Predictive State Representations with Options

Britton Wolfe BDWOLFE@UMICH.EDU
Satinder Singh BAVEJA@UMICH.EDU
University of Michigan, Computer Science and Engineering, 2260 Hayward Drive, Ann Arbor, MI 48109 USA

Abstract

Recent work on predictive state representation (PSR) models has focused on using predictions of the outcomes of open-loop action sequences as state. These predictions answer questions of the form "What is the probability of seeing observation sequence $o_1, o_2, \ldots, o_N$ if the agent takes action sequence $a_1, a_2, \ldots, a_N$ from some given history?" We would like to ask more expressive questions in our representation of state, such as "If I behave according to some policy until I terminate, what will be my last observation?" We extend the linear PSR framework to answer questions like these about *options* – temporally extended, closed-loop courses of action – bounding the size of the linear PSR needed to model questions about a certain class of options. We introduce a *hierarchical PSR* (HPSR) that can make predictions about both options and primitive action sequences and show empirical results from learning HPSRs in simple domains.

Existing work with predictive state representations (PSRs) focuses on using predictions about open-loop action sequences as state. These predictions answer questions of the form "What is the probability of seeing observation sequence $o_1, o_2, \ldots, o_N$ if the agent takes action sequence $a_1, a_2, \ldots, a_N$ in some given history?" Littman et al. (2002) showed that predictions of this form are *sufficient* in that they can perfectly capture state and can be used to make any prediction, i.e., answer any question, about the system. In general, the number of predictions in the state vector grows linearly with the number of underlying or hidden system states and this can be too large for practical purposes. Of course, if one truly wants a perfect model that can answer any question about the system at all, one cannot in general circumvent this impracticality. In practice, we may be interested only in a subset of all possible questions, e.g., questions relating to predictions useful for effective planning. A fundamental motivation for this research is the following: can we build predictive models whose complexity scales not with the complexity of the underlying system but with the complexity of the set of questions we want to be able to answer? In this paper, we take a first step towards an answer by showing that PSR models that are restricted to making predictions about options (known to be useful for planning) can be much smaller than full PSR models and hence much more efficiently learnable. In addition, we define and test a hierarchical PSR (HPSR) that can make predictions both about options and primitive action sequences.

Appearing in *Proceedings of the 23rd International Conference on Machine Learning*, Pittsburgh, PA, 2006.

1. Background and Definitions

Before we show how to construct a PSR that makes predictions about options, we introduce some notation and state our assumptions. We assume that the agent is in a discrete-time environment with a set of discrete, primitive actions A and a set of discrete observations O. At each time step x, the agent chooses some action $a^x \in A$ to execute and then receives some observation $o^x \in O$. A *primitive history* is a possible sequence of primitive actions and observations $a^1 o^1 a^2 o^2 \ldots a^k o^k$ from the beginning of time.

In addition to the primitive actions, we assume that the agent has a set of options Ω. An *option* $\omega \in \Omega$ can be viewed as a temporally extended action that prescribes a closed-loop way of behaving until some (stochastic) termination condition is met. Each option has three components: 1) a policy that gives a probability distribution over actions for any history; 2) a termination condition that assigns a probability to each history (the probability that the option will terminate given that it reached that history); and 3) an initiation set (a set of histories from which the option can be started). Options were defined by Sutton

et al. (1999) for MDPs, where state is used in place of history; our definition generalizes theirs to partially observable settings.

We define an *option history* as a possible sequence of options and observations $\omega^1 o^1 \omega^2 o^2 \ldots \omega^k o^k$ from the beginning of time, where o^i in this context is the last observation during the execution of option ω^i. We will assume that the agent chooses how to act by selecting an option, executing it until completion, selecting another option, and so on.[1] So at any point in time, the agent can view its history in two ways: 1) a high-level view as an option history (through the last completed option); or 2) a low-level view as a primitive history.[2] Note that a particular option history could correspond to any one of several possible primitive histories. Thus the option history provides some abstraction from the primitive history.

A *primitive test* is a sequence of possible future primitive actions and observations $a_1 o_1 \ldots a_k o_k$. An *option test* is similarly defined by replacing actions with options; the observations in the test correspond to the last observation of each option. We define the *prediction* for a primitive test $t = a_1 o_1 \ldots a_k o_k$ from a primitive history $h = a^1 o^1 \ldots a^j o^j$ as the probability of seeing the observations of t given that the actions of t are taken from history h. Formally, this prediction is

$$p(t|h) \equiv \prod_{i=1}^{k} Pr(O^{j+i} = o_i | A^1 = a^1, O^1 = o^1, \ldots A^j = a^j, O^j = o^j, A^{j+1} = a_1, O^{j+1} = o_1, \ldots A^{j+i} = a_i),$$

where A^x and O^x are the random variables for the action and observation of the xth time step. A prediction for an option test $\omega_1 o_1 \ldots \omega_k o_k$ from an option history h^ω is similarly defined as the probability that $o_1, \ldots, o_k$ are the *last* observations, respectively, of the options $\omega_1, \ldots, \omega_k$ taken from option history h^ω.

1.1. The System-dynamics Matrix and Linear PSRs

As shown by Singh et al. (2004), a fundamental construct in defining PSRs is the system dynamics matrix. The size of the PSR model for a system, and hence its learnability, is determined by the rank of this conceptual matrix. Thus, throughout this paper we will define the system dynamics matrix that corresponds to the questions we are interested in answering for that system and define the complexity of our model from the rank of the resulting matrix. In this subsection, we define this matrix $\mathcal{D}$ for primitive tests and histories as in the original PSR work. This matrix (Figure 1) is $\infty \times \infty$ and has one row for each possible history (including the empty or null history h_ϕ) and one column for each possible test.[3] The entry in a particular row and column is the prediction for that column's test from that row's history. We will denote the submatrix of $\mathcal{D}$ for a set of tests T and histories H as $p(T|H)$.

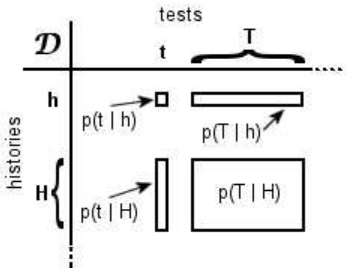

Figure 1. A system-dynamics matrix. Here h is a history, H a set of histories, t a test, and T a set of tests.

For a large class of systems (including POMDPs with a finite number of hidden states), the $\mathcal{D}$ matrix will have some finite rank n. For these systems, one can find a set Q of n linearly independent columns such that all other columns are linearly dependent upon Q. The tests corresponding to these columns (also denoted Q) are called *core tests*. At any history h, the prediction for any test t is a linear function of the predictions for Q. Thus the predictions for Q from any h (written as a row vector $p(Q|h)$) form the state vector of a linear PSR at history h. This state vector is called the *prediction vector*. We use m_t for the column vector of weights such that $\forall h, p(t|h) = p(Q|h)m_t$; such an m_t exists for any t. A *linear PSR* is composed of the predictions for Q from the null history of the system, and the *update parameters* used to update the prediction vector as the agent moves to new histories. The update parameters are the m_t's for the tests $\{a \in A, o \in O, q_i \in Q : ao, aoq_i\}$. The update procedure is to obtain $p(Q|hao)$ from $p(Q|h)$ after taking a and seeing o from h. For any $q_i \in Q$, one can use the existing state vector $p(Q|h)$ and the update parameters m_{ao} and m_{aoq_i} to calculate

$$p(q_i|hao) = \frac{p(aoq_i|h)}{p(ao|h)} = \frac{p(Q|h)m_{aoq_i}}{p(Q|h)m_{ao}}.$$

[1] A primitive action a can be "wrapped" in an option that simply executes a and then terminates, if the agent needs to be able to take a directly.

[2] Our work deals with a two-level hierarchy of actions (primitive and options); it is straightforward to add levels to this hierarchy (i.e. options over options over primitive actions).

[3] The tests (or histories) can be arranged in length-lexicographical ordering to make a countable list.

2. Extending Linear PSRs to Options

Now we return to the issue of constructing a PSR that can make predictions about options. Recall that we want to construct such a model because it may be smaller and easier to learn than a linear PSR that makes primitive predictions for the whole system, and because the predictions about options should be useful for planning. For a given system and set Ω of options, all option predictions can be arranged in an option-level system dynamics matrix $\mathcal{D}^\Omega$ by listing all option histories for the rows and all option tests for the columns. In general, however, this $\mathcal{D}^\Omega$ could have larger rank than the $\mathcal{D}$ matrix for the primitive system.[4] We show this by example, by constructing an option test t^ω whose predictions are linearly independent of all primitive tests (i.e. all columns of $\mathcal{D}$). One can construct $\mathcal{D}^\Omega$ to contain $\mathcal{D}$ as a submatrix by allowing Ω to contain an option for each primitive action that just executes that primitive action and then terminates. Since this $\mathcal{D}^\Omega$ contains the columns of $\mathcal{D}$ and the column for t^ω, which is linearly independent of $\mathcal{D}$, its rank will be larger than $\mathcal{D}$. The primary idea behind the following example is that one can construct t^ω by using an option policy that is a *non-linear* function of the prediction vector.

Example: The POMDP system has 2 states (s_1 and s_2); 4 actions, ($\alpha, \beta, swap$, and $stay$); and 4 observations ($o_\alpha, o_\beta, o_1, o_2$). Action α always produces observation o_α and leaves the state unchanged. Similarly, action β always produces observation o_β and leaves the state unchanged. The $swap$ action changes the state, and $stay$ leaves the state unchanged. The observation after $swap$ or $stay$ depends upon the ending state s': if $s' = s_1$, then the observation is o_1 with probability p and o_2 with probability $1-p$. If $s' = s_2$, the probabilities are reversed (p for o_2 and $1-p$ for o_1). The initial state distribution is $\{0.5, 0.5\}$.

For $p = 0.2$, $rank(\mathcal{D}) = 2$, and one set of core tests for this system is $\{\alpha\, o_\alpha, stay\, o_1\}$. Consider an option ω with a policy that chooses α with probability $(p(stay\, o_1|h))^2$ (a non-linear function of the prediction vector), and chooses β otherwise; after one action, ω terminates. The option test $p(\omega\alpha|h)$ is not a linear function of the prediction vector:

		pred. vec.		
tests →		$\alpha\, o_\alpha$	$stay\, o_1$	$\omega\alpha$
	$\beta\, o_\beta$	1.0	0.5	0.25
histories	$stay\, o_1$	1.0	0.68	0.4624
	$stay\, o_2$	1.0	0.32	0.1024

This rank-3 matrix is a submatrix of $\mathcal{D}^\Omega$, so $rank(\mathcal{D}^\Omega) \geq 3 > rank(\mathcal{D}) = 2$. Note that one can define other options that choose α based upon other non-linear functions of the prediction vector, further increasing the rank of $\mathcal{D}^\Omega$. □

Despite the fact that $rank(\mathcal{D}^\Omega)$ can be more than $rank(\mathcal{D})$ in general, if we limit ourselves to a particular class of options, then $rank(\mathcal{D}^\Omega) \leq rank(\mathcal{D})$ (see Theorem 4 in the Appendix). In fact, $rank(\mathcal{D}^\Omega)$ can be much less than $rank(\mathcal{D})$, as demonstrated empirically in Section 4. This class of options has two properties: 1) policies depend only on the history since the option began executing (which still permits them to be closed-loop); and 2) termination conditions are deterministic functions of the observations since the option began executing. For the remainder of this paper, we will limit our attention to this class of options; examples are presented in Section 4. Since $rank(\mathcal{D}^\Omega) \leq rank(\mathcal{D})$, one can construct a linear PSR to model $\mathcal{D}^\Omega$ just as for $\mathcal{D}$, and the number of core tests needed for the model of $\mathcal{D}^\Omega$ will be no more than for the model of $\mathcal{D}$.

3. Hierarchical PSRs

Building a model of $\mathcal{D}^\Omega$ allows one to make predictions about the outcomes of options, providing an abstract model of the system that can be smaller and easier to learn than a primitive model of the whole system. However, the $\mathcal{D}^\Omega$ model does not allow one to make primitive predictions, which the agent may need in order to learn other options, for instance. We introduce a *hierarchical PSR* (HPSR) that can make both primitive and option-level predictions. We will first describe what an HPSR is and how it can be used, and later describe how one can learn an HPSR.

The HPSR Model: An HPSR is composed of several linear PSRs: one high-level model $\mathcal{M}^\Omega$ that makes predictions about options, and one primitive-level model $\mathcal{M}^{\omega_i}$ for each option $\omega_i \in \Omega$ that makes primitive predictions *only while ω_i is executing*. The agent uses an HPSR in the following way: At the first time step, it chooses some option ω_i to execute, and initializes $\mathcal{M}^\Omega$ and $\mathcal{M}^{\omega_i}$ with their respective initial prediction vectors. At each time step through the termination of ω_i, the agent updates the prediction vector of $\mathcal{M}^{\omega_i}$ according to the standard linear PSR update procedure and uses $\mathcal{M}^{\omega_i}$ to make predictions about any primitive tests. When ω_i terminates (call this time t^*), the prediction vector for $\mathcal{M}^\Omega$ is updated based upon the last observation of ω_i.[5] Also at time t^*, the agent chooses another option ω_j to begin executing. The agent initializes $\mathcal{M}^{\omega_j}$ by asking $\mathcal{M}^{\omega_i}$ to make a prediction about each of the core tests of

[4] While $\mathcal{D}$ contains enough information to compute $\mathcal{D}^\Omega$, the *linear* rank of $\mathcal{D}$ may be less than that of $\mathcal{D}^\Omega$.

[5] Since one only updates the high-level model upon options' termination, it always makes predictions based upon the state at the last option's termination. This abstraction is necessary since it is unclear how to update the prediction for an option test as one takes primitive actions.

$\mathcal{M}^{\omega_j}$; these predictions are then the prediction vector for $\mathcal{M}^{\omega_j}$. Until ω_j terminates, $\mathcal{M}^{\omega_j}$ is responsible for any primitive predictions, and its prediction vector is updated at each time step. This process continues as the agent chooses options to execute.

The HPSR Component Models: Each linear PSR within an HPSR models some system-dynamics matrix, as described in Section 1.1, so we will describe each linear PSR by the system-dynamics matrix which it models. The high-level linear PSR (that makes predictions about options) models the option-level system-dynamics matrix $\mathcal{D}^{\Omega}$. The primitive-level linear PSR for an option $\omega_i \in \Omega$ models a system-dynamics matrix that we will call $\mathcal{D}^{\omega_i}$, which contains predictions for all primitive tests from a subset of primitive histories that we call $H^{\omega_i}_{trunc}$; we now address $H^{\omega_i}_{trunc}$ in more detail.

Because the agent chooses how to act by selecting an option, executing it until completion, selecting another option, and so on, we do not need the HPSR to make primitive predictions from every primitive history, but only those that could be generated by some sequence of options. For any such primitive history h, there is some option $\omega_i \in \Omega$ that was being executed (or just completed) at h. We define H^{ω_i} to be the primitive histories where ω_i was being executed, just finished being executed, or could be initiated.[6] One can divide each $h \in H^{\omega_i}$ into two parts h^1 and h^2 such that h^2 is the history since ω_i began, and h^1 is the history prior to that point.[7] The set of histories $H^{\omega_i}_{trunc}$ for the system-dynamics matrix $\mathcal{D}^{\omega_i}$ is the set of h^2's for each $h \in H^{\omega_i}$.

We want the h^2 row of $\mathcal{D}^{\omega_i}$ to be predictions from when ω_i has generated sequence h^2. However, these predictions could depend upon what has happened before the execution of ω_i. The history before ω_i's execution (h^1) can be captured in the null history (h_ϕ) row of $\mathcal{D}^{\omega_i}$ by setting it equal to the h^1 row of $\mathcal{D}$. The dynamics of the system then constrain the h row of $\mathcal{D}^{\omega_i}$ to be equal to the h^1h row of $\mathcal{D}$, for each $h \in H^{\omega_i}_{trunc}$. But note that the history before ω_i's execution will be different each time ω_i executes, leaving us with the question "Which history should we choose as h^1?" In practice, we choose an empirical average over the histories from which ω_i starts. Thus each row of $\mathcal{D}^{\omega_i}$ is a weighted average of some rows of $\mathcal{D}$.[8]

Note that a model of $\mathcal{D}^{\omega_i}$ will make predictions based upon the *average* history from which ω_i starts. However, during an execution of ω_i that started from some h^1, we want our model to make predictions *given the fact that ω_i started from h^1*. Otherwise, the predictions will not reflect the actual primitive history of the agent. A model for $\mathcal{D}^{\omega_i}$ can – in certain cases which we describe in the Appendix – make predictions that *do* reflect the actual primitive history of the agent. This is done by using an initial prediction vector that reflects the specific history from which ω_i begins; each initial prediction vector (for each different history from which ω_i starts) can be computed online as described above. Note that the model update parameters and core tests are learned from $\mathcal{D}^{\omega_i}$ and are the same each time ω_i executes.

Sizes of Component Models: We now show bounds on the ranks of $\mathcal{D}^{\Omega}$ and each $\mathcal{D}^{\omega_i}$ modeled by the HPSR; these ranks are important because it is often difficult to learn a linear PSR for a high-rank system-dynamics matrix. For $\mathcal{D}^{\Omega}$, Theorem 4 shows that $rank(\mathcal{D}^{\Omega}) \leq rank(\mathcal{D})$. For each $\mathcal{D}^{\omega_i}$, we first note that each row of $\mathcal{D}^{\omega_i}$ is a linear combination of the H^{ω_i} rows of $\mathcal{D}$ (by construction), which implies that $rank(\mathcal{D}^{\omega_i}) \leq rank(\mathcal{D})$. For some options ω_i, we can obtain a tighter upper bound on the rank of $\mathcal{D}^{\omega_i}$ by bounding the rank of the H^{ω_i} rows of $\mathcal{D}$, which will also be a bound for $rank(\mathcal{D}^{\omega_i})$. In the proof below, we will refer to the H^{ω_i} rows of $\mathcal{D}$ as $\mathcal{D}_i$.

Theorem 1. *For a POMDP system with a set of hidden states S, $rank(\mathcal{D}_i) \leq |S_i|$, where $S_i \subseteq S$ is the set of hidden states in which ω_i could be executing.*

Proof. The proof is similar to the bound on $rank(\mathcal{D})$ for POMDPs given by Singh et al. (2004). Let T be the set of all tests and let $p(T|s_i)$ be the row vector of probabilities that each test in T succeeds from initial state $s_i \in S$. Let U be the $|S| \times \infty$ matrix formed by stacking the $p(T|s_i)$ vectors for each $s_i \in S$. Let B be the $\infty \times |S|$ matrix such that the jth row is the POMDP belief state (i.e. distribution over hidden states S) given the jth history of H^{ω_i}. Then $\mathcal{D}_i = BU$, by conditioning upon the hidden state at each history. Note that the number of non-zero columns of B is no more than $|S_i|$. Thus, $rank(\mathcal{D}_i) \leq rank(B) \leq |S_i|$. □

Therefore, if an option ω_i only traverses a subset of the state space, then the number of core tests in our primitive-level model for that option will depend only on the size of that subset, rather than the size of the

[6] Since multiple option histories could generate a primitive history h, a given h could be in H^{ω_i} for multiple i's.

[7] Some h's could be divided in more than one way and still satisfy the condition that ω_i generated h^2; each such division is considered in the construction of $H^{\omega_i}_{trunc}$. Also note that h^1 or h^2 (or both) could be the null history h_ϕ.

[8] Specificially, if h has weight $f(h)$ in the empirical average of histories from which ω_i starts, then the h' row of $\mathcal{D}^{\omega_i}$ is $\sum_h f(h)\mathcal{D}(hh')$, where $\mathcal{D}(hh')$ is the hh' row of $\mathcal{D}$.

whole state space. So one can use options (in conjunction with an HPSR) to restrict the portion of the state space that one wishes to model.

Learning an HPSR: Learning an HPSR consists of learning each of its linear PSR components, each of which models some system-dynamics matrix. Given an estimate $\hat{\mathcal{D}}$ of some system-dynamics matrix $\mathcal{D}$ (or $\mathcal{D}^{\omega_i}$), one can use the algorithm from James and Singh (2004) and Wolfe et al. (2005) to find core tests and calculate update parameters of a linear PSR. The question then is how to estimate predictions (portions of $\hat{\mathcal{D}}$). If one has multiple experience trajectories, one can use the reset algorithm (James & Singh, 2004) to estimate some $p(t|h)$ as the number of sequences that begin with ht divided by the number of sequences that begin with h, divided by a policy correction term $g(t,h)$. To define $g(t,h)$, let $\overline{\pi}(h,a)$ be the agent's average (over the training data) probability of taking action a from history h, and let h_i be the history h followed by the first i time steps of t. Then $g(t,h) = \prod_{i=1}^{n} \overline{\pi}(h_{i-1}, a_i)$ for $t = a_1 o_1 \dots a_n o_n$. If there is only one experience trajectory, one can use the suffix-history algorithm (Wolfe et al., 2005), which treats each suffix of that trajectory as an experience trajectory, and uses the reset algorithm's method to estimate $p(t|h)$ from those suffixes.

To learn an HPSR from a single trajectory of experience, one can learn the model of $\mathcal{D}^{\Omega}$ by using suffix-history on the option-level view of the experience. For each $\mathcal{D}^{\omega_i}$ model, one can use the reset algorithm (James & Singh, 2004) because there are multiple executions of each option in the training sequence. Each time an option executes, it generates some sequence of primitive actions and observations; these sequences are the trajectories used by the reset algorithm.

4. Experiments

We tested the HPSR learning algorithm on two domains, measuring the HPSR's accuracy in making both option and primitive predictions as the amount of training data (measured in time steps) increases. Both training and test sequences were generated by having the agent repeatedly choose from the available options uniformly at random. Our error measure is a mean squared error of one-step predictions: for primitive-level predictions it is $\frac{1}{|O|L} \sum_{j=1}^{L} \sum_{i=1}^{|O|} (p(a^{j+1}o_i|h_j) - \hat{p}(a^{j+1}o_i|h_j))^2$, where h_j is the history after time step j, L is the number of primitive actions taken in the test sequence, $\hat{p}(a^{j+1}o_i|h_j)$ is the estimate computed by the learned model, and $p(a^{j+1}o_i|h_j)$ is the true prediction. This is the same measure used by Wolfe et al. (2005). The error measure for option predictions is similar, replacing a^{j+1} with the $(j+1)$st option executed, replacing h_j with the history through the end of the jth option, and replacing L with the number of options taken in the test sequence. Our testing sequence was 10000 time steps long, during which the entries of each model's prediction vector were clipped as needed to fall in [0,1]; note that this is not clipping the predictions that we measure for accuracy, just the state vector itself. Both suffix-history and reset algorithms take a single parameter which tunes how conservatively they estimate the rank of a submatrix of $\mathcal{D}$. We ran several trials with a broad range of parameters and report the results from the best parameter settings. Finally, because a set of one-step core tests exists for any MDP, we modified the core search algorithm to only look at one-step tests.

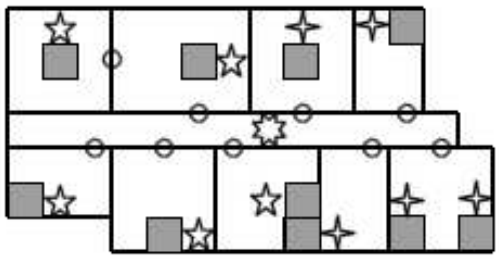

Figure 2. Rooms domain. The grid squares are the size of the gray blocks, which are obstacles. The starred locations are the destinations of options, and the circles denote doors. There are four primitive actions (N, S, E, W), which fail to move the agent with probability 0.1, and the observation is the square in which the agent lands.

Rooms Domain: We tested the HPSR learning algorithm on an MDP grid-world domain with 78 states shown in Figure 2. The 11 starred locations are the states that the options go between. We provided options that went from the hallway star to each other star, and options to go the opposite way. Each of the five-point stars had an option to get to each other five-point star; similarly, the four-point stars had options to reach each other. There were 60 options in all. The number of core tests needed for a $\mathcal{D}^{\omega_i}$ model ranged from 3 to 13, in contrast to the 78 required for a primitive model of the whole system. The option-level model also needs significantly fewer core tests (11). The Markovian property allowed us to estimate the prediction $p(a_1 o_1 \dots a_j o_j | h a^k o^k)$ as 0 if $h a^k o^k a_1 o_1 \dots a_j o_j$ never occurred and $\hat{p}(a_1 o_1 | o^k) \prod_{i=2}^{k} \hat{p}(a_i o_i | o_{i-1})$ otherwise, with $\hat{p}(a_1 o_1 | o^k)$ being the fraction of times that o_1 followed $o^k a_1$ in the training sequence; this is a form of intra-option learning, as we use experience from

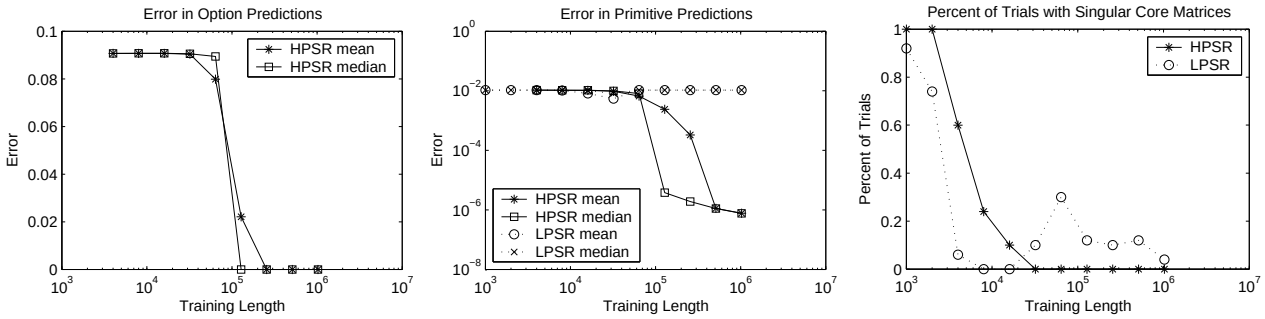

Figure 3. Rooms domain results. Means and medians are taken over those of the 50 trials which did not produce singular core matrices, which prevent the calculation of the update parameters. "LPSR" is a linear PSR for the whole domain.

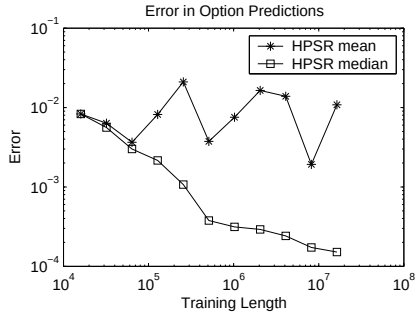

Figure 4. Taxi domain results for 20 trials.

one option to estimate the behavior of other options. The results are shown in Figure 3. The error for the option predictions reaches 0.0, demonstrating how an abstract, option-level model can be easy to learn. For the primitive predictions, we compare the HPSR against a primitive-level linear PSR that models the whole system, learned using the suffix-history algorithm. The linear PSR was comparable to the HPSR for the smaller training lengths, but as the training length increased, its error stayed nearly constant, several orders of magnitude higher than the HPSR's error.

Taxi Domain: We also ran some experiments to learn an HPSR for a modification of the taxi domain from Dietterich (1998) without fuel; this domain is a 25-location grid world with a passenger that can be in the taxi (i.e. the agent) or at one of 4 colored locations. The passenger also has a destination that is one of the four colors. The options were to go from each color to each other color, to pickup the passenger, and to drop off the passenger (14 total options). Upon a successful drop off, the passenger would randomly choose a new destination. This domain has 500 states and would require 500 core tests to model with a single linear PSR; in contrast, the high-level model of the system requires 80 core tests, and each primitive model within an HPSR requires no more than 180 core tests. Learning a linear PSR for the whole system was computationally impractical due to the high number of core tests required. We were, however, able to learn HPSR models of the system; the accuracy of the option predictions are shown in Figure 4; the mean and median primitive prediction error were between 0.000365 and 0.000395 for all but the smallest training length (where they were 0.0004575 and 0.0004446, respectively), and a model was learned on every trial (i.e. no singular core matrices).

In each domain, learning all of the component models of an HPSR was computationally faster than learning a linear PSR for the whole system. However, if many of the options given to the agent traverse a large portion of the system's state space, then the $\mathcal{D}^{\omega_i}$ models could have sizes near that of a model for the whole system, making HPSR learning slower.

5. Related Work

There has been much work on hierarchical reinforcement learning (Barto and Mahadevan (2003) give a good overview), but little work combining PSRs and hierarchical RL. Sutton et al. (2006) used temporal difference networks (a form of PSR) to estimate predictions about options, but they did not show that the options' predictions are actually computable in their architecture. In contrast, we proved that a linear PSR can make accurate predictions about a class of options. James et al. (2005) used the idea of dividing up all primitive histories into several submatrices of $\mathcal{D}$ with smaller rank. In their work, the division was done by the last observation of history, rather than by the last option executing. Finally, the abstract MDP of Hauskrecht et al. (1998) is related to our option-level model in that both provide an abstract model of the system that is potentially smaller than a detailed model of the system; note that one can build an option-level model even in partially observable do-

mains.

6. Conclusion

We have shown that a linear PSR can make predictions about a class of options. We bounded the number of core tests needed for such a model and gave examples where modeling the system at the option level provides a smaller, abstract model of the system. We showed how a set of options can implicitly divide a state space for modeling at the primitive level, using this principle as the basis for a hierarchical PSR that can predict the outcomes of both options and primitive actions.

Appendix

Bounding $rank(\mathcal{D}^\Omega)$: In this section, we show that $rank(\mathcal{D}^\Omega) \leq rank(\mathcal{D})$ when the options of Ω have 1) policies that depend only on the history since the option began executing (which still permits them to be closed-loop); and 2) termination conditions that are deterministic functions of the observations since the option began executing. Note that, although we did not use this fact in our experiments, Lemma 3 proves that one can ask more general questions about options than just about the last observation, and the answer will still be a linear function of the prediction vector of a primitive-level model. The question need only be a function of the primitive observation sequence seen during the option.

Lemma 2. *Let T be a set of tests (option tests or primitive tests) and let H be the set of all primitive histories. Let h^ω be any option history. Then the vector $p(T|h^\omega) = v_{h^\omega}^T p(T|H)$ for some vector v_{h^ω}.*

Proof. Note that $\mathcal{D} = p(T|H)$ when T is the set of all primitive histories. Since $p(T|h^\omega) = \sum_{h \in H} Pr(h^\omega \equiv h) p(T|h)$, the ith entry of v_{h^ω} is the probability that h is the primitive actions/observations of h^ω. □

Lemma 3. *Let T be the set of all primitive tests and H be all primitive histories, and let t^ω be an option test. Then $p(t^\omega|H) = p(T|H)v_{t^\omega}$ for some vector v_{t^ω}.*

Proof. Let ω_1 be an option that executes the options of t^ω in order. We define an indicator variable $\psi_1 : O^* \to \{0, 1\}$ that returns "success" or "failure" based upon the observation sequence seen during ω_1. This function ψ_1 can be defined such that it is 1 if and only if the last observations of the respective options of t^ω are equal to the observations of t^ω (due to the deterministic termination conditions of the options). Then $p(\omega_1\psi_1|h) \equiv Pr(\psi_1|h, \omega_1)$ equals $p(t^\omega|h)$. (All of the probabilities in this proof are conditioned upon the policy of ω_1, which we do not write for brevity.)

Let $\mathbf{A}$ and $\mathbf{O}$ be random vectors for the primitive actions and observations, respectively, seen during the execution of ω_1 from h.[9] Then $p(\omega_1\psi_1|h) =$

$$\sum_{\mathbf{a} \in A^*} \sum_{\mathbf{o} \in O^*} \psi_1(\mathbf{o}) Pr(\mathbf{A} = \mathbf{a}, \mathbf{O} = \mathbf{o}|h)$$

by conditioning on the random variables $\mathbf{A}, \mathbf{O}$. We will use $\mathbf{a}$ as shorthand for $\mathbf{A} = \mathbf{a}$, and a_i as shorthand for $\mathbf{A}_i = a_i$ (where $\mathbf{A}_i$ is the ith element of $\mathbf{A}$); similar notation will be used for observations. We will now focus on a particular $\mathbf{a}, \mathbf{o}$, using n as their lengths. We can break apart $Pr(\mathbf{a}, \mathbf{o}|h)$ into the probability of ω_1 generating $\mathbf{a}, \mathbf{o}$ times the probability that ω_1 terminates after $\mathbf{a}, \mathbf{o}$, given that $\mathbf{a}, \mathbf{o}$ was generated. We use $\beta(\mathbf{a}, \mathbf{o})$ to denote the latter, and the former is

$$\begin{aligned} & Pr(a_1|h) Pr(o_1|ha_1) Pr(a_2|ha_1o_1) \\ & \quad \cdots Pr(o_n|ha_1o_1 \ldots a_{n-1}o_{n-1}a_n) \\ = \; & (Pr(a_1|h) \cdots Pr(a_n|ha_1 \ldots a_{n-1}o_{n-1})) \\ & \quad \cdot (Pr(o_1|ha_1) \cdots Pr(o_n|ha_1 \ldots o_{n-1}a_n)) \\ = \; & Pr(a_1|h) Pr(a_2|ha_1o_1) \cdots \\ & \quad Pr(a_n|ha_1o_1 \ldots a_{n-1}o_{n-1}) p(t|h) \end{aligned}$$

for t corresponding to $\mathbf{a}, \mathbf{o}$. Each of the factors $Pr(a_i|h \ldots)$ is equivalent to a corresponding $Pr(a_i|\ldots)$ because the policy within ω_1 is independent of history given the actions and observations within ω_1. If we define $w_{\mathbf{a},\mathbf{o}} = Pr(a_1) Pr(a_2|a_1o_1) \cdots Pr(a_n|a_1o_1 \ldots a_{n-1}o_{n-1})$, then $p(\omega_1\psi_1|h) =$

$$\begin{aligned} & \sum_{\mathbf{a} \in A^*} \sum_{\mathbf{o} \in O^*} \psi_1(\mathbf{o}) Pr(\mathbf{A} = \mathbf{a}, \mathbf{O} = \mathbf{o}|h) \\ = \; & \sum_{\mathbf{a} \in A^*} \sum_{\mathbf{o} \in O^*} \psi_1(\mathbf{o}) \beta(\mathbf{a}, \mathbf{o}) w_{\mathbf{a},\mathbf{o}} p(\mathbf{a}, \mathbf{o}|h) \\ = \; & p(T|h) v_{t^\omega}. \end{aligned}$$

For each test $\mathbf{a}, \mathbf{o}$ in T, the corresponding entry of v_{t^ω} is $\psi_1(\mathbf{o})\beta(\mathbf{a}, \mathbf{o})w_{\mathbf{a},\mathbf{o}}$. Since this equation holds for any history, $p(\omega_1\psi_1|H) = p(t^\omega|H) = p(T|H)v_{t^\omega}$. □

Theorem 4. *For system-dynamics matrices $\mathcal{D}$ and $\mathcal{D}^\Omega$ defined as above, $rank(\mathcal{D}^\Omega) \leq rank(\mathcal{D})$.*

Proof. To get the matrix $\mathcal{D}^\Omega$ from $\mathcal{D}$, we apply two linear transformations, which cannot increase the rank. The first transformation, which is linear because of Lemma 3, replaces the primitive tests of $\mathcal{D}$ with all option tests. The second transformation, which is linear because of Lemma 2, replaces the primitive histories with all option histories. □

[9] We can use random-length random vectors because there is a mapping from action (or observation) sequences to integers (e.g. length-lex ordering).

Changing the Initial Condition: In this section, we note cases where one can learn a model for a system with one initial condition and use that to model the system under different initial conditions, changing only the initial prediction vector. The following lemma shows that if one builds a linear PSR of an MDP from a distribution over initial states, then one can use that PSR to model the system that starts in any one of those states, as long as that starting state is reachable (under the agent's policy) after the initial time step.

Lemma 5. *For an MDP system with states S, let b_0 be some initial distribution over S, and let b_0' be a different initial distribution such that $\forall s_i \in S, b_0'(s_i) > 0 \Rightarrow (b_0(s_i) > 0$ and s_i is reachable at some time $t > 0)$. Then learning a linear PSR under initial condition b_0 will give valid core tests and update parameters for the system under initial condition b_0'.*

Proof. We let $\mathcal{D}$ and $\mathcal{D}'$ be the system-dynamics matrices for the initial conditions b_0 and b_0', respectively. For an MDP system, changing the initial condition does not change any row of $\mathcal{D}$ other than the null history row, due to the Markovian property, but it may render some histories unreachable that were previously reachable, or vice versa. Our condition $\forall s_i \in S, b_0'(s_i) > 0 \Rightarrow b_0(s_i) > 0$ ensures that any state reachable under b_0' is also reachable under b_0. Thus, if we ignore the null history row, $\mathcal{D}'$ is a submatrix of $\mathcal{D}$, so all columns of $\mathcal{D}'$ are linearly dependent upon those of Q (core tests for $\mathcal{D}$), in all rows but the null history row. We now show that the null history row of $\mathcal{D}'$ is a linear combination of the other rows of $\mathcal{D}'$, which implies that the columns of $\mathcal{D}'$ are linearly dependent upon those of Q, making Q valid core tests for $\mathcal{D}'$. Let T be the set of all primitive tests and $p(T|s_i)$ be the vector of predictions for each primitive test from state s_i. For an initial condition b_0', the null history row of $\mathcal{D}'$ is $\sum_{s_i \in S} b_0'(s_i) p(T|s_i)$, by conditioning upon the starting state. Our condition $[\forall s_i \in S, b_0'(s_i) > 0 \Rightarrow s_i$ is reachable at some time $t > 0]$ means that each $p(T|s_i)$ with non-zero weight in the previous sum is also a row in $\mathcal{D}'$. Thus, the null history row of $\mathcal{D}'$ is linear combination of other rows of $\mathcal{D}'$. Finally, since the weights of the linear relationships between the columns of $\mathcal{D}'$ are the same as those for $\mathcal{D}$, the update parameters from $\mathcal{D}$ are valid in $\mathcal{D}'$. □

Two other cases where one can use data from one initial distribution to model the system under other initial distributions were shown by Wolfe et al. (2005) and are restated here:

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 $\mathcal{D}$.*

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 $\mathcal{D}$.*

Acknowledgements: This work is supported by the National Science Foundation under Grant Number IIS-0413004. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

Barto, A., & Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, *13*, 341–379.

Dietterich, T. G. (1998). The MAXQ method for hierarchical reinforcement learning. *Proceedings of ICML 1998* (pp. 118–126).

Hauskrecht, M., Meuleau, N., Kaelbling, L. P., Dean, T., & Boutilier, C. (1998). Hierarchical solution of markov decision processes using macro-actions. *UAI* (pp. 220–229).

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).

James, M. R., Wolfe, B., & Singh, S. (2005). Combining memory and landmarks with predictive state representations. *Proceedings of IJCAI 2005* (pp. 734–739).

Littman, M. L., Sutton, R. S., & Singh, S. (2002). Predictive representations of state. *Advances in Neural Information Processing Systems 14* (pp. 1555–1561).

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

Sutton, R. S., Precup, D., & Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, *112*, 181–211.

Sutton, R. S., Rafols, E. J., & Koop, A. (2006). Temporal abstraction in temporal-difference networks. *Advances in Neural Information Processing Systems 18*. To appear.

Wolfe, B., James, M. R., & Singh, S. (2005). Learning predictive state representations in dynamical systems without reset. *Proceedings of ICML 2005* (pp. 985–992).