these questions for PSR-IP, we make use of policy trees. In the context of PSRs, a policy tree will define a linear function over PSR prediction vectors, as well as the policy as defined for POMDP policy trees above. We now show that this linear value function represents the expected reward for that policy. We do this with an inductive proof.

Lemma 1 *For PSRs, every policy tree has an associated function that calculates the expected discounted reward at every PSR prediction vector. This function is linear in the PSR prediction vector.*

Proof **(inductive step)** Given a policy tree ρ with initial action a and policy trees ρ^o for every observation o. Assume that the expected discounted reward functions for all ρ^o are linear functions of the prediction vector so can be written as $(n \times 1)$ vectors w_{ρ^o}. The expected discounted reward for tree ρ at history h is

$$\begin{aligned}
&V_\rho(h) \\
&\quad = R(h,a) + \gamma \sum_{o \in \mathcal{O}} prob(o|a,h) V_{\rho^o}(hao) \\
&\quad = p(Q|h)^T n_a + \gamma \sum_{o \in \mathcal{O}} \left(p(Q|h)^T m_{a,o} \right) \left(p(Q|hao)^T w_{\rho^o} \right) \\
&= p(Q|h)^T n_a + \gamma \sum_{o \in \mathcal{O}} p(Q|h)^T m_{a,o} \left(\frac{p(Q|h)^T M_{a,o}}{p(Q|h)^T m_{a,o}} \right) w_{\rho^o} \\
&\quad = p(Q|h)^T n_a + \gamma \sum_{o \in \mathcal{O}} p(Q|h)^T M_{a,o} w_{\rho^o} \\
&\quad = p(Q|h)^T n_a + \gamma p(Q|h)^T \sum_{o \in \mathcal{O}} M_{a,o} w_{\rho^o} \\
&\quad = p(Q|h)^T \left(n_a + \gamma \sum_{o \in \mathcal{O}} M_{a,o} w_{\rho^o} \right)
\end{aligned} \quad (10)$$

note that

$$w_\rho = n_a + \gamma \sum_{o \in \mathcal{O}} M_{a,o} w_{\rho^o} \quad (11)$$

is a $(n \times 1)$ vector, and so $V_\rho(h)$ is a linear function of $p(Q|h)$.

(initial step) A one-step policy tree ρ^1 specifies a single action, so the expected reward for ρ^1 is defined by equation 3, which shows it to be a linear function of the prediction vector. □

Theorem 1 *For PSRs, the optimal value function V_i is a piecewise linear function over prediction vectors.*

Proof From Lemma 1, Equation 11 defines the policy tree value function for PSRs as the vector w_ρ. Moreover, given a set S_i of vectors for i step policy trees, at a given prediction vector, the policy tree with highest expected reward define the optimal value function at that prediction vector. Therefore, V_i is defined by the upper surface of the expected reward functions for all policy trees, and is a piecewise linear function over PSR prediction vectors. □

Given these facts, it is now possible to implement PSR-IP by using equation 11 to calculate the policy tree value function for PSRs.

The equations used in this proof are all based on the PSR core tests, rather than nominal-states as used by POMDPs. In effect, we have changed the basis for calculating the value of policy trees from POMDP nominal-states to PSR core tests. This change retains the important properties of the policy tree vector w_ρ, while moving away from the unobservable nominal-states to observable core tests. Additionally, we have shown that the value function V_i remains a piecewise linear function, although now it is a function of PSR prediction vector.

Although this version of PSR-IP is theoretically correct, the use of prediction vectors introduces a problem. Consider the purge subroutine of POMDP-IP. For a given policy tree, the routine finds a belief state for which that policy tree has a higher value than all others. The same routine is used with PSRs, except that a prediction vector is found. While there are simple constraints on when a belief state is valid[2] (all entries are between 0 and 1, and the sum of all entries is 1), the problem is that there is not a simple analogue to these constraints for prediction vectors. For instance, all entries of a prediction vector must be between 0 and 1, but this alone does not guarantee that the prediction vector will make legal predictions.

The result is that the purge routine for a given policy tree may return an invalid prediction vector. Thus, some trees may be added which are not optimal for any valid prediction vector. This does not invalidate the PSR-IP algorithm (the optimal policy trees for all valid prediction vectors will remain), but it does mean that extraneous trees may be added. This may have a highly detrimental effect on the efficiency of the algorithm. This effect can be mitigated by adding additional constraints to the linear program used in the purge routine. For instance, a possible constraint may be that all one-step predictions must be between 0 and 1. We present a list of potential constraints for prediction vector p; these are based on probabilistic equations which must hold for valid dynamical system behavior.

1. For every entry p_i of p: $0 \leq p_i \leq 1$.
2. For every action sequence $a_1 ... a_n$: $\sum_{o_1,...o_n,r_1...r_n} p^T m_{a_1,(o_1,r_1)...a_n(o_n,r_n)} = 1.0$.
3. For every action/result sequence $a_1, (o_1, r_1), ...a_n, (o_n, r_n)$: $0 \leq p^T m_{a_1,(o_1,r_1)...a_n(o_n,r_n)} \leq 1.0$.
4. Constrain all one-step extensions of the core tests q^i. This ensures that every next prediction vector will have entries within the range (0-1). For every core test q^i and action/result pair $a, (o, r) : 0 \leq p^T m_{a,(o,r)q^i} \leq 1.0$.
5. A stricter upper bound on 4 can be found by noting that, for every core test q^i and action/result pair $a, (o, r) : 0 \leq p^T m_{a,(o,r)q^i} \leq p^T m_{a,(o,r)}$.
6. The prediction vector must correctly predict each core test. For core test q^i with corresponding entry p_i, $p^T m_{q^i} = p_i$.

[2]We say that a belief state or prediction vector is valid when all future predictions satisfy the laws of probability. Examples are given in the list of possible constraints.

Experimentation on the utility of these constraints is presented in section 6.1. The use of these constraints on valid prediction vectors is key in the development of value iteration algorithms for PSRs.

Although in this paper we focused on IP, the main ideas developed here can be used to extend many other POMDP value iteration-based planning algorithms to PSRs. Next, we turn to another classical planning algorithm.

5 Q-learning on PSRs

Many successful applications of reinforcement learning, e.g., [11, 5] use a model-free learning algorithm like Q-learning on simulated experience from a model of the environment. Q-learning applied in this way becomes a planning algorithm and we extend this idea to PSR-based models of dynamical systems.

We implemented the Q-learning algorithm [12], using prediction vectors as the state representation. The prediction vectors exist in a continuous multidimensional space, and so function approximation was necessary. We used a separate action-value function approximator for each action. The function approximators used were CMACs [1], a grid-based method that uses r overlapping grids, each spanning the entire space of prediction vectors, and each offset by a different amount. Every grid partitions the space of prediction vectors into equal-sized segments, and a value is learned for each segment of each grid. Every prediction vector falls into exactly one segment per grid. The action-value of a prediction vector p and action a is the sum over all grids of each grid's action-value for the prediction vector, i.e.,

$$Q(p, a) = \sum_g v_{g,a}(i_g(p)) \tag{12}$$

where $i_g(p)$ returns the index of the partition in grid g that p falls into, and $v_{g,a}(i)$ returns the value of partition i in grid g for action a. Given this function approximation technique, Q-learning works as follows. Given a current prediction vector p, action a, reward r, and next prediction vector p', let

$$\delta = r + \gamma \max_{a'} Q(p', a') - Q(p, a). \tag{13}$$

Update the appropriate partition of each grid g by

$$v_{g,a}(i_g(p)) = v_{g,a}(i_g(p)) + \alpha\delta \tag{14}$$

where α defines the "per grid" learning rate. This update is applied in an online fashion during interaction with the PSR model of the dynamical system. The actions for this interaction are chosen by an ϵ-greedy

Table 1: Test Problems and Results of IP on both POMDPs and PSRs

Problem	POMDP nominal-states	PSR tests	POMDP trees	POMDP stages	PSR trees	PSR stages
1D maze	4	4	4	70	5	71
4x3	11	11	434	8	465	8
4x3CO	11	11	4	367	4	410
4x4	16	16	23	374	167	6
Cheese	11	11	14	373	16	399
Paint	4	4	9	339	10	371
Network	7	7	549	15	5	463
Shuttle	8	7	482	7	380	7
Tiger	2	2	9	68	9	75

policy [10]. The implementation of this algorithm was a straightforward application of Q-learning using the PSR prediction vector as the state representation.

6 Empirical Results

Although we showed that PSR-IP is theoretically sound, it wasn't clear how this extension would perform in practice. In particular, without empirical testing it was unknown how the use of constraints would affect the performance of the algorithm. The Q-learning algorithm was straightforward, but it was still unknown how the effects of the CMAC function approximation would affect performance and how the two algorithms would perform relative to each other. In order to answer these questions, we ran the algorithms on a set of 9 standard POMDP test problems. The problems are listed in Table 1, and all problem definitions are available at [2].

We now present results for both algorithms: PSR-IP and Q-learning on PSR prediction vectors using CMAC function approximation.

6.1 Incremental Pruning with PSRs

Section 4.2 presents a list of constraints that may be used in deciding whether a prediction vector is valid during the purge routine. Constraint 1 is essential, and can be added to purge's linear program with no additional overhead, and so it is always used. The other constraints all require extra computation in order to be added to the linear program. There is a tradeoff between the usefulness of a constraint and the amount of time it requires to compute. We tested the constraints on the 4x4 problem, and found that constraint 4 had the best tradeoff between limiting invalid prediction vectors and the amount of time required, and the results for PSR-IP make use of this constraint.

As a baseline for PSR-IP, we also present the results of running POMDP-IP for the same problems. Results for both PSR-IP and POMDP-IP were generated by running the problem for 8 hours, or until the maximal difference between two successive value functions fell below 1E-9, in which case we say that the problem has completed. The number of trees reported is for the last successful stage.

Table 1 presents the experimental results of both PSR-IP and POMDP-IP. For POMDP-IP, the 4x3, Network, and Shuttle problems did not complete; while for PSR-IP, the 4x3, 4x4, and Shuttle problems did not complete. The Network problem completed for PSR-IP, but not for POMDP-IP. Investigating this led to an interesting observation. POMDP models update belief states using only the observations and not the rewards while PSR models update prediction vector states using both the observations and rewards. In some controlled dynamical systems, e.g., the Network problem, it may be that the reward provides information about state that is not obtained from the observations alone. This can lead PSRs to have different and perhaps more accurate state than POMDPs. This accounts for the better performance of PSR-IP on the Network problem. On the other hand, the 4x4 problem completed for POMDP-IP, but not for PSR-IP. This is due to a lack of suitable constraints[3], and so the number of policy trees corresponding to invalid prediction vectors grew too large, slowing the algorithm severely. Additionally, many of the completed problems had one or two more policy trees for PSR-IP than for POMDP-IP. This is also due to inadequate constraints. The development of better constraints is an important area for future work on value iteration algorithms on PSRs.

[3] All the constraints listed in Section 4.2 were tried for the 4x4 problem, but none allowed the problem to complete.

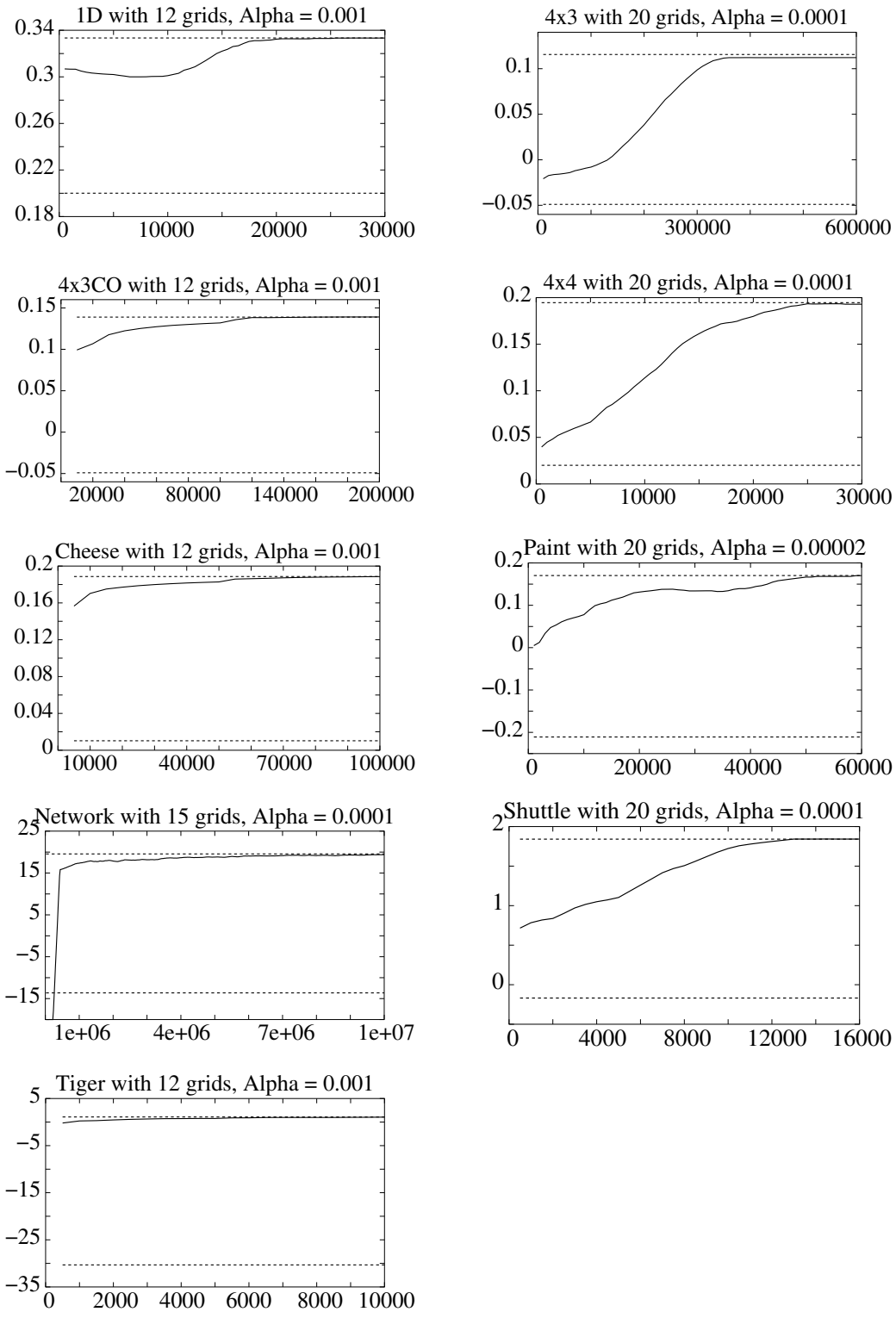

Figure 1: Results of Q-learning with CMAC on Test Problems: These graphs show the average reward per time step given by the learned policies. The y-axis on all graphs is the average reward, and the x-axis is the number of steps taken by the learning algorithm. Each graph is labeled with the problem, number of CMAC grids, and the per grid learning rate.

To summarize, our empirical results show that PSR-IP performed similarly to POMDP-IP for most problems. In the next section, we will use the final policy trees obtained by PSR-IP in order to define the comparison policy against Q-learning.

6.2 Q-learning on PSRs

This section presents the results of Q-learning with CMAC on PSRs on the nine problems of the previous section. The Q-learning algorithm ran for a large number of steps (using a uniform random exploration policy), stopping at regular intervals to test the current learned policy. Testing was performed by executing the greedy learned policy for a large number of steps[4] without learning, and recording the average reward per time step. Ten runs of this experiment were executed per problem, resulting in the graphs seen in Figure 1.

Also presented in these graphs are the average reward found by following the final policy found by PSR-IP[5] (the upper dashed line), and the average reward found by following a uniform random policy (the lower dashed line). These numbers were generated by averaging the reward received under these two policies in 10 runs of length $1,000,000$ each.

The parameters of the Q-learning algorithm included the number of CMAC grids, the number of partitions per dimension for each grid, and α, the per grid learning rate. The number of grids and α are listed in Figure 1 for each problem. The number of partitions for every problem was 10, except for the 4x3CO problem, where we used 20 grids.

As can be seen in the graphs, the Q-learning algorithm performed quite well. On each problem, the policy for Q-learning approached the same level of performance as the best policy found by PSR-IP. For 7 of the 9 problems, the problems on which PSR-IP converged (and 4x4), this corresponds to an optimal policy. Thus, for every problem with a known optimal policy, Q-learning on PSRs converges to policies that are optimal or nearly optimal.

7 Conclusion

We presented two algorithms for planning in PSRs. We showed how the theoretical properties of POMDP-IP algorithms extend to the PSR-IP algorithm. Empirical comparison of the more straightforward Q-learning with CMACs on PSRs with PSR-IP showed that they achieved similar asymptotic performance.

[4]this was set to a minimum of 100,000 in order to obtain a large sample.

[5]For 4x4, we used the optimal policy found by POMDP-IP.

As future work, we are pursuing combining these planning algorithms with methods for learning PSR models in unknown controlled dynamical systems.

Acknowledgements: Satinder Singh and Michael R. James were funded by NSF grant CCF 0432027.

References

[1] J.S. Albus. A theory of cerebellar function. *Mathematical Biosciences*, 10:25–61, 1971.

[2] A. Cassandra. Tony's pomdp page. http://www.cs.brown.edu/research/ai/pomdp/index.html, 1999.

[3] Anthony Cassandra, Michael L. Littman, and Nevin L. Zhang. Incremental Pruning: A simple, fast, exact method for partially observable Markov decision processes. In Dan Geiger and Prakash Pundalik Shenoy, editors, *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–97)*, pages 54–61, San Francisco, CA, 1997. Morgan Kaufmann Publishers.

[4] Anthony R. Cassandra, Leslie Pack Kaelbling, and Michael L. Littman. Acting optimally in partially observable stochastic domains. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, volume 2, pages 1023–1028, Seattle, Washington, USA, 1994. AAAI Press/MIT Press.

[5] R. H. Crites and Andrew G. Barto. Improving elevator performace using reinforcement learning. In *Advances in Neural Information Processing Systems: Proceedings of the 1995 Conference*, pages 1017–1023, 1996.

[6] Masoumeh T. Izadi and Doina Precup. A planning algorithm for predictive state representations. In *Eighteenth International Joint Conference on Artificial Intelligence*, 2003.

[7] Michael L. Littman, Richard S. Sutton, and Satinder Singh. Predictive representations of state. In *Advances In Neural Information Processing Systems 14*, 2001.

[8] Michael Lederman Littman. Algorithms for sequential decision making. Technical Report CS-96-09, 1996.

[9] Satinder Singh, Michael L. Littman, Nicholas K. Jong, David Pardoe, and Peter Stone. Learning predictive state representations. In *The Twentieth International Conference on Machine Learning (ICML-2003)*, 2003.

[10] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: an introduction*. MIT press, 1998.

[11] Gerald Tesauro. Td-gammon, a self-teaching backgammon program, achieves master-level play. *Communications of the ACM*, 38:58–68, 1995.

[12] C.J.C.H. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, 1989.

[13] Nevin L. Zhang and Wenju Liu. Planning in stochastic domains: Problem characteristics and approximation. Technical Report HKUST-CS96-31, 1996.