# Learning Predictive State Representations in Dynamical Systems Without Reset

**Britton Wolfe** BDWOLFE@UMICH.EDU
**Michael R. James** MRJAMES@UMICH.EDU
**Satinder Singh** BAVEJA@UMICH.EDU
Computer Science and Engineering, University of Michigan, Ann Arbor, MI 48109

## Abstract

Predictive state representations (PSRs) are a recently-developed way to model discrete-time, controlled dynamical systems. We present and describe two algorithms for learning a PSR model: a Monte Carlo algorithm and a temporal difference (TD) algorithm. Both of these algorithms can learn models for systems without requiring a reset action as was needed by the previously available general PSR-model learning algorithm. We present empirical results that compare our two algorithms and also compare their performance with that of existing algorithms, including an EM algorithm for learning POMDP models.

## 1. Introduction

Predictive state representations or PSRs is a recently developed formalism (Littman et al., 2002) for modeling agent-environment interactions as discrete-time controlled dynamical systems. PSRs provide an alternative to more traditional models common to AI and machine learning such as hidden Markov models (HMMs) or partially observable Markov decision processes (POMDPs). A number of representational advantages of PSRs relative to POMDPs have been shown: 1) PSRs can model all environments at least as compactly as POMDPs (Singh et al., 2004), 2) PSRs can model some environments exponentially more compactly than POMDPs (Rudary & Singh, 2004), and 3) PSR models use only observable quantities while POMDP models use unobserved or hidden variables (Littman et al., 2002). These relative advantages together offer the hope of more robust and efficient methods for learning models of agent-environment interaction than are available for POMDPs.

A few methods for learning PSR models have already been developed in the still-early phase of exploration in this direction. For example, Singh et al. (2003) developed a gradient method, James and Singh (2004) developed a method for learning PSR models in dynamical systems in which it is possible to reset the environment to an initial configuration, and Rosencrantz et al. (2004) developed a method that uses principle-components analysis for learning PSR models in uncontrolled dynamical systems. In this paper we describe two learning algorithms for PSR models in controlled dynamical systems: a Monte Carlo algorithm called suffix-history and a temporal difference (TD) algorithm. Neither algorithm requires the availability of a reset action, in contrast to the previously available general algorithm for PSR-model learning (James & Singh, 2004). Finally, we empirically compare their performance to each other and to existing applicable methods.

## 2. PSRs

The key idea behind PSRs, and the closely related prior work on observable operator models or OOMs (Jaeger, 1998) as well as the more recent work on TD networks (Sutton & Tanner, 2005), is to represent the state of the environment as a set of predictions about observations that the agent could see as it performs some tests or experiments. A PSR-model maintains a state vector of predictions that allow the agent to make any prediction about future events. We develop the PSR formalism below.

### 2.1. Definitions and Notation

In this paper, we restrict attention to discrete-time dynamical systems. An agent's interaction with such

Appearing in *Proceedings of the 22$^{nd}$ International Conference on Machine Learning*, Bonn, Germany, 2005. 

*Table 1.* A system-dynamics matrix. In this example, the set $Q = \{t_1, t_3, t_4\}$ forms a set of core tests. The initial prediction vector is shown in boldface. The equations in the $t_i$ column show how any entry on a row can be computed from the prediction vector of that row.

| $\mathcal{D}$ | $t_1$ | $t_2$ | $t_3$ | $t_4$ | ... | $t_i$ | ... |
|---|---|---|---|---|---|---|---|
| $\phi_h$ | $\mathbf{p(t_1}\|\phi_\mathbf{h})$ | $p(t_2\|\phi_h)$ | $\mathbf{p(t_3}\|\phi_\mathbf{h})$ | $\mathbf{p(t_4}\|\phi_\mathbf{h})$ | | $p(t_i\|\phi_h) = p(Q\|\phi_h)m_{t_i}$ | |
| $h_1$ | $p(t_1\|h_1)$ | $p(t_2\|h_1)$ | $p(t_3\|h_1)$ | $p(t_4\|h_1)$ | | $p(t_i\|h_1) = p(Q\|h_1)m_{t_i}$ | |
| $h_2$ | $p(t_1\|h_2)$ | $p(t_2\|h_2)$ | $p(t_3\|h_2)$ | $p(t_4\|h_2)$ | | $p(t_i\|h_2) = p(Q\|h_2)m_{t_i}$ | |
| $\vdots$ | | | | | | | |
| $h_j$ | $p(t_1\|h_j)$ | $p(t_2\|h_j)$ | $p(t_3\|h_j)$ | $p(t_4\|h_j)$ | | $p(t_i\|h_j) = p(Q\|h_j)m_{t_i}$ | |
| $\vdots$ | | | | | | | |

a system can be written as a sequence of alternating actions and observations, $a^1o^1a^2o^2\ldots$, where we use $a^i$ and $o^i$ to refer to the action and observation of timestep $i$, respectively. The actions are selected from a set $A$, and the observations are members of a set $O$.

A *history* is a sequence of alternating actions and observations $a^1o^1a^2o^2\ldots a^no^n$ that describes an agent's experience in the system up through timestep $n$. A *test* is a sequence of alternating actions and observations $a_1o_1a_2o_2\ldots a_mo_m$ that describes a possible sequence of future actions and observations. Note the distinction between $a_i$ and $a^i$; the former is a variable that represents the $i$th action in a test, while the latter is a variable representing the action of the $i$th timestep. A test *succeeeds* if the observations of the test are obtained, given that the test's actions are taken. A *prediction* for a test $t = a_1o_1\ldots a_mo_m$ starting from a given history $h$ is the probability that $t$ will succeed when its actions are executed immediately following $h$. Formally, the prediction for a test $t = a_1o_1\ldots a_mo_m$ from some history $h$ of length $n$ is

$$p(t|h) = Pr(o^{n+1} = o_1, o^{n+2} = o_2, \ldots o^{n+m} = o_m \mid h, a^{n+1} = a_1, a^{n+2} = a_2, \ldots a^{n+m} = a_m).$$

For ease of notation, we will use the following shorthand in the remainder of the paper: for a set of tests $T = \{t_1, t_2, \ldots t_n\}$, $p(T|h) = [p(t_1|h)\ p(t_2|h)\ \ldots\ p(t_n|h)]$ is a row vector of predictions for the tests in $T$ from a history $h$. Similarly, for a set of histories $H = \{h_1, h_2, \ldots h_n\}$, $p(t|H) = [p(t|h_1)\ p(t|h_2)\ \ldots\ p(t|h_n)]^T$ is a column vector of predictions for test $t$ from the histories in $H$. And lastly, $p(T|H)$ is a $|H| \times |T|$ matrix such that the $(i,j)$th entry is $p(t_j|h_i)$.

### 2.2. System-Dynamics Matrix

A *system-dynamics matrix* $\mathcal{D}$ completely specifies a discrete-time dynamical system (Singh et al., 2004). It is a theoretical matrix with an infinite number of rows and an infinite number of columns (see Table 1). Each row corresponds to a history, including the empty or null history $\phi_h$. Each column corresponds to a test. The entry $\mathcal{D}_{ij}$ is defined to be $p(t_j|h_i)$.[1] More detail about the system-dynamics matrix can be found in Singh et al. (2004); this introduction is merely intended to facilitate the later discussion of the learning algorithms.

If a system-dynamics matrix $\mathcal{D}$ has finite rank $n$, then there exists some set $C$ of $n$ columns of $\mathcal{D}$ that spans the image of $\mathcal{D}$. That is, all columns of $\mathcal{D}$ are linearly dependent upon $C$. As noted in Singh et al. (2004), any dynamical system that can be modeled as a POMDP with a finite number of states $m$ has a system-dynamics matrix of rank no more than $m$. So restricting our attention to finite-rank system-dynamics matrices is a mild assumption that will be made for the remainder of this paper.

A set of linear *core tests* for a system is a set $Q = \{q_1, \ldots, q_n\}$ of tests such that the columns of $\mathcal{D}$ corresponding to $Q$ are maximally linearly independent, i.e., all columns of $\mathcal{D}$ are linearly dependent on the columns corresponding to $Q$. Similarly, a set of linear *core histories* for a system is a set $H$ of $n$ histories such that the rows of $\mathcal{D}$ corresponding to $H$ are maximally linearly independent. At least one valid set of core tests (or histories) exists for any system $\mathcal{D}$ of finite rank.

### 2.3. Linear PSR Model

This section will describe one type of PSR called a *linear PSR*. The algorithms that we will present later learn a linear PSR (or variant) as a model of the environment. In the remainder of this paper we will drop the "linear" qualifier. We will begin by describing the components of a PSR and then describe its operation.

A *PSR* model of a system is completely defined by a

[1] The histories (and similarly tests) can be ordered by length, and ordered lexicographically within the same length, giving a totally-ordered, countable list of histories (or tests).

pair of objects: an *initial prediction vector* or *initial state* $p(Q|\phi_h)$, and a set of matrices $\{\forall a, o : M_{ao}\}$ and vectors $\{\forall a, o : m_{ao}\}$ called the *model update parameters*. In order to understand the semantics of these components, it is useful to keep in the mind the fact that the system and its system-dynamics matrix $\mathcal{D}$ are interchangeable, i.e., a PSR model of a system should be able to generate $\mathcal{D}$.

- The initial prediction vector $p(Q|\phi_h)$ is a row vector such that the $i$th element of $p(Q|\phi_h)$ is $p(q_i|\phi_h)$, for each core test $q_i \in Q$.
- For each action $a$ and observation $o$, the vector $m_{ao}$ is a column vector such that $p(Q|h)m_{ao} = p(ao|h)$ for all $h$. Such a vector exists and is unique because of how the core tests are defined: The column of $\mathcal{D}$ that corresponds to the test $ao$ is a linear combination of the core tests' columns. The $m_{ao}$ vector is composed of the weights in that linear combination.
- For each action $a$ and observation $o$, the matrix $M_{ao}$ is a $|Q| \times |Q|$ matrix with the $i$th column equal to $m_{aoq_i}$, where $m_{aoq_i}$ satisfies $p(aoq_i|h) = p(Q|h)m_{aoq_i}, \forall h$. The $m_{aoq_i}$ vectors exist and are uniquely determined by $Q$ and $\mathcal{D}$ in the same way as the $m_{ao}$ vectors. In fact, since all columns are linearly dependent upon the core tests' columns, there exists an $m_t$ such that $p(Q|h)m_t = p(t|h), \forall h$ for any test $t$ (Table 1).

The fact that $m_t$ exists for any $t$ shows that $p(Q|h)$ is a sufficient statistic for the history $h$. That is, given $p(Q|h)$, it is possible to compute the accurate prediction for any test from the history $h$, without using $h$ itself. Thus, the *prediction vector* $p(Q|h)$ at any history $h$ is the state of the system after history $h$.

The prediction vector is updated according to

$$p(Q|hao) = \frac{[p(aoq_1|h) \ \cdots \ p(aoq_n|h)]}{p(ao|h)} = \frac{p(Q|h)M_{ao}}{p(Q|h)m_{ao}}$$

upon taking action $a$ and seeing observation $o$ from history $h$.

## 3. Learning Algorithms

The problem that we wish to focus on is simply, *How can an agent learn a PSR model of its environment?*

In the remainder of this section, we will describe our two algorithms: the suffix-history algorithm and the temporal difference (TD) algorithm for learning a PSR model of an environment. Both of our algorithms learn from one continuous interaction between agent and system. This is in contrast to the closest prior work in James and Singh (2004) that requires several interactions, each one beginning after resetting the environment to its initial configuration. Of course, in many real-world environments a reset action may not be available.

### 3.1. Suffix-History Algorithm

Suppose that one had access to an oracle that could be queried for entries in the system-dynamics matrix $\mathcal{D}$. We will first describe how one can build a PSR model using such an oracle. Then we will show how queries to the oracle can be replaced by estimates of the entries of $\mathcal{D}$ from the data.

#### 3.1.1. Using an Oracle

Suppose that one knew a set of core tests $Q$ and a set of core histories $H$. Then for all tests $t$, $p(t|H) = p(Q|H)m_t$ by definition of $m_t$. Since $Q$ and $H$ are core, the matrix $p(Q|H)$ is invertible, so one can compute $m_t = p^{-1}(Q|H)p(t|H)$ for any $t$. This includes $\{m_{ao}\}$ and $\{m_{aoq_i}\}$, which are the update parameters needed for the PSR model. Thus, one can build a PSR model by asking the oracle for $p(Q|\phi_h)$, $p(Q|H)$, and $\{\forall a, o, i : p(ao|H), p(aoq_i|H)\}$.

Of course the core tests and histories are not known to begin with. These can be determined through an iterative procedure. On the $i$th iteration, the algorithm examines a submatrix $p(T_i|H_i)$ of $\mathcal{D}$, computing its rank $r_i$. It finds $r_i$ linearly independent rows and columns corresponding to some histories $H'_i$ and tests $T'_i$. If $r_i = r_{i-1}$, then the procedure stops and the core tests (histories) are $T'_i$ ($H'_i$). Otherwise ($r_i \neq r_{i-1}$), $T_{i+1}$ and $H_{i+1}$ are computed as $\{\forall a, o, t \in T'_i : ao, aot\}$ and $\{\phi_h\} \cup \{\forall a, o, h \in H'_i : ao, hao\}$, respectively. Then the next iteration begins. The initial set $T_1$ ($H_1$) is all tests (histories) of length 1 or less.

This iterative method for choosing $Q$ and $H$ is not guaranteed (James & Singh, 2004) to find a full set of core tests (histories), but the tests (histories) that it does find will be linearly independent. Despite this limitation, this procedure often works well in practice, as seen in James and Singh (2004) as well as in our results below.

#### 3.1.2. Without an Oracle

Without an oracle, one could estimate an entry $p(t|h)$ in $\mathcal{D}$ by performing a Bernoulli trial: execute the actions of $t$ starting from history $h$, and do this multiple times. The fraction of individual trials in which $t$ suc-

ceeded is then $\hat{p}(t|h)$. This is the method used in James and Singh (2004). The problem with this method is that it requires being in history $h$ multiple times. We have only a single long history $S = a^1o^1a^2o^2 \cdots a^no^n$ of interaction as our data, so we see each history at most once.

The suffix-history algorithm gets around this issue by treating every suffix of $S$ as if it were a separate training sequence. The estimate $\hat{p}(t|h)$ is then the proportion of these training sequences (suffixes of $S$) where the observations of $t$ are seen given that the actions of $t$ are taken following $h$. This is equivalent to $\hat{p}(t|h) = Succeeded(t,h)/Executed(t,h)$, where $Succeeded(t,h)$ is the number of times the sequence $ht$ was seen in $S$ and $Executed(t,h)$ is the number of times the actions of $t$ followed the sequence $h$ in $S$. This method allows us to obtain information about many entries in $\mathcal{D}$ from any subsequence of $S$.

Our method of using suffixes as histories is similar to one used by Jaeger (1998) for uncontrolled dynamical systems. One drawback of both these methods is that the suffix-histories are not independent, as components of a true Bernoulli trial should be. However, this is ameliorated by the fact that suffix-histories that are far enough apart are almost independent. Another potential issue is that suffix-histories do not start in the initial configuration of the system. However, if the system $\mathcal{D}$ has a stationary distribution (induced by following a fixed policy), then we can view the suffix-histories as starting from that stationary distribution instead of the initial history $\phi_h$ of $\mathcal{D}$. Thus we would learn a model of a slightly different system $\mathcal{D}'$ whose dynamics is the same as $\mathcal{D}$ except for the initial configuration. In the appendix we prove that under certain conditions the core tests and the model-update parameters of $\mathcal{D}'$ are the same as for $\mathcal{D}$.

When using estimates $\hat{\mathcal{D}}$ instead of the accurate values $\mathcal{D}$ from an oracle, one cannot perform strict linear independence tests (as required in the oracle method) because of the noise in the estimates. Instead, we examine singular values to test for linear independence and use thresholds or cutoffs (on singular values) computed based on the number of samples available for each estimate. The more samples we have in our estimates the less conservative the singular value cutoff becomes. The details of this procedure are identical to that used in James and Singh (2004) and we omit them here.

### 3.2. TD Algorithm

The suffix-history algorithm is a Monte Carlo algorithm in the sense that an estimated prediction $\hat{p}(t|h)$ is obtained by carrying out all the actions of $t$ from $h$. One then notes if $t$ succeeded, and $\hat{p}(t|h)$ is an average across several trials. Considerable experience in the field of reinforcement learning has shown the superiority of temporal difference (TD) methods over Monte Carlo methods for predicting value functions (see, e.g., Singh & Sutton, 1996). In a recent paper Sutton and Tanner (2005) showed that TD algorithms can be adapted to predictions of tests in a class of models they call TD networks that are related to PSRs (though the precise relationship is not yet fully developed). Here we adapt their TD algorithm to PSR models.

#### 3.2.1. The Temporal Difference Idea

The basic idea behind TD methods is to update a guess of a long-term outcome on the basis of a guess at the next time step instead of waiting until the end as in Monte Carlo methods. For example, suppose the outcome of interest is a prediction $p(t|h)$ for $t = a_1o_1a_2o_2a_3o_3$ and for some history $h$ of length $k$. After observing $h$, the current model can compute an estimate $\hat{p}(t|h)$ of $p(t|h)$. If the agent takes action $a_1$ at timestep $k+1$, it gains information about $p(a_1o_1|h)$ as it sees the next observation $o^{k+1}$ and compares it with $o_1$. This information allows it to compute a new one-step delayed estimate of $p(t|h)$,

$$\tilde{p}(t|h) = \begin{cases} \hat{p}(a_2o_2a_3o_3|ha_1o_1), & o^{k+1} = o_1 \\ 0, & o^{k+1} \neq o_1 \end{cases}.$$

Note that $E[\tilde{p}(t|h)] = p(a_1o_1|h) * \hat{p}(a_2o_2a_3o_3|ha_1o_1)$, which equals $p(t|h)$ if $\hat{p}(a_2o_2a_3o_3|ha_1o_1)$ is accurate. So the TD error is $\tilde{p}(t|h) - \hat{p}(t|h)$, and the model parameters can be updated based on this error. This will work well if, on average, $\tilde{p}(t|h)$ is more accurate than $\hat{p}(t|h)$. We would expect this to be the case because $\hat{p}(t|h)$ is a prediction for a $|t|$-step test, while $\tilde{p}(t|h)$ is a prediction for a $(|t|-1)$-step test. The $(|t|-1)$-step test, say $a_2o_2a_3o_3$, will occur in the agent's experience at least as often as – and probably more often than – the $t$-step test $a_1o_1a_2o_2a_3o_3$. So we expect the estimate for $a_2o_2a_3o_3$ to be more accurate than that for $a_1o_1a_2o_2a_3o_3$.

#### 3.2.2. TD for learning PSR models

To support TD learning, the minimal PSR model description of Section 2.3 needs to be extended a bit as follows. The set of tests for use in the state representation is expanded to include not only the core tests but also all suffixes of the core tests; the latter are needed to provide the one-step delayed estimates. We will call the expanded set of tests $Y$ and write the expanded state vector in some history $h$ as $p(Y|h)$. The TD al-

gorithm learns a model that uses the following state vector update equation (see Sutton and Tanner (2005) for motivation):

$$p(Y|hao) = g\left(\frac{p(Q|h)W^{ao} + b^{ao}}{d_{ao}(h)}\right)$$

where $Q \subseteq Y$ is the set of "core" tests,[2] $W^{ao}$ is a $|Q| \times |Y|$ weight matrix, $b^{ao}$ is a bias vector, and $g$ is the logistic function that is applied element-by-element to its argument. One can use several different options for the denominator $d_{ao}(h)$: it can always be 1 (no denominator); it can be computed as in linear PSRs as $p(Q|h)w^{ao}$ for some weight vector $w^{ao}$ (denominator weights); or it can be $\hat{p}(ao|h)$ as computed in the current state vector (state for denominator). The different modifications of the update equation define slightly different types of PSR-models; we implemented all of these modifications but in Section 4 present results only for the state for denominator choice that worked approximately best across domains. Note that the update equation for the TD algorithm's PSR-model is similar to that of the original linear PSR, for which $Y = Q$ are the set of core tests, $g(x) = x$, $b^{ao} = \mathbf{0}$, and $d_{ao}(h)$ is computed using denominator weights. The parameters of the model that must be learned by the algorithm are $W^{ao}$, $b^{ao}$, and the denominator weights $w^{ao}$ (if applicable). In this paper the sets of tests $Y$ and $Q$ are given to the algorithm, i.e., the TD learning algorithm only has to learn the model update parameters and thus faces an easier task than the suffix-history learning algorithm above which needs to discover the set of core tests $Q$ from the data.

*Table 2.* Domain and Core Search Statistics. The *Asymp* column denotes the approximate asymptote for the percent of required core tests found during the trials for suffix-history (with parameter 0.1). The *Training* column denotes the approximate smallest training size at which the algorithm achieved the asymptote value.

| *Domain* | $\|A\|$ | $\|O\|$ | $\|Q\|$ | *Asymp* | *Training* |
|---|---|---|---|---|---|
| Tiger | 3 | 2 | 2 | 100 % | 4000 |
| Paint | 4 | 2 | 2 | 100 % | 4000 |
| Cheese | 4 | 7 | 11 | 82 % | 32000 |
| Network | 4 | 2 | 7 | 43 % | 2048000 |
| Bridge | 12 | 5 | 5 | 100 % | 1024000 |
| Shuttle | 3 | 5 | 7 | 100 % | 1024000 |
| Maze 4x3 | 4 | 6 | 10 | 90 % | 1024000 |

[2]More generally we can use any subset of $Y$ that includes $Q$.

# 4. Results

In this section we present results from experiments using several simulated domains with varying complexity from Cassandra's web site (1999) that were also used to test PSR learning in James and Singh (2004) and Singh et al. (2003). Summary statistics for each domain are presented in Table 2. Our experiments provide several comparisons: 1) suffix-history learning with TD learning (again, note that the latter is given the tests to begin with and thus faces an easier task), 2) the two algorithms described here with the results from the older PSR-learning algorithm on these same domains, and finally 3) the performance of all the PSR learning algorithms to the performance of EM-based POMDP learning.

## 4.1. Experimental Setup for each Domain

Our experiments aim to answer the following question: *How accurate of a model can be learned for a given length of training data?* Thus for each algorithm, we test the performance with several different training-sequence lengths. The results are shown in Figure 1 and are detailed below.

For each trial in our experiment, we generated a training data sequence and a test data sequence using a uniformly-random action at each time step. Our evaluation criterion for the algorithms is the accuracy of the learned model's predictions on the test sequence. The error at history $h$ in the test sequence was computed as $\frac{1}{|O|}\sum_{i=1}^{|O|}(p(ao_i|h) - \hat{p}(ao_i|h))^2$, where $a$ happens to be the action chosen in history $h$, $\hat{p}(ao_i|h)$ is the estimate computed by the learned model, and $p(ao_i|h)$ is the true prediction. This is a normalized version of the measurement used in Singh et al. (2003) and James and Singh (2004). The error for each trial is the average (per time step) error over the test sequence, including the first timesteps of testing where the model learned for the system's stationary distribution may be quite inaccurate. Despite this fact, we obtained good results. During testing the learned models' state vector entries were bounded in the $[0, 1]$ range of valid probabilities, but the $\hat{p}(ao_i|h)$ values used to evaluate the models were not bounded and sometimes fell outside the $[0, 1]$ range.

### 4.1.1. EM Algorithm for POMDP learning

To compare PSR and POMDP learning, we used EM (Baum-Welch)[3] to learn a POMDP model for each domain and computed the same prediction error from the learned POMDP model as for PSR-learning. For

[3]We modified the code from Murphy (2004).

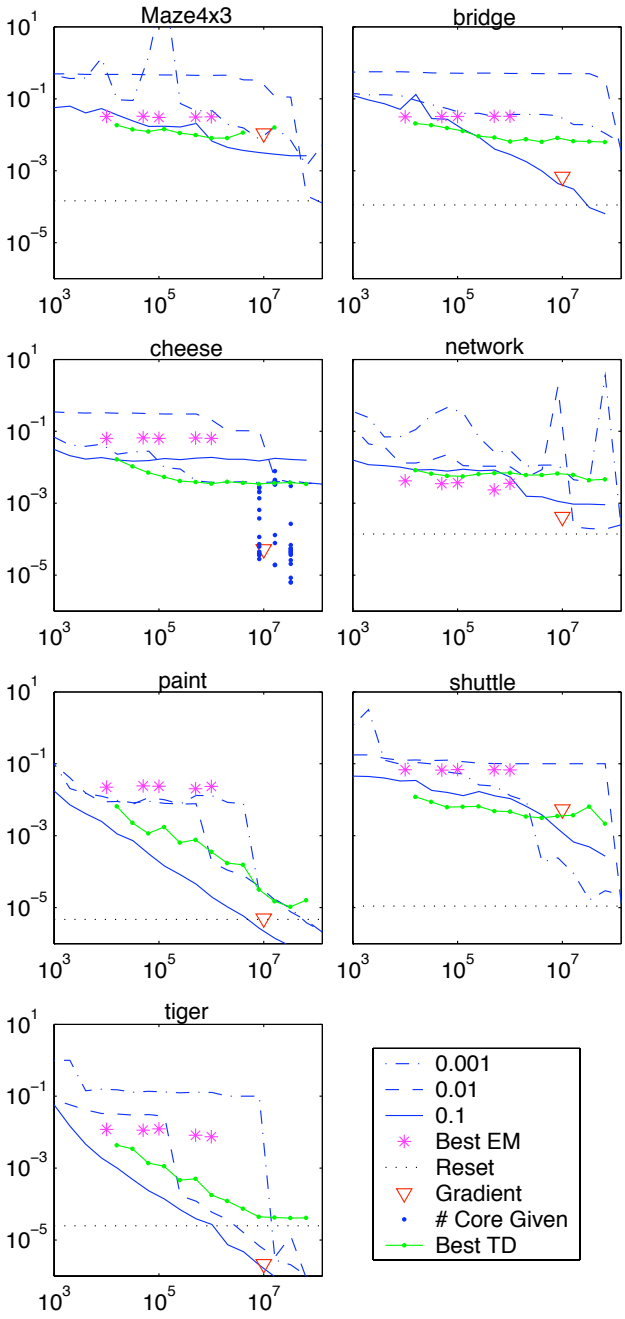


*Figure 1.* Error vs. training length. The scale is identical for each y-axis. The horizontal dotted line represents the reset algorithm's results; the amount of training data used for those results was not available, so the x-axis has no meaning for that series. The lines labeled 0.001, 0.01 and 0.1 are for the suffix-history algorithm with those choices of the scalar parameter. For the cheese domain the error of the reset algorithm is lower than $6 \times 10^{-6}$. Other algorithms' results are labeled in the legend. See text for further detail.

each domain, we used the correct number of states in EM and ran it for 200 iterations upon the training sequence. After each iteration, we evaluated the current POMDP model on the test sequence. The *lowest* error over the 200 iterations in a trial is the error assigned to EM for that trial. Furthermore, for each training-sequence length, the error reported is the *lowest* error over 10 trials. This error is plotted in each graph of Figure 1 as a function of training-sequence length. Note that we gave EM-based POMDP-learning every chance to outperform PSR-learning methods by taking the minimum error over all iterations and all trials as well as by giving it the correct number of states to begin with.

#### 4.1.2. Reset and Gradient Algorithms

The gradient algorithm results presented here are taken from Singh et al. (2003), and the reset algorithm results are taken from James and Singh (2004).

### 4.2. Comparing Algorithms

The different graphs in Figure 1 correspond to different domains. Each graph contains results for three versions of the suffix-history algorithm defined by a scalar parameter with choices 0.001, 0.01 and 0.1; the parameter determines how conservative the algorithm is in using cutoffs in SVD analysis for discovering core tests (with smaller numbers implying more conservative). Overall the suffix-history algorithm with parameter setting 0.1 performs very well when compared with the other algorithms (Figure 1).

**[versus EM for POMDP learning]** The suffix-history algorithm outperforms EM in all domains except for the network domain. In all domains including network, the suffix-history algorithm improves its performance as the training-sequence length increases, whereas the EM algorithm's error curves are relatively flat. Thus it may be the case that the EM algorithm would perform better than suffix-history for small training sizes.

**[versus TD]** The TD algorithm may have a slight edge over the suffix-history algorithm for the smaller training sizes (in the cheese and shuttle domains, e.g.), but the TD algorithm generally performs worse than the suffix-history algorithm, especially for the larger training sizes. This was surprising to us and we explore this further below.

**[versus Gradient]** Suffix-history outperforms the gradient algorithm for all the domains except network (for which the two algorithms were close) and cheese (which was not very close). We explore the performance discrepancy in the cheese domain further be-

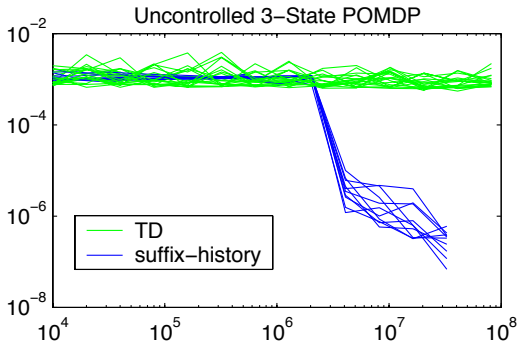


*Figure 2.* Error vs. training length. Each line is a trial of the TD or suffix-history algorithm.

low.

**[versus Reset]** A straightforward comparison against the reset algorithm is not possible because it required the use of a reset action. Nevertheless we plot the lowest error found by the reset algorithm on the same domains from James and Singh (2004). In all domains except for cheese, suffix-history is able to match or outperform reset for sufficiently large training lengths.

Next we sought an explanation for our observation above that in the cheese and network domains suffix-history does not outperform the other algorithms. It turns out that in these problems suffix-history is not able to find enough core tests. Of course, larger training sizes generally resulted in a greater percentage of the required number of core tests being found. In Table 2 we list the approximate asymptote of the % of required core tests found as the training size increases and the training size at which the asymptote was reached. When comparing with the error plots in Figure 1, one can see that the domains for which the algorithm eventually finds a full set of core tests — tiger, paint, shuttle, and bridge — suffix-history performs very well when compared with the EM and gradient algorithms. In the cheese domain, where suffix-history did not find all the core tests and fell well short of the gradient algorithm's performance, we did a separate experiment in which we gave suffix-history the number of core tests in the true model to see if that would make a difference. The graph for the cheese domain shows that indeed, with this prior knowledge suffix-history outperforms all the other algorithms (see points labeled # Core Given). Recall that gradient, TD, and EM all had access to the true number of core-tests and states in all the experiments.

To further clarify the performance of TD versus the Monte Carlo-like suffix-history algorithm, we decided to run an experiment on the simplest of problems in which we would expect TD to outperform Monte Carlo: a 3-state POMDP in which one core-test is 2-steps long and 2 core tests are 1-step long. We expect the TD algorithm to have an advantage over Monte Carlo methods when learning predictions for tests longer than one step (because with only 1-step tests the two algorithms are essentially the same). The comparison between suffix-history and the TD algorithm is shown in Figure 2. For the smaller training sizes, the suffix-history algorithm finds models with *only one core test*, but was still comparable with the TD algorithm. For the larger training sizes, its error drops significantly below the TD algorithm's error, as it finds models with the full number (3) of required core tests. This is consistent with our results in Figure 1. We note that these results seem inconsistent with the results of Sutton and Tanner (2005) who showed that TD outperforms Monte Carlo in a related setting, though this discrepancy may in part be explained by the fact that they focused mostly on fully observable domains while we focused on partially observable domains.

## 5. Conclusions and Future Work

Methods for learning PSRs from data are clearly at an early stage of development; we would characterize them at roughly the stage of learning lookup table-like models. This exploration is necessary to clarify basic issues such as relative advantages of TD versus Monte Carlo, and the relative advantages of learning PSR models versus learning POMDP models using EM. In this paper we presented the suffix-history algorithm, a basic PSR learning algorithm for controlled dynamical systems that does not assume a reset action unlike the previously available general algorithm, as well as a basic TD-inspired algorithm for learning PSR-models. We showed that suffix-history outperforms both TD-learning of PSR models and EM-learning of POMDP models at least with large amounts of data.

As future work there continue to be some basic issues to explore in the simple lookup table-like setting of this paper, e.g., the use of eligibility traces with TD methods and finding more efficient methods for computing prediction estimates from sampled data. Finally, making PSR-model learning practical for real-world applications would require the use of function approximation.

### Appendix

The suffix-history algorithm interacts with a system $\mathcal{D}$ and builds a model for a different system $\mathcal{D}'$ that is derived from $\mathcal{D}$ by changing the initial condition of the system. Specifically, the initial condition for $\mathcal{D}'$

is the stationary distibution of $\mathcal{D}$ under the uniform random policy. We now show two sets of conditions under which core tests and model update parameters for $\mathcal{D}'$ are valid core tests and parameters for $\mathcal{D}$; only the initial prediction vector is different. In the following proofs, we will use $p(\cdot)$ $(Q)$ and $p'(\cdot)$ $(Q')$ to denote predictions (core tests) for $\mathcal{D}$ and $\mathcal{D}'$ respectively. Also, we will use the result that for a matrix $Z = XY$, $rank(Z) \leq min(rank(X), rank(Y))$ several times below.

**Theorem 5.1.** *Let $\mathcal{D}$ be a system with finite rank $n$ that can be modeled by a POMDP with $n$ hidden states. Let $\mathcal{D}'$ be the system obtained by replacing the initial belief state of $\mathcal{D}$ with a new initial belief state. If the rank of $\mathcal{D}'$ is also $n$, then any set of core tests and update parameters for $\mathcal{D}'$ are a valid set of core tests and update parameters for $\mathcal{D}$.*

*Proof.* (Sketch) Let $H$ denote the set of all histories. We define $B$ $(B')$ to be an $\infty \times n$ matrix with the $i$th row being the belief state for $\mathcal{D}$ $(\mathcal{D}')$ in history $h_i$. We also define $p(Q'|S) = p'(Q'|S)$ as an $n \times n$ matrix with the $(i, j)$th element being the probability of core test $j$ succeeding from POMDP state $i$, where $S$ denotes the set of POMDP states. Note that the ranks of $B$, $B'$, $p(Q'|S)$ and $p'(Q'|S)$ are upper bounded by $n$.

By assumption, $|Q| = |Q'| = n$ and therefore $rank(p(Q|H)) = rank(p'(Q'|H)) = n$. From the equation $p(Q|H) = Bp(Q|S)$, we obtain that $rank(B) = n$. Similarly, from the equation $p'(Q'|H) = B'p(Q'|S)$ we obtain that $rank(p(Q'|S)) = n$. Finally, given that $p(Q'|H) = Bp(Q'|S)$, and that $p(Q'|S)$ is a square matrix with full rank, we obtain that $B = p(Q'|H)p^{-1}(Q'|S)$ which in turn implies that $rank(p(Q'|H)) = n$ because $rank(B) = n$ and the maximum possible rank of $p(Q'|H)$ is $n$. Therefore, the set of core tests $Q'$ found for system $\mathcal{D}'$ must also be core tests for the original system $\mathcal{D}$. The update parameters for $\mathcal{D}'$ are valid for $\mathcal{D}$ because they do not depend upon the initial belief state, but only the core tests and the POMDP dynamics (Littman et al., 2002). □

**Theorem 5.2.** *Let $\mathcal{D}$ be a system-dynamics matrix and $h^*$ be a history for $\mathcal{D}$. Let $\mathcal{D}'$ be a system-dynamics matrix such that $\mathcal{D}'$ has the same dynamics as $\mathcal{D}$, but its first row is identical to the row of $\mathcal{D}$ from history $h^*$. If $rank(\mathcal{D}) = rank(\mathcal{D}')$, then any set of core tests and update parameters for $\mathcal{D}'$ are a valid set of core tests and update parameters for $\mathcal{D}$.*

*Proof.* (Sketch) Note that the row corresponding to any history $h$ in $\mathcal{D}'$ is identical to the row corresponding to history $h^*h$ in $\mathcal{D}$. Thus all the rows of $\mathcal{D}'$ are rows of $\mathcal{D}$. Therefore, any set of tests whose columns are linearly independent in $\mathcal{D}'$ must have columns that remain linearly independent in $\mathcal{D}$. Finally, given that $rank(\mathcal{D}) = rank(\mathcal{D}')$, we obtain that $Q'$ are also core tests for $\mathcal{D}$. Let $H'$ denote a set of linearly independent histories of size $|Q'|$ for system $\mathcal{D}'$, and let $h^*H'$ denote the corresponding histories in $\mathcal{D}$. By assumption, for any test $t$ and history $h$, $p(t|h^*h) = p(t|h)$. The update parameters for any $t$, $m_t$ in $\mathcal{D}$ are $p^{-1}(Q'|h^*H')p(t|h^*H')$ and $m'_t$ in $\mathcal{D}'$ are $p'^{-1}(Q'|H')p'(t|H')$, and are thus equal. So the update parameters found for $Q'$ in $\mathcal{D}'$ are also update parameters for $Q'$ in $\mathcal{D}$. □

**Acknowledgements** The authors' research was supported by NSF grant IIS-0413004.

## References

Cassandra, A. (1999). Tony's pomdp file repository page. `http://www.cs.brown.edu/research/ai/pomdp/examples/index.html`.

Jaeger, H. (1998). *Discrete-time, discrete-valued observable operator models: A tutorial* (Technical Report 42). German National Research Center for Information Technology.

James, M. R., & Singh, S. (2004). Learning and discovery of predictive state representations in dynamical systems with reset. *Proceedings of ICML 2004* (pp. 417–424).

Littman, M. L., Sutton, R. S., & Singh, S. (2002). Predictive representations of state. In *Advances in neural information processing systems 14*, 1555–1561.

Murphy, K. (2004). Hidden markov model (hmm) toolbox for matlab. `http://www.ai.mit.edu/~murphyk/Software/HMM/hmm.html`.

Rosencrantz, M., Gordon, G., & Thrun, S. (2004). Learning low dimensional predictive representations. *Proceedings of ICML 2004* (pp. 695–702).

Rudary, M., & Singh, S. (2004). A nonlinear predictive state representation. In *Advances in neural information processing systems 16*, 791–798.

Singh, S., James, M. R., & Rudary, M. (2004). Predictive state representations: A new theory for modeling dynamical systems. *Proceedings of UAI 2004* (pp. 512–519).

Singh, S., Littman, M., Jong, N., Pardoe, D., & Stone, P. (2003). Learning predictive state representations. *Proceedings of ICML 2003* (pp. 712–719).

Singh, S., & Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning, 22*, 123–158.

Sutton, R. S., & Tanner, B. (2005). Temporal-difference networks. In *Advances in neural information processing systems 17*, 1377–1384.