Combining Memory and Landmarks with Predictive State Representations

Michael R. James and **Britton Wolfe** and **Satinder Singh**
Computer Science and Engineering
University of Michigan
{mrjames, bdwolfe, baveja}@umich.edu

Abstract

It has recently been proposed that it is advantageous to have models of dynamical systems be based solely on observable quantities. Predictive state representations (PSRs) are a type of model that uses predictions about future observations to capture the state of a dynamical system. However, PSRs do not use memory of past observations. We propose a model called memory-PSRs that uses both memories of the past, and predictions of the future. We show that the use of memories provides a number of potential advantages. It can reduce the size of the model (in comparison to a PSR model). In addition many dynamical systems have memories that can serve as landmarks that completely determine the current state. The detection and recognition of landmarks is advantageous because they can serve to reset a model that has gotten off-track, as often happens when the model is learned from samples. This paper develops both memory-PSRs and the use and detection of landmarks.

1 Introduction

This paper explores the use of two types of observable quantities — the history of past observations and predictions about future observations — in creating models for complex dynamical systems. Models that use only predictions of future observations to capture state, called predictive state representations or PSRs [Littman *et al.*, 2001], have been shown (surprisingly) to be at least as expressive and compact [Singh *et al.*, 2004] as classical models such as partially observable Markov decision processes (POMDPs) that use hidden state variables. Models that only use observation history, such as k-order Markov models, are known to be not as expressive and hence not as widely applicable as PSRs or POMDPs. On the other hand, history-based models are easier to learn from data than either PSRs and POMDPs. In this paper, we propose an extension to PSRs, called memory-PSRs (*mPSRs*), that combines past observations (*memories*) with predictions about the future to define the state of a dynamical system.[1] Our goal is to provide a model class that is as expressive as PSRs but that has some of the efficient learning properties of history-based models.

In addition to accelerating learning, mPSRs can also exploit *landmarks* or observations that capture the state of the system uniquely, by "resetting" the approximate state of the learned model during prediction whenever these landmark-observations are encountered. We present a method for finding landmarks in mPSRs, develop the basic theory of mPSRs and provide preliminary empirical results exploring the relative efficiency of learning mPSRs and PSRs.

[1] Sutton and Tanner [2004] have also recently proposed a model that combines memory and predictions.

2 PSRs and memory-PSRs

PSRs depart from other models of dynamical systems in that their representation of state is a vector of predictions of the outcomes of *tests* that may be performed on the dynamical system. A test $t = a_1 o_1, ...a_k o_k$ is a sequence of alternating actions $a_i \in \mathcal{A}$ and observations $o_j \in \mathcal{O}$, and its *prediction*, $p(t)$, is the conditional probability that the observation sequence occurs, given that the action sequence is taken, and so $p(t) = prob(o_1, ...o_k | a_1, ...a_k)$. Of course, the prediction of a test is dependent on the actions and observations that have occurred so far, called the *history*. The prediction of a test t at history h is $p(t|h) = prob(o_1, ...o_k | h a_1, ...a_k)$.

A PSR's state representation consists of predictions of a special set Q of tests, called the *core* tests. The core tests are special because at any history, the predictions for *any* test can be computed as a linear function of the predictions of the core tests. The prediction vector $p(Q|h)$ is the $(n = |Q| \times 1)$ vector of predictions for the tests in Q at history h. Thus, $p(Q|h)$ is the PSR's representation of state and is the counterpart to belief-states in POMDPs and the last k observations in k-order Markov models.

In addition to the set of core tests, a PSR has model parameters: a set of $(n \times n)$ matrices M_{ao}, and $(n \times 1)$ vectors m_{ao}, for all a, o. The model parameters allow linear computation of the prediction for any test $t = \{a_1 o_1 \ldots a_k o_k\}$ as follows:

$$p(t|h) = p(Q|h)^T M_{a_1 o_1} ... M_{a_{k-1} o_{k-1}} m_{a_k o_k}. \qquad (1)$$

Updating the current state, or prediction vector, when action a is taken in history h and o is observed, is accomplished by

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

The matrices M_{ao} and vectors m_{ao} have a special form. The i^{th} column of M_{ao} is the $(n \times 1)$ constant vector that computes the prediction of the (ao) one-step extension of the i^{th} core test $q_i \in Q$, i.e., of test aoq_i. The vector m_{ao} is the constant $(n \times 1)$ vector that computes the prediction for the one-step test $t = ao$. The fact that the model parameters have these meanings is the foundation of existing PSR learning algorithms [James and Singh, 2004; Rosencrantz *et al.*, 2004] as well as the new mPSR learning algorithm presented here.

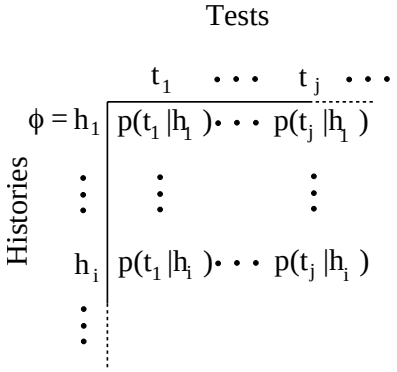

Figure 1: The system dynamics matrix.

System Dynamics Matrix

The theoretical development of mPSRs is best explained using the system dynamics matrix, D, that was developed in Singh *et al.* [2004]. The matrix D, shown in Figure 1, has all possible histories as rows and all possible tests as columns. The columns (rows) are arranged by test (history) length and within the same length in lexicographic ordering. The entry $D_{ij} = p(t_j|h_i)$ is the prediction for the j^{th} test at the i^{th} history. There is a one to one correspondence between system dynamics matrices and dynamical systems. *The rank of D is the dimension of the corresponding dynamical system and is the number of core tests needed by the PSR model.* The core tests of a PSR model correspond to a maximal set of linearly independent columns. Similarly, we can define a notion of *core histories* that correspond to a maximal set of linearly independent rows. We will only be interested in finite dimensional systems. This assumption is not terribly restrictive because all POMDPs with n underlying or nominal states are dynamical systems of dimension at most n. A full derivation of PSRs through use of the system dynamics matrix is presented in Singh *et al.* [2004].

memory-PSRs

The central idea behind mPSRs is to *partition* D into a set of m submatrices $D^1 \dots D^m$ by partitioning the set of histories (rows) but not the tests (columns), i.e., each submatrix D^i contains all the columns but only a subset of the histories. How do memories enter this picture? We will use memory of past observations to partition histories, and every memory will correspond to a partition; we elaborate on this later. For now, note that each submatrix will have its own rank and its own core tests and core histories as well as model-update parameters. The set of memories and core tests for each partition form the basis of mPSR models. It is straightforward that the rank of each submatrix is at most the rank of the full matrix D. In the worst case, the ranks of all the submatrices are the same as the rank of D, in which case the resulting mPSR model may have (many) more parameters than the PSR model. But in other cases the ranks of the submatrices may be much smaller than the rank of D and then the resulting mPSR model may be more compact than the PSR model. We provide examples of both cases in Section 5. The size of the model is important because a model with fewer parameters should in general be more efficient (i.e., require less data) to learn. We also test this empirically in Section 5.

In this paper we do not address the question of automatically discovering useful ways of partitioning histories and instead assume that partitions correspond to history suffixes of some fixed length.

Definition of memory: For this paper, given an mPSR using length k suffixes, a *memory* is a specific sequence of length k that identifies a partition. So, when considering a history h and memories of length one, the memory at h is just the last (most recent) observation in h, and there will be $|\mathcal{O}|$ memories in the mPSR. For memories of length two, the memory at h is the last (most recent) action-observation pair in h, and there will be $|\mathcal{A}| * |\mathcal{O}|$ memories in the mPSR. In general, the set of possible memories in an mPSR that uses length-k suffixes is the set of all length-k sequences of alternating actions and observations that end in an observation; we will denote the size of such a set as m_k.

Let $\mu_1 \dots \mu_{m_k}$ represent all the distinct memories in an mPSR model. Also, let the memory at history h be denoted $\mu(h)$. Each memory μ_i has a corresponding submatrix D^{μ_i} created by the partition of histories corresponding to μ_i. The core tests for partition D^μ are referred to as μ-core tests to distinguish them from the core tests of the PSR model of the same system. Thus, the μ-core tests for the submatrices $D^{\mu_1} \dots D^{\mu_{m_k}}$ are denoted $Q^{\mu_1} \dots Q^{\mu_{m_k}}$ respectively. The prediction vector $p(Q^{\mu(h)}|h)$ contains the predictions for the μ-core tests $Q^{\mu(h)}$ at memory μ_h.

Definition of mPSR state: The *mPSR state* at history h is denoted by the concatenation of the memory at h and the prediction vector for that memory at h, i.e., as $[\mu(h), p(Q^{\mu(h)}|h)]$. Thus, the mPSR state contains both a memory of the past as well as predictions about the future while a PSR state contains only predictions about the future.

Definition of mPSR update parameters: For each memory, μ_i, we will keep a set of matrices $M_{ao}^{\mu_i}$ and vectors $m_{ao}^{\mu_i}$ for all a, o. There is a subtle issue in defining these parameters. It is not the case that each submatrix is a separate dynamical system. Indeed if the memory corresponding to current history h is say μ_i and we take action a and observe o, the memory corresponding to the next history hao could be different from μ_i. Lets say it is μ_j. Thus the update parameters $M_{ao}^{\mu_i}$ must transform the current prediction vector that makes prediction for μ-core tests Q^{μ_i} in history h to the prediction vector for μ-core tests Q^{μ_j} in history hao. Note that all histories belonging to memory μ_i will transition to the same memory μ_j for action-observation pair ao, i.e., j is uniquely deter-

mined by i and the pair ao; and so the model parameter $M_{ao}^{\mu_i}$ need not explicitly reference to the next memory μ_j. Thus one can define the state update for mPSRs as follows: upon taking action a in history h and observing o

$$p(Q^{\mu_j}|hao) = \frac{p(aoQ^{\mu_j}|h)}{p(ao|h)} = \frac{p(Q^{\mu_i}|h)^T M_{ao}^{\mu_i}}{p(Q^{\mu_i}|h)^T m_{ao}^{\mu_i}} \quad (3)$$

where $\mu(h) = \mu_i$ and $\mu(hao) = \mu_j$. The matrix $M_{ao}^{\mu_i}$ is of size $(|Q^{\mu_i}| \times |Q^{\mu_j}|)$ and the vector $m_{ao}^{\mu_i}$ is of size $(|Q^{\mu_i}| \times 1)$. As in PSRs the update parameters in mPSRs have meaning. For instance, the k^{th} column of $M_{ao}^{\mu_i}$ is the $(|Q^{\mu_i}| \times 1)$-sized constant parameter vector that computes the prediction for the test that is the k^{th} μ-core test for memory μ_j.

The mPSR model parameters allow linear computation of the prediction for any test $t = \{a_1 o_1 \dots a_k o_k\}$ at history h as follows:

$$p(t|h) = p(Q|h)^T M_{a_1 o_1}^{\mu_0} \dots M_{a_{k-1} o_{k-1}}^{\mu_{k-2}} m_{a_k o_k}^{\mu_{k-1}}. \quad (4)$$

where μ_0 is the memory corresponding to history h, μ_1 is the memory corresponding to history ha_1o_1 and so on.

Definition of mPSR: An mPSR model is defined by the tuple $\langle \mathcal{A}, \mathcal{O}, \mu_1 \dots \mu_{m_k}, Q^{\mu_1} \dots Q^{\mu_{m_k}}, \mathcal{M}, [\mu_0, p(Q^{\mu_0}|\emptyset)] \rangle$ where $\mathcal{A}$ is the set of actions; $\mathcal{O}$ is the set of observations; $\mu_1 \dots \mu_{m_k}$ are the possible memories; Q^{μ_i} is the set of μ-core tests for memory μ_i; $\mathcal{M}$ is the set of update parameters $M_{ao}^{\mu_i}$ and $m_{ao}^{\mu_i}$ for all a, o, μ_i; μ_0 is the initial memory; and $p(Q^{\mu_0}|\emptyset)$ is the initial prediction vector.

A special and interesting case arises when a memory by itself serves as state and no predictions are needed.

Definition of landmark: Given an mPSR, a *landmark* is a memory that serves as state, i.e., is a sufficient statistic of history. Landmarks can be quite beneficial for making accurate predictions. We will discuss the identification of landmarks and exploiting them for prediction in Section 3.1.

3 Basic Theory of mPSRs

Our first result is to show that for any dynamical system it is possible to find μ-core tests for all memories in an mPSR model from the set Q of core tests for the corresponding PSR model.

Lemma 1. *Let a dynamical system be modeled as a PSR with core tests Q. Then there always exists an mPSR model for the system such that for each memory μ_i the corresponding μ-core tests Q^{μ_i} satisfy $Q^{\mu_i} \subset Q$.*

Proof. We provide a constructive proof which shows how to derive the subset Q^{μ_i} from Q. Recall that all columns of D are linear combinations of the columns corresponding to Q in D. Consider the submatrix D^{μ_i} for memory μ_i. It must be the case that all columns of D^{μ_i} are linear combinations of the columns corresponding to Q in D^{μ_i}. Thus, there exists $Q^{\mu_i} \subset Q$. □

In this paper, we don't exploit Lemma 1 for two reasons: 1) the core tests of PSRs are not unique, and 2) in any case when learning a model of a dynamical system from experience data we do not have a PSR model and thus its core tests Q to begin with.

Lemma 2. *For any dynamical system of finite dimension and any choice of (fixed-length suffix) memories, the size of the resulting mPSR model for that system is at most the number of memories times the size of a PSR model for that system.*

Proof. In the worst case, the rank of each of the submatrices in the mPSR is exactly the rank of the full system dynamics matrix. In such a case, if we happen to find different μ-core tests for each submatrix, then the model for each submatrix will be exactly as large as the PSR model. □

The above lemma holds no matter how one chooses the memories. But what if we can choose the memories to use in constructing an mPSR judiciously (by which we mean to minimize the size of the resulting model)?

Theorem 1. *With a judicious choice of memories, the size of the corresponding mPSR for any dynamical system is at least as compact as the size of the PSR for that dynamical system. Furthermore, there exist dynamical systems for which some choice of memories leads to an mPSR model that is more compact than the PSR model of the same dynamical system.*

Proof. The proof of the first statement follows from the fact that a PSR is also an mPSR with the null set as memories. We prove the second statement by constructing such a dynamical system. Table 1 compares the size of mPSRs and PSRs for various standard test dynamical systems (based on POMDPs). In all instances, a suffix of length one was used to partition the histories. For at least three problems, Cheese, Shuttle and Four-Three, the mPSR is more compact than the PSR. □

3.1 Landmarks

We show that landmarks are equivalent to memories for which there is only one μ-core test and furthermore that the prediction of any test at a landmark is the same at all histories that map to that landmark.

Lemma 3. *For an mPSR model of a dynamical system,*

- *any memory that is a landmark has only one μ-core test, and every memory that has a μ-core set of size one is a landmark.*
- *for all histories that map to a landmark the prediction vector is a constant, i.e., independent of history.*
- *for all histories that map to a landmark the prediction of any test t is a constant, i.e., independent of history.*

Proof. If μ is a landmark, then the predictions for all tests are fully determined when μ is known. Therefore, at any two histories that have memory μ, the prediction of every test is the same at both histories. This means that every row of D^{μ} must be identical, so the rank of D^{μ} is one, and only one μ-core test exists for μ.

For a memory μ that has a single μ-core test, q^{μ}, let H^{μ} denote the histories that map to memory μ. For any $h \in H^{\mu}$, and any action a, it must hold that $\sum_o p(ao|h) = 1$ which implies that $\sum_o p(q^{\mu}|h)^T m_{ao}^{\mu} = 1$, which in turn implies that $p(q^{\mu}|h)^T = 1/\sum_o m_{ao}^{\mu}$. Recall that m_{ao}^{μ} are independent of history, and thus the prediction of q^{μ} must be independent of

Table 1: Comparison of the size of PSRs and mPSRs

Problem	# Core Tests		# Param. Entries	
	PSR	mPSR	PSR	mPSR
Tiger	2	2,2	36	72
Paint	2	2,2	48	96
Cheese	11	1,1,1,1,2,2,3	3696	792
Network	7	4, 6	448	480
Bridge	5	2,2,4,4,5	1800	4488
Shuttle	7	1,1,2,2,4	840	450
Four Three	10	1,1,1,1,3,4	2640	748
Float Reset	5	1,5	120	96

the history. This calculation exploits the fact that the prediction vector and update parameters are scalars.

Furthermore, because the prediction of q^μ is independent of history, and the update vector m_t^μ for any test t is independent of history, the prediction $p(t|h) = p(q^\mu|h)^T m_t^\mu$ of t at history h must also be independent of history. So, all rows of D^μ must be identical, and μ is therefore a landmark. □

Landmarks come in handy because they can be used to keep a learned — and therefore only approximately correct — model from progressively drifting farther and farther away from reality as we make longer and longer term predictions from such a model. Every time we observe a landmark memory, we can reset the prediction vector to the constant corresponding to the landmark irrespective of the actual observed history. This can keep the error in the prediction for very long tests from growing with the length of the test. We exploit this ability in our empirical results below.

4 Learning mPSR models from data

In this section we present an algorithm for learning mPSR models from data under the assumption that the algorithm has the ability to reset the dynamical system being modeled to an initial state. We present the algorithm in two stages. In the first stage, we will assume that the algorithm has access to an oracle that if given a history h and test t will return the prediction $p(t|h)$ for the dynamical system. In the second stage we will show how the oracle can be replaced by the use of sample data from the dynamical system with reset. Because we wish to minimize the amount of sample data needed by our algorithm, we will attempt to minimize the number of calls to the oracle in the first stage. Our algorithm is a relatively straightforward adaptation to mPSRs of the algorithm presented by James and Singh [2004] for PSRs.

4.1 Oracle based algorithm

Recall that there are two components to an mPSR: the μ-core tests for the different memories and the model-update parameters for the memories. In our empirical work below, we always use memories of length one. Clearly the model parameters for a memory depend on the choice of μ-core tests for that memory (recall that the μ-core tests are not unique). The process of computing μ-core tests and update parameters for each memory is identical and so we can just discuss how to do this for any one memory, say μ_i.

Determining μ-core tests and histories

The algorithm proceeds in iterations. At iteration t, the algorithm has a current set of linearly independent or μ-core tests and histories for each memory. Let $H_t^{\mu_i}$ ($T_t^{\mu_i}$) be the set of μ-core histories (μ-core tests) found by iteration t for memory μ_i. These μ-core tests (histories) start off as the empty set at the first iteration. We also keep a set of candidate tests and histories at each iteration. The set of candidate tests for memory μ_i is the set of one-step tests; and for every ao pair, the set of ao-extensions for all the current μ-core tests for memory μ_j, where μ_j is the next memory when action a is taken in memory μ_i and o is observed. The set of candidate histories is similarly generated. We ask the oracle for the predictions of all the candidate tests at all the candidate histories. We compute the rank of this oracle-generated matrix, denoted X_t. The linearly independent rows and columns of X_t become the new $H_{t+1}^{\mu_i}$ and $T_{t+1}^{\mu_i}$ respectively. In selecting the new μ-core tests and histories we always include the previous μ-core tests and histories. We stop at iteration s if the rank of X_s is the same as the rank of X_{s-1} for all memories.

The above algorithm is an adaptation of the similar algorithm for determining core test and histories for PSRs by James and Singh [2004]. Just like for PSRs this algorithm is not guaranteed to find all μ-core tests and histories, but also just like for PSRs it seems to work for all empirical work done so far.

Computing the model-update parameters

Now we discuss how to compute the model-update matrix M_{ao}^μ for any ao pair and any memory μ. We define a matrix A for memory μ that contains the predictions for all the μ-core tests at all the μ-core histories for memory μ. This matrix will have full rank and is computed in the algorithm above. Let μ_j be the memory achieved when action a is taken in memory μ and o is observed. The k^{th} column of M_{ao}^μ, denoted x, is the constant parameter vector that computes the prediction for the ao-extension of the k^{th} μ-core test, denoted y, of memory μ_j. Let b be the column vector of predictions of test y for the μ-core histories of memory μ. We ask the oracle for the entries of b. Then from Equation 3 above, $Ax = b$. Since A is full rank, it is invertible and hence $x = A^{-1}b$. We can use the same idea to compute the model-update parameter vector m_{ao}^μ.

In the next section we show how the oracle needed for the above algorithm can be replaced by using data from the dynamical system with reset.

4.2 Learning mPSRs from sample data

We show how to obtain an estimate for the prediction $p(t|h)$, something we went to the oracle for in the previous algorithm. We generate samples from the distribution $p(t|h)$ by first taking the action sequence in h after a reset. If the observation sequence in h happens, then we have succeeded in generating history h. If we don't succeed in generating the history, we reset the system and try again. If we do succeed in generating the history, we take the action sequence in test t. If the observation sequence in test t happens, the test succeeds, else the test fails. We get an estimate of $p(t|h)$ from the empirical success rate of the test t at history h. Of course, this is ex-

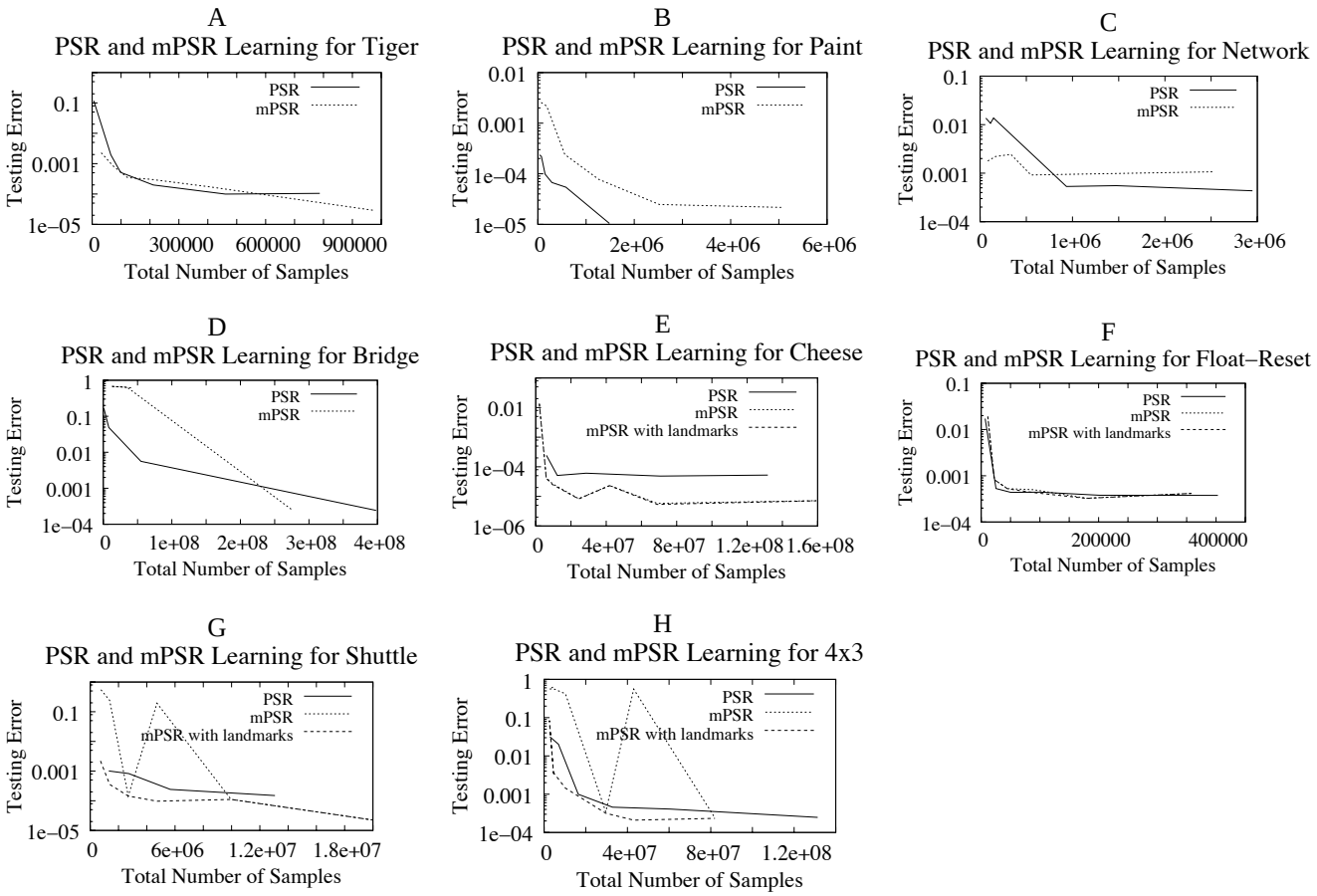

Figure 2: Results of running mPSR learning algorithms on a suite of test problems. See text for details.

tremely wasteful in the number of data samples needed. We implement this basic idea much more efficiently by executing the action sequence in ht regardless of the observations obtained and then mining the data generated for all possible history and test pairs that can be extracted from it.

One problem that occurs with this estimation technique is that the sampled entries of the system dynamics matrix will almost certainly result in inaccurate calculations of rank (needed in the above oracle-based algorithm). We use a more robust rank calculation procedure developed in the PSR learning algorithm by James and Singh [2004]. The central idea is that we can use the number of samples that went into each entry of the matrix we wish to find the rank of to generate a threshold on singular values obtained via singular value decomposition. This threshold is more conservative when we have fewer samples and generally leads to a smaller rank than otherwise. As more samples are included, the threshold becomes less conservative and the calculated rank is closer to the straightforward calculation of rank.

5 Empirical results

We conducted experiments with a view towards validating four major ideas presented above. The first experiments are meant to compare the relative size of PSRs and mPSRs. The second is to test whether the oracle-based algorithm for computing mPSRs finds exact mPSR models. The third is to test the efficacy of the sample-data based learning algorithm for mPSRs. Finally, we are also interested in whether using landmarks, where available, to reset prediction vectors during the process of computing predictions for long test sequences provides a measurable benefit. All of these experiments were on a suite of standard POMDP-based dynamical systems [Cassandra, 1999] that have been used for both POMDP-learning as well for PSR-learning.

Again, for all of the results presented below, the memories used for the mPSR models were the most recent observation.

5.1 Comparing PSRs and mPSRs

For PSRs and mPSRs, the size of the model can be measured both in terms of the number of core tests (μ-core tests), and in the number of entries in the model parameters. Table 1 compares the sizes of PSR and mPSR models for the suite of test problems. For mPSRs, the number of μ-core tests at each memory is listed. Fully half of the test problems have landmarks (memories with only one μ-core test), indicating that this may be a fairly common situation.

The number of entries in the model parameters are listed for both PSRs and mPSRs. On three of the problems (Cheese,

Shuttle, and Four-Three) there are significantly fewer entries needed for the mPSR model than for the PSR model; on three other problems (Tiger, Bridge, and Paint) there are significantly more; and on the two remaining problems there are approximately an equal number needed. This proves that mPSRs can be more compact than PSRs. This also illustrates that the unless we choose memories wisely, mPSRs can also be less compact than PSRs.

5.2 Constructing mPSRs from exact predictions

We tested the oracle algorithm for computing mPSRs by providing access to the true predictions computed analytically for the test problems. As we mentioned in Section 4.1, this algorithm is not guaranteed to find all μ-core tests for every memory. We found that in fact for all the test problems, the algorithm does indeed find all the μ-core tests and histories for all the models. We also verified that the resulting mPSR models were perfect, in the sense that any prediction error was due to machine round-off error. Furthermore, because correct sets of μ-core tests were discovered, all landmarks were identified.

5.3 Learning mPSRs from samples

Here we present the results of learning mPSR models for all our dynamical systems. As mentioned above we assumed a reset. This makes our results directly comparable to the results of James and Singh [2004] on the same set of problems under the same reset assumption. We applied the algorithm of Section 4.2, stopping every so often during learning to test the model learned so far. For each test of the learned model, we asked the model to make predictions while tracking the system for $100,000$ steps (with randomly chosen actions). The test error reported is the average error in the one-step predictions of the model relative to the true predictions. This is the same error measure as used in James and Singh [2004].

The results of this testing are presented in Figure 2. We compare our results to the results for PSR learning for the same dynamical systems from James and Singh [2004]. Plots A to D present the results for the systems without any landmarks, and show the results of PSR and mPSR learning. In two of these systems, PSR learning is more efficient while in the other two they are roughly equivalent. We note that in all of these problems the PSR models are more compact or equal in size than the mPSR models. Plots E to H are for systems with landmarks. Here we plot three results: PSR learning, mPSR learning and mPSR learning with landmarks. The last algorithm, mPSR learning with landmarks, uses the mPSR learning algorithm of Section 4.2 but during testing resets the prediction vector when a landmark memory is encountered to the constant prediction for that memory. In three of the systems corresponding to graphs E, G and H, the size of the mPSR model is smaller than the PSR model. As we surmised in Section 2 this leads to mPSR learning being more efficient than PSR learning. In the system for plot F the two are roughly equivalent as are their sizes. The effect of using landmarks is most easily seen in plots G and H. Here mPSR learning without using landmarks is very noisy for if the system gets off-track in its predictions the error accumulates rapidly as the test sequence lengthens, while using the landmarks stabilizes the error and gives the best results.

6 Conclusion

In this paper we proposed a new class of models, mPSRs, that combines memory of past observations and predictions of future observations into its state representation. We showed that mPSRs are as expressive as and can be more compact than PSRs that solely use predictions of future observations in its state representation. Our preliminary empirical results showed that in dynamical systems where the mPSR model of the system is more compact than the PSR model, learning the mPSR model is more efficient than learning the PSR model. Finally, we formalized the notion of landmark memories and demonstrated how to find them and how to exploit them in making long-term predictions.

As future work, we will explore the interesting challenge of automatically finding good memories to combine with predictions of tests to make compact and efficiently learnable mPSR models.

Acknowledgements The research reported in this paper was supported by NSF grant IIS-0413004. The authors also thank Rich Sutton for helpful comments.

References

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

[Golub and Van Loan, 1996] G.H. Golub and Ch. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, 3rd edition, 1996.

[James and Singh, 2004] Michael R. James and Satinder Singh. Learning and discovery of predictive state representations in dynamical systems with reset. In *The 21st International Conference on Machine Learning*, 2004.

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

[Rosencrantz *et al.*, 2004] Matthew Rosencrantz, Geoff Gordon, and Sebastian Thrun. Learning low dimensional predictive representations. In *The Twenty-First International Conference on Machine Learning*, 2004.

[Singh *et al.*, 2003] 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*, 2003.

[Singh *et al.*, 2004] Satinder Singh, Michael R. James, and Matthew R. Rudary. Predictive state representations, a new theory for modeling dynamical systmes. In *20th Conference on Uncertainty in Artificial Intelligence*, 2004.

[Sutton and Tanner, 2004] R. S. Sutton and B. Tanner. Temporal-difference networks. In *Advances in Neural Information Processing Systems 17*, 2004.