Modeling Dynamical Systems with Structured Predictive State Representations

by

Britton D. Wolfe

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2009

To my wonderful wife, Abby

ACKNOWLEDGEMENTS

"Now to God, who is able to do immeasurably more than all we ask or imagine, according to his power that is at work within us, to him be glory . . . for ever and ever!"

— The Bible, Ephesians 3:20,21

First and foremost, I wish to acknowledge the Lord Jesus Christ, who has blessed me with the ability to complete this work. It is only through his gracious provision that this has been possible.

He has worked through many people to bless me and help me both before and during my time at the University of Michigan. My wife Abby has been a tremendous source of help and encouragement, doing everything from listening to me practice conference talks, proof-reading my papers, picking me up from work when I missed the last bus, or just offering an encouraging word. Nothing I can write could possibly capture how grateful I am to her.

I also want to acknowledge my parents, who have sacrificed so much for me in the years leading up to my time at Michigan. To them I owe much of who I am and what I do. My mom's encouragement has been never-ending, no matter how overwhelming my research seemed. My dad's example of hard work and perseverance has been a great inspiration to me throughout my life, especially during my time in graduate school.

The rest of my family — Tanya, Mike, Grandma, aunts and uncles, Ed, Leann, Chel, and Ian — has also been a great blessing to me in many different ways.

I truly appreciate all of the guidance that my advisor Satinder has given me these past years, and I am grateful for his patience with me. My fellow computer science graduate students have been great friends and colleagues these past years: thank you to Bart, Mike James, Matt, Vishal, David, Julie, Mark, Erik, Nick, Jon, Dave, Mike Maxim, and many others.

Finally, I want to thank my local spiritual family for their prayers, friendship, and Christ-centered perspective. Thank you to Chuck, Dianne, Ashley, Heather, Carson, Gene, Jon, Georgia, Andrew, Shinobu, Mike, the Hibblers, the Hammonds, the Sanders, the Bucks, the Chapmans, the Sowashes, the Dunnings, the Martellis, and the many others who have been a blessing to me.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF APPENDICES

CHAPTER I

Introduction and Background

An agent interacts with its environment by taking actions and receiving observations from the environment. One can view the environment as a controlled dynamical system, and an agent can use a model of the system to predict what it will observe if it takes a sequence of actions. The agent can potentially use these predictions to determine good courses of action for many different tasks, due to the generality of the model. This flexibility is one advantage that model-based agents have over agents that focus on learning how to achieve one particular task.

Given that one wants to build a model of a dynamical system, one must decide what type of model to build. Before describing some possible model types, I will formally describe dynamical systems themselves and the general framework for a model of a dynamical system (Section 1.1). The remainder of this chapter goes on to describe some existing classes of models, including motivation for using predictive state representations (PSRs) to model dynamical systems.

This chapter provides the necessary background and motivation for my contributions, which I describe in the following chapters. These contributions center around the goal of developing new PSR models that scale well to large, complex dynamical systems. Section 2.1 motivates the use of PSR models through a model-learning algorithm that outperforms traditional methods for small systems. However, the type of PSR model that this algorithm learns is not suited for large systems (Section 3.1). Therefore, I developed new classes of PSRs that are suited for large systems and algorithms for learning the PSR parameters from an agent's experience in the system (Chapters IV, V, and VI).

1.1 Dynamical Systems

My work focuses on discrete-time dynamical systems with a finite set $\mathcal{A}$ of discrete actions and a finite set $\mathcal{O}$ of discrete observations. This section formally describes dynamical systems and a general view of state-based models of dynamical systems. These descriptions provide the background for describing specific models, including the predictive state representations that are the focus of my work.

An agent in a dynamical system proceeds in the following fashion: at each time step $\tau \in \{1, 2, 3, \ldots\}$, the agent chooses some action a_τ to take and then sees some observation (vector) o_τ. For a given sequence of actions $a_1 \ldots a_\tau$, the agent may see any of several sequences of observation values $o_1 \ldots o_\tau$; the dynamical system stochastically determines the particular sequence that the agent actually sees. While models could be used in many ways, the models of dynamical systems that I develop in this dissertation predict the probabilities of seeing different observation sequences conditioned upon taking some actions.

Of course, the probability of seeing an observation sequence will generally be different depending upon what has already occurred in the system. If τ time steps have already elapsed in the system, then at each one of those time steps, the agent took some action and received some observation from the system. The sequence $a_1 o_1 a_2 o_2 a_3 o_3 \ldots a_\tau o_\tau$ of these actions and observations is called the *history* at time

τ. The probabilities of seeing future observation sequences depend upon *all* of history, in general. As time progresses and the agent continues to take actions and see observations, the length of history will grow without bound. Thus, an agent cannot literally store all of its history and use that information to calculate the probabilities of future observations. Instead, an agent can use a model to summarize the information from history that is needed to predict the future observations. The general form of such a model is the topic of the next section.

1.1.1 The General Form of a Model

This section provides a unifying view of state-based models of dynamical systems. As discussed, a viable model of a system cannot store the entire history (which grows without bound) but must maintain a summary of history that allows it to make predictions as if it did have access to the full history. Such a summary is called *state* because the future is conditionally independent of history given the state. The particular form that a model uses to represent state is one aspect that defines a class of models. Section 1.2 discusses particular state representations; this section is fully general with respect to state representation.

One can illustrate the progression of a dynamical system model over time with a *directed graphical model*. In a directed graphical model, each node in the graph represents a random variable, and the edges often correspond to a notion of causality or influence: the distribution of a random variable is influenced by its parents in the graph. (More formally, a random variable in the graph is conditionally independent of its non-descendants given its parents.)

Figure 1.1 is a graphical model representation of the progression of a general, state-based model of a dynamical system, where time moves from left to right. There are three types of variables in this graph, with one variable of each type for each time

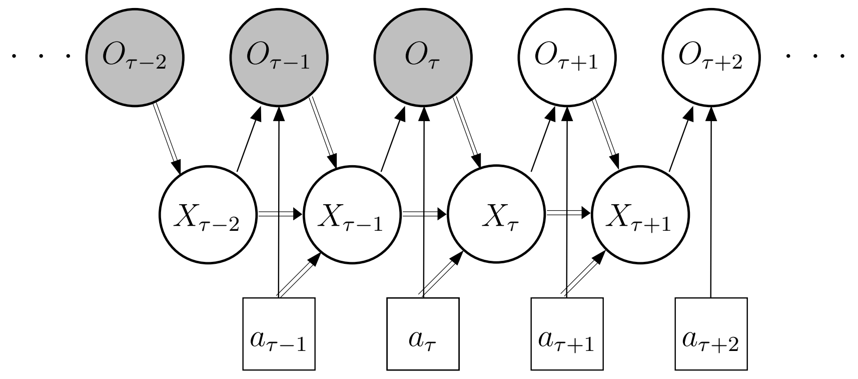


Figure 1.1: A dynamical system at time τ with generic model states X_i, actions a_i, and observations O_i. (The X_i variables are not POMDP states, which are typically depicted in a graphical model of a dynamical system (cf. Figure 1.3). Rather, for a POMDP model, the X_i variables would be belief states.) The determinism of the state update function is indicated by the double arrows. The distribution of the actions is not part of the dynamical system itself, so the action nodes are shown as squares.

step, starting from the first time step and going out through the infinite future.

The O_i variables are random variables for the observations at each time step i. In Figure 1.1, the current time step is τ, so the observations through time τ have been observed (in contrast with the future observations, which are yet to be observed); this is indicated by the shading of O_i for $i \leq \tau$.

The a_i variables are the actions at each time step i. Note that these actions are represented as squares in the figure, to indicate that they are not random variables. However, they do influence the distribution of other nodes in the graph. The actions are not modeled as random variables because the agent determines their distribution, not the dynamical system itself. Thus, a model of the dynamical system should not specify a distribution for the actions, but should be able to account for any action sequence the agent chooses to take.

Finally, the X_i variables are the *model states* at each time step i. In order for a model's state representation to be of practical use, the model must be able to update its state to reflect the current history. That is, the model must always be able to compute $X_{\tau+1}$ from X_τ, $a_{\tau+1}$, and $o_{\tau+1}$. (Since the motivation behind a state-based model is to avoid storing the entire history, the function to compute the new state $X_{\tau+1}$ cannot take history as an input, but only the last state X_τ and the most recent action $a_{\tau+1}$ and observation $o_{\tau+1}$.) I will use the term *state update function* to refer to the function that a model uses to compute $X_{\tau+1}$ from X_τ, $a_{\tau+1}$, and $o_{\tau+1}$.

The state X_i at time i has the following important property: *the future observations $O_{i+1}, \ldots$ are conditionally independent of the historical observations $O_1, \ldots, O_i$ given the state X_i.* This can be visually verified in the graphical model by noting that for any $j, k > 0$, any undirected path between O_{i+k} and O_{i-j} must pass through X_i. This property is necessary because the model does not use history to make predictions about the future observations; instead of history, it uses its state to make predictions about the future. In order for the model to make the same predictions about the future as one would make if the full history were (hypothetically) available, the future observations must be independent of history given the model state.

There is another important property to note about the X_i variables: the value of X_{i+1} is a *deterministic function* of the values of X_i, a_{i+1}, and O_{i+1}. That is, given X_i, a_{i+1}, and o_{i+1}, there is a single next state X_{i+1}. This determinism is denoted by the double arrows in Figure 1.1.

This deterministic update assumption may seem restrictive, but it turns out to be quite natural when one recalls that the state $X_{\tau+1}$ is a summary of the history through time $\tau+1$. When the agent has observed the history through $\tau+1$, it should have exactly one summary of that history. The deterministic update assumption simply captures the fact that the model has one state for any given history.

The deterministic update assumption is not a restriction on the class of dynamical systems that one can model, but rather on the state representation that one uses to model the system. Put another way, the dynamical system can be stochastic, but the model will update its state in a deterministic way, given the latest action and observation. Figure 1.1 illustrates this: the distribution of $O_{\tau+1}$ given X_τ and $a_{\tau+1}$ captures the stochasticity of the dynamical system, while the distribution of $X_{\tau+1}$ given X_τ, $a_{\tau+1}$, and $O_{\tau+1}$ captures the deterministic update mechanism of the model.

In summary, Figure 1.1 illustrates the general framework for a state-based model of a dynamical system. Such a model consists of the following:

- the initial state of the system (i.e., X_0)
- parameters for updating the state as time progresses (i.e., as the agent takes actions and receives observations)
- parameters for computing predictions about the future from the current state.

To define a specific class of models, one selects a state representation, a functional form for updating state, and a functional form for computing predictions. (The functions for updating state and for computing predictions may be related, so there may be some overlap between the parameters of these functions.)

1.1.2 The System-dynamics Matrix

Before describing some example model classes, I will present a fully general representation of a dynamical system called the *system-dynamics matrix*, which was introduced by Singh, James, and Rudary (2004). This representation will aid in the explanation of PSRs, and it will be used throughout this work to characterize the complexity of different dynamical systems.

The system-dynamics matrix is a theoretical construct that contains all predictions of the form *"What is the probability of seeing* $o_{\tau+1}, o_{\tau+2}, \ldots o_{\tau+k}$ *if I take actions* $a_{\tau+1}, a_{\tau+2}, \ldots a_{\tau+k}$*,* conditioned upon *the fact that my current experience through* *time τ is $a_1o_1a_2o_2\ldots a_\tau o_\tau$?"* A model of a dynamical system should be able to make any prediction of this form. These predictions are the numbers that comprise the system-dynamics matrix, denoted as $\mathcal{D}$. Littman, Sutton, and Singh (2001) introduced some definitions to precisely define these predictions.

Definition 1.1. The agent's experience $a_1o_1a_2o_2\ldots a_\tau o_\tau$ through the current time step τ is called the *history.* I will commonly use the variable h to denote an arbitrary history.

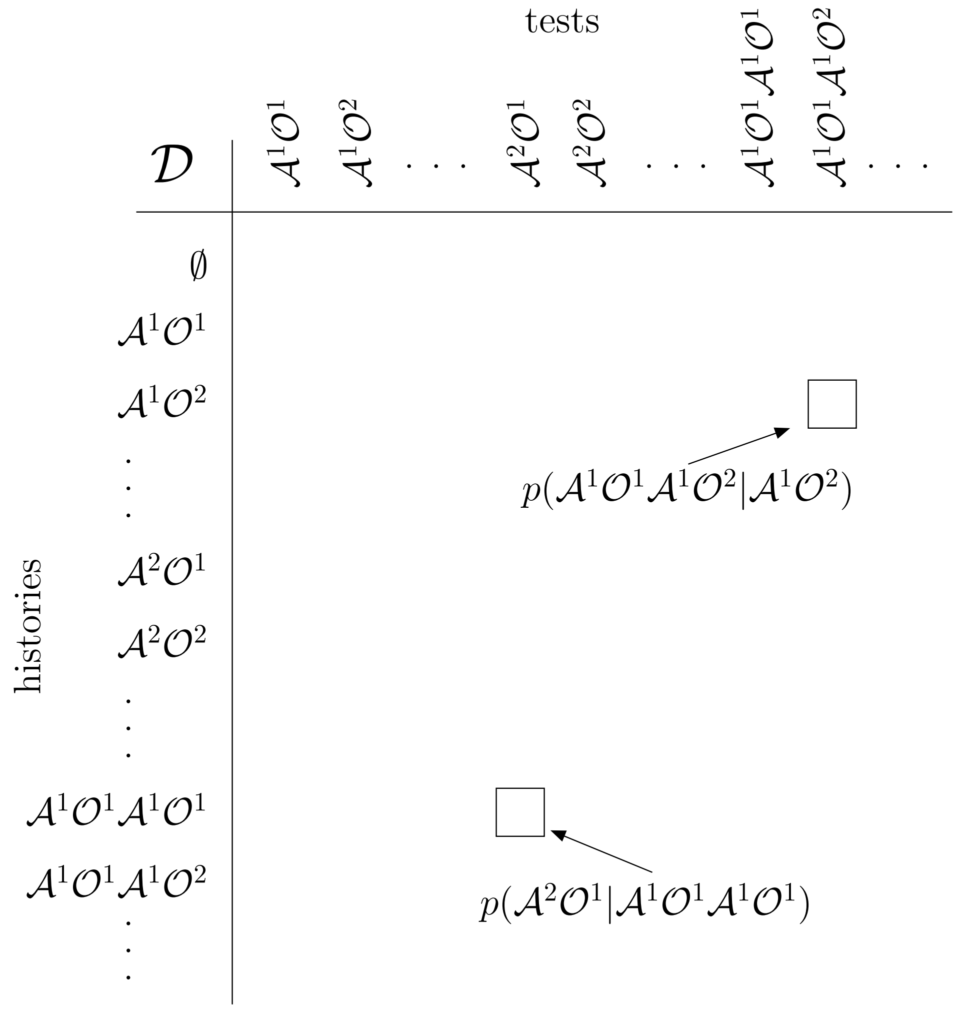
Definition 1.2. A *test* is a hypothetical sequence of *future* actions and observations $a_{\tau+1}o_{\tau+1}a_{\tau+2}o_{\tau+2}\ldots a_{\tau+k}o_{\tau+k}$. I will commonly use the variable t to denote an arbitrary test.

Definition 1.3. The *prediction* for a test t from a history h is the likelihood of seeing the observations of t when the agent takes the actions of t from the history h. Letting $h = a_1o_1a_2o_2\ldots a_\tau o_\tau$ and $t = a_{\tau+1}o_{\tau+1}a_{\tau+2}o_{\tau+2}\ldots a_{\tau+k}o_{\tau+k}$, the formal definition of a prediction is

$$p(t|h)\overset{\text{def}}{=}\prod_{i=\tau+1}^{\tau+k} Pr(o_i|a_1o_1a_2o_2\ldots a_{i-1}o_{i-1}a_i). \tag{1.1}$$

It is worth pointing out that the notation for predictions includes the actions of the test on the left-hand side of the bar (e.g., $p(a_2o_2|a_1o_1)$), but in the conditional probability notation the actions are listed on the right-hand side of the bar (e.g., $Pr(o_2|a_1o_1a_2)$).

The matrix $\mathcal{D}$ has one row for each possible history and one column for each possible test (Figure 1.2). Without loss of generality, the histories and tests are arranged in length-lexicographical ordering. The correspondence between tests and columns means that one can use tests to index columns of $\mathcal{D}$. Similarly, one can use



tests
histories
$\mathcal{D}$
$\mathcal{A}^1\mathcal{O}^1$
$\mathcal{A}^1\mathcal{O}^2$
$\mathcal{A}^2\mathcal{O}^1$
$\mathcal{A}^2\mathcal{O}^2$
$\mathcal{A}^1\mathcal{O}^1\mathcal{A}^1\mathcal{O}^1$
$\mathcal{A}^1\mathcal{O}^1\mathcal{A}^1\mathcal{O}^2$
$\emptyset$
$p(\mathcal{A}^1\mathcal{O}^1\mathcal{A}^1\mathcal{O}^2|\mathcal{A}^1\mathcal{O}^2)$
$p(\mathcal{A}^2\mathcal{O}^1|\mathcal{A}^1\mathcal{O}^1\mathcal{A}^1\mathcal{O}^1)$

Figure 1.2: The system-dynamics matrix $\mathcal{D}$. I use $\mathcal{A}^i$ to denote the i^{th} element in the set of possible actions $\mathcal{A}$, and I use $\mathcal{O}^i$ to denote the i^{th} element in the set of possible observations $\mathcal{O}$. I list the specific predictions corresponding to several entries in the matrix as examples.

histories to index rows of $\mathcal{D}$. The entry of $\mathcal{D}$ in row h and column t is defined to be the prediction $p(t|h)$.

It is worth noting that there is a one-to-one correspondence between dynamical systems and system-dynamics matrices: any $\mathcal{D}$ that respects the axioms of probability will completely specify a dynamical system, and there exists a unique $\mathcal{D}$ for any dynamical system.

As stated above, a model of the system given by $\mathcal{D}$ must be able to compute any entry of $\mathcal{D}$. Since $\mathcal{D}$ has an infinite number of rows and an infinite number of columns, the only hope for a finite model is if $\mathcal{D}$ contains redundant information. Fortunately, this is indeed the case. For instance, the row of $\mathcal{D}$ corresponding to the empty history is a complete specification of the dynamical system: it specifies the joint distribution over the next k observations for any $k > 0$ and any action sequence of length k. Any other entry in $\mathcal{D}$ is determined by this first row. Thus, a model needs to capture only the information in the empty-history row of $\mathcal{D}$. However, this row still has an infinite number of columns. Nevertheless, it turns out that for many systems — including finite-state partially observable Markov decision processes (POMDPs) — there is a *finite* portion of $\mathcal{D}$ that completely determines the remainder of $\mathcal{D}$, which permits a finite model of $\mathcal{D}$. I address this topic in further detail in Section 2.2.

Sub-matrices of $\mathcal{D}$ form an integral part of many of the contributions presented throughout this dissertation. I will use the following notation to describe sub-matrices of $\mathcal{D}$. For a set of tests $T = \{t^1, t^2, \ldots t^k\}$ and a set of histories $H = \{h^1, h^2, \ldots h^j\}$, $p(T|H)$ is the matrix of predictions for each test in T from each history in H where, as in $\mathcal{D}$, the rows correspond to histories and columns to tests:

(1.2)
$$p(T|H) \stackrel{\text{def}}{=} \begin{bmatrix} p(t^1|h^1) & p(t^2|h^1) & \dots & p(t^k|h^1) \\ p(t^1|h^2) & p(t^2|h^2) & \dots & p(t^k|h^2) \\ \vdots & \vdots & \ddots & \vdots \\ p(t^1|h^j) & p(t^2|h^j) & \dots & p(t^k|h^j) \end{bmatrix}.$$

When T consists of a single test, this matrix reduces to a vector:

(1.3)
$$p(t|H) \stackrel{\text{def}}{=} \begin{bmatrix} p(t|h^1) \\ p(t|h^2) \\ \vdots \\ p(t|h^k) \end{bmatrix}.$$

When H is a single history h, $p(T|H) = p(T|h)$ is also a vector. In keeping with the standard practice in linear algebra, vectors are column vectors unless otherwise noted. Thus, $p(T|h)$ will denote a *column* vector, even though it corresponds to part of a row in $\mathcal{D}$:

(1.4)
$$p(T|h) \stackrel{\text{def}}{=} \begin{bmatrix} p(t^1|h) \\ p(t^2|h) \\ \vdots \\ p(t^k|h) \end{bmatrix}.$$

Hereafter, when talking about a test $t = a_{\tau+1}o_{\tau+1} \dots a_{\tau+k}o_{\tau+k}$, I will often drop the "$\tau+$" from the subscripts for brevity when there is no risk of confusion with history (i.e., the first action of the test would be denoted by a_1, etc.).

1.2 Model Classes

The system-dynamics matrix is a theoretical construct that is valid for any discrete-time dynamical system. However, it is not a useful computational model of a system because it has infinite size. A model of the dynamical system must capture all of the information of $\mathcal{D}$ with a finite number of parameters. This section describes several popular classes of models for discrete-time dynamical systems. For each of these model classes, I describe its state representation, parameterization, procedure for updating state, and procedure for making predictions about the future. These model classes can be divided into two sets — predictive and latent state representations — corresponding to the following two subsections. The final subsection compares the two sets of models, motivating the use of predictive state representations, which are the focus of subsequent chapters.

1.2.1 Predictive State Representations

A *predictive state representation* (PSR) is any model that represents state as a vector of statistics about *future* actions and observations. The state vector of a PSR is called a *prediction vector* because it is composed of predictions about the future.

Linear PSRs

Linear PSRs are a particular model class that uses a predictive state representation. They were first introduced by Littman et al. (2001). Linear PSRs are based upon linear dependencies among columns of the system-dynamics matrix $\mathcal{D}$. Despite the fact that $\mathcal{D}$ has an infinite number of rows and an infinite number of columns, it has a finite rank n for a broad class of systems, including partially observable Markov decision processes (described in Section 1.2.2) with a finite number of states. For any system where $\mathcal{D}$ has finite rank n, one can find a set Q of n linearly independent columns of $\mathcal{D}$ such that all other columns are linearly dependent upon Q.[1] The tests corresponding to these columns (also denoted Q) are called *minimal linear core tests.*

[1]In linear algebra terms, Q forms a minimal basis of the column space of $\mathcal{D}$.

Definition 1.4. A set of tests T are *linear core tests* if and only if there exists a history-independent vector m_t for any test t such that $p^\top(T|h)m_t = p(t|h)$ for all histories h. If $|T| = \text{rank}(\mathcal{D})$, then T is a set of *minimal linear core tests.*

Hereafter, I will drop the "linear" qualifier on core tests.

The prediction vector (i.e., the state vector) of a linear PSR at history h is the vector of predictions $p(Q|h)$ for some set of minimal core tests Q. This vector of predictions constitutes state because any prediction about the future can be made as a history-independent function of the predictions for Q; specifically, $p(t|h) = p^\top(Q|h)m_t$ for any test t.

Furthermore, the linear PSR can efficiently update its state as time progresses. The update procedure computes the new state $p(Q|hao)$ from the old state $p(Q|h)$ after taking the action a and seeing the observation o from history h. For any $q_i \in Q$, one can calculate

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

The parameters needed to perform this computation for any q_i, a, o are the m_t's for each one-step test (ao) and each one-step extension (aoq_i) of each core test $q_i \in Q$. These m_t's are called the *update parameters* of the linear PSR. A *linear PSR* consists of these update parameters and the *initial prediction vector* $p(Q|\emptyset)$.

To write the state update procedure concisely, one can arrange the m_{aoq_i} for all i into a matrix M_{ao}:

$$M_{ao} \overset{\text{def}}{=} \begin{bmatrix} | & | & & | \\ m_{aoq_1} & m_{aoq_2} & \cdots & m_{aoq_n} \\ | & | & & | \end{bmatrix}.$$

Using this notation, the state update is

$$p^\top(Q|hao) = \frac{p^\top(aoQ|h)}{p(ao|h)} = \frac{p^\top(Q|h)M_{ao}}{p^\top(Q|h)m_{ao}} \tag{1.5}$$

where $p^\top(aoQ|h)$ denotes the row vector $[p(aoq_1|h) \ \ \ldots \ \ p(aoq_n|h)]$.

Thus, the parameters of a linear PSR model will include at least the update parameters and the initial prediction vector. It turns out that these are the only parameters needed for a complete linear PSR model. To verify this, recall that a complete model must be able to compute any prediction $p(t|h)$. Since one can compute $p(t|h)$ from the initial prediction vector and the update parameters (Littman et al., 2001), they constitute a complete model. To see how this is possible, note that one can compute $p(t|h)$ as $p^\top(Q|h)m_t$. This prediction vector $p(Q|h)$ can be computed by starting with $p(Q|\emptyset)$ and performing the state update procedure for each time step of h, and the required m_t for an arbitrary $t = a_1o_1a_2o_2\ldots a_ko_k$ can be computed as

$$m_t = M_{a_1o_1}M_{a_2o_2}\ldots M_{a_{k-1}o_{k-1}}m_{a_ko_k}. \tag{1.6}$$

One can verify that this satisfies the definition of m_t (i.e., $p(t|h) = p^\top(Q|h)m_t$ for any h) by using the fact that $p^\top(Q|h)M_{ao} = p(ao|h)p^\top(Q|hao)$ (Equation 1.5):

$$\begin{array}{cccccc}
p^\top(Q|h) & m_t & & & & \\
= p^\top(Q|h) & M_{a_1o_1} & M_{a_2o_2} & \ldots & M_{a_{k-1}o_{k-1}} & m_{a_ko_k} \\
= p(a_1o_1|h) & p^\top(Q|ha_1o_1) & M_{a_2o_2} & \ldots & M_{a_{k-1}o_{k-1}} & m_{a_ko_k} \\
= p(a_1o_1|h) & p(a_2o_2|ha_1o_1) & p^\top(Q|ha_1o_1a_2o_2) & \ldots & M_{a_{k-1}o_{k-1}} & m_{a_ko_k} \\
 & \vdots & & & \vdots & \\
= p(a_1o_1|h) & p(a_2o_2|ha_1o_1) & \ldots & \ldots & p^\top(Q|ha_1o_1\ldots a_{k-1}o_{k-1}) & m_{a_ko_k} \\
= p(a_1o_1|h) & p(a_2o_2|ha_1o_1) & \ldots & \ldots & p(a_ko_k|ha_1o_1\ldots a_{k-1}o_{k-1}) & \\
= p(t|h) & & & & &
\end{array}$$

where the last step follows by the definition of $p(t|h)$ (Equation 1.1).

Therefore, a linear PSR consists of the update parameters and the initial prediction vector. Together, these can be used to compute any prediction $p(t|h)$, as is required of a complete model.

There are some constraints on the parameters that must be satisfied in order for the predictions made by the linear PSR to satisfy the axioms of probability (e.g., certain predictions must sum to 1.0, all predictions must be non-negative, etc.). These constraints are listed in Appendix B.

Other Predictive State Models

In addition to the linear PSR, several other models use the idea of predictive state. This section gives a brief description of these models. Rivest and Schapire (1994) introduced the diversity representation for modeling *deterministic* systems. The diversity representation of the state of a system is the set of tests that will succeed from the current history. (When dealing with deterministic systems, tests either will or will not succeed. There is no need to talk about the probability of success, which is either 1.0 or 0.0.) The tests used in the diversity representation are called *end-tests* or *e-tests*.[2] An e-test consists of a sequence of actions followed by an observation, such as $a_1 a_2 \ldots a_k o$. The prediction for the e-test is the probability of seeing the observation after taking the sequence of actions.[3] Even though there are an infinite number of e-tests, Rivest and Schapire (1994) showed that there are a finite number of equivalence classes of e-tests for a finite underlying system. Two tests are equivalent if they succeed from the same (unobservable) system states. The authors provide an example of a system where the number of equivalence classes (which they call the *diversity* of the system) is the logarithm of the rank of the system-dynamics matrix, making the diversity representation much more compact than a linear PSR.[4] However, the authors also provide an example system where

[2]Rudary and Singh (2003) introduced the name "e-test".

[3]An e-test prediction can be constructed from the predictions of standard tests, since an e-test is a special kind of set test (cf. Section A.2).

[4]On that example system, the factored PSR presented in Chapter V would also have size logarithmic in the rank of the system-dynamics matrix.

the diversity is exponentially larger than the rank of the system-dynamics matrix. Thus, the diversity representation may be more compact than a linear PSR for some deterministic systems, but the linear PSR may be more compact for other systems (and linear PSRs can model stochastic systems).

The e-test-based PSR (EPSR) introduced by Rudary and Singh (2003) is a predictive state model that subsumes the diversity representation. The EPSR is based upon the idea of core e-tests, which are a set Q_e of e-tests such that the prediction for any e-test is a (history-independent) linear function of the predictions for Q_e. While the concept of core e-tests is equally valid in stochastic and deterministic systems (unlike the diversity representation), there is no known general form for maintaining the predictions of the core e-tests in stochastic systems. However, in deterministic systems, Rudary and Singh (2003) showed that one can maintain the predictions for the core e-tests, making the EPSR a viable model. Furthermore, they showed that for deterministic systems, the number of core e-tests is no greater than the *minimum* of the diversity of the system and the rank of the system-dynamics matrix (i.e., the number of core tests in the linear PSR). Thus, the EPSR can be exponentially more compact than a linear PSR, but it is viable only for deterministic systems.

Observable operator models (OOMs) and input/output OOMs (IO-OOMs) (Jaeger, 1998) are types of models for uncontrolled and controlled dynamical systems, respectively. There are interpretable and uninterpretable versions of both OOMs and IO-OOMs; the interpretable versions use predictive state. For uncontrolled systems, Singh et al. (2004) showed that OOMs and linear PSRs have equivalent expressive power — i.e., OOMs with state vectors of size k can model the same systems as linear PSRs with state vectors of size k. James (2005) presents methods for translating OOMs to linear PSRs for uncontrolled systems and vice versa. For controlled systems, Singh et al. (2004) showed that there are systems that can be modeled by linear PSRs that cannot be modeled by IO-OOMs. That is, linear PSRs are more general that IO-OOMs for controlled systems.

Transformed predictive state representations (TPSRs) (Rosencrantz, Gordon, & Thrun, 2004) are a generalization of linear PSRs where the entries in the state vector are allowed to be *linear combinations* of tests' predictions. In contrast, in the linear PSR, each entry is a prediction for a single test. Any linear PSR is also a TPSR, because a single test's prediction is also a linear combination of tests' predictions. Rosencrantz et al. (2004) showed how one can use singular value decomposition to choose the linear combinations that will form the state of the TPSR. Given a desired size k for the state vector, the singular value decomposition allows one to easily choose the best k linear combinations of tests to use as the state vector of the TPSR. Because the TPSR allows linear combinations of predictions as state vector entries, the state is no longer a vector of probabilities, so it is not always apparent when the state is invalid. In contrast, in a linear PSR, one can adjust the state vector of an approximate model whenever the entries fall outside the $[0, 1]$ range of valid probabilities.

Temporal-difference networks (TDNets) (Sutton & Tanner, 2004) are both a type of predictive state model and an algorithm for learning the model. The network part of the TDNet is a directed graph, where edges denote temporal relationships between nodes. Each node has some value that changes over time. The values of the nodes of the TDNet are what comprise the model state. The nodes' values are defined as predictions about (functions of) tests, which is what qualifies the TDNet as a predictive state model. The nodes' values are defined recursively (by specifying edges in the network), in order to be amenable to temporal-difference learning algorithms.

Appendix C shows how the prediction for any test can be defined recursively, so one can include the prediction for any test as part of the state of a TDNet.

The TDNet architecture is very general, allowing for arbitrary functions of predictions to be included in the network. However, much of the work with TDNets has been limited to using e-tests as the nodes in the network. A couple of exceptions to this are worth noting briefly. Wolfe, James, and Singh (2005) performed some experiments with TDNets that included standard tests as nodes, though the learning algorithm did not perform very well. Sutton, Rafols, and Koop (2005) included temporally extended predictions in their TDNet by using macro-actions (called options) in place of regular actions when defining some of their nodes. However, the tests they were predicting only had observations at the end of the tests, like e-tests.

In addition to allowing very general node definitions, TDNets technically allow general forms for updating the model state. However, in practice the new state vector is often computed by multiplying the old state vector by a matrix that depends upon the most recent action and observation, then passing each entry of that resulting vector through a sigmoid function (e.g., Sutton & Tanner, 2004; Sutton et al., 2005; Tanner & Sutton, 2005). This state update form is taken from neural networks; indeed, TDNets can be viewed as recursive neural networks. However, there are no guarantees that an accurate TDNet with the linear-sigmoid update even exists for a given system. This is one of the reasons that I do not address TDNets in great detail in the rest of this dissertation. In contrast, there exists a perfectly accurate linear PSR for any system whose system-dynamics matrix has finite rank (Singh et al., 2004).

Even though my work focuses upon upon discrete-observation systems, it is worth mentioning some predictive state models that are designed to model systems with real-valued observations. These include the predictive linear Gaussian (PLG) model (Rudary, Singh, & Wingate, 2005) and several extensions and adaptations of the PLG model (e.g., Wingate & Singh, 2006b, 2006a; Rudary & Singh, 2006), in addition to the exponential family PSR (Wingate & Singh, 2007, 2008).

1.2.2 Latent State Representations

An alternative to predictive state representation is latent state representation. The traditional models in reinforcement learning — partially observable Markov decision processes (POMDPs) — fall under the umbrella of *latent state* representations. This term describes models that assume that there is some set of latent states such that the system is always in exactly one of these latent states. However, the agent will not, in general, get to observe the current latent state of the system. Since the agent cannot observe the latent state, the state representation of a latent-state model is a *distribution over latent states*, representing the probability that the system is in each latent state given the current history.

It is important to distinguish between the model's state representation and the latent states of the system (also called *hidden states*). To reiterate, the model's state representation is a distribution over latent states. This distribution is called the *belief state*.

While my research deals with predictive state models, latent-state models provide a basis of comparison, both for empirical and theoretical results. For example, in Section 2.1, I will empirically compare the accuracy of learned predictive state models and learned latent-state models, demonstrating an advantage of using predictive state models. There are also theorems that relate the sizes of latent-state models and predictive-state models of the same system, showing that predictive-state models are comparable in size (James, 2005).

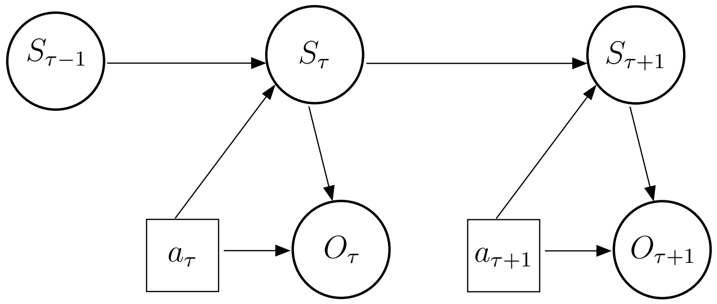


Figure 1.3: The graphical model for two time steps of a POMDP model. The S_i are *latent* states, the a_i are actions, and the O_i are observations. The model states (i.e., the belief states) are not shown in this figure.

The following sections describe several classes of latent state models that I mention in subsequent chapters. I provide the details of these latent-state models here for completeness.

Partially Observable Markov Decision Processes

The first example class is *partially observable Markov decision processes* (POMDPs) (Drake, 1962; Astrom, 1965; Monahan, 1982), which are capable of representing stochastic, partially observable dynamical systems.

A POMDP model describes a system that is in one of a finite set of latent states at each time step. I use the S variables to denote latent state, in contrast to the X variables that denote model state in Figure 1.1. The system evolves in the following way (Figure 1.3): the current latent state $S_{\tau-1}$ and an action a_τ determine the probability distribution over the next latent state S_τ, and an observation is emitted according to a probability distribution that depends upon the action a_τ and the new latent state S_τ. At the next time step, this process repeats, with the system transitioning to a new latent state $S_{\tau+1}$ from S_τ based upon the action $a_{\tau+1}$; the next observation depends upon $a_{\tau+1}$ and $S_{\tau+1}$. This process repeats for every time step.

The following formal description of a POMDP describes the model parameters and the state representation of the model. A POMDP is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{O}, \Omega)$ where

- $\mathcal{S}$ is the finite set of latent states
- $\mathcal{A}$ is the finite set of actions available to the agent
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ gives the probability $\mathcal{T}(\mathcal{S}^i, \mathcal{A}^j, \mathcal{S}^k)$ of transitioning to $\mathcal{S}^k \in \mathcal{S}$ when taking action $\mathcal{A}^j$ from $\mathcal{S}^i \in \mathcal{S}$.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ gives the expected immediate reward $\mathcal{R}(\mathcal{S}^i, \mathcal{A}^j)$ for taking action $\mathcal{A}^j$ from state $\mathcal{S}^i$.
- $b_0 : \mathcal{S} \rightarrow [0, 1]$ gives the probability $b_0(\mathcal{S}^i)$ of the system starting in state $\mathcal{S}^i$
- $\mathcal{O}$ is the finite set of possible observations
- $\Omega : \mathcal{A} \times \mathcal{S} \times \mathcal{O} \rightarrow [0, 1]$ gives the probability $\Omega(\mathcal{A}^j, \mathcal{S}^k, \mathcal{O}^i)$ of observing $\mathcal{O}^i \in \mathcal{O}$ after taking action $\mathcal{A}^j$ and transitioning to $\mathcal{S}^k$.

In the POMDP literature, the "states" of a POMDP model refer to the latent states $\mathcal{S}$ of the system, while the actual state representation of the POMDP model is a distribution over latent states called the *belief state*.

As the agent takes actions in a dynamical system, it will need to update its model state after every time step. That is, the agent will need to compute the model state at history hao from the model state at history h, using the model parameters. For a POMDP model, the agent updates the belief state $b(h)$ for some history h to obtain a new belief state $b(hao)$ after taking action a and seeing the observation o. The agent computes $b(hao)$ according to the following equation:

$$b(hao) = \frac{O_{a,o}\mathcal{T}_a b(h)}{\mathbf{1}^\top O_{a,o}\mathcal{T}_a b(h)}$$

where $O_{a,o}$ is a diagonal matrix with $\Omega(a,i,o)$ as the i^{th} diagonal element, and the *transition matrix* $\mathcal{T}_a$ has $\mathcal{T}(j,a,i)$ as its $(i,j)^{th}$ entry, and $\mathbf{1}$ is a vector of 1's.

In order to be a valid model, the POMDP should be able to compute a prediction for any test from any history. Equation 1.1 shows that any such prediction can be made by multiplying together an appropriate set of predictions for length-one tests, so if the POMDP can make a prediction for any length-one test from any history, then it can make any prediction. Any one-step prediction $p(ao|h)$ can be computed as $\mathbf{1}^\top O_{a,o}\mathcal{T}_a b(h)$,[5] so the POMDP model can make any prediction.

Markov Decision Processes

Markov decision processes (MDPs) (Bellman, 1957) are a special case of POMDP models where the agent gets to observe the state of the system (which is hidden/latent in the general POMDP) after each time step. This means that the set of observations is the same as the set of states, and the observation upon transitioning to some state will simply be that state. For an MDP, the belief state at any history (except the null history) will be a distribution with all of its mass on one state, since history reveals the state of the system. Thus, the state representation of the MDP can be

[5]This matrix product is equal to

$$\sum_{S_h}\sum_{S_{hao}} Pr(o|a,S_{hao})Pr(S_{hao}|S_h,a)Pr(S_h|h)$$

where S_h is the latent state after history h and S_{hao} is the latent state after history hao. The probabilities in this expression are as follows:

- $Pr(o|a,S_{hao})$ is the probability of observing o after transitioning to state S_{hao} by action a. This is the S_{hao} diagonal element of $O_{a,o}$.
- $Pr(S_{hao}|S_h,a)$ is the probability of transitioning to state S_{hao} from S_h under action a. This is the element of $\mathcal{T}_a$ in the S_{hao} row and S_h column.
- $Pr(S_h|h)$ is the probability of being in state S_h after history h. This is the S_h element of the vector $b(h)$.

The summations in the expression above marginalize the latent states S_h and S_{hao}, leaving $Pr(o|h,a) = p(ao|h)$, as desired.

simplified to the current state of the system.

Updating this state representation is completely straightforward and requires no parameters: the updated state at hao is just o. As with any complete model, an MDP can use its parameters to make predictions about any test from any history. For any history hao of length at least one, the predictions are independent of h given the most recent observation o. Thus, one can compute the prediction for an arbitrary test as

$$p(a_1o_1a_2o_2\dots a_ko_k|hao) = \mathcal{T}(o, a_1, o_1)\prod_{i=2}^{k}\mathcal{T}(o_{i-1}, a_i, o_i).$$

To make predictions from the only history $h = \emptyset$ of length zero, one conditions upon the initial latent state $S_\emptyset$:

$$p(a_1o_1a_2o_2\dots a_ko_k|\emptyset) = \sum_{S_\emptyset\in\mathcal{O}} b_0(S_\emptyset)\mathcal{T}(S_\emptyset, a_1, o_1)\prod_{i=2}^{k}\mathcal{T}(o_{i-1}, a_i, o_i).$$

Structured Models

With the models that have been described so far (including the POMDP, MDP, and linear PSR models), the observation at each time step is a member of a single, unstructured set of discrete observations $\mathcal{O}$. When the number of possible observations in a system is very large, the number of parameters required for one of these models can be intractably large, as the number of parameters is linear in the number of observations.

However, in many systems, the observation at each time step has some structure that a model of the system can exploit. For example, the observation at each time step can often be naturally represented by a vector of several values (each with a discrete domain), instead of a single value. The size of the joint observation space (i.e., the number of possible observation vectors) might still be prohibitively large. However, the number of possible values for a small sub-vector of the observation vector is often much less. For instance, if each element of the length-j observation vector can take on k possible values, the number of observation vectors is k^j, exponential in j. This is in stark contrast to the k^2 values that a length-2 sub-vector could take. With the POMDP, MDP, and linear PSR models, there is a separate set of parameters for every *joint* observation value, so the number of parameters is also exponential in j.

However, there are model classes where the number of parameters is not linear in the number of *joint* observations, but in the number of possible sub-vectors of some length. I describe one such class of models — dynamic Bayesian networks (DBNs) — in the next section. DBNs are latent-state models, but the *factored PSRs* that I describe in Chapter V are a class of predictive-state models that also exploit structure in observation vectors.

Dynamic Bayesian Networks

Dynamic Bayesian networks (DBNs) (Dean & Kanazawa, 1989) extend POMDPs by allowing both the latent state and the observation at each time step to be vectors. For a given system, a DBN model can have significantly fewer parameters than a POMDP model, because a DBN can capture conditional independencies among the dimensions of the latent state and observation. These conditional independencies allow compression in the representation of the transition function $\mathcal{T}$ and the observation function Ω, as discussed below.

Formally, a DBN consists of the following:

- the set of latent state vectors $\mathcal{S}$, which is the Cartesian product of m sets $({}^1S, {}^2S, \dots {}^mS)$; each ${}^iS$ is the finite domain of the i^{th} dimension of the latent state vector.

- a set of observation vectors $\mathcal{O}$, which is the Cartesian product of z sets $({}^1O, {}^2O, \dots {}^zO)$; each ${}^iO$ is the finite domain of the i^{th} dimension of the observations.
- the set of actions $\mathcal{A}$, the transition function $\mathcal{T}$, reward function $\mathcal{R}$, starting distribution b_0, and observation function Ω. Each of these functions is defined as for a POMDP but with vector-valued state and observation domains.

The compactness of a DBN over a POMDP comes from its *representation* of the functions $\mathcal{T}$, $\mathcal{R}$, and Ω. A DBN represents each of these functions as a product form in order to exploit conditional independencies among the random variables. A graphical model such as in Figure 1.4 captures these conditional independencies. This graph G contains a variable for the actions at times τ and $\tau + 1$, a variable for each dimension of latent state for time steps τ and $\tau + 1$, and a variable for each dimension of the observation at τ.[6] The conditional independencies in the graph G lead to the following form for the transition function for action a (i.e., the distribution of $\mathcal{S}_{\tau+1}$ given $\mathcal{S}_\tau$, parameterized by a):

$$\begin{aligned} Pr(\mathcal{S}_{\tau+1}|\mathcal{S}_\tau; a) &= Pr(\mathcal{S}^1_{\tau+1}, \mathcal{S}^2_{\tau+1}, \dots \mathcal{S}^m_{\tau+1}|\mathcal{S}^1_\tau, \mathcal{S}^2_\tau, \dots \mathcal{S}^m_\tau; a) \\ &= \prod_{i=1}^{m} Pr(\mathcal{S}^i_{\tau+1}|\mathrm{Par}(\mathcal{S}^i_{\tau+1}); a) \end{aligned}$$

where $\mathrm{Par}(\mathcal{S}^i_{\tau+1})$ are the parents of the variable $\mathcal{S}^i_{\tau+1}$ in the graph G. To see how the factored form allows a compact model, consider the following: if the distribution over $\mathcal{S}_{\tau+1}$ was stored in tabular form for each value of $\mathcal{S}_\tau$, one would need $|\mathcal{S}|(|\mathcal{S}|-1)$ parameters. Note that $|\mathcal{S}| = \prod_{i=1}^m |{}^iS|$ grows exponentially in the number of state variables m. In contrast, the parameters needed for the factored form are just the conditional probability tables (CPTs) that specify the distributions $Pr(\mathcal{S}^i_{\tau+1}|\mathrm{Par}(\mathcal{S}^i_{\tau+1}); a)$ for each i. If k is an upper bound on the number of parents and $c \overset{\text{def}}{=} \max_i |{}^iS|$ is the

[6]The reward at each time step is implicitly included as one of the observation variables.

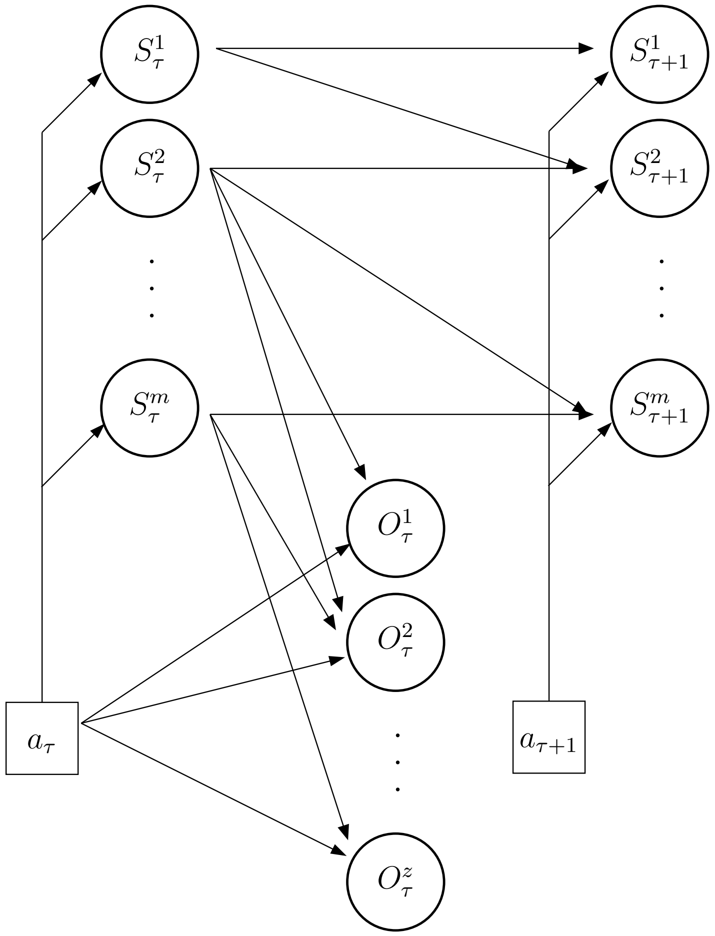


Figure 1.4: A time-slice view of a DBN, showing the *latent* state variables at times τ and $\tau + 1$, the actions at times τ and $\tau + 1$, and the observation variables at time τ. The conditional independencies between latent state and observation dimensions allow a compact representation of the transition and observation functions (e.g., O_τ^z is independent of the other latent state variables given S_τ^m). The model states (i.e., belief states) are not shown in this figure; they will not generally be factored (cf. the introduction of Chapter V).

largest domain size for any $|^iS|$, then the number of parameters in these CPTs is less than $c^{k+1}m$. Note that this is exponential in k, the number of parents, rather than the number of state variables. The observation function Ω can be similarly compressed by using a product form. This compression allows DBNs to tractably represent larger systems than a simple POMDP model. However, even when the latent state transition function and observation function are factorable, the *model state* (i.e., belief state) will generally *not be factorable*. The introduction to Chapter V explains this in more detail.

1.2.3 Comparison of State Representations

Now that I have described some examples of latent state models and predictive state models, in this section I highlight some appealing aspects of using predictive state models, providing motivation for my work on PSRs in the following chapters. A primary difference between latent and predictive state lies in the semantics or *definition* of the state vector. Latent state representations define the state vector as a distribution over latent random variable(s). PSRs define the state vector in terms of actions the agent can take and observations that the agent can see. Thus, a latent state describes a distribution from which the agent will *never get a sample*, because it never observes the latent state (in general). However, a predictive state describes a distribution over *observations*, from which the agent *can get a sample*, thus revealing information about the distribution described by the state. The motivating hypothesis behind the work on PSRs is that a state representation defined in terms of an agent's observations will enable better models than a state representation defined in terms that may be meaningful to people but are not defined in terms of the agent's sensors or actuators.

One example where this hypothesis has been supported is in the area of learning a model from experience. I compared the accuracy of a linear PSR and POMDP learned from a single trajectory of experience in several simple systems, and found that the linear PSR was more accurate in making predictions by several orders of magnitude (Wolfe et al., 2005). I describe these results in more detail in Section 2.1. Rudary and Singh (2006); Rudary et al. (2005) obtained similar results when comparing PSRs to latent state models in systems with real-valued observations. This advantage of PSRs over latent state models comes from the fact that the model parameters are defined in terms of the actions and observations of the agent. On the other hand, the parameters of a latent state model are defined in terms of the latent states, which have an unknown (to the agent) relationship to the data (i.e., actions and observations).

Another motivation for the PSR approach comes from the system-dynamics matrix, which provides some insight into the information that a model's state must capture. Recall that state is a summary of history that allows a model to make any prediction about the future as if the model knew the current history. That is, if one were to give the model the state (in whatever representation the model was using) for some history h, then the model must be able to compute any entry in the h row of the system-dynamics matrix $\mathcal{D}$ without knowing what h is. Thus, the state at h must capture all of the information in the h row of $\mathcal{D}$. From this viewpoint, a PSR is a very natural model of the system: it captures the information in a row of $\mathcal{D}$ by using some of the entries in that row (or, more generally, a function of some of the entries in that row). On the other hand, latent state models' belief state does not come directly from $\mathcal{D}$, but makes an assumption about the system (i.e., that there exists a finite set of latent states of the system).[7]

[7]The existence of a finite set of latent states in an assumption that is not true for all systems. As I mention in the next section, James (2005) provides an example system that can be modeled

Rafols, Ring, Sutton, and Tanner (2005) demonstrate yet another advantage of using predictive state models, providing evidence that predictive state is useful for generalization. Generalization involves taking what is known about one particular situation and applying it to similar situations. For an agent to function well in a large, complex environment, generalization will be essential, since the agent cannot possibly learn about every different situation independently but must use what it has already learned about similar situations.

POMDPs and Linear PSRs

In addition to general differences between PSRs and latent state representations, there are several specific comparisons to make between linear PSRs and POMDPs. The introductory paper on linear PSRs (Littman et al., 2001) showed that any system that could be represented as a POMDP with n latent states could also be represented as a linear PSR with no more than n core tests. James (2005) extended this result by providing examples of systems that can be modeled by a linear PSR with n core tests but not by any POMDP with n latent states (or, in one example, by any POMDP with any finite number of states). Put another way, the state representation of a linear PSR model is never larger than the belief state of a POMDP model of the same system, and in some cases it can be (infinitely) smaller.

Another measure for comparing model sizes is the number of free parameters in a model. A linear PSR with n core tests needs more parameters than a POMDP model with n latent states, by roughly a factor of n. If $|\mathcal{A}|$ and $|\mathcal{O}|$ are the number of possible actions and observations, respectively, in the system, then a POMDP has $(n-1)\ n\ |\mathcal{A}|$ free parameters for the transition probabilities, $(|\mathcal{O}|-1)\ n\ |\mathcal{A}|$ free parameters for the observation probabilities, and $n-1$ free parameters for the initial

by a finite linear PSR but not by any POMDP with a finite number of latent states.

belief state, for $O(|\mathcal{A}|\ n\ (n + |\mathcal{O}|))$ total parameters. A linear PSR has $|\mathcal{A}|\ |\mathcal{O}|\ n^2$ parameters for the M_{ao} matrices, $|\mathcal{A}|\ |\mathcal{O}|\ n$ parameters for the m_{ao} vectors, and n parameters for the initial prediction vector, for $O(|\mathcal{A}|\ n\ (n\ |\mathcal{O}|))$ total parameters.

However, not all of these parameters are free parameters; Appendix B lists necessary and sufficient constraints on the linear PSR parameters. One disadvantage of the linear PSR is that there is no known, finite set of constraints that forms both necessary and sufficient conditions for potential linear PSR parameters to constitute a valid linear PSR model. (A valid linear PSR model is a model that makes predictions that are consistent with *some* system.) In contrast, for a POMDP model there are a finite number of constraints on the parameters that are easily checked (i.e., the parameters must be in the range $[0.0, 1.0]$ and certain subsets of parameters must sum to 1.0).

1.3 Learning a Linear PSR from Experience: Background

This section provides a background on one of the central issues that my work addresses: "How can an agent use its experience in a dynamical system to build a model of that system?" Answers to this question take the form of *learning algorithms*, which construct a model from the agent's experience. Different classes of models lend themselves to different learning algorithms.

The remainder of this section describes the *reset algorithm*, an existing algorithm for learning a linear PSR model. I use concepts from the reset algorithm in several learning algorithms that I develop in later chapters.

The reset algorithm requires multiple trajectories of experience from the initial state of the system, which can be accomplished by "resetting" the system to its initial state as needed. In practice, an agent will often not have the ability to reset the dynamical system in which it finds itself. This motivates my extension of the reset algorithm, the *suffix-history algorithm* (Section 2.1), which does not require that the agent be able to reset the system. Section 2.1 also includes an empirical comparison of both the reset and suffix-history algorithms with other model-learning algorithms, including a POMDP-learning algorithm. Those results help motivate the use of linear PSR models.

The Reset Algorithm for Learning a Linear PSR

The reset algorithm for learning a linear PSR can be divided into three main stages:

1. Estimate predictions (i.e., entries of the system-dynamics matrix $\mathcal{D}$) from the agent's experience in the system.

2. Using those estimates, find a set of core tests Q. (Recall that the predictions for the tests Q form the state vector of the linear PSR model.) The algorithm also finds a set of core histories (defined below) in this stage.

3. Using the estimated predictions, the set of core tests, and the set of core histories, solve for the parameters of the linear PSR model.

The details of the reset algorithm rely upon the concept of *linear core histories*. Just as core tests are useful for analyzing the relationship among columns of $\mathcal{D}$, linear core histories are useful for analyzing the relationship among rows of $\mathcal{D}$:

Definition 1.5. A set of histories H are called *linear core histories* if and only if there exists a test-independent vector w_h for any history h such that $p(t|h) = p^\top(t|H)w_h$ for all tests t. If $|H| = \text{rank}(\mathcal{D})$, then H is a set of *minimal linear core histories*.

As with core tests, I will hereafter drop the "linear" qualifier on core histories. Analogous to the core tests, a set of histories H are core histories if and only if any row of $\mathcal{D}$ is a linear combination of the H rows, and core histories H are *minimal* if and only if the H rows are linearly independent (or equivalently, $|H| = \text{rank}(\mathcal{D})$).

The next part of this sub-section describes the three stages of the reset algorithm in reverse order:

- First, I describe the computation of the model parameters (i.e., the third stage), assuming that the estimated predictions from stage 1 are correct (i.e., are equal to the true values of the predictions), and assuming a valid set of core tests and histories have been found.
- Second, I describe the process for finding core tests and histories (i.e., the second stage), assuming that the estimated predictions from stage 1 are correct.
- Finally, I describe the method for estimating predictions from the agent's experience (i.e., the first stage), as well as some adjustments that are made to the other parts of the algorithm to account for the fact that it uses estimated predictions instead of the true predictions.

Stage 3: Parameter estimation given core tests Q, core histories K, and the true predictions

The parameters needed to compose a linear PSR are the initial state vector $p(Q|\emptyset)$ and the m-vectors for several tests. Recall that the m-vector m_t for a test t is the unique vector such that

$$p(t|h) = p^\top(Q|h)m_t$$

for all histories h; note that m_t does not depend upon h. The parameters for the linear PSR include the m-vectors for the one-step tests and all one-step extensions of each core test: $\{\forall a, o, q_i \in Q : m_{ao}, m_{aoq_i}\}$.

One can solve for m_t for any test t — including those needed for the linear PSR parameters — using the following procedure. For any test t, any set of histories K, and any set of core tests Q,

$$p(t|K) = p(Q|K)m_t.$$

When K and Q are minimal core histories and tests, respectively, $p(Q|K)$ is invertible,[8] so

$$m_t = p(Q|K)^{-1}p(t|K).$$

Thus, the reset algorithm uses this equation to solve for the m-vector parameters of the linear PSR, using the true predictions $p(Q|K)$ and $\{\forall a, o, q_i \in Q : p(ao|K), p(aoq_i|K)\}$. The only remaining parameters of the linear PSR are the elements of the initial state vector, $p(Q|\emptyset)$, which are directly available from the predictions calculated in stage 1.

Stage 2: Finding core tests and histories using the true predictions

The reset algorithm uses an iterative procedure to find core tests and core histories. On the i^{th} iteration, the algorithm examines a sub-matrix $p(T_i|H_i)$ of $\mathcal{D}$, computing its rank r_i. It then finds r_i linearly independent rows and columns of that matrix, corresponding to some histories H_i' and tests T_i'. If $r_i = r_{i-1}$, the procedure stops and returns T_i' as the core tests and H_i' as the core histories. Otherwise, the algorithm computes T_{i+1} and H_{i+1} as the one-step extensions of T_i' and H_i', respectively. Specifically, $T_{i+1} \stackrel{\text{def}}{=} \{ao, aot : \forall a, o, t \in T_i'\}$ and $H_{i+1} \stackrel{\text{def}}{=} \{\emptyset\} \cup \{ao, hao : \forall a, o, h \in H_i'\}$.

[8] I prove this explicitly in Lemma F.3, found in Section F.1.

This begins the next iteration. The initial set T_1 is all tests of length one, and H_1 is all histories of length one and the null history $\emptyset$.

This iterative method for choosing Q and K is not guaranteed to find a full set of core tests or histories, but the tests or histories that it does find will be linearly independent (James & Singh, 2004). Despite this limitation, this procedure often works well in practice, as seen in empirical results using the suffix-history and reset algorithms (e.g., James & Singh, 2004; Wolfe et al., 2005; Wolfe & Singh, 2006).

The full algorithm: using predictions that are estimated from data

Since the reset algorithm does not have access to the true predictions for the system, it must estimate the needed predictions from data. The algorithm uses multiple trajectories of experience to estimate a prediction $p(t|h)$ in the following way. Let $\#h$ be the number of trajectories that begin with h, and let $\#ht$ be the number of trajectories that begin with ht. Let $\pi(h', a)$ be the probability that the agent took action a from history h'.[9] Then for a test $t = a_1 o_1 \ldots a_k o_k$, the estimated prediction is

$$\hat{p}(t|h) = \frac{\#ht}{\#h} \prod_{i=1}^{k} \frac{1}{\pi(ha_1 o_1 \ldots a_{i-1} o_{i-1}, \quad a_i)}. \tag{1.7}$$

The first term $\frac{\#ht}{\#h}$ estimates the probability that both the actions and observations of t follow h, which depends upon the agent's policy for choosing actions. Multiplying the first term $\frac{\#ht}{\#h}$ by the remaining terms cancels out the effect of the agent's policy, making $\hat{p}(t|h)$ an unbiased estimator of $p(t|h)$ (Bowling, McCracken, James, Neufeld, & Wilkinson, 2006). Since the variance of $\hat{p}(t|h)$ is proportional to $\frac{1}{\#h}$ (Bowling et al., 2006), the variance of $\hat{p}(t|h)$ goes to 0.0 as the amount of training data increases.

[9]If the agent's policy π is available to the learning algorithm, then the algorithm can use the true policy values $\pi(h', a)$ for estimating the predictions. Otherwise, the algorithm approximates $\pi(h', a)$ as the fraction of trajectories where the agent took action a from history h'.

Therefore, $\hat{p}(t|h)$ is not only an unbiased estimator, but it is also a *consistent* estimator of $p(t|h)$ (cf. Theorem 9.1 of Wackerly, Mendenhall, and Scheaffer (2002)).

Adapting the core search process to use estimated predictions: If the algorithm had exact values for predictions, the search for core tests and histories (i.e., stage 2) could perform strict rank calculations on matrices of predictions to find linearly independent tests or histories. However, noise in the estimated predictions $\hat{p}(t|h)$ invalidates strict rank calculations. The reset algorithm uses the singular values of the estimated prediction matrices to calculate their rank, using a method from Golub and Van Loan (1996). Appendix D presents the details of how this method is incorporated into the core search stage of the reset algorithm.

Algorithm Properties

The running time of the reset algorithm is the sum of the running times for its three stages. For estimating the predictions from data (stage 1), Section 2.2 proves that the algorithm need not estimate the prediction for any history/test pair with combined length more than $2n+1$, where n is the rank of the system-dynamics matrix. The algorithm uses the training data to calculate statistics that can then be used to estimate the predictions for *all* history/test combinations of length $2n+1$ or less. If T is the number of time steps of training data, then calculating those statistics takes time $O(Tn)$. For finding the core tests and histories (stage 2), Appendix D shows that the running time is $O(n^5(|\mathcal{A}||\mathcal{O}|)^3)$, where $|\mathcal{A}|$ and $|\mathcal{O}|$ are the numbers of possible actions and observations, respectively. Finally, solving for the model parameters given the core tests and histories (stage 3) takes time $O(n^3+n^3|\mathcal{A}||\mathcal{O}|)$, which is dominated by the running time of finding the core tests and histories (stage 2). Therefore, *the running time of the reset algorithm is polynomial*: $O(Tn + n^5(|\mathcal{A}||\mathcal{O}|)^3)$. This is in stark contrast to the running time of simply listing all of the tests of length n (i.e., potential core tests), which is exponential in n: $O((|\mathcal{A}||\mathcal{O}|)^n)$.

While the polynomial running time of the reset algorithm is important, it is equally important that the algorithm learn accurate models. One of the important qualities of the reset algorithm is that it can learn increasingly accurate PSR models as the amount of training data increases. Since the estimated predictions are consistent estimators of the true predictions, the estimated predictions will converge in probability to the true predictions as the amount of training data grows. With a true set of core tests Q, a true set of core histories K, and the values of $p(Q|\emptyset)$, $p(Q|K)$, and $\{\forall a, o, q_i : p(ao|K), p(aoq_i|K)\}$, one can solve for the exact PSR parameters. Thus, *the reset algorithm can learn increasingly accurate PSR parameters as its training data increases.*

1.4 Summary

This chapter has laid out some of the fundamental concepts that are used throughout this work:

- state-based models of dynamical systems
- the systems-dynamics matrix view of a dynamical system
- descriptions of several classes of models for dynamical systems, including linear PSRs, POMDPs, MDPs, and DBNs.

Each of these models summarizes history using some state representation. The distinguishing feature of a PSR model is that it represents state as a set of statistics about future actions and observations. PSRs are a promising method for modeling dynamical systems, especially with respect to parameter estimation, as their state representation is defined in terms of the actions and observations of the agent rather than latent random variables.

This chapter has also presented the *reset algorithm* for learning a linear PSR model of a dynamical system from a set of trajectories of experience in the system. The reset algorithm estimates the linear PSR parameters from sample statistics of the experience trajectories. As these sample statistics converge to their true values, the model parameters learned by the reset algorithm will become increasingly accurate. Therefore, the reset algorithm does not suffer from local optima, but can learn an accurate linear PSR in the limit of experience, provided that a valid set of core tests and histories are found.

The next chapter describes my work in extending the early work on PSRs presented in this chapter.

CHAPTER II

Linear PSR Contributions

This chapter describes several contributions pertaining to linear PSRs, including the *suffix-history algorithm* for parameter estimation of linear PSRs (Section 2.1). The suffix-history algorithm generalizes the reset algorithm (Section 1.3) to the case where an agent does not have the ability to reset the system to its initial configuration. This generalization is important because an agent will not have the ability to reset many real-world systems. Section 2.1 also gives an empirical comparison of parameter estimation in PSRs and POMDPs on some simple problems; the fact that the PSR models are generally more accurate motivates the use of predictive state models instead of latent-state models such as POMDPs.

The other contributions presented in this chapter address the identification of core histories and core tests, which is an important step in the existing algorithms for learning a linear PSR model from data. Both the reset and suffix-history algorithms use the same iterative process to find core histories and core tests. On the i^{th} iteration, histories and tests of length i or less are considered for inclusion as core histories and core tests. Thus, a bound on the length of histories and tests that constitute a set of core histories or tests is also a bound on the number of iterations required in the core search process. This motivates the primary results of Section 2.2, which provide upper bounds on the lengths of tests and histories that an algorithm needs to examine in order to find core tests or histories.

2.1 The Suffix-history Algorithm for Parameter Estimation in Linear PSRs

This section describes the *suffix-history algorithm* for parameter estimation of linear PSRs, which was published by Wolfe et al. (2005). This algorithm generalizes the reset algorithm (Section 1.3) to the case where there is only one trajectory of experience from which to learn the parameters. This will be the case in many real-world situations: the agent is placed in some dynamical system (i.e., some environment), and it interacts with the system for some period of time. It must then build a model of the system from its experience in the system. Without the ability to reset the system or start over from the beginning, the agent has but a single trajectory of experience from the system, and it must use that to build a model.

I describe the suffix-history algorithm by noting how it differs from the reset algorithm. Recall that the reset algorithm constructs a linear PSR in three main stages:

1. Estimate predictions about the system from the agent's experience with the system.

2. Using those estimates, find a set of core tests Q and core histories K.

3. Using the estimated predictions and the sets of core tests and core histories, solve for the parameters of the linear PSR model.

Since the difference between the reset and suffix-history algorithms lies in the form of the agent's experience in the dynamical system (i.e., many trajectories vs. one trajectory), only the first stage needs to be adapted. Once the first stage is adapted to estimate predictions from a single trajectory of experience, the latter two stages can proceed as in the reset algorithm, using the estimated predictions just as if they were calculated from multiple trajectories of experience.

2.1.1 Estimating Predictions from a Single Trajectory of Experience

The way in which the reset algorithm estimates a prediction $p(t|h)$ requires that the agent has been in history h multiple times. This is not possible when there is only a single long sequence $d = a_1o_1a_2o_2 \cdots a_ko_k$ of experience available, because each history occurs at most once in the single sequence d.

The suffix-history algorithm gets around this issue by treating every suffix of d as if it were a separate trajectory of experience. Specifically, the suffix-history algorithm estimates $p(t|h)$ using the same method as the reset algorithm, where the suffixes of d are the trajectories of experience from which $p(t|h)$ is estimated. This method of using suffixes of the experience d as histories (i.e., trajectories of experience) is similar to one used by Jaeger (1998) for uncontrolled dynamical systems. While these methods make efficient use of the data by using each sub-sequence of d to obtain information about many predictions, the suffix histories are not actually independent trajectories. Nevertheless, this has not posed a problem empirically (e.g., see Sections 2.1.2 and 5.3), in part because only the first few time steps of each suffix are actually needed to learn a model. Therefore, if the system $\mathcal{D}$ has a stationary distribution (induced by following a fixed policy), the first few time steps of the suffix histories that begin far apart in the original, single data sequence can be viewed as trajectories that are sampled starting from the stationary distribution.

In general, that stationary distribution is not the same as the initial distribution of the system. Thus, the suffix-history algorithm learns a model of a slightly different system $\mathcal{D}'$ that has the same dynamics as the original system $\mathcal{D}$, but it has a different initial distribution. Theorems 2.1, 2.2, and 2.3 state conditions under which the core tests and the model-update parameters for $\mathcal{D}'$ are accurate core tests and model-update parameters for $\mathcal{D}$; that is, only the initial prediction vector will differ between linear PSR models of $\mathcal{D}$ and $\mathcal{D}'$. I provide proofs for Theorems 2.1, 2.2, and 2.3 in Appendix E.

Theorem 2.1. *Let $\mathcal{D}$ be a system-dynamics matrix with finite rank n that can be modeled by a POMDP P with n hidden states. Let P' be a POMDP model that is identical to P except for a different initial belief state, and let $\mathcal{D}'$ be the system-dynamics matrix generated by the model P'. If the rank of $\mathcal{D}'$ is also n, then any set of core tests and update parameters for either $\mathcal{D}'$ or $\mathcal{D}$ are valid for both $\mathcal{D}'$ and $\mathcal{D}$.*

Theorem 2.2. *Let $\mathcal{D}$ be a system-dynamics matrix and h^* be a history of $\mathcal{D}$ such that $p(h^*|\emptyset) > 0.0$. Let $\mathcal{D}'$ be a system-dynamics matrix such that $\mathcal{D}'$ has the same dynamics as $\mathcal{D}$, but its null history is identical to history h^* in $\mathcal{D}$. If $\text{rank}(\mathcal{D}) = \text{rank}(\mathcal{D}')$, then any set of core tests and update parameters for either $\mathcal{D}'$ or $\mathcal{D}$ are valid for both $\mathcal{D}'$ and $\mathcal{D}$.*

There is an additional set of conditions that applies to fully-observable systems. Specifically, the following theorem states that if one builds a linear PSR of an MDP that has some initial state distribution, then that PSR will be an accurate model (after the initial time step) of the system that starts in any one of those possible initial states (assuming that start state is reachable at *some* point on or after the first time step, which will be true if the start state is ever seen as an observation).

Theorem 2.3. *Let $\mathcal{D}$ be a system-dynamics matrix that can be modeled by an MDP M with states S. Let b_0 be the initial state distribution of M, and let M' be an MDP that is identical to M except for its initial state distribution b'_0. If b'_0 satisfies the*

Table 2.1: Domain Statistics. The number of actions $|\mathcal{A}|$, the number of observations $|\mathcal{O}|$, and the number of required core tests $|Q|$ help determine the complexity of each domain.

Domain	$\|\mathcal{A}\|$	$\|\mathcal{O}\|$	$\|Q\|$
Tiger	3	2	2
Paint	4	2	2
Cheese	4	7	11
Network	4	2	7
Bridge	12	5	5
Shuttle	3	5	7
Maze 4x3	4	6	10

following property — $\forall s_i \in S$ such that $b'_0(s_i) > 0$, it holds that $b_0(s_i) > 0$ and s_i is reachable at some time $t > 0$ — then core tests and update parameters for $\mathcal{D}$ are valid core tests and update parameters for the system-dynamics matrix $\mathcal{D}'$ given by M'.

2.1.2 Empirical Results

This section presents results from running the suffix-history algorithm to learn linear PSR models of several simple domains from the POMDP literature that were collected by Cassandra (1999). These domains were also used to test PSR learning by James and Singh (2004) and Singh, Littman, Jong, Pardoe, and Stone (2003). Summary statistics for each domain are presented in Table 2.1.

These experiments evaluate the accuracy of linear PSR models learned from a single trajectory of experience using the suffix-history algorithm. I also evaluated three other model-learning algorithms on these domains, providing a basis of comparison with the suffix-history algorithm.[1] These three algorithms are:

- The expectation-maximization (EM) algorithm (Dempster, Laird, & Rubin, 1977) for learning a POMDP model.

  The special case of the EM algorithm for hidden Markov models (HMMs) is

[1]Wolfe et al. (2005) also compare the suffix-history algorithm with temporal-difference (TD) learning for TD networks (Sutton & Tanner, 2004), a class of PSR models. The TD networks did not perform as well as the linear PSRs learned by the suffix-history algorithm.

often called the Baum-Welch algorithm (Baum, Petrie, Soules, & Weiss, 1970). The algorithm also applies to POMDPs, which are just HMMs with actions. Note that the algorithm is *given the number of latent states for the POMDP model*, and it is susceptible to local optima, regardless of the amount of experience available for learning.

- The reset algorithm for learning a linear PSR model.

  The reset algorithm *requires multiple trajectories of experience*. Comparing the suffix-history algorithm with the reset algorithm demonstrates the empirical effect of the suffix-history approximation (i.e., treating each suffix of the single sequence of experience as a trajectory).

- The gradient algorithm (Singh et al., 2003) for learning a linear PSR model from a single trajectory of experience *given a set of core tests*.

  The gradient algorithm *must be given a set of core tests*. It begins with some initial model parameters and performs online adjustments to those parameters based upon the stochastic gradient of prediction error.

Each algorithm is given a trajectory (or, for the reset algorithm, several trajectories) of training data and learns a model from that training data.

For simplicity, I generated each training sequence and testing sequence using a uniform random policy (i.e., at each time step, each action was chosen with equal probability). One could use another stationary policy[2] to generate data for the suffix-history algorithm, as long as the policy explored the system sufficiently well (i.e., conditions for at least one of Theorems 2.1, 2.2, or 2.3 should be met).

I evaluated each learned model at every time step of a test sequence, using a

[2]A non-stationary policy would invalidate Equation 1.7, requiring a significantly different computation and additional information about the agent's policy in order to estimate predictions.

squared error of predictions. Specifically, I computed the model's error at history h in the test sequence as $\frac{1}{|\mathcal{O}|}\sum_{i=1}^{|\mathcal{O}|}(p(ao_i|h) - \hat{p}(ao_i|h))^2$, where a is 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 by 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. During testing, I bounded the elements of the learned PSRs' state vectors in the $[0, 1]$ range of valid probabilities, but I did not bound the $\hat{p}(ao_i|h)$ values in the error calculation (which sometimes fell outside the $[0, 1]$ range).

I provided the EM algorithm with the true number of latent states in the system and ran it for 200 iterations or until convergence. I ran 50 independent trials and averaged the results. For the suffix-history algorithm, I tried three choices for the free parameter θ, and the best value for each domain is presented. The parameter θ determines how conservatively the algorithm estimates the rank of a matrix of estimated predictions (which is used when searching for core tests). The error values for the reset and gradient algorithms are the values reported in their respective publications (James and Singh (2004) and Singh et al. (2003)).

Figure 2.1 presents the results from these experiments. The different plots correspond to different domains. James and Singh (2004) did not report the amount of training data they used for their experiments with the reset algorithm, so the x-axis has no meaning for that series.

The first comparison to note is between the EM algorithm for learning a POMDP model and the other three algorithms, which learn linear PSR models. The POMDP models learned by EM have higher error in every domain than the linear PSRs from the gradient, reset, and suffix-history algorithms. While in some domains, EM learns

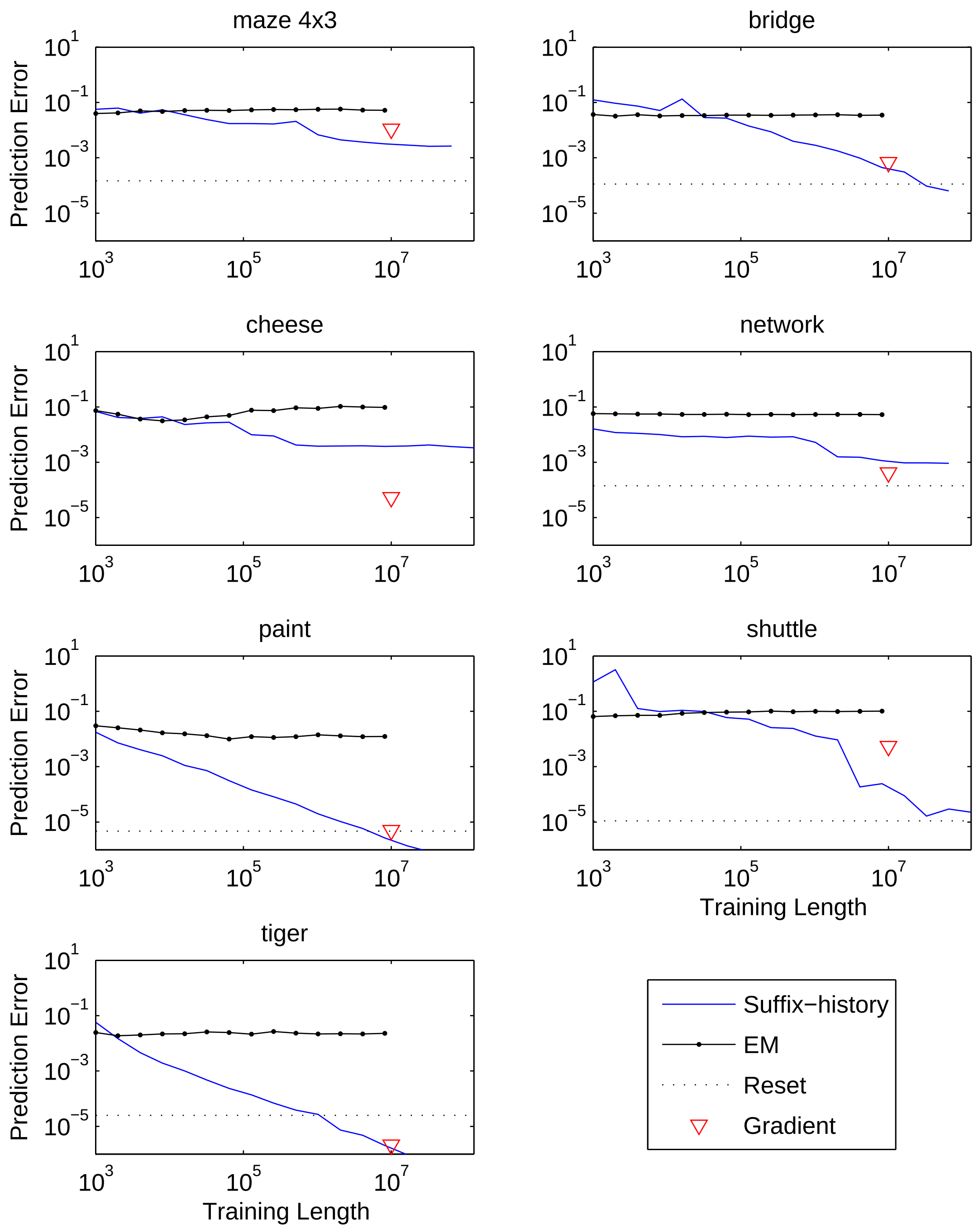


Figure 2.1: Prediction error of linear PSRs learned by the suffix-history algorithm. The horizontal dotted line represents the reset algorithm's results; the amount of training data used for those results was not reported, so the x-axis has no meaning for that series. For the cheese domain the error of the reset algorithm is lower than 6×10^{-6}. The error of the linear PSRs generally decreases as the amount of training data increases.

a more accurate model than suffix-history for the smaller training lengths, the error of the EM models does not decrease very much as the amount of training data increases. This is in contrast to suffix-history, which learns increasingly accurate linear PSR models as the amount of training data increases. For the larger amounts of training data, suffix-history outperforms EM in each of the domains.

Among the linear PSR learning algorithms, suffix-history outperforms the gradient algorithm on all but the cheese and network domains; this is investigated more thoroughly below. This achievement of suffix-history is in spite of the fact that the gradient algorithm is provided with a set of true core tests, while the suffix-history algorithm must learn a set of core tests from the data. It is also worth noting that the suffix-history algorithm achieves error as low or lower than the reset algorithm on four of the seven domains, empirically justifying the approximation of treating suffixes of the experience trajectory as independent trajectories.

Thus, the suffix-history algorithm can learn linear PSRs that have comparable error to linear PSRs learned by other algorithms, despite the fact that the other algorithms either are given a set of core tests (instead of learning them) or assume that the agent has the ability to reset the system in order to get multiple trajectories of experience. Furthermore, the linear PSRs learned by suffix-history are more accurate than POMDP models learned by EM.

Finding enough core tests: Nevertheless, in some domains, the relative performance of the suffix-history algorithm is better than in other domains. For example, in the network and cheese domains, the suffix-history algorithm is not as accurate as the gradient algorithm. In these domains and the maze 4x3 domain, the suffix-history algorithm does not achieve error as low as does the reset algorithm. Again, the reset and gradient algorithms are given multiple trajectories of experience

Table 2.2: Core Test Search Statistics. The *Asymptote* column denotes the approximate asymptote for the percent of required core tests that the suffix-history algorithm found. The *Training* column denotes the approximate smallest amount of training data at which the algorithm achieved the asymptote value.

Domain	$\|Q\|$	*Asymptote*	*Training*
Tiger	2	100 %	4000
Paint	2	100 %	4000
Cheese	11	82 %	32000
Network	7	43 %	2048000
Bridge	5	100 %	1024000
Shuttle	7	100 %	1024000
Maze 4x3	10	90 %	1024000

and a full set of core tests, respectively, so it is expected that they would outperform suffix-history. However, it is worth searching for a general principle to determine when suffix-history performs better in relation to the other algorithms.

The relative performance of the suffix-history algorithm seems to be determined by whether or not it finds enough core tests. Generally speaking, the algorithm found a greater percentage of the required number of core tests as the amount of training data increased. Table 2.2 lists the approximate asymptote of the percent of required core tests found as the training size increases, as well as the amount of training data at which the asymptote was reached. When comparing with the error plots in Figure 2.1, 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. In the cheese domain, where suffix-history did not find all the core tests and fell well short of the gradient algorithm's performance, I performed a separate experiment in which suffix-history was given the number of core tests in the true model but not the tests themselves. This target number of core tests was used as the stopping criterion for the iterative core search procedure instead of the usual stopping criterion (i.e., stopping when the estimates for the system-dynamics matrix rank were identical on two successive iterations). The graph for the cheese

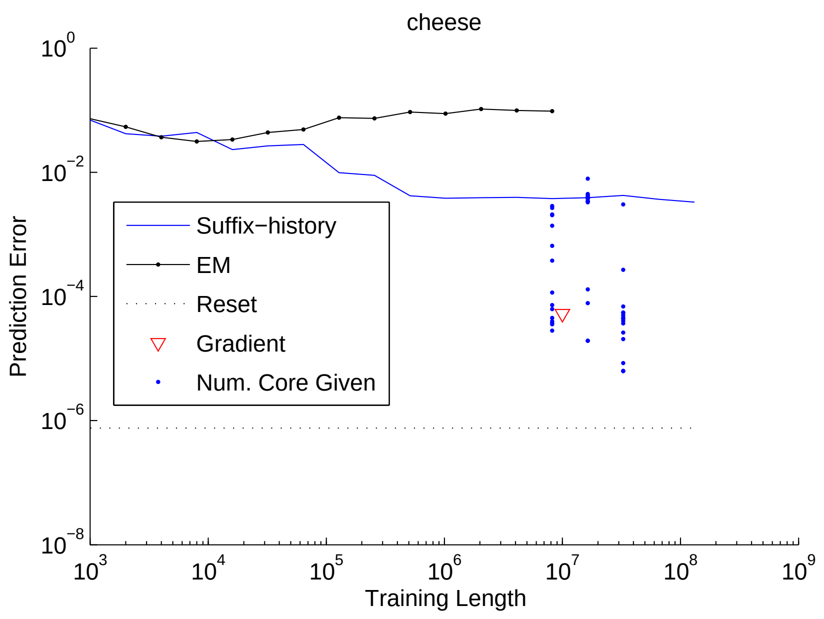


Figure 2.2: Prediction error of linear PSRs learned by the suffix-history algorithm for the cheese maze. The horizontal dotted line represents the reset algorithm's results; the amount of training data used for those results was not reported, so the x-axis has no meaning for that series. This plot adds the "Num. Core Given" series to the results from Figure 2.1, for the trials where the suffix-history algorithm was given the *number* of core tests required for a model (but not the tests themselves). That information greatly improved the accuracy of the models.

domain (Figure 2.2) shows that indeed, when given the number of core tests needed, suffix-history performs much better (see the points labeled "Num. Core Given"), comparable with the gradient algorithm.

2.1.3 Summary of Suffix-history Algorithm

The suffix-history algorithm extends the reset algorithm by removing the requirement that multiple trajectories of experience are available. Multiple trajectories will not be available when the agent cannot reset the system to its initial state. For example, an agent may simply be placed in a dynamical system and have one contiguous sequence of experience from which to build a model. The suffix-history algorithm is able to learn a linear PSR from a single sequence of experience.

The suffix-history algorithm estimates parameters for a system with a different initial condition than the system from which the training data was taken. In this section, I have listed several sets of conditions under which the update parameters of a linear PSR are the same in both systems, providing proofs for these theorems in Appendix E.

Empirically, in several domains the suffix-history algorithm learns linear PSRs that have comparable error to linear PSRs learned by other algorithms, despite the fact that the other algorithms are either given a set of core tests (instead of learning them) or assume that the agent has the ability to reset the dynamical system in order to get multiple trajectories of experience. Furthermore, the linear PSRs learned by suffix-history are more accurate than POMDP models learned by EM, even though the EM algorithm is given the number of latent states needed to model the system.

2.2 Bounding the Longest Required Core Test and History

As with the reset algorithm, the suffix-history algorithm iteratively searches tests and histories of increasing length to find core tests and histories. If there exists a set of core tests and core histories of length k or less, then the core search algorithm only needs to complete k iterations. Since each iteration involves several costly singular value decompositions, the number of iterations has a significant practical impact on the time it takes to learn a linear PSR model. The results of this section bound the length of the longest required core history and test; the greater of these two values is also a bound on the number of iterations needed to find core tests and histories.

In addition to bounding the number of core search iterations, bounds on the lengths of the longest required core history and test will also define a finite part of the system-dynamics matrix $\mathcal{D}$ that allows one to build a linear PSR model of the system. In particular, if there is a set of core tests and histories of length k or less, one only needs to estimate predictions for tests of length $k+1$ or less and histories of length k or less. These predictions are all that is needed to solve for the parameters of a linear PSR (cf. stage 3 of the reset algorithm, Section 1.3). Therefore, a model-learning algorithm for a linear PSR will never need to estimate a prediction for a history/test combination of length greater than $2k+1$. For example, this means that the suffix-history algorithm only needs to keep track of the frequency of sequences of length $2k+1$ or less. Given an upper bound on k, the suffix-history algorithm can calculate the frequencies of these sequences *online*, so the amount of memory required will not grow with the size of the training data.

The *primary results of this section* bound the length of the longest required core history and test in terms of the complexity of the system. There are two different complexity measures used here: the value of k for k^{th}-order Markov systems defines how far back in history one must remember in order to predict the future accurately; and the rank of the system-dynamics matrix corresponds to the number of latent states in a POMDP model and the number of core tests required in a linear PSR. Depending upon the dynamical system, either k or the rank of the system-dynamics matrix may be smaller and may constitute the tighter bound.

For k^{th}-order Markov systems, the longest required core test is no longer than k (Theorem 2.7 in Section 2.2.2). The bound for the longest required core history depends upon the initial condition of the system (Theorem 2.10 in Section 2.2.2). In *all systems where* $\text{rank}(\mathcal{D})$ *is some finite value* n (including any k^{th}-order Markov systems), there exists a set of core tests and core histories of length no greater than n. (Theorems 2.4 and 2.5 in Section 2.2.1). These results supplement the result from Littman et al. (2001) that provides a bound of n on the length of the longest required core history or test for POMDP systems with n latent states.

2.2.1 Bound Based Upon Rank

The following theorems (i.e., Theorems 2.4 and 2.5) show that the rank of the system-dynamics matrix $\mathcal{D}$ is an upper bound on the length of the longest required core test and history. I prove these theorems in Appendix F.

Theorem 2.4. *For any system-dynamics matrix of rank n, there exists some set T of core tests such that no $t \in T$ has length greater than n.*

Theorem 2.5. *For any system-dynamics matrix of rank n, there exists some set H of core histories such that no $h \in H$ has length greater than n.*

Wiewiora (2005) presents an independently-obtained proof sketch of the result in Theorem 2.4. The proof I present in Appendix F is more rigorous and follows a different line of argument. That line of argument permits an analogous proof for core histories (i.e., Theorem 2.5).

2.2.2 Linear PSRs for Markovian Systems

For k^{th}-order Markov systems – where the future is independent of the past given the k most recent observations – I prove an additional bound on the length of the longest required core history and test. The bound leverages the following property of $\mathcal{D}$ for Markovian systems:

Lemma 2.6. *For a k^{th}-order Markov system, any column t of $\mathcal{D}$ is a non-negative scalar multiple of a single column for a test t' of length no more than k.*

Proof. If t is itself of length no more than k, then the theorem trivially holds. Otherwise, let t' be the k-step prefix of t, such that $t = t'a_{k+1}o_{k+1} \ldots a_{k+j}o_{k+j}$. Then for any history h, $p(t|h) = p(t'|h)p(a_{k+1}o_{k+1} \ldots a_{k+j}o_{k+j}|ht')$. By the Markov property, the last term $p(a_{k+1}o_{k+1} \ldots a_{k+j}o_{k+j}|ht')$ is equal to $p(a_{k+1}o_{k+1} \ldots a_{k+j}o_{k+j}|t')$, which is a scalar that is independent of h. Thus the t column of $\mathcal{D}$ is $p(a_{k+1}o_{k+1} \dots a_{k+j}o_{k+j}|t')$ times the t' column of $\mathcal{D}$. $\square$

Theorem 2.7. *For a k^{th}-order Markov system, there exists a set of core tests Q such that no test in Q has length greater than k.*

Proof. Let T be the set of all tests of length k or less. Then T are core tests (though they may not be a minimal set), because any column of $\mathcal{D}$ is a linear combination of the T columns (Lemma 2.6). $\square$

The following bound on the required length of core histories uses the fact that the predictions from many different histories are exactly the same in a k^{th}-order Markov system. In particular, any two reachable histories with the same length-k suffix will have identical predictions (i.e., identical rows in $\mathcal{D}$). A history h is defined as "reachable" if $p(h|\emptyset) > 0.0$.

One might think that all reachable histories of length k or less — call them H_k — would constitute a set of core histories because the row in $\mathcal{D}$ for any history longer than k would be identical to the row for its length-k suffix. However, this is not the case, because a sequence h may be the length-k suffix of a reachable history $h'h$ even if h is not itself a reachable history. In this case, the $h'h$ row of $\mathcal{D}$ is not necessarily identical to the row for any history in H_k. Furthermore, in some systems, the $h'h$ row of $\mathcal{D}$ can be linearly independent of the rows for H_k, precluding H_k from being a valid set of core histories. (A simple example of such a system is a Markov chain with $n > 2$ states that starts in state s^1 and deterministically transitions from state s^i to state s^{i+1} until it reaches an absorbing state s^n.) Thus, some systems require core histories that are longer than k time steps.

I call a sequence h "possible" if there is some history h' such that $p(h|h') > 0.0$.

Note that all reachable histories are possible sequences, but not the other way around. The variable τ^* in the following definition captures the amount of time that must pass until all possible length-k sequences have been possible from a reachable history.

Definition 2.8. For a k^{th}-order Markov system $\mathcal{D}$, let T_k be the set of length-k tests that have non-zero probability of success in at least one history. Let τ^* be the smallest integer such that, for each $t \in T_k$, there is some history h_t of length no greater than τ^* such that $p(t|h_t) > 0$.

The following lemma proves that the worst-case value for τ^* is the rank of the system-dynamics matrix.

Lemma 2.9. *For a k^{th}-order Markov system $\mathcal{D}$, the value of τ^* (as defined in Definition 2.8) is less than or equal to* $\text{rank}(\mathcal{D})$.

Proof. The proof is by contradiction. Let n be the rank of $\mathcal{D}$. Suppose there exists some t of length k such that $p(t|h) = 0.0$ for all $|h| \leq n$ but there is some h^* with length greater than n such that $p(t|h^*) > 0.0$. Let K be a set of core histories of length n or less; Theorem 2.5 (Section 2.2.1) proves that such a set exists. Since K are core histories, there is some vector w_{h^*} such that $p(t|h^*) = w_{h^*}^{\top} p(t|K)$. However, by assumption, $p(t|K)$ is a vector of 0's and $p(t|h^*) > 0.0$, so there can be no such w_{h^*}. Thus, there is no length-k sequence t that is impossible from all histories of length n or less but is possible from some history of length greater than n. Then $\tau^* \leq n$ by definition. $\square$

The following theorem uses the variable τ^* to bound the length of core histories for k^{th}-order Markov systems.

Theorem 2.10. *For a k^{th}-order Markov system $\mathcal{D}$, there exists a set of core histories K such that no history in K has length greater than $\tau^* + k$.*

Proof. Let K be all reachable histories of length $\tau^* + k$ or less. The first step in the proof is to show that every reachable history h not in K has the same length-k suffix as some history in K. Since $h \notin K$, it will have some length $|h| > \tau^* + k$, so one can break h into a length-k suffix t and the remainder h' (i.e., $h = h't$). Because h is reachable, the sequence t is a member of the set T_k of possible length-k sequences that is used to define τ^*. By definition of τ^*, there must be some history h^* of length τ^* or less such that h^*t is a *reachable history*. The history h^*t is in K because it has length less than $\tau^* + k$. Since h and h^*t have the same length-k suffix t, this completes the proof that every reachable history not in K has the same length-k suffix as some history in K.

This means that any history's row in $\mathcal{D}$ is identical to the row of some history in K (by the k^{th}-order Markov property). Therefore, K forms a set of core histories (though not necessarily a minimal set). $\square$

If every length-k sequence that can occur is possible from the initial condition of the system, then $\tau^* = 0$, and the bounds on the lengths of core tests and core histories are both equal to k. Depending on the values of τ^*, k, and $\text{rank}(\mathcal{D})$, either Theorem 2.10 or Theorem 2.5 may provide the tighter bound on the length of core histories.

Other Results for Markovian Systems

One might wonder if the parameters of linear PSRs for Markovian systems exhibit any special properties that one could leverage when learning linear PSRs in Markovian systems. The following results disprove the possibility of two types of structure: a full set of core tests with the same action or model parameters that are all non-negative. If there were a full set of core tests with the same action, that would provide additional guidance to the core search procedure of the reset and suffix-history algorithms. The existence of non-negative model parameters might also be useful when learning linear PSR parameters. However, neither of these types of structure is required, even in 1^{st}-order Markov systems (i.e., MDPs), as shown by Theorems 2.11 and 2.12. I prove these theorems in Appendix G.

Theorem 2.7 (Section 2.2.2) proved that there is a set of one-step core tests Q for any MDP. Theorem 2.11 proves that for some systems, any such Q must contain tests with different actions.

Theorem 2.11. *There exist MDP systems where no set of one-step core tests Q exists such that each test in Q contains the same action.*

Theorem 2.12. *There exist MDP systems where, for any set of minimal core tests Q, there is some action a and observation o such that the parameter vector m_{ao} of a linear PSR contains at least one negative entry.*

Theorem 2.13. *There exist* deterministic *MDP systems where, for any set of minimal core tests Q, there is some action a and observation o such that the parameter vector m_{ao} of a linear PSR contains at least one negative entry.*

2.3 Summary

This chapter has presented several of my contributions to the work on linear PSRs. Section 2.1 presented the suffix-history algorithm for learning a linear PSR from a single trajectory of experience, providing evidence that using predictive state can aid in learning accurate model parameters.

Section 2.2 focused on bounding the length of the longest required core test and history, proving separate bounds based upon the rank of the system-dynamics matrix and the order of k^{th}-order Markov systems. These bounds have two immediate practical implications: they define a finite portion of the system-dynamics matrix that an algorithm needs to estimate in order to learn a linear PSR, and they translate directly into bounds on the number of iterations that the reset and suffix-history algorithms need to find core tests and histories.

CHAPTER III

General Principles for Modeling Large Systems Using PSRs

While the PSR approach to learning models of dynamical systems has shown promise, the results presented thus far have been on simple domains. For domains of practical interest, the rank of the system-dynamics matrix is too large for learning and using a linear PSR model (e.g., the rank can grow combinatorially in the number of objects in the system). Despite this fact, the theory of linear PSRs developed in the previous chapters is not wasted. On the contrary, it forms the foundation for the theory of alternative PSR models that I have developed, which I present in Chapters IV, V, and VI.

When dealing with large systems, learning an exact model is intractable. There are several alternatives to an exact model, including

1. a model that can make (close to) exact predictions, but from only a subset of histories;

2. an *accurate, partial* model that can make (close to) exact predictions about a subset of tests that are deemed important; predictions about other tests may never be queried or may be estimated crudely;

3. and an *approximate, full* model that can make approximate predictions about any test.

Requiring a model to make predictions from only a subset of histories (Option 1) can reduce the number of core tests needed for a linear PSR model (e.g., James, Wolfe, & Singh, 2005; Wolfe & Singh, 2006), which also reduces the time needed to learn the model. The hierarchical PSR that I present in Chapter IV is a specific model that exploits this fact.

The two sections in this chapter address the other two options in this list — an accurate, partial model, and an approximate, full model. The results in this chapter apply generally to all PSR models that would embody these techniques, while I present specific PSR models that employ these techniques in the upcoming chapters.

3.1 Partial Models

In some modeling tasks, a *full* model of the system — one that can *accurately* make the prediction for any test from any history — may not be required, but only a *partial* model is needed. I use the term "partial model" to refer to a model that makes accurate predictions for a subset of tests. Such a partial model may be tractable even when an accurate, full model of the dynamical system would be too large or complex to learn or use. The learnability of a partial model will depend upon several factors: the dynamical system itself, the set of tests T for which the partial model should make accurate predictions, and the type of model used as the partial model. The primary result of this section is the following: *for certain types of models, an accurate, partial model is as difficult to learn as an accurate, full model,* for any non-degenerate dynamical system and set of tests T that the agent wants predicted accurately. This is because, for those types of models, any partial model that can accurately make predictions for a set of tests T will also be able to accurately make predictions for *any* test, thereby making it an accurate, full model.

The insight that leads to this result is the following: even though an agent may require that a partial model needs only to make accurate predictions about some tests T, in order to make those predictions accurately at any history, the model will need to be able to maintain accurate state for any history. To maintain accurate state, the model will need to accurately make any predictions that it uses to update its state.[1] It turns out that for certain classes of models, the accurate predictions that are used to maintain accurate state can be used to accurately calculate *any* prediction about the system.

In particular, there are model classes such as the linear PSR and the POMDP model that use the prediction $p(ao|h)$ for the most recent action a and observation o to update their state from history h to history hao. Since the agent could take any action a from any history, *any* prediction of the form $p(ao|h)$ could be used to update the model state.

[1]Even if a model class uses some predictions about a set of tests Z to update its state, it is possible that there exists a model $\hat{\mathcal{M}}$ that makes correct predictions for a subset of tests T while making inaccurate predictions about Z. However, the existence of $\hat{\mathcal{M}}$ would depend upon the particular system one is modeling (except for degenerate cases such as when T contains no tests). Nevertheless, assuming that such a $\hat{\mathcal{M}}$ exists, it must fall into one of the following two cases:

- Despite inaccurate predictions for Z, the state of $\hat{\mathcal{M}}$ is always accurate. This would imply that the state update is actually independent of the likelihood of the last observation (which is the case in some degenerate systems), or that the inaccuracy of each prediction in Z that is used to update state is precisely balanced with inaccuracies in other quantities of the state update (e.g., model parameters). While this latter situation may be possible in some systems, it is a dubious principle upon which to base a model-learning algorithm, particularly when one considers that $\hat{\mathcal{M}}$ must make accurate predictions for the tests T.
- The other possibility is that inaccuracies in the predictions that $\hat{\mathcal{M}}$ makes about Z produce inaccuracies in the state vector of $\hat{\mathcal{M}}$. Again, it would require some system-specific balancing of inaccuracies in the model parameters in order to use inaccurate state to make accurate predictions for T *from every history*.

Since it is unreasonable to presume that there is any model class such that an "accidentally accurate" $\hat{\mathcal{M}}$ exists for any dynamical system, I will assume the following: any model class that uses some predictions Z in its state update cannot accurately predict T unless it also accurately predicts Z.

Definition 3.1. The predictions of the form $p(ao|h)$ for all actions a, observations o, and histories h are all of the predictions that could be made for all the tests of length one, so they are called the "one-step predictions."

The following theorem highlights the fact that accurate one-step predictions can be used to accurately make *any prediction about the system.*

Theorem 3.2. *Any model that can accurately make any one-step prediction from any history can accurately make any prediction about the system.*

Proof. The prediction for an arbitrary test $t = a_1o_1a_2o_2 \dots a_ko_k$ from a history h is the product of predictions for one-step tests from several histories:

$$p(t|h) = \prod_{i=1}^{k} p(a_io_i|ha_1o_1a_2o_2 \dots a_{i-1}o_{i-1}).$$

Thus, if all of the one-step predictions are accurate, then $p(t|h)$ can be computed accurately as well. $\square$

While this result is not new, and in fact is quite intuitive, it is important to state explicitly because it eliminates certain model classes from consideration if one wants to build an accurate, partial model that is smaller than an accurate, full model. In particular, if a PSR model includes a prediction for a test t in its state, then the state update calculation

$$p(t|hao) = \frac{p(aot|h)}{p(ao|h)}$$

uses the one-step prediction $p(ao|h)$ for the most recent action and observation. Using this state update form requires that the model is not partial (i.e., $p(ao|h)$ can always be computed accurately) or is inaccurate. That is, a partial, accurate, PSR model cannot use this state update form, which precludes a linear PSR from being a partial, accurate model.

This technique of dividing by the probability of the most recent observation is not unique to PSR models. The state update of a POMDP model also includes a division by the likelihood of the most recent observation, which implies that a POMDP cannot be a accurate, partial model without also being an accurate, full model.

Therefore, when faced with a large dynamical system for which an accurate, full model is intractable, one must either

- use a model class that does not use one-step predictions in its state update,
- or relax the requirement for accurate predictions

in order to construct a tractable model. This is the case even if one needs the model to make accurate predictions only for certain tests. Chapters IV, V, and VI each present alternative PSR models that use combinations of these two techniques to avoid building a full, accurate model of the system.

3.2 Approximate Predictive State Representation

It is possible to build a predictive state model that uses the one-step predictions about the most recent action and observation to update its state but is smaller than a full model. However, as mentioned in the previous section, such a model would be an approximate model — it would not be able to make the one-step predictions accurately, which would affect the accuracy of its state and any other predictions that it makes. The resulting inaccurate predictions may suffice for an agent's purposes if the inaccuracy is not too great. This section provides a bound on the inaccuracy of an approximate PSR model's state, under certain conditions.

One question of importance when using inaccurate predictions to update the model state is "Will the inaccuracy or error of the approximated state continue to grow larger and larger over time, as the approximations made during the state update compound upon each other?" This section provides conditions under which the answer to this question is "no": the error of an approximate PSR's state will not grow over time but remains bounded. The work described in this section was published by Wolfe, James, and Singh (2008).

For complex systems, the number of simple tests required to form the state of a PSR is prohibitively large, so one must use some approximate, compact representation of the predictive state (e.g., a product of factors). An approximate PSR consists of an approximate state representation and some approximate model $\hat{\mathcal{M}}$ to update the state. To bound the error in the approximate state, one must first specify how the state is updated; the update process is illustrated on the left-hand side of Figure 3.1. The approximate predictive state $\tilde{x}_{\tau-1}$ at time $\tau - 1$ is updated to get $\tilde{x}_\tau$ in two steps. The first step passes $\tilde{x}_{\tau-1}$ through the update process of $\hat{\mathcal{M}}$ for $a_\tau o_\tau$ and yields some predictive state $\hat{x}_\tau$. However, $\hat{x}_\tau$ may not be in the desired compact form, so a second step is needed to map $\hat{x}_\tau$ to a predictive state $\tilde{x}_\tau$ that does have the desired compact form. The function that maps $\hat{x}_\tau$ to $\tilde{x}_\tau$ is called F.

The quality of approximation of a given $\tilde{x}_\tau$ is measured with respect to the true predictive state x_τ at time τ, which is determined by the true PSR model $\mathcal{M}$ (Figure 3.1). The primary result of this section (Theorem 3.6) is a bound on the error of $\tilde{x}_\tau$ that is independent of τ when $\hat{\mathcal{M}}$ is $\mathcal{M}$; of course, selecting an $\hat{\mathcal{M}}$ that is optimized for F will do no worse.

Boyen and Koller (1998) obtained a similar bound for approximate latent-state models (i.e., DBNs), which I use to obtain the bound for PSRs. In order to formally apply the bound for approximate latent state models to approximate PSR models, I define the approximate latent state update, which I follow with an explanation of

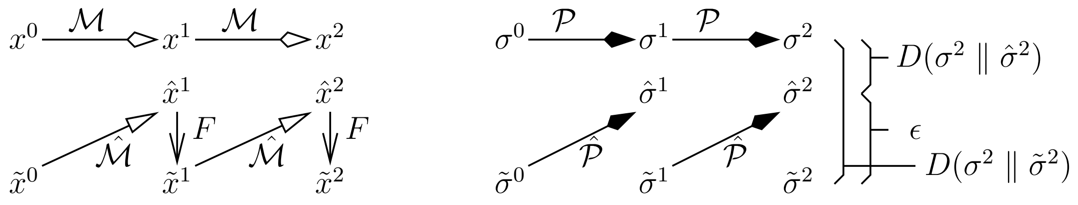

$x^0 \xrightarrow{\mathcal{M}} x^1 \xrightarrow{\mathcal{M}} x^2$
$\hat{x}^1$, $\hat{x}^2$, F, $\hat{\mathcal{M}}$
$\tilde{x}^0$, $\tilde{x}^1$, $\tilde{x}^2$
$\sigma^0 \xrightarrow{\mathcal{P}} \sigma^1 \xrightarrow{\mathcal{P}} \sigma^2$
$\hat{\sigma}^1$, $\hat{\sigma}^2$, $\hat{\mathcal{P}}$
$\tilde{\sigma}^0$, $\tilde{\sigma}^1$, $\tilde{\sigma}^2$
$D(\sigma^2 \parallel \hat{\sigma}^2)$
ϵ
$D(\sigma^2 \parallel \tilde{\sigma}^2)$

Figure 3.1: The approximation process of an approximate PSR. See text for full details. The belief states on the right correspond to the predictive states on the left. Arrow style and label indicates which model or function performs the transformation. The state updates (performed by the models $\mathcal{M}$, $\hat{\mathcal{M}}$, $\mathcal{P}$, and $\hat{\mathcal{P}}$), are each a function of the most recent action and observation (omitted for clarity). Note that each $\tilde{\sigma}_\tau$ is determined by its correspondence with $\tilde{x}_\tau$ and not necessarily as a function of $\hat{\sigma}_\tau$. The errors shown on the right side illustrate an F that incurs error ϵ at each time step.

the relationship between predictive state and latent state models.

Approximate latent state models: Latent-state models represent state as *belief state*, a distribution over a set of latent states. After each time step τ the latent state model $\mathcal{P}$ calculates the new belief state σ_τ from the most recent action and observation and the old belief state $\sigma_{\tau-1}$. One can pass an approximate belief state $\tilde{\sigma}_{\tau-1}$ through the true belief state update $\mathcal{P}$ to get an estimate $\hat{\sigma}_\tau$ of the true σ_τ (Figure 3.1). Boyen and Koller (1998) showed that the error $D(\sigma_\tau \parallel \hat{\sigma}_\tau)$ is a constant factor less than the error $D(\sigma_{\tau-1} \parallel \tilde{\sigma}_{\tau-1})$, where $D(v \parallel w)$ is the Kullback-Leibler (KL) divergence of two stochastic vectors v and w: $D(v \parallel w) \stackrel{\text{def}}{=} \sum_i v_i \frac{v_i}{w_i}$.

Relating predictive state and latent state: I use the error bound on the approximate belief state update to bound the predictive state error, employing a belief state approximation procedure that is *implicitly* defined by the approximate PSR. The proof associates each $\tilde{x}_\tau$ with some belief state $\tilde{\sigma}_\tau$ that implies the same future predictions as $\tilde{x}_\tau$ (Figure 3.1). (In this discussion, I assume the existence of at least one such $\tilde{\sigma}_\tau$ for each $\tilde{x}_\tau$. This is satisfied if each $\tilde{x}_\tau$ lies in the convex hull of possible predictive states.)

The *outcome matrix* U (Singh et al., 2004) relates each $\tilde{x}_\tau$ and $\tilde{\sigma}_\tau$. The U matrix has one row for each latent state and one column for each test in x_τ. The $(i,j)^{th}$ entry is the probability of test j succeeding when starting from latent state i. The relationship between $\tilde{x}_\tau$ and $\tilde{\sigma}_\tau$ is that $\tilde{x}_\tau = U^\top \tilde{\sigma}_\tau$. Because of this relationship, when $\hat{\mathcal{M}} = \mathcal{M}$, the fact that $\tilde{x}_\tau = U^\top \tilde{\sigma}_\tau$ implies $\hat{x}_{\tau+1} = U^\top \hat{\sigma}_{\tau+1}$ (Littman et al., 2001). Thus, each predictive state in Figure 3.1 has a corresponding belief state.

Bounding the approximation error in predictive state: Two steps remain to bound the error in the approximate predictive state $\tilde{x}_\tau$: 1) bound the error in the approximate belief state $\tilde{\sigma}_\tau$ and 2) relate that to the error in $\tilde{x}_\tau$. The bound for $\tilde{\sigma}_\tau$ requires that F meets the following condition:

Definition 3.3. A series $\tilde{x}_0, \tilde{x}_1, \tilde{x}_2, \ldots$ of approximate predictive states *incurs error* ϵ at each time step τ if for all τ, the implicit, approximate belief states $\tilde{\sigma}_\tau$ and $\hat{\sigma}_\tau$ corresponding to $\tilde{x}_\tau$ and $\hat{x}_\tau$ satisfy $D(\sigma_\tau \parallel \tilde{\sigma}_\tau) - D(\sigma_\tau \parallel \hat{\sigma}_\tau) \leq \epsilon$ (Figure 3.1).

The remainder of this section assumes that this condition is met. The following two results are from Boyen and Koller (1998). I use Lemma 3.4 to define the rate of contraction for Lemma 3.5, which provides the bound on the error of $\tilde{\sigma}_\tau$.

(Boyen and Koller (1998)) Lemma 3.4. For any row-stochastic matrix C, let

$$\gamma_C \overset{\text{def}}{=} \min_{i_1,i_2} \sum_j \min(C_{i_1 j}, C_{i_2 j}).$$

Then for any stochastic vectors v and w, $D(C^\top v \parallel C^\top w) \leq (1-\gamma_C) D(v \parallel w)$.

(Boyen and Koller (1998)) Lemma 3.5. For any τ, $E_{h_\tau}[D(\sigma_\tau \parallel \tilde{\sigma}_\tau)] \leq \frac{\epsilon}{\gamma}$, where h_τ is the history through time τ and $\gamma \overset{\text{def}}{=} \min_a \gamma_{\mathcal{T}_a^\top}$, with $\mathcal{T}_a$ being the latent state transition matrix for action a.

I now proceed to define the error of the approximate predictive state $\tilde{x}_\tau$ in terms of the error of the corresponding approximate belief state $\tilde{\sigma}_\tau$. Since $\tilde{x}_\tau$ is not necessarily a stochastic vector, one cannot directly use KL divergence to measure its error. However, $\tilde{x}_\tau$ implies a set of stochastic vectors, one for each unique action sequence of its core tests. To see this, note that the predictions for all of the possible tests with a given action sequence must sum to 1. Thus, one can partition the entries of $\tilde{x}_\tau$ according to their tests' action sequences. One can implicitly add a complement test to each partition, which succeeds if and only if no other test in that partition succeeds (assuming that the action sequence is taken).[2] Each partition (with the implicit complement test) then forms a stochastic vector. For simplicity, I will assume that all tests in the state vector fall in the same partition; the bound on the KL divergence of the single partition is easily extended to each of multiple partitions.

Define the error $\mathcal{E}(\tilde{x}_\tau)$ of $\tilde{x}_\tau$ as $D(y_\tau \parallel \tilde{y}_\tau)$, where y is just x augmented with the complement test prediction. To translate KL divergence of belief states to KL divergence of predictive state, I use the relationship $\tilde{x}_\tau = U^\top \tilde{\sigma}_\tau$, where U is the outcome matrix described earlier. Let V be the matrix formed by adding a column to U for the complement test. Since V is a stochastic matrix, Lemma 3.4 gives a contraction rate γ_V which I use in the bound on $\mathcal{E}(\tilde{x}_\tau)$. This bound (Theorem 3.6) is the main result of this section, showing that the error of the approximate predictive state does not grow without bound over time.

Theorem 3.6. *For any τ, $E_{h_\tau}[\mathcal{E}(\tilde{x}_\tau)] \leq (1-\gamma_V)\frac{\epsilon}{\gamma}$, where $\gamma \overset{def}{=} \min_a \gamma_{\mathcal{T}_a^\top}$.*

Proof. By definition of V, $E_{h_\tau}[\mathcal{E}(\tilde{x}_\tau)] = E_{h_\tau}[D(V^\top \sigma_\tau \parallel V^\top \tilde{\sigma}_\tau)]$. By Lemma 3.4, this quantity is less than or equal to $(1-\gamma_V)E_{h_\tau}[D(\sigma_\tau \parallel \tilde{\sigma}_\tau)]$, which itself is less than or equal to $(1-\gamma_V)\frac{\epsilon}{\gamma}$ because of Lemma 3.5. □

In addition to being the first error bound for approximate predictive state repre-

[2]The complement test is an example of a set test (Wingate, Soni, Wolfe, & Singh, 2007), described in Section A.2.

sentations, Theorem 3.6 has two potential advantages over the corresponding bound on belief state error (i.e., Lemma 3.5). First, the bound itself is smaller by a factor of γ_V. Second, there are a couple of degrees of freedom to obtain a good ϵ: one can choose the latent-state model $\mathcal{P}$ of the system and the belief states $\tilde{\sigma}_\tau$ (such that $\tilde{x}_\tau = U^\top \tilde{\sigma}_\tau$) that give the best ϵ, since $\mathcal{P}$ and $\tilde{\sigma}_\tau$ are just tools for theoretical analysis.

3.3 Summary

This chapter has focused upon two possibilities for scaling PSRs to large systems: building a model that can make accurate predictions about some tests, and building a model that can make approximate predictions about all tests. I presented conditions under which the error in the state vector of an approximate PSR will not grow without bound. In addition, I showed that in order for an accurate, partial model to be more compact than an accurate, full model, it cannot accurately predict every one-step test from every history. In particular, this precludes using a linear PSR as a small, accurate, partial model.

CHAPTER IV

Hierarchical PSRs

As shown in Section 3.1, a straightforward application of linear PSRs will not be a viable model for large systems, even if an agent needs the model to make predictions about only a few tests. Both the size of the linear PSR model and the running time of learning a linear PSR model are polynomial in the rank of the system-dynamics matrix. Because the rank of the system-dynamics matrix can grow combinatorially in the number of latent state variables (e.g., state variables in a DBN model) of a system, learning a linear PSR model will be intractable for systems for which a DBN model has more than a handful of latent state variables. This is also the case with learning a POMDP model, where the size of the latent state space is combinatorial in the number of latent-state variables of a DBN model of the same system.

This chapter introduces *hierarchical PSRs* (HPSRs), a class of PSR models that leverages the advantages of linear PSRs (e.g., learnability of small models via the suffix-history algorithm (Section 2.1)), while avoiding the intractability of learning a single linear PSR to model the whole system. The work described in this chapter was published by Wolfe and Singh (2006).

An HPSR is an aggregate model, made up of several component models that make predictions about the system at different levels of temporal resolution. In particular,

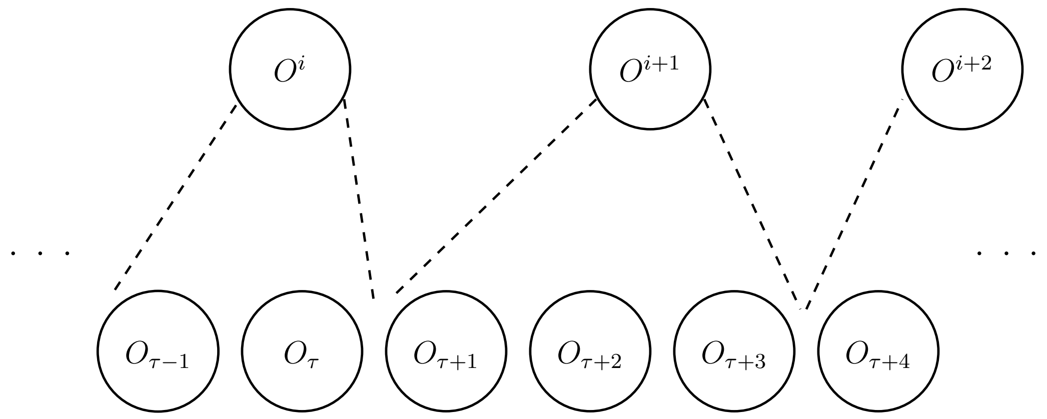


Figure 4.1: An illustration of temporal abstraction. The abstract observations are denoted by the superscripts, while the system observations are denoted by subscripts. Each of the abstract observations O^i, O^{i+1}, O^{i+2} summarizes several observations from the system, shown by the dashed lines.

one of the component models is a model of the entire system at a temporally abstract level. This temporally abstract model can be smaller and easier to learn than a non-abstract model of the entire system.

A temporal abstraction of a system takes blocks of subsequent actions and observations and summarizes them with a single step in the abstract view. For example, in the uncontrolled system depicted in Figure 4.1, the abstract observation O^{i+1} summarizes the observations $O_{\tau+1}, O_{\tau+2}, O_{\tau+3}$. Note that the domain of the temporally abstract observations $\ldots, O^i, O^{i+1}, O^{i+2}, \ldots$ does not need to be the same as that of the system observations $\ldots, O_{\tau-1}, O_\tau, O_{\tau+1}, \ldots$. For instance, a temporal abstraction of a robot navigating in a building may summarize the many camera images received while moving to the end of a hallway with the single, simple observation "at the end of the hall."

4.1 Definitions

I use *options* (Sutton, Precup, & Singh, 1999) to formally define a temporal abstraction of a dynamical system. An option is a macro-action or temporally extended action that specifies a closed-loop way of choosing actions until some (possibly stochastic) stopping condition is met. For instance, an option may specify that a robot is to move forward until its sensors detect some obstacle within one meter of its current position. Options can be useful when an agent is planning which action to take (Sutton et al., 1999), and Perkins and Precup (1999) have shown that options can be useful in transferring knowledge learned in one environment to similar environments. These features of options motivate the development of a PSR model that can make predictions about options.

To distinguish between levels of abstraction in the HPSR, I use the term "primitive" to refer to the actions and observations that an agent takes directly in the dynamical system, and I use the term "option" or "option-level" to refer to the higher-level, temporally abstract actions and observations. I call the tests and histories used in previous chapters "primitive tests" and "primitive histories," to distinguish from option tests and option histories (defined below).

In addition to the primitive actions, the HPSR model assumes that the agent has a set of options Ω. Each option $\omega \in \Omega$ has three components:

- a *policy* that specifies a way of behaving for any history; in general, the policy is a probability distribution over actions for any history
- a *termination condition* that assigns a probability to each history, which is the probability that the option will terminate given that it reached that history
- an *initiation set*, which is 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. In partially observable systems, the latent state of the system is not observable, so the option policy, termination condition, and initiation set are not expressed in terms of latent states, but are instead functions of history. To my knowledge, this is the first definition for options in partially observable settings.

The upcoming definitions of option histories and option tests provide temporally abstract counterparts to the primitive histories and tests whose predictions define a dynamical system. Predictions for these option histories and tests will similarly define a temporally abstract view of the dynamical system.

Definition 4.1. An *option history* is 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 refers to the last observation during the execution of option ω^i.

In this definition, each o^i refers to the last observation during the execution of option ω^i. However, the primary theoretical results about an option-level PSR generalize in a straightforward fashion to allow o^i to be a function of *any part of the primitive sequence during the execution of* ω^i (and not just the last observation), as long as that function has a finite set of outputs (see Section H.1 for more details). In the remainder of this chapter, I assume that o^i is the last observation of ω^i's execution, for concreteness and simplicity.

A *critical assumption* of the HPSR is that the agent always executes options until completion (i.e., options are not interrupted). The HPSR also assumes that the agent is always acting according to some option in the set Ω. This is not a significant restriction, because any primitive action a can be made available to the agent by including an option in Ω that simply executes a and then terminates, thereby creating an option that is equivalent to the primitive action a. However, such options provide no temporal abstraction over the primitive actions, so if Ω includes options that are equivalent to each primitive action, the HPSR will be as complex as an unstructured model (e.g., a linear PSR).

Because the agent is always acting according to some option, the agent can view its history (at any point in time) in two ways:

- a high-level view as an option history (through the last completed option)
- a low-level view as a primitive history.[1]

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. For example, if an agent were told that it was at option history $\omega^1 o^1 \omega^2 o^2$, it would not know the current primitive history, in general.

To make this more concrete, the following example lists some of the primitive-level histories that could correspond to the option history $\omega^1 o^1 \omega^2 o^2$, for certain definitions of the options ω^1 and ω^2. In the list of these primitive histories, a_i denotes the i^{th} action in the set of possible primitive actions (so the i does not reference time). Similarly, o_i denotes the i^{th} observation in the set of possible primitive observations. These notation changes apply only for the discussion of this example.

Suppose that the policy of option ω^1 chooses actions uniformly randomly, and ω^1 deterministically terminates after 2 time steps. Also, let option ω^2 always choose action a_1 and terminate with probability 0.9 at each time step. Finally, let the observations o^1 and o^2 of the option history have the specific values $o^1 = o_3$ and $o^2 = o_5$. Then the option history $\omega^1 o^1 \omega^2 o^2 = \omega^1 o_3 \omega^2 o_5$ could correspond to any of the following primitive histories:

- $\underbrace{a_1 o_7 a_3 o_3}_{\omega^1 o_3} \underbrace{a_1 o_5}_{\omega^2 o_5}$
- $\underbrace{a_3 o_6 a_2 o_3}_{\omega^1 o_3} \underbrace{a_1 o_5}_{\omega^2 o_5}$

[1]I describe an HPSR with two levels of temporal resolution (primitive and options), but it is straightforward to add other levels of temporal resolution, such as options Ω' that have policies over options Ω, that have policies over primitive actions.

- $\underbrace{a_3o_6a_2o_3}_{\omega^1 o_3}\underbrace{a_1o_5a_1o_5}_{\omega^2 o_5}$
- $\underbrace{a_3o_6a_2o_3}_{\omega^1 o_3}\underbrace{a_1o_3a_1o_5}_{\omega^2 o_5}$
- $\underbrace{a_1o_2a_1o_3}_{\omega^1 o_3}\underbrace{a_1o_3a_1o_5a_1o_2a_1o_8a_1o_5}_{\omega^2 o_5}.$

Of course, there are many more primitive histories that could correspond to the given option history; this is just a sampling of the possibilities. Note that the last observation of the first option (which is always at the second time step) is o_3 in each of these examples, because the o_3 in the option history $\omega^1 o_3 \omega^2 o_5$ must match the primitive observation at the last time step of the execution of ω^1. Similarly, the last observation of the second option is always o_5, and the actions of the second option are always a_1. This restriction on the actions of the second option is given by the policy of ω^2.

Thus, knowing that one is in the option history $\omega^1 o_3 \omega^2 o_5$ reveals *some* information about the primitive history, but the exact primitive history is not revealed by knowing the option history (in general). This is how the option-level history provides an abstraction from the primitive-level history. This abstraction does not necessarily lose important information from history; for example, it may not be important what observation occurred during the first time step of ω^1 as long as it terminates in o_3.

The amount of information that an option history hides about the underlying history can be tailored by adjusting the specificity of the options' termination conditions and policies. For example, the fact that the policy of ω^1 allows for any action means that knowing $\omega^1 o_3$ reveals nothing about the actions that happened during the execution of ω^1, but it is known that exactly two actions were taken. On the other hand, the policy of ω^2 only takes action a_1, but it may take that action any number of time steps. Again, the definitions of the options determine what information is abstracted when using option-level histories instead of primitive-level histories.

Just as option histories are an abstraction of primitive histories, option tests are an abstraction of primitive tests.

Definition 4.2. An *option test* is a sequence of options and observations $\omega^1 o^1 \ldots \omega^k o^k$, where the observations in the test correspond to the last observation of each option.

I define a prediction for an option test $t^\omega \overset{\text{def}}{=} \omega^1 o^1 \ldots \omega^k o^k$ from an option history h^ω similar to a primitive prediction: 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.

Option Predictions in Terms of Primitive Predictions

In order to prove that one can build a linear PSR model of a system at an option level, it is important to relate option predictions to primitive predictions. I relate an option-level prediction $p(t^\omega|h^\omega)$ to primitive-level predictions by conditioning upon the primitive sequences that underlie the option history and summing over all possible primitive sequences that underlie the option test. To write an expression for the option prediction $p(t^\omega|h^\omega)$, let $Pr(h^\omega = h|h^\omega)$ be the probability that h is the primitive history that underlies h^ω, and let $Pr(t \wedge t^\omega|h, \pi(t^\omega))$ be the probability that *both t and t^ω concurrently succeed* in the same number of time steps following h, given the policies $\pi(t^\omega)$ of the option test t^ω. Then

$$p(t^\omega|h^\omega) = \sum_{h,t} Pr(h^\omega = h|h^\omega) Pr(t \wedge t^\omega|h, \pi(t^\omega))$$

where the sum is over all primitive histories h and all primitive tests t.

Because the primitive sequences corresponding to a given option history or test could have different lengths, different action sequences, and different observation sequences, the formal relationship between option-level predictions and primitive-level predictions is somewhat tedious. For an intuitive understanding of the HPSR, the details of how option predictions relate to primitive predictions are not strictly necessary, though I use the precise relationship in proofs of some of the upcoming theorems. Section H.1 develops the precise expression for an option-level prediction in terms of primitive-level predictions.

4.2 Extending Linear PSRs to Options

With the definitions of option tests and histories in place, in this section I prove that one can construct an option-level linear PSR that makes accurate predictions about only option tests from option histories. This option-level PSR can provide abstraction over a full system model, making it more compact and easier to learn.

Just as a primitive linear PSR is derived from the primitive system-dynamics matrix, the option-level PSR is derived from an option-level system-dynamics matrix. For a given system and a finite 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. The entry of $\mathcal{D}^{\Omega}$ for a column t^{ω} and a row h^{ω} is just the prediction $p(t^{\omega}|h^{\omega})$. Section H.2 shows that this definition of $\mathcal{D}^{\Omega}$ forms a valid system-dynamics matrix, which leads to the following theorem.

Theorem 4.3. *Let $\mathcal{D}^{\Omega}$ be an option-level system-dynamics matrix for a finite set of options Ω. If $\mathcal{D}^{\Omega}$ has finite rank n, then there exists a finite linear PSR with n core tests that can accurately make any prediction in $\mathcal{D}^{\Omega}$. That is, there is an accurate linear PSR model of the option-level system.*

Proof. This result follows from the fact that there exists a linear PSR with n core tests that can accurately model any system-dynamics matrix of rank n (Singh et al., 2004). The fact that $\mathcal{D}^{\Omega}$ is an abstracted version of another system does not change this fact, because $\mathcal{D}^{\Omega}$ is a valid dynamical system (Theorem H.1 in Section H.2).

Because the number of options is finite and the number of observations is finite, the number of parameters in a linear PSR model of $\mathcal{D}^\Omega$ is also finite. $\square$

The rank of $\mathcal{D}^\Omega$ will play a critical role in learning and using an option-level model of the system, just as the rank of $\mathcal{D}$ is critical when learning and using a primitive-level model of the system. In particular, the rank of $\mathcal{D}^\Omega$ is an upper bound on

- the number of core option tests required in the option-level linear PSR
- the length of the option histories and option tests to be examined in order to find core option histories and core option tests (which are then used to learn the model parameters)
- the number of iterations that the reset or suffix-history algorithms require to find core option histories and core option tests.

This latter point is particularly important in practice, as each iteration of searching for core histories and tests involves several costly singular value decompositions, making it the primary bottleneck for learning a linear PSR with the suffix-history and reset algorithms. Furthermore, the number of parameters of a linear PSR is quadratic in the number of core tests, so the rank of $\mathcal{D}^\Omega$ also affects the size of the option-level model.

Since the option-level model is an abstract model of the system, one would think that it would be smaller and easier to learn than a primitive model. That is, one would think that the rank of $\mathcal{D}^\Omega$ would be smaller than that of $\mathcal{D}$. However, it turns out that the relationship between rank($\mathcal{D}^\Omega$) and rank($\mathcal{D}$) depends upon the system and the set of options Ω. The following theorem proves that it is possible for rank($\mathcal{D}^\Omega$) to be greater than rank($\mathcal{D}$), contrary to first intuition.

Theorem 4.4. *There exists a system $\mathcal{D}$ and a set of options Ω such that the option-level system-dynamics matrix $\mathcal{D}^{\Omega}$ has greater rank than $\mathcal{D}$.*

The proof is by example, in Appendix H. The line of argument is that one can construct an option policy that is a *non-linear* function of the prediction vector of the original system, thus adding to its linear rank. Note that $\mathcal{D}$ always contains enough information to compute $\mathcal{D}^{\Omega}$, even though the linear rank of $\mathcal{D}$ may be less than that of $\mathcal{D}^{\Omega}$.

Despite the fact that rank($\mathcal{D}^{\Omega}$) can be more than rank($\mathcal{D}$) in general, if one is limited to a particular class of options, then rank($\mathcal{D}^{\Omega}$) $\leq$ rank($\mathcal{D}$).

Theorem 4.5. *Let Ω be a set of options such that neither the termination conditions nor the policies of the options in Ω depend upon history prior to the execution of the option. For a primitive system-dynamics matrix $\mathcal{D}$ and an option-level system-dynamics matrix $\mathcal{D}^{\Omega}$ for Ω, rank($\mathcal{D}^{\Omega}$) $\leq$ rank($\mathcal{D}$).*

I prove this result in Section H.4. This relationship between rank($\mathcal{D}$) and rank($\mathcal{D}^{\Omega}$) is compatible with the intuition that the temporally abstract $\mathcal{D}^{\Omega}$ should be no more complex than the full system $\mathcal{D}$. Examples of options with the properties given by Theorem 4.5 are presented in Section 4.4. The remainder of this chapter will assume that the options Ω have these properties.

Even though, in the worst case, rank($\mathcal{D}^{\Omega}$) and rank($\mathcal{D}$) could be equal, Section 4.4 demonstrates empirically that rank($\mathcal{D}^{\Omega}$) can be much less than rank($\mathcal{D}$). In such situations, learning a model of $\mathcal{D}^{\Omega}$ will generally be much more efficient than learning a full model at the primitive level. This is of particular importance when dealing with dynamical systems where rank($\mathcal{D}$) is so large that learning a full model at the primitive level is intractable, but learning the option-level model may be possible. (I present an example of such a system in Section 4.4.) Again, the efficiency of modeling a system-dynamics matrix with small rank is realized in the search for core tests and histories, the length of those tests and histories, the processing of training data, and the size of the linear PSR model (cf. Sections 1.2.1 and 1.3).

4.3 Hierarchical PSRs

A model of $\mathcal{D}^\Omega$ makes 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 make primitive predictions, which the agent may need in order to learn other options, among other things. One possibility for getting primitive predictions would be to learn an option-level linear PSR and a primitive-level linear PSR to make option and primitive predictions, respectively. However, this is harder than just learning a linear PSR of the primitive system to begin with. Since learning a linear PSR of the primitive system will be intractable in many systems of interest, I developed the *hierarchical PSR* (HPSR) model, which takes a different approach to making both primitive and option-level predictions. The remainder of this section describes the HPSR model itself and an algorithm for learning an HPSR.

4.3.1 The HPSR Model

An HPSR consists of several linear PSRs (Figure 4.2): 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.* This limit on the responsibility of $\mathcal{M}^{\omega_i}$ is important, as it can enable tractable learning of $\mathcal{M}^{\omega_i}$ even when learning a full primitive-level linear PSR is intractable (cf. Section 4.4).

The agent uses an HPSR in the following way. At the first time step, it chooses

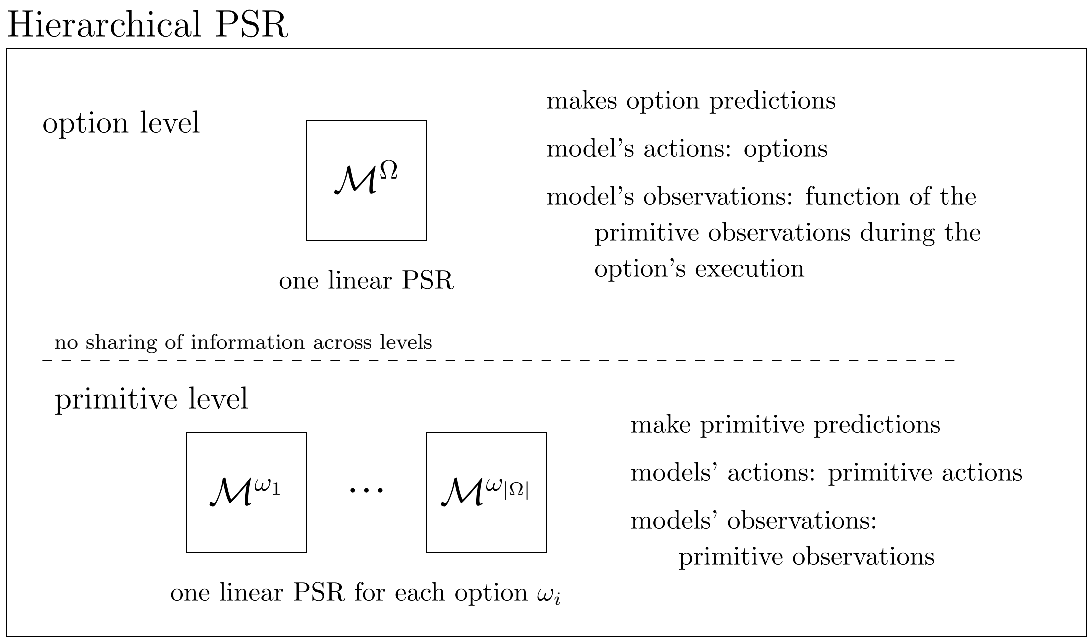

Hierarchical PSR
option level
$\mathcal{M}^{\Omega}$
one linear PSR
makes option predictions
model's actions: options
model's observations: function of the primitive observations during the option's execution
no sharing of information across levels
primitive level
$\mathcal{M}^{\omega_1}$
$\mathcal{M}^{\omega_{|\Omega|}}$
one linear PSR for each option ω_i
make primitive predictions
models' actions: primitive actions
models' observations: primitive observations

Figure 4.2: The component models within an HPSR. Each square represents a linear PSR component model. There is one temporally abstract, high-level linear PSR that makes option predictions, and there is one primitive, low-level linear PSR for each option that makes predictions while its option is executing.

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 agent updates the prediction vector for $\mathcal{M}^{\Omega}$ based upon the last observation of ω_i.[2] 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 $\mathcal{M}^{\omega_j}$. 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 the agent updates its prediction vector at each time step. This process continues as the agent chooses options to execute.

[2]Since the agent only updates the high-level model upon options' termination, it always makes predictions based upon the state at the most recent option's termination. This abstraction is necessary when using a linear PSR as $\mathcal{M}^{\Omega}$ since it cannot update its states' predictions about option tests at every time step without being a full model of the system (Section 3.1).

4.3.2 The HPSR Component Models

Each linear PSR within an HPSR models some system-dynamics matrix, so I describe the linear PSR component models by the system-dynamics matrices that they model. The high-level linear PSR (that makes predictions about options) models the option-level system-dynamics matrix $\mathcal{D}^{\Omega}$ that I described previously.

The primitive-level linear PSR for an option $\omega_i \in \Omega$ models a system-dynamics matrix called $\mathcal{D}^{\omega_i}$, which contains predictions for all primitive tests from a *subset* of primitive histories denoted as $H^{\omega_i}_{trunc}$. The fact that $H^{\omega_i}_{trunc}$ does not include all primitive histories is what underlies the divide-and-conquer approach of the HPSR: a component PSR for a particular option only needs to make predictions from histories in which that option could be executing. This allows the component models to be smaller than a whole-system model, but together the component models can make accurate predictions for the whole system.

The following discussion describes $\mathcal{D}^{\omega_i}$ in more detail. To begin, let $H^{\omega_i}_{trunc}$ be defined as all the possible primitive-level sequences that could be generated by a partial or full execution of ω_i. If an agent picks ω_i as its first option to execute, any history that agent could possibly see up through the termination of ω_i is included in $H^{\omega_i}_{trunc}$. Thus, a model of $\mathcal{D}^{\omega_i}$ can make any prediction during an execution of ω_i that started at the null history of the system. However, in the HPSR, the model $\mathcal{M}^{\omega_i}$ of $\mathcal{D}^{\omega_i}$ should be able to make any prediction during *any* execution of ω_i, regardless of when it started. The critical insight that allows the HPSR to overcome this discrepancy is the following: the *dynamics* of the system during *all* executions of ω_i can be learned separately from the parameters that capture the state of the system at the beginning of a particular execution of ω_i.

In linear PSR terms, one can learn the update parameters (M_{ao} and m_{ao}) that are valid for *all* executions of ω_i; the initial prediction vector is the only part of the model that differs depending upon the history prior to ω_i. When using the HPSR model, an "initial" prediction vector will be needed each time an option begins. When ω_i is the option that is beginning, the prediction vector of $\mathcal{M}^{\omega_i}$ is initialized from predictions supplied by the model $\mathcal{M}^{\omega_j}$ whose option ω_j just ended, as described above.

In order to learn the update parameters that apply for *all* executions of ω_i, one can change the initial condition of $\mathcal{D}^{\omega_i}$ from the null history of the system to an empirical average of all the histories from which ω_i was executed.[3] Specifically, define $f(h)$ as the number of times ω_i was executed from history h divided by the total number of times ω_i was executed. Then the null history row of $\mathcal{D}^{\omega_i}$ is $\sum_h f(h)\mathcal{D}(h)$, where $\mathcal{D}(h)$ is the h row of $\mathcal{D}$. This completely specifies $\mathcal{D}^{\omega_i}$ since the entire system-dynamics matrix can be computed from the null history row. This includes the rows that comprise $\mathcal{D}^{\omega_i}$ (i.e., the rows for the histories $H^{\omega_i}_{trunc}$).

Theorems 2.1, 2.2, and 2.3 (introduced in Section 2.1 in the context of the suffix-history algorithm) establish conditions under which the update parameters learned for a system with one initial condition (e.g., the system $\mathcal{D}^{\omega_i}$) will be valid under different initial conditions (e.g., a particular execution of ω_i). The proofs of Theorems 2.1, 2.2, and 2.3 are given in Appendix E.

(**Theorem 2.1**). Let $\mathcal{D}$ be a system-dynamics matrix with finite rank n that can be modeled by a POMDP P with n hidden states. Let P' be a POMDP model that is identical to P except for a different initial belief state, and let $\mathcal{D}'$ be the system-dynamics matrix generated by the model P'. If the rank of $\mathcal{D}'$ is also n, then any set of core tests and update parameters for either $\mathcal{D}'$ or $\mathcal{D}$ are valid for both $\mathcal{D}'$ and $\mathcal{D}$.

[3]One could potentially use data from options other than ω_i to estimate the entries of $\mathcal{D}^{\omega_i}$ as long as the data was consistent with an execution of ω_i. The details of such intra-option learning are left for future work.

(**Theorem 2.2**). Let $\mathcal{D}$ be a system-dynamics matrix and h^* be a history of $\mathcal{D}$ such that $p(h^*|\emptyset) > 0.0$. Let $\mathcal{D}'$ be a system-dynamics matrix such that $\mathcal{D}'$ has the same dynamics as $\mathcal{D}$, but its null history is identical to history h^* in $\mathcal{D}$. If $\text{rank}(\mathcal{D}) = \text{rank}(\mathcal{D}')$, then any set of core tests and update parameters for either $\mathcal{D}'$ or $\mathcal{D}$ are valid for both $\mathcal{D}'$ and $\mathcal{D}$.

The following theorem states that if one builds a linear PSR of an MDP that has some initial state distribution, then (after the initial time step) that PSR will be an accurate model of the system that starts in any one of those possible initial states (assuming that start state is reachable at *some* point on or after the first time step, which will be true if the start state is ever seen as an observation in the training data).

(**Theorem 2.3**). Let $\mathcal{D}$ be a system-dynamics matrix that can be modeled by an MDP M with states S. Let b_0 be the initial state distribution of M, and let M' be an MDP that is identical to M except for its initial state distribution b_0'. If b_0' satisfies the following property — $\forall s_i \in S$ such that $b_0'(s_i) > 0$, it holds that $b_0(s_i) > 0$ and s_i is reachable at some time $t > 0$ — then core tests and update parameters for $\mathcal{D}$ are valid core tests and update parameters for the system-dynamics matrix $\mathcal{D}'$ given by M'.

4.3.3 Sizes of the HPSR Component Models

As mentioned above, the rank of the system-dynamics matrix that a linear PSR models plays a very important role in the size of the model and the amount of time required to learn the model. The HPSR consists of linear PSRs to model $\mathcal{D}^{\Omega}$ and each $\mathcal{D}^{\omega_i}$, so their ranks will determine the learnability of the HPSR: lower ranks lead to smaller models that are easier to learn. The ranks of $\mathcal{D}^{\Omega}$ and $\mathcal{D}^{\omega_i}$ will depend not only upon the options themselves, but also upon the system $\mathcal{D}$ being modeled. For the option-level model, Theorem 4.5 (Section 4.2) proved that $\text{rank}(\mathcal{D}^{\Omega}) \leq \text{rank}(\mathcal{D})$.

For the primitive-level models, the following theorem establishes an upper bound on the rank of $\mathcal{D}^{\omega_i}$ in terms of a latent-state model of the system. (Even though latent-state models can be difficult to learn, they can be useful for theoretical analysis of the system itself. Results obtained in this way can then be applied to *any* model of the system.)

Theorem 4.6. *For a dynamical system $\mathcal{D}$ that can be modeled by a POMDP with a set of hidden states $\mathcal{S}$, let $\mathcal{S}_i \subseteq \mathcal{S}$ be the set of hidden states in which ω_i could be executing. Then* $\text{rank}(\mathcal{D}^{\omega_i}) \leq |\mathcal{S}_i|$.

Proof. The first step is to show that changing the initial belief state of a POMDP model of $\mathcal{D}$ results in a POMDP model of $\mathcal{D}^{\omega_i}$. Let $b(h)$ be the belief state for history h in $\mathcal{D}$. Define the initial belief state $b'(\emptyset)$ for $\mathcal{D}^{\omega_i}$ as $\sum_h f(h)b(h)$, where the $f(h)$ values are those used in the definition of $\mathcal{D}^{\omega_i}$. The POMDP model with initial belief state $b'(\emptyset)$ will make the same null-history predictions as in the definition of $\mathcal{D}^{\omega_i}$. To see this, note that for any test t, the prediction $p(t|h)$ is a history-independent linear function of the POMDP belief state (Littman et al., 2001). For a given t, let w_t be a vector such that $p(t|h) = w_t^\top b(h)$ for all histories h. Littman et al. (2001) showed that w_t does not depend upon the initial belief state of the POMDP model.

Therefore, the null history prediction for t using the new belief state $b'(\emptyset)$ is

$$\begin{aligned} p'(t|\emptyset) &= w_t^\top b'(\emptyset) \\ &= w_t^\top \left(\sum_h f(h) b(h) \right) \\ &= \sum_h f(h) w_t^\top b(h) \\ &= \sum_h f(h) p(t|h). \end{aligned}$$

This is equal to the null-history prediction given by the definition of $\mathcal{D}^{\omega_i}$. Since the null-history predictions completely define the system, using the initial belief state $b'(\emptyset)$ results in a valid model of $\mathcal{D}^{\omega_i}$.

Let $b'(h)$ be the belief state of the POMDP model at history h when starting with initial belief state $b'(\emptyset)$. That is, $b'(h)$ is the belief state corresponding to the h row of $\mathcal{D}^{\omega_i}$.

The next step of the proof is to show that for any h corresponding to a row of $\mathcal{D}^{\omega_i}$ (i.e., any $h \in H^{\omega_i}_{trunc}$), the belief state $b'(h)$ has non-zero elements only for states in which ω_i could be executing after exactly $|h|$ time steps. The proof is by induction on the length of h. For the base case $h = \emptyset$, the belief state $b'(\emptyset)$ is the convex sum of several belief states $b(h)$. Even though the sum in the definition of $b'(\emptyset)$ is over all histories, the $f(h)$ values are only non-zero for histories where ω_i was initiated. Therefore, each $b(h)$ corresponding to a non-zero $f(h)$ will only have non-zero elements for states in which ω_i could begin (i.e., states in which ω_i could be executing after exactly $|h| = 0$ time steps).

For the inductive step, I begin with the fact that $b(hao)$ is proportional to $\mathcal{O}_{a,o}\mathcal{T}_a b(h)$, using the POMDP notation of Section 1.2.2. Thus, for any s^* such that the s^* element of $b(hao)$ is non-zero, it must be the case that the s^* element of $\mathcal{T}_a b(h)$ is also non-zero (because $\mathcal{O}_{a,o}$ is a diagonal matrix). The s^* element of $\mathcal{T}_a b(h)$ is equal to $\sum_s \mathcal{T}(s, a, s^*)[b(h)]_s$, where $[b(h)]_s$ is the element of $b(h)$ corresponding to s. In order for the sum to be non-zero, there must be some s such that $\mathcal{T}(s, a, s^*) > 0$ and $[b(h)]_s > 0$, because all of the $\mathcal{T}(s, a, s^*)$ and $b(h)$ values are non-negative. By inductive hypothesis, the fact that $[b(h)]_s > 0$ implies that ω_i could be in state s after $|h|$ time steps. Because this theorem is only concerned with $hao \in H^{\omega_i}_{trunc}$, the option ω_i has some non-zero probability of taking action a from history h, which means there is some chance of ω_i being in state s^* after $|hao|$ time steps. This completes the inductive step.

The remainder of the proof is similar to the bound on rank($\mathcal{D}$) for POMDPs given by Singh et al. (2004). Let B be the $\infty \times |\mathcal{S}|$ matrix such that the j^{th} row is $b'(h_j)$, where h_j is the history for the j^{th} row of $\mathcal{D}^{\omega_i}$. Note that the non-zero entries of these belief states are only going to be in the columns of B corresponding to the latent states $\mathcal{S}_i$ in which ω_i could be executing (as shown above). Thus, the rank of B will be no more than $|\mathcal{S}_i|$.

Let T be the set of all tests and let $p^\top(T|s_i)$ be the row vector of probabilities that each test in T succeeds from hidden state $s_i \in \mathcal{S}$. Let U be the $|\mathcal{S}| \times \infty$ matrix formed by stacking the $p^\top(T|s_i)$ vectors for each $s_i \in \mathcal{S}$. Then $\mathcal{D}^{\omega_i} = BU$, by conditioning upon the hidden state at each history. Thus, rank($\mathcal{D}^{\omega_i}$) $\leq$ rank(B) $\leq |\mathcal{S}_i|$. $\square$

Therefore, if an option ω_i only traverses a subset of the latent state space, then the rank of the system-dynamics matrix for that option will depend only on the size of that subset, rather than the size of the whole latent state space (which determines the rank of the primitive system-dynamics matrix). So one can use options (in conjunction with an HPSR) to restrict the portion of the latent state space that one wishes to model, resulting in smaller primitive-level models within the HPSR.

For many systems, there is some POMDP model of the system with the same number of latent states as the rank of $\mathcal{D}$. In such systems, the rank of $\mathcal{D}^{\omega_i}$ is no more than $\mathcal{D}$ (in the worst case, where the option ω_i could be in any latent state).

Corollary 4.7. *For a system $\mathcal{D}$ such that $\operatorname{rank}(\mathcal{D}) = n$, if there exists a POMDP model of $\mathcal{D}$ with n latent states then $\operatorname{rank}(\mathcal{D}^{\omega_i}) \leq \operatorname{rank}(\mathcal{D})$.*

Proof. Even if ω_i could be executing in all of the latent states, Theorem 4.6 bounds the rank of $\mathcal{D}^{\omega_i}$ by the total number of latent states n. Since $\operatorname{rank}(\mathcal{D}) = n$, $\operatorname{rank}(\mathcal{D}^{\omega_i}) \leq \operatorname{rank}(\mathcal{D})$. □

Because the representational power of POMDP models is so great, Theorem 4.6 covers the vast majority of systems of practical interest. Nevertheless, I also provide a bound on the rank of $\mathcal{D}^{\omega_i}$ without reference to a POMDP model of the system, using the following lemma.

Lemma 4.8. *Let $\mathcal{D}_i$ be the rows of $\mathcal{D}$ for the histories at which ω_i could be initiated, terminated, or executing. If minimal core tests and update parameters for $\mathcal{D}_i$ are also core tests and update parameters for $\mathcal{D}^{\omega_i}$, then any row of $\mathcal{D}^{\omega_i}$ is a linear combination of rows of $\mathcal{D}_i$.*

Proof. Let Q be a set of minimal core tests for $\mathcal{D}_i$, with parameters m_t for any test t such that $p(t|h) = p^\top(Q|h)m_t$ for any h corresponding to a row in $\mathcal{D}_i$. Let M be the (infinite) matrix with columns corresponding to the m_t vectors for every test. Then the h row of $\mathcal{D}_i$ is equal to $p^\top(Q|h)M$. Since the parameters from $\mathcal{D}_i$ are valid in $\mathcal{D}^{\omega_i}$ (by assumption), the h row of $\mathcal{D}^{\omega_i}$ is equal to $p'^\top(Q|h)M$, where $p'(\cdot)$ are the predictions in $\mathcal{D}^{\omega_i}$.

Let K be a set of core histories for $\mathcal{D}_i$, and let n be the number of core tests Q. Because Q is a minimal set of core tests in $\mathcal{D}_i$, the rows of $p(Q|K)$ form a basis of

$\mathbb{R}^n$, from which one can form the predictions for the core tests in $\mathcal{D}^{\omega_i}$. That is, for any history h, there exists some vector w_h such that $w_h^\top p(Q|K) = p'^\top(Q|h)$. Then the h row of $\mathcal{D}^{\omega_i}$ is equal to

$$\begin{aligned} & p'^\top(Q|h)M \\ =& w_h^\top p(Q|K)M \\ =& w_h^\top \mathcal{D}_i(K) \end{aligned}$$

where $\mathcal{D}_i(K)$ are the K rows of $\mathcal{D}_i$. □

Because $\mathcal{D}^{\omega_i}$ is formed by combining information from parts of $\mathcal{D}_i$, I expect that the condition of this lemma — minimal core tests for $\mathcal{D}_i$ are also core for $\mathcal{D}^{\omega_i}$ — will generally hold true. Then the following theorem provides a worst-case bound on the rank of $\mathcal{D}^{\omega_i}$.

Theorem 4.9. *Let $\mathcal{D}_i$ be the rows of $\mathcal{D}$ for the histories at which ω_i could be initiated, terminated, or executing. If minimal core tests and update parameters for $\mathcal{D}_i$ are also core tests and update parameters for $\mathcal{D}^{\omega_i}$, then* $\text{rank}(\mathcal{D}^{\omega_i}) \leq \text{rank}(\mathcal{D}_i) \leq \text{rank}(\mathcal{D})$.

Proof. The result immediately follows from the fact that each row of $\mathcal{D}^{\omega_i}$ is a linear combination of rows of $\mathcal{D}_i$ (Lemma 4.8), which are rows of $\mathcal{D}$. □

While the theorems in this section provide upper bounds on the ranks of the component models in the HPSR, the upcoming experiments demonstrate empirically that the component model ranks can be much less than that of the original system.

4.3.4 Learning an HPSR

Learning an HPSR consists of learning each of its linear PSR components, each of which models some system-dynamics matrix. As with learning a single linear PSR, it is important to be able to learn the model from a single trajectory of experience, as that may be all that is available to an agent. One can learn the model of $\mathcal{D}^{\Omega}$ by using the suffix-history algorithm (Section 2.1) on the option-level view of the experience. For each $\mathcal{D}^{\omega_i}$ model, one can use the reset algorithm (Section 1.3) because there are multiple executions of each option in the training sequence.[4] Each time an option executes, it generates some sequence of primitive actions and observations; these sequences are the trajectories used by the reset algorithm.

4.4 Experiments

In this section, I present experiments that demonstrate that learning an HPSR from a single sequence of experience can be faster and more accurate than learning a single linear PSR. I performed this evaluation on two domains, one of which is an order of magnitude larger (measured by the rank of the system-dynamics matrix) than the domains on which I demonstrated the suffix-history algorithm for learning linear PSRs (Section 2.1.2). Thus, these experiments represent a step toward modeling complex dynamical systems with PSR models.

Experimental Details: I measured the HPSR's accuracy in making both option and primitive predictions as the amount of training data increases. I measured the amount of training data in primitive-level time steps. I generated both the training and test sequences by having the agent repeatedly choose from the available options uniformly at random. The 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, a^{j+1} is the action taken at time step $j+1$ of the testing sequence, L is the number of primitive actions

[4] As mentioned earlier, even though the different executions of an option ω_i may begin from different states, one can use the trajectories from all executions of ω_i to learn the parameters of the linear PSR $\mathcal{M}^{\omega_i}$, given that the conditions from one of Theorems 2.1, 2.2, or 2.3 are satisfied.

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 j^{th} option, and replacing L with the number of options taken in the test sequence. I used a testing sequence of 10000 time steps, during which the entries of each model's prediction vector were clipped as needed to fall in the range $[0, 1]$ of valid probabilities. (I did not clip the predictions that are measured for accuracy, just the state vector itself.) Both the suffix-history and reset algorithms take a single parameter which tunes how conservatively they estimate the rank of a sub-matrix of $\mathcal{D}$. I ran several trials with a broad range of parameters, and I report the results from the best parameter settings. Finally, because a set of one-step core tests exists for any MDP (Theorem 2.7 in Section 2.2.2), I modified the core test search algorithm to only consider one-step tests.

In these experiments, I provided the options to the HPSR learning algorithm (i.e., the options themselves were not learned). The options used by the HPSR can have a significant impact on the learning time for the component models. The number of latent states in which *any* option could terminate is the most important aspect of the options for learning the high-level model in the HPSR (with lower numbers of states being generally better for learning efficiency). The number of latent states each option could traverse is the most important aspect of the options when it comes to learning the low-level models in the HPSR (cf. Theorem 4.6). Şimşek and Barto (2008) provide an overview of methods for learning options that would lead to efficient HPSR learning (along with being useful for other types of temporal abstraction).

Rooms Domain Details: The first evaluation domain is an MDP grid world

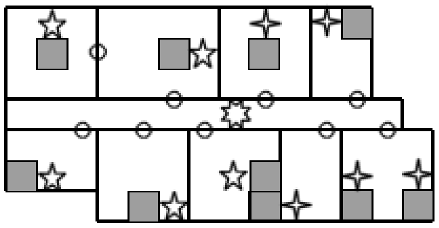


Figure 4.3: The rooms domain. The grid squares are the size of the gray blocks, which are obstacles. The starred locations are the destinations of options, and the circles denote doors. There are four primitive actions (N, S, E, W), which fail to move the agent with probability 0.1, and the observation is the square in which the agent lands.

domain with 78 states shown in Figure 4.3. The 11 starred locations are the states between which the options go. The set of options consisted of 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 Markovian property allowed the algorithm to estimate the prediction $p(a_1o_1 \ldots a_jo_j|hao)$ as 0 if $haoa_1o_1 \ldots a_jo_j$ never occurred and $\hat{p}(a_1o_1|o) \prod_{i=2}^{k} \hat{p}(a_io_i|o_{i-1})$ otherwise, with $\hat{p}(a_1o_1|o)$ being the fraction of times that o_1 followed oa_1 in the training sequence; this is a form of intra-option learning, as experience from one option is used to estimate the behavior of other options.[5]

Rooms Domain Results: 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) than a model of the whole system. As mentioned above, the smaller number of core tests not only leads to smaller models, but it also helps speed the learning process.

Figure 4.4 shows the accuracy results from learning an HPSR model of this domain. The error for the option predictions reaches 0.0, demonstrating that an ab-

[5]As mentioned earlier, the details of intra-option learning for partially observable domains is left for future work.

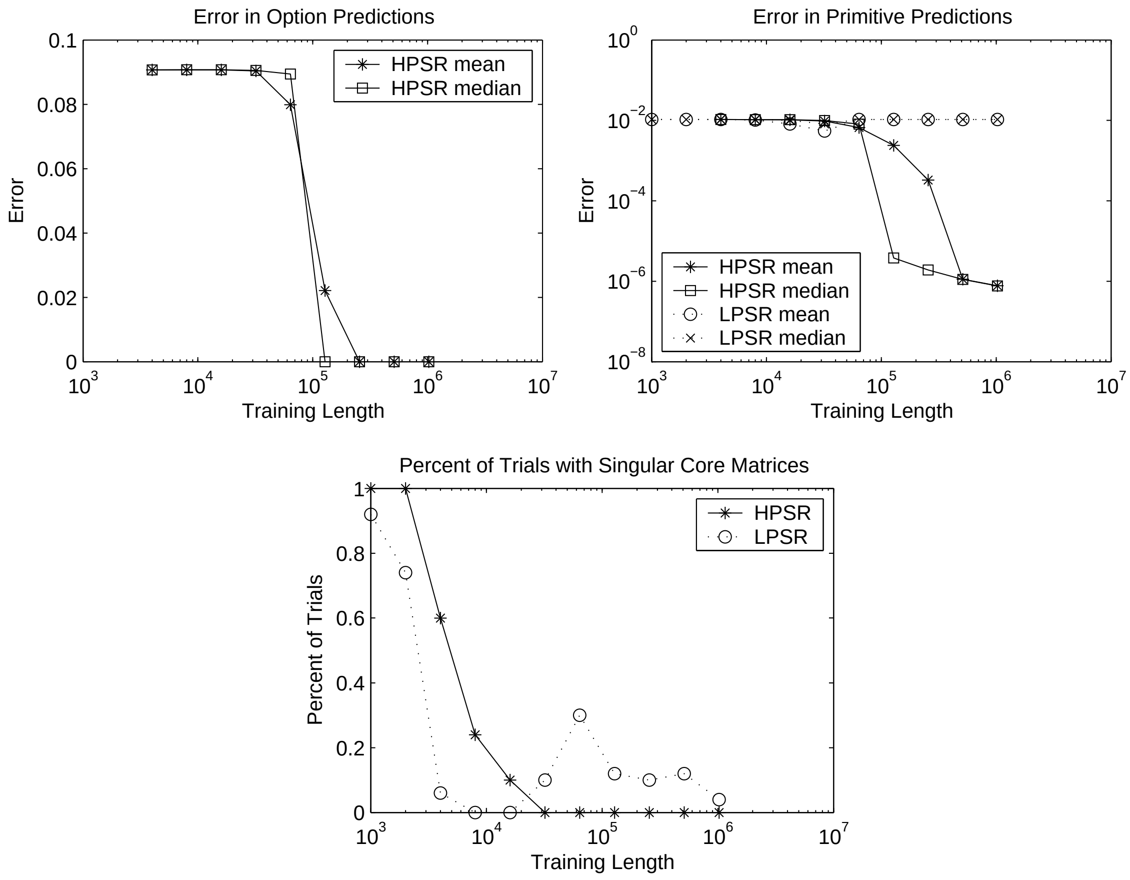


Figure 4.4: Rooms domain results for the HPSR. Means and medians are taken over those of the 50 trials which did not produce singular core matrices, which prevent the calculation of the model parameters. "LPSR" denotes the results from a single linear PSR that models the entire system. The errors in the first two plots are the mean-squared errors of predictions about options and primitive actions, respectively. Note that the temporal abstraction afforded by the option-level model results in perfectly accurate predictions once enough training data is used.

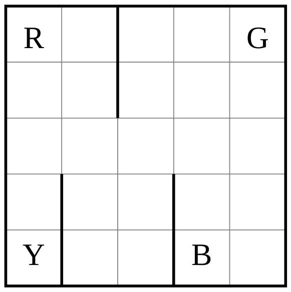


Figure 4.5: The taxi domain. The squares are different positions in which the taxi (i.e., the agent) could be, with walls indicated by the thick lines. The letters denote the colored locations at which a passenger could be picked up or dropped off.

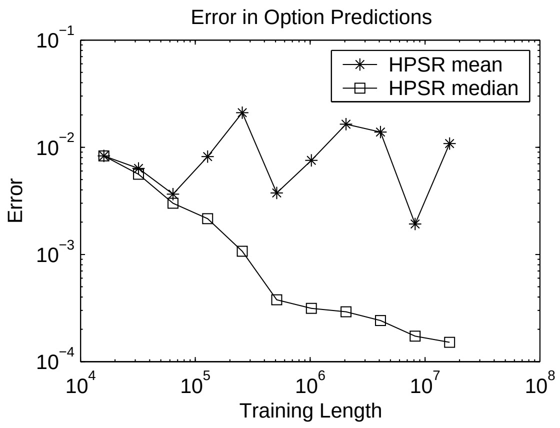


Figure 4.6: Mean squared error of the learned HPSR in making option predictions in the taxi domain. The error is plotted versus the amount of training data (in primitive time steps). The mean/median is taken over 20 trials. The results illustrate that the predictions are more accurate for higher amounts of training data.

stract, option-level model can be easy to learn. For the primitive predictions, I compared the HPSR with 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 Details: The second evaluation domain was a modification of the taxi domain from Dietterich (1998) but 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).

Taxi Domain Results: The taxi domain has 500 states and would require 500 core tests to model with a single linear PSR. However, the high-level model of the system requires 80 core tests, and each primitive model within the 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. In contrast, the learning algorithm for the HPSR was able to learn a model of the system. The total time for learning HPSR models for five different amounts of training data (1024000, 2048000, 4096000, 8192000, 16384000) was approximately four hours on a 2.2GHz desktop PC with 1GB of RAM, including the time to generate the training data, learn the HPSR models, and evaluate the models' accuracy. The accuracy of the option predictions are shown in Figure 4.6. 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).

Summary of Empirical Results

The HPSR framework is designed to model larger dynamical systems than would be possible with a single linear PSR. I verified this ability empirically on the 500-state taxi domain, where the HPSR model makes increasingly accurate predictions as the amount of training data increases. To compare the accuracy of an HPSR model with a linear PSR model, I used a smaller domain, though it was still larger than those previously modeled with linear PSRs. In that rooms domain, the HPSR model was more accurate than a linear PSR model.

In the rooms domain, learning all of the component models of an HPSR was computationally faster than learning a linear PSR for the whole system (which was intractable for the taxi system). The total time for learning PSR models from eleven different amounts of training data (1000, 2000, 4000, ..., 1024000 primitive time steps) was approximately 1 minute for the HPSR models and approximately 15 minutes for the linear PSR models, including the time to generate the training data, learn the models, and evaluate the models' accuracy. 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.

4.5 Related Work

The HPSR is related to hierarchical methods that use different resolutions at different levels in the hierarchy. There has been much work on hierarchical reinforcement learning (Barto and Mahadevan (2003) give a good overview), including a hierarchical POMDP model (Theocharous, Rohanimanesh, & Mahadevan, 2001). However, there has been little work combining PSRs and hierarchical methods. (The multi-mode PSR model described in Chapter VI also combines these two techniques.) The known related work using PSR models with temporal abstraction is that of Sutton et al. (2005), who used temporal difference networks (a form of PSR) to estimate predictions about options. However, they did not show that the options' predictions are actually computable by their model class. In contrast, I proved that a linear PSR can make accurate predictions about a class of options (Theorem 4.3).

Similar to the HPSR's division of primitive histories based upon options, James et al. (2005) used the idea of dividing up all primitive histories into several sub-matrices 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.

4.6 Summary

I proved that a linear PSR can make accurate predictions about a class of options, forming a temporally abstract model of a system (Theorem 4.3). I also proved conditions under which the rank of the temporally abstract system, which affects the learnability and the size of the PSR model, is no more than the rank of the original system (Theorem 4.5). I demonstrated empirically that the temporally abstract system can have significantly lower rank — in fact, the difference is enough that modeling a system at the temporally abstract level was tractable, while modeling the non-abstract system was intractable.

The HPSR is able to make both temporally abstract and primitive predictions. It accomplishes this by combining a temporally abstract model with several primitive-level models, using a divide-and-conquer approach to modeling the system at the primitive level. This division is done according to a set of options, which can implicitly divide a state space for modeling at the primitive level. Empirically, the division results in primitive-level models that are smaller, more easily learnable, and more accurate than a single PSR model of the whole system. This enables the HPSR to model dynamical systems that are too large to model with a linear PSR.

CHAPTER V

Factored PSRs

In order to scale to large systems, models *must* exploit structure in the system; an unstructured model would have too many parameters and be too difficult to learn for large systems. The previous chapter presented the hierarchical PSR (HPSR) as a compact, structured alternative to the unstructured, linear PSR (which does not scale well to large dynamical systems). This chapter presents another class of PSRs called *factored PSRs* that exploit structure to build a compact model. While the HPSRs leveraged a temporal structure in the environment, factored PSRs leverage a structure among the observation dimensions of a dynamical system with vector-valued observations. In particular, the factored PSRs are designed to exploit conditional independence between subsets of observation dimensions. One can use a factored PSR that makes certain conditional independence assumptions even if those assumptions do not strictly hold in the system. In that case, the factored PSR would be an approximate model. One of the features of the factored PSR framework is the ability to trade off model compactness for accuracy by making more or less strict independence assumptions.

Like DBNs (Section 1.2.2), factored PSRs use a factored form for the joint probability distribution of a set of random variables. However, the random variables for the factored PSR are components of observation vectors, whereas the random variables for a DBN also include latent state variables. This means that the *factorization in a DBN is conditioned upon unobservable random variables, while the factored PSR deals only with actions and observations.* This difference turns out to have a significant impact on the sizes of the models' state representations.

With a DBN, the state representation is called the *belief state*, which is a distribution over possible latent state values. The latent state at a given time step is represented as several latent state variables (Figure 1.4), and conditional independence among the latent state variables and the observation variables is what leads to the compactness of a DBN model (cf. Section 1.2.2). However, predictions about latent states or observations many time steps in the future will often be dependent upon *all* of the current latent state variables (Boyen & Koller, 1998). The intuition behind this is the following: consider the i^{th} latent state variable at the current time step. It will affect some subset of latent state variables at the next time step; that subset will affect another subset of variables at the following time step; and so on. Generally speaking, the subset of variables that can be traced back to the i^{th} latent state variable at the current time step will continue to grow as one looks further forward in time. Consequently, in general, the belief state of a DBN — which tracks the joint distribution of the latent state variables — *is not factored*, and has size *exponential* in the number of latent state variables.[1] In contrast, the factored PSR does not reference latent state, but only actions and observations. Thus, its state is *always factored*, as seen in the upcoming model description.

In both factored PSRs and DBNs, *the factored form can significantly reduce the model complexity* (i.e., the number of parameters) when compared with their non-

[1]One can use an approximate, factored belief state in a DBN, even if the true belief state is not factored (cf. Boyen & Koller, 1998).

factored or flat counterparts (linear PSR and POMDP, respectively). In the extreme case, this can be a reduction from an exponential number of parameters to a linear number of parameters (with respect to the number of factors in the model). Not only is fewer parameters better from a storage standpoint, but it can also result in more accurate models: large numbers of parameters relative to the amount of training data can lead to overfitting.

The remainder of this chapter describes factored PSRs, building up the supporting theory and then describing some empirical results on learning factored PSRs from data. Both simulated and real-world data is used in these experiments, which include modeling the complex behavior of cars on a highway. This work has been published by Wolfe et al. (2008).

A factored PSR models a discrete-time dynamical system with a set of discrete actions $\mathcal{A}$ and a set of n observation variables, each of which has a discrete, finite domain. At each time step τ, the agent chooses some action $a_\tau \in \mathcal{A}$ to execute and then receives some n-dimensional observation $o_\tau = [o_\tau^1, o_\tau^2, \ldots o_\tau^n]$. The observations for several time steps are illustrated in Figure 5.1, where capital O's are used to denote random variables, as opposed to lower-case o's, which denote a particular instantiation of the random variables.

5.1 A Simple Approximation

I begin describing factored PSRs with a special case: the most compact factored PSR, which assumes that any two future observation variables in different dimensions are conditionally independent given history. This is a strong assumption that I will rescind later when I present factored PSRs in general, but for now it provides a simple illustration. The conditional independence of the different observation dimensions

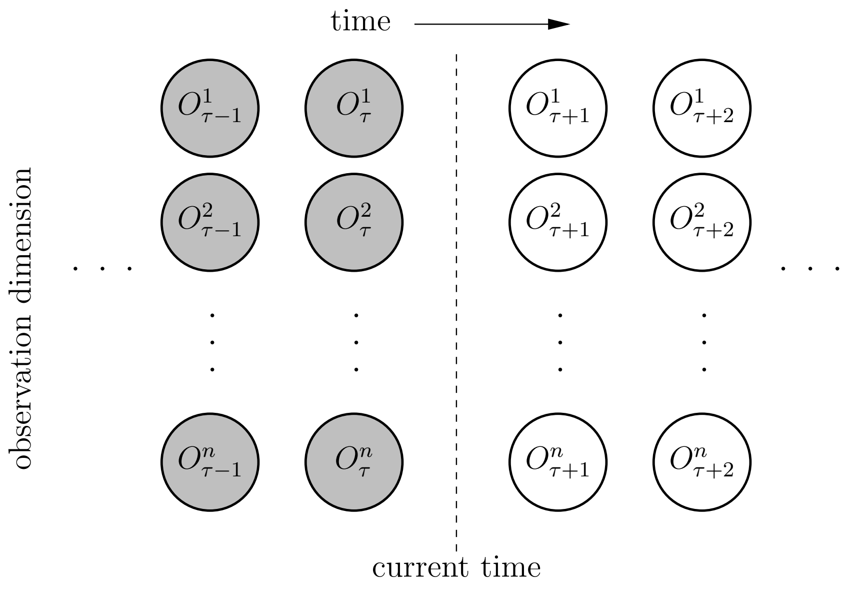


Figure 5.1: The observation variables for each of several time steps in a dynamical system with an n-dimensional observation vector at each time step. Superscript denotes the dimension and subscript denotes time. The shading indicates that the observations of history have been observed, while the unshaded observations of the future are yet to be observed.

means that one can represent predictions about future observation vectors as the product of predictions for each dimension.

To formally state the conditional independence assumption, I define a set of functions $\{g^i : 1 \leq i \leq n\}$ such that g^i selects the i^{th} observation dimension from a history, test, or observation vector. For example, for a test $t = a_1 o_1 \dots a_k o_k$, $g^i(t) = a_1 o^i_1 \dots a_k o^i_k$. The test $g^i(t)$ does not specify values for the full observation vector but only the i^{th} dimension; this leaves the other observation dimensions as wild cards, so $g^i(t)$ is a *set test* (Wingate et al., 2007). Set tests are so named because the prediction for a set test is the sum of the predictions of a set of tests: the set generated by filling in the wild cards with each possible observation value. Thus, the prediction $p(g^i(t)|h)$ for the set test $g^i(t)$ is a marginal probability of the observation variables of $g^i(t)$ conditioned upon the history h (Figure 5.2).[2]

The following equation expresses the assumption that any two future observation

[2]Section A.2 defines set tests and their predictions in more detail.

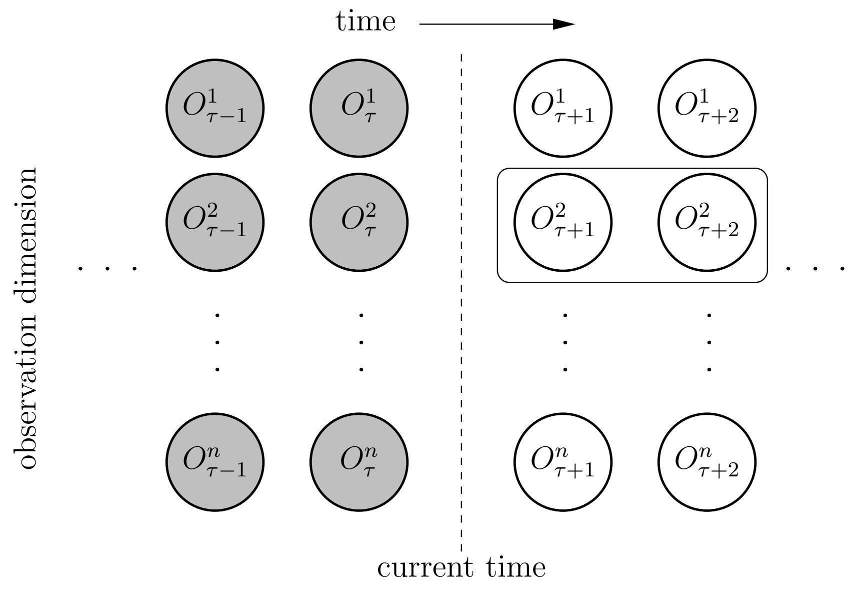


Figure 5.2: The prediction $p(g^2(t)|h)$ for a two-step test t. The shaded nodes indicate conditioning upon the history h, while the box indicates the observation variables that are predicted.

variables in different dimensions are conditionally independent given history:

$$\forall h, t : p(t|h) = \prod_{i=1}^{n} p(g^i(t)|h), \tag{5.1}$$

for any sequence or set test t. This equation suggests a compact state representation: the compact state would consist of the predictions for several tests of the form $g^i(t)$, which could be combined to compute predictions for exponentially many (non-set) tests. The set tests to comprise state would be chosen so that their predictions could be used to compute the predictions for a set of core tests, which form a complete representation of state (Section 1.2.1). In addition to a factored *state representation*, the upcoming Theorem 5.1 will show that a factored *model* can be used as well, under the condition of Equation 5.1. This can result in a significantly more compact model of the system than a full, non-factored model of the entire joint observation space. In particular, since the number of parameters in a linear PSR model is linear in the number of possible observations, modeling a single observation dimension instead of the joint observation space can exponentially reduce the number of parameters in the model.

One can use a compact, factored model because Equation 5.1 implies that the prediction of $g^i(t)$ is independent of part of history: the observation dimensions not selected by g^i.

Theorem 5.1. *The condition* $\forall h, t : p(t|h) = \prod_{i=1}^{n} p(g^i(t)|h)$ *(Equation 5.1) implies* $p(g^i(t)|h) = p(g^i(t)|g^i(h))$ *for all histories* h *and tests* t*.*

Proof. By the definition of prediction, $p(g^i(t)|h) = \frac{p(hg^i(t)|\emptyset)}{p(h|\emptyset)}$. Apply Equation 5.1 to both numerator and denominator to get

$$\frac{\prod_{j=1}^{n} p(g^j(hg^i(t))|\emptyset)}{\prod_{j=1}^{n} p(g^j(h)|\emptyset)} = \frac{p(g^i(ht)|\emptyset)}{p(g^i(h)|\emptyset)} \cdot \prod_{j \neq i} \frac{p(g^j(h)|\emptyset)}{p(g^j(h)|\emptyset)}.$$

This step factored out the case $i = j$ where $g^j(hg^i(t))$ is $g^i(ht)$ from the cases $i \neq j$ where $g^j(hg^i(t)) = g^j(h)$ (i.e., $g^i(t)$ has no overlap with the j^{th} dimension). Then

$$\frac{p(g^i(ht)|\emptyset)}{p(g^i(h)|\emptyset)} \cdot 1 = \frac{p(g^i(h)g^i(t)|\emptyset)}{p(g^i(h)|\emptyset)} = \frac{p(g^i(t)|g^i(h))\, p(g^i(h)|\emptyset)}{p(g^i(h)|\emptyset)} = p(g^i(t)|g^i(h)).$$

□

Combining Theorem 5.1 with Equation 5.1 implies $p(t|h) = \prod_{i=1}^{n} p(g^i(t)|g^i(h))$. Thus, under the conditional independence assumption of Equation 5.1, each observation dimension can be modeled completely independently. This is the basis for the following completely factored model.

A Completely Factored Model: The completely factored model for a dynamical system with n observation dimensions consists of n linear PSRs ($\mathcal{M}^1 \dots \mathcal{M}^n$). Even if all of these models require the same number of core tests m as a full model of the system, the factored PSR model still has significantly fewer parameters than a full model of the system. If k is the maximum number of values an observation dimension could take, then the factored model has $O(nkm^2|\mathcal{A}|))$ parameters, versus

$O(k^n m^2|\mathcal{A}|)$ for a full model. Note that this is a difference between *linear and exponential* in the number of observation dimensions n. Fewer parameters means that the model will (in general) require less data to learn, because there will be less tendency to overfit the training data.

To be a complete model, the factored model must be able to make predictions for any test t. The prediction for a test t is computed by obtaining the prediction for each $g^i(t)$ from the respective $\mathcal{M}^i$, multiplying those predictions together as in Equation 5.1 to get an estimate for $p(t|h)$. The $\mathcal{M}^i$ only makes predictions about observation dimension i, so it can ignore all the observations from history except dimension i (Theorem 5.1).

Learning this factored model consists of learning each $\mathcal{M}^i$. One learns the $\mathcal{M}^i$ by passing only dimension i of an agent's experience into a linear PSR learning algorithm such as the suffix-history algorithm (Section 2.1). Thus, the core tests and the state update procedure for $\mathcal{M}^i$ are restricted to dimension i: at history h, the prediction vector of $\mathcal{M}^i$ is $p(Q^i|g^i(h))$ for core tests Q^i that are set tests in dimension i. To update the state of the factored model, each $\mathcal{M}^i$ computes

$$p(q|g^i(hao)) = p(q|g^i(h)ao^i) = \frac{p(ao^i q|g^i(h))}{p(ao^i|g^i(h))}$$

for each $q \in Q^i$ upon taking action a and seeing observation o.

As mentioned throughout this dissertation, the number of core tests in a linear PSR model — equal to the rank of the system-dynamics matrix that it models — is a crucial factor in determining not only the number of parameters in the model (which is quadratic in the number of core tests), but also how efficiently the model can be learned from data (cf. Section 1.3). The number of core tests for $\mathcal{M}^i$ is rank$(\mathcal{D}^i)$, where $\mathcal{D}^i$ is a system-dynamics matrix with one row for each $g^i(h)$ and one column for each $g^i(t)$; the entry in that row and column is $p(g^i(t)|g^i(h))$. Theorem 5.2 (below) shows that rank($\mathcal{D}^i$) $\leq$ rank($\mathcal{D}$), where $\mathcal{D}$ is the system-dynamics matrix of the whole system. In practice, rank($\mathcal{D}^i$) can be much less than rank($\mathcal{D}$), as illustrated in Section 5.3. The lower rank of $\mathcal{D}^i$ enables the learning of factored PSR models of systems that are too complex for learning unstructured models (e.g., linear PSRs).

5.2 Factored PSRs

The completely factored model presented in the previous section is based upon the strong assumption that any two future observation variables in different dimensions are conditionally independent given history (Equation 5.1). This section describes *factored PSRs*, a generalization of the completely factored model that does not make such a strong independence assumption. Like the completely factored model, a factored PSR consists of n linear PSRs ($\mathcal{M}^1 \ldots \mathcal{M}^n$), one for each observation dimension. A factored PSR is a generalization of the completely factored model because each $\mathcal{M}^i$ is allowed to model any subset of the observation dimensions that includes dimension i, rather than modeling just dimension i. Let f^i be the function that selects the observation dimensions for $\mathcal{M}^i$, defining $f^i(a_1 o_1 \ldots a_k o_k)$ as $a_1 f^i(o_1) \ldots a_k f^i(o_k)$. One can view $f^i(h)$ as selecting somewhere between $g^i(h)$ and the full h.

The learning and state update procedures for a factored PSR with a given $(f^1, \ldots, f^n)$ are the same as for the completely factored model, except each g^i is replaced with f^i. For making predictions, one will still use $\mathcal{M}^i$ to calculate the probability of success for observation dimension i. Unlike the completely factored model, here $\mathcal{M}^i$ models more than just dimension i, so its predictions for dimension i can be conditioned upon other dimensions that it models, improving their accuracy (proven in the upcoming Theorem 5.3).

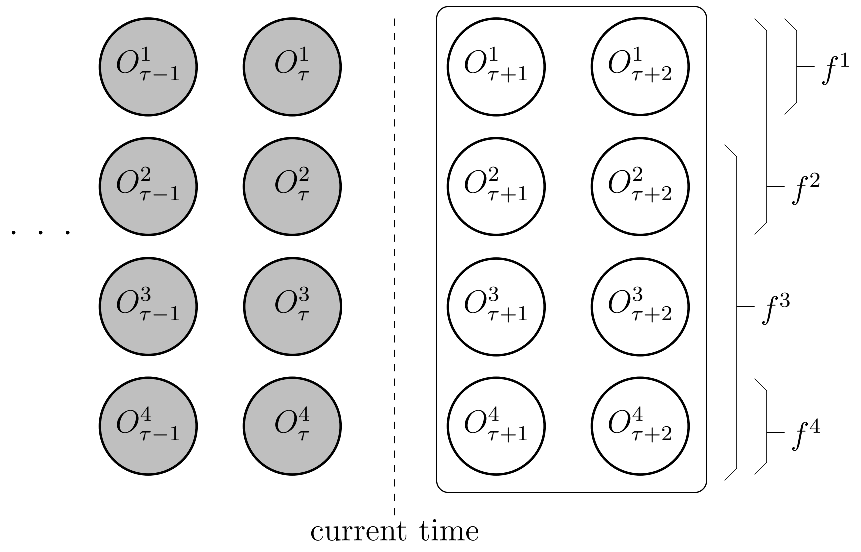


Figure 5.3: The prediction for a two-step test. The box illustrates the observation variables that must be predicted. The scopes of the component models are indicated by the f^i.

In order to be a complete model, the factored PSR must have some way to compute the prediction for any test. As with the fully factored case, a prediction for a test of k time steps is calculated as the product of kn marginal predictions, one for each time step-observation dimension pair. However, because there may be overlap between the observation dimensions of different $\mathcal{M}^i$ models, it is more complicated to compute the prediction for a test with a general factored PSR. The panels of Figure 5.4 illustrate the conditional, marginal predictions that are multiplied together to get the joint prediction of a two-step test, shown in Figure 5.3. If the marginal prediction for $o^i_{\tau+k}$ was conditioned upon all observations through time $\tau + k - 1$ (including those in the history and those prescribed by the test) and the observation variables $o^1_{\tau+k} \dots o^{i-1}_{\tau+k}$, then the product of all such marginals would be the exact joint prediction (by the chain rule of probability). However, the factored PSR makes each marginal prediction conditioned upon a subset of the preceding observation variables, thus making an approximate prediction.

Specifically, to make the prediction for a test $t = a_1o_1 \dots a_ko_k$ from some history

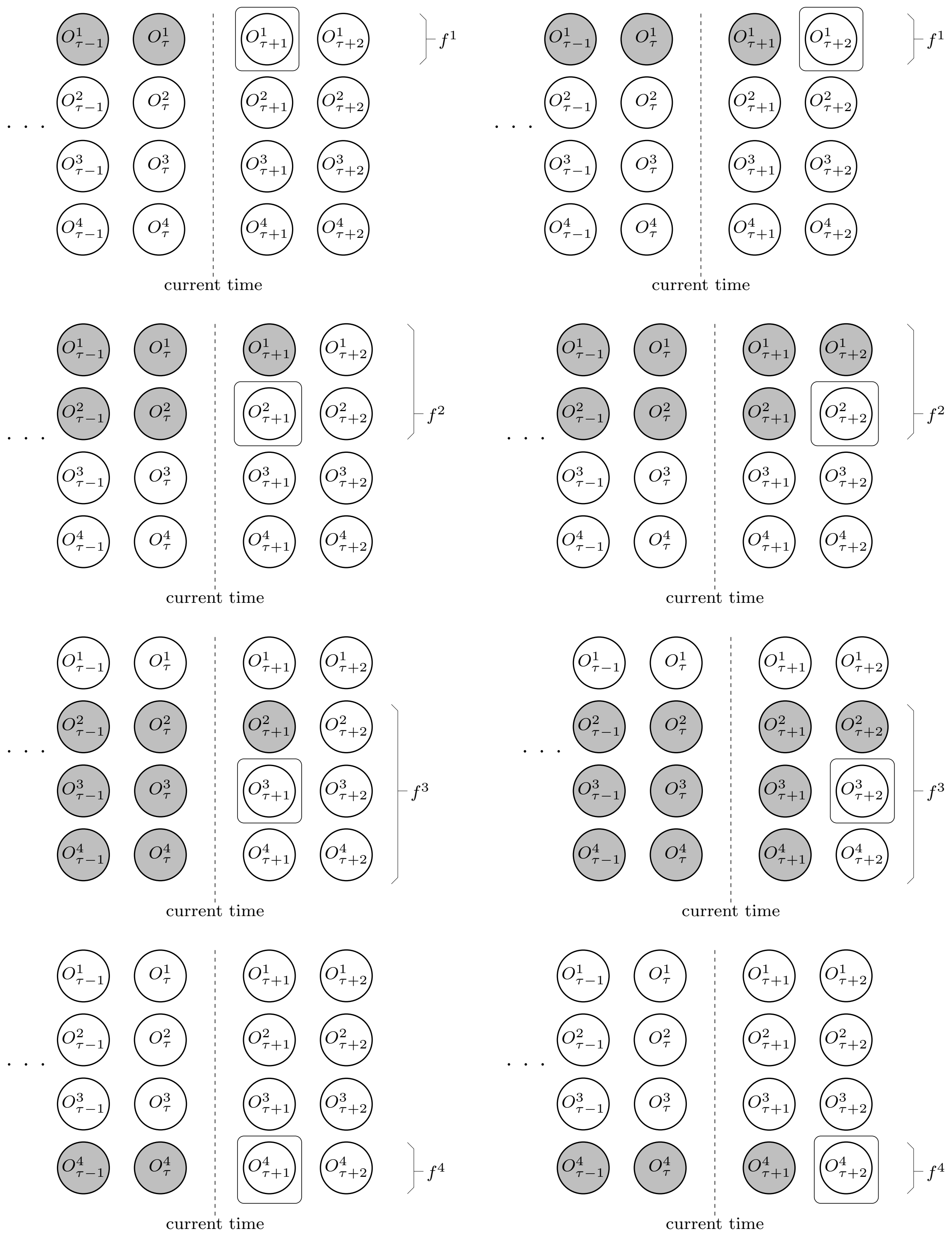

$O^1_{\tau-1}$ O^1_{τ} $O^1_{\tau+1}$ $O^1_{\tau+2}$
$O^2_{\tau-1}$ O^2_{τ} $O^2_{\tau+1}$ $O^2_{\tau+2}$
$O^3_{\tau-1}$ O^3_{τ} $O^3_{\tau+1}$ $O^3_{\tau+2}$
$O^4_{\tau-1}$ O^4_{τ} $O^4_{\tau+1}$ $O^4_{\tau+2}$
. . .
current time
f^1
f^2
f^3
f^4

Figure 5.4: The product of each marginal prediction shown here forms a factored PSR's estimate for the prediction of the two-step test in Figure 5.3. The scope of each component model determines the variables upon which each marginal prediction is conditioned (indicated by the shading), with $\mathcal{M}^i$ making the marginal predictions for dimension i.

h, $\mathcal{M}^i$ will compute a probability for o^i_τ for each $1 \leq \tau \leq k$. The probability for o^i_τ is conditioned upon $f^i(h_{\tau-1})$ and the dimensions of $f^i(o^i_\tau)$ that are less than i, where h_τ is defined as $ha_1o_1 \ldots a_\tau o_\tau$. Formally, this probability is

$$\frac{Pr(f^i([o^1_\tau \; o^2_\tau \; \ldots \; o^i_\tau])|f^i(h_{\tau-1})a_\tau)}{Pr(f^i([o^1_\tau \; o^2_\tau \; \ldots \; o^{i-1}_\tau])|f^i(h_{\tau-1})a_\tau)}.$$

The approximate prediction for $p(t|h)$ from the factored PSR is the product of these probabilities for each i and each τ.

This prediction method comes from applying the chain rule of probability to the set $\{o^i_\tau : 1 \leq \tau \leq k, 1 \leq i \leq n\}$ in ascending order of τ, and in ascending order of i within each τ. The choice of ascending i within each τ is arbitrary; another order will yield a different estimate for $p(t|h)$, in general. One can obtain estimates for $p(t|h)$ using different orderings from the same factored PSR, smoothing those estimates to get an overall prediction.

Choosing f^i: There is a trade-off to consider when choosing the observation dimensions that $\mathcal{M}^i$ should model, which are determined by f^i: having f^i select more of the observation vector can lead to better predictions, but having f^i select less of the observation vector can decrease the model size, which typically makes it easier to learn. One extreme choice is $f^i(h) = h$, which makes $\mathcal{M}^i$ a model for the whole system. The other extreme choice is $f^i = g^i$, which completely ignores information in the observation dimensions other than i. Upcoming theorems prove two results about moving between these extremes: Theorem 5.2 shows that decreasing the scope of f^i will never increase the number of core tests for $\mathcal{M}^i$, and thus never increase the size of the model. Later on in this section, Theorem 5.3 will show that increasing the scope of f^i will not make $\mathcal{M}^i$ less accurate in predicting the future of observation dimension i. To prove these results, I first formalize the notion of *scope*, illustrated in Figure 5.5: *$f^{i'}$ has larger scope than f^i* (written $f^i \subseteq f^{i'}$) if $f^i(o)$ is a sub-vector

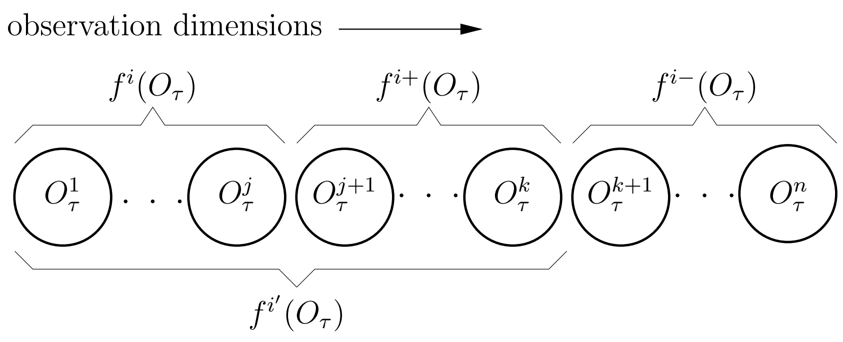


Figure 5.5: How the f functions select from the observation vector at time step τ. The dimensions selected by each function do not need to be contiguous, but are shown that way here for convenience.

of $f^{i'}(o)$ for any o.

Theorem 5.2 proves that decreasing the scope of f^i will never increase the number of core tests for $\mathcal{M}^i$.

Theorem 5.2. *For f^i and $f^{i'}$ with respective system-dynamics matrices $\mathcal{D}^i$ and $\mathcal{D}^{i'}$, if $f^i \subseteq f^{i'}$, then* $\text{rank}(\mathcal{D}^i) \leq \text{rank}(\mathcal{D}^{i'})$.

Proof. This proof describes matrices V and W such that $\mathcal{D}^i = V\mathcal{D}^{i'}W$, which implies $\text{rank}(\mathcal{D}^i) \leq \text{rank}(\mathcal{D}^{i'})$. The matrix V will combine the rows of $\mathcal{D}^{i'}$, yielding an intermediate matrix $\mathcal{X}$ of predictions that has rows for the histories of $\mathcal{D}^i$ and columns for the tests of $\mathcal{D}^{i'}$. The matrix V exists because the row for any history $f^i(h)$ in $\mathcal{X}$ is equal to a linear combination of rows of $\mathcal{D}^{i'}$. This can be seen by conditioning upon the observation dimensions of h selected by $f^{i'}$ but not by f^i, denoted as $f^{i+}(o)$ (Figure 5.5).

The column for any test $f^i(t)$ in $\mathcal{D}^i$ is equivalent to a sum of columns of $\mathcal{X}$ because $f^i(t)$ is a set test in $\mathcal{X}$. It has wild cards for the f^{i+} observations at each time step of t. Therefore, there exists a matrix W such that $\mathcal{D}^i = \mathcal{X}W$, which equals $V\mathcal{D}^{i'}W$. $\square$

As mentioned earlier, applying this theorem with $f^{i'}(h) = h$ proves that $\text{rank}(\mathcal{D}^i) \leq \text{rank}(\mathcal{D})$ for any f^i. (Recall that $\text{rank}(\mathcal{D}^i)$ and $\text{rank}(\mathcal{D})$ are the numbers of core tests needed for $\mathcal{M}^i$ and a full model $\mathcal{M}$, respectively.) Section 5.3 shows that the empirical estimate for rank($\mathcal{D}^i$) can be significantly less than that of rank($\mathcal{D}^{i'}$), which supports making the scope of f^i small. As mentioned in previous sections, the efficiency of modeling a system-dynamics matrix with small rank is realized in the search for core tests and histories, the length of those tests and histories, the processing of training data, and the size of the linear PSR model (cf. Sections 1.2.1 and 1.3).

On the other hand, having f^i with a larger scope can give better predictions. The intuition is that conditioning predictions upon more of history cannot yield worse predictions, in expectation. Specifically, an f^i with larger scope will not increase the expected Kullback-Leibler divergence of the predictions given $f^i(h)$ from the predictions given the full history h. The *Kullback-Leibler (KL) divergence* of two stochastic vectors v and w is defined as

$$D(v \parallel w) \overset{\text{def}}{=} \sum_i v_i \frac{v_i}{w_i}.$$

Note that v and w can each be viewed as a probability distribution over a finite domain. I prove Theorem 5.3 in Appendix I.

Theorem 5.3. *For $f^i \subseteq f^{i'}$ and any subset $\mathbf{X}$ of future observations,*

$$E_H[D(\mathbf{X}|H \parallel \mathbf{X}|f^i(H)) - D(\mathbf{X}|H \parallel \mathbf{X}|f^{i'}(H))] \geq 0$$

where H is the random variable for history of some length.

The theorem compares the difference between the approximation error $D(\mathbf{X}|H \parallel \mathbf{X}|f^i(H))$ when using f^i and the approximation error $D(\mathbf{X}|H \parallel \mathbf{X}|f^{i'}(H))$ when using $f^{i'}$. The fact that the expected value is greater than zero means that the expected error when using f^i is greater, because it has smaller scope.

```
{Initialize the scopes $f^i$ to the completely-factored case}
for $i = 1$ to $n$ do
    $f^i := \{O^i\}$
end for
$\mathcal{M}$ := learn_factored_PSR($f^1, f^2, \ldots, f^n$)
repeat
    any_improvement := false
    for $i = 1$ to $n$ do {loop over the component models}
        best_new_score := $-\infty$
        for all dimensions $1 \leq j \leq n$ such that $O^j \notin f^i$ do
            $f^i_{new} := f^i \cup \{O^j\}$
            $\mathcal{M}_{new}$ := learn_factored_PSR($f^1, \ldots, f^{i-1}, f^i_{new}, f^{i+1}, \ldots, f^n$)
            if evaluation_score($\mathcal{M}_{new}$) > best_new_score then
                best_new_scope := $f^i_{new}$
                best_new_score := evaluation_score($\mathcal{M}_{new}$)
                best_new_model := $\mathcal{M}_{new}$
            end if
        end for
        if best_new_score > evaluation_score($\mathcal{M}$) then
            any_improvement := true
            $\mathcal{M}$ := best_new_model
            $f^i$ := best_new_scope
        end if
    end for
until any_improvement == false
return $\mathcal{M}$
```

Figure 5.6: Local search algorithm for learning component model scopes

A Simple Learning Algorithm for Selecting Model Scopes

As mentioned earlier, one can learn a factored PSR from training data by using existing linear PSR learning algorithms to learn each component model of the factored PSR. To do so, the scopes $(f^1, f^2, \ldots, f^n)$ for each of the n component models must be specified, since they determine what subsets of the observation dimensions each component model should be trained upon. One could manually select the scopes for the component models based upon domain knowledge. Alternatively, this section describes a simple local search algorithm for choosing the component model scopes automatically from the training data.

The algorithm (Figure 5.6) is based upon Theorems 5.2 and 5.3, which state that the simplest factored PSR model will be the one with smallest scopes for its component models, and more accurate component models can be obtained by adding observation dimensions to the scopes. In practice, adding dimensions to the scope will not always lead to a more accurate model, because the additional dimensions may make the model more complex. When trying to learn the more complex model from a finite amount of training data, the accuracy may suffer to the point where the learned component model with smaller scope actually works better within the factored PSR.

Therefore, the algorithm begins with the simplest factored PSR model — the completely-factored case, where f^i just includes dimension i. The algorithm iterates through the component models, testing additions to the scope of each component model in order to improve the accuracy of the overall factored PSR. For each component model, the algorithm tests each observation dimension that is not already in the scope to see if adding it will lead to a more accurate model. The algorithm then adds to that component model's scope the dimension that improves accuracy the most. If none of the additional dimensions improve the accuracy, then the component model's scope remains unchanged. This iterative improvement process continues until the algorithm makes a full pass through all of the component models without changing any scopes.

In order to measure the accuracy of each candidate factored PSR, a portion of the training data is held out from the data that is used to learn the linear PSR component models. Each time a component model's scope is changed, the resulting factored PSR is evaluated on the held-out data, using the average likelihood of the observation at each time step as the evaluation measure (i.e., the "evaluation_score" function in

Figure 5.6). The local search algorithm considers the factored PSR with the higher average likelihood to be the more accurate model. Note that this evaluation measure does not require an accurate model of the data (in contrast to comparing the model's predictions with accurate predictions). This property is important because it allows the learning algorithm to be used even on complex systems where an accurate model is not available.

The "learn_factored_PSR" function in Figure 5.6 consists of learning the component linear PSRs with the given scopes, which can be done using the suffix-history or reset algorithms. Once a component model has been learned for a given scope, that model can be reused whenever a component model with that scope is requested. This reuse significantly reduces the most time-consuming portion of the learning process — learning the component models — because each call to "learn_factored_PSR" (except the initial call) learns at most one component model.

5.3 Experimental Results

This section describes results for learning factored PSRs to model three domains of varying complexity. The data for the first two domains comes from simulations, while the data for the last domain comes from cameras overlooking a section of a highway.

5.3.1 Simulated Domains

The simulated domains provide some empirical illustrations of the factored PSR theorems on two example systems. The first system is smaller in order to permit a comparison with exact predictions, while the second system is an example where approximate models and state representations are required for model learning. Both systems highlight the trade-off between the accuracy and compactness of a factored

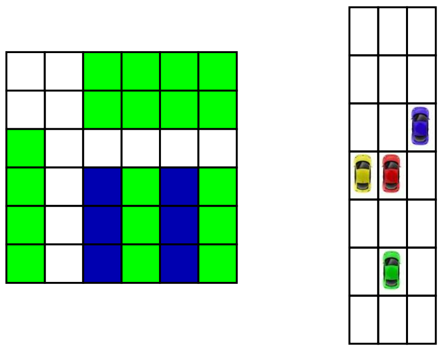


Figure 5.7: Grid world and traffic domains. The agent in the grid world observes the colors of four tiles adjacent to its current position. The traffic system has discrete positions for the cars, illustrated by the grid.

PSR that is made by choosing which observation dimensions each component model selects (i.e., the choice of each f^i). The first few experiments use manually specified scopes for the component models in order to compare a "baseline," completely-factored model that uses $f^i = g^i$ with an "augmented" model that uses an $f^{i'}$ with larger scope. As part of the approximation scheme for the learned models, I clipped the entries of the prediction vector to fall in $[\epsilon, 1]$ after each time step of the testing sequence. To account for any compounding of the approximation error over time, I evaluated each learned model throughout a testing sequence on the order of 10000 time steps.

The first system is a grid world (Figure 5.7) in which the agent can move any of the four cardinal directions and observes the colors of the adjacent floor tiles or a wall in each of the four cardinal directions (a four-dimensional observation). The baseline f^i selects just the i^{th} observation dimension, while the augmented $f^{i'}$ also selects the dimension corresponding to the tile 90 degrees counterclockwise from the i^{th} dimension. I learned each component model $\mathcal{M}^i$ using the suffix-history algorithm (Wolfe et al., 2005) applied to f^i (or $f^{i'}$) of the agent's single experience trajectory.

To evaluate the predictive accuracy of the model, I used the mean squared error of predictions for one-step tests, where the tests evaluated at time τ were those with action a_τ and any observation (as done by Wolfe et al. (2005)). When making joint predictions with the augmented model, I used five random orderings for predicting the observation dimensions to get five estimates for $p(t|h)$. I used the median of these estimates as the model's prediction. The results are shown in Figure 5.8. For the larger training sizes, the augmented model makes better predictions than the baseline model (as suggested by Theorem 5.3). The augmented model also makes better predictions than a single linear PSR for the whole system learned using suffix-history (Figure 5.8). This illustrates that more data can be required to learn a reasonably accurate full model than to learn a reasonably accurate approximate model (which is more compact). The increased accuracy of the augmented model over the baseline model requires more core tests (Figure 5.9), as suggested by Theorem 5.2. This demonstrates that the factored PSR architecture allows one to choose the scopes of the component models to trade off model simplicity and accuracy.

I also learned factored PSRs to model a second dynamical system, which simulates one direction of a three-lane freeway (Figure 5.7). This system simulates the problem faced by an agent in a vehicle whose goal is to track and predict the movements of the other vehicles around it. The agent is given the current positions of all cars within some range, as would be returned from a radar system with multiple sensors. However, some simplifications were introduced. Cars take discrete positions in this system, and the field of view is defined relative to the agent's vehicle: the agent can see all three lanes for three spaces in front and behind its current location. Since this work focuses on modeling rather than control, the agent simply maintains a constant velocity in the middle lane. At each time step, the agent observes only the positions of each other car in its field of view (i.e., velocities are not given as observations).

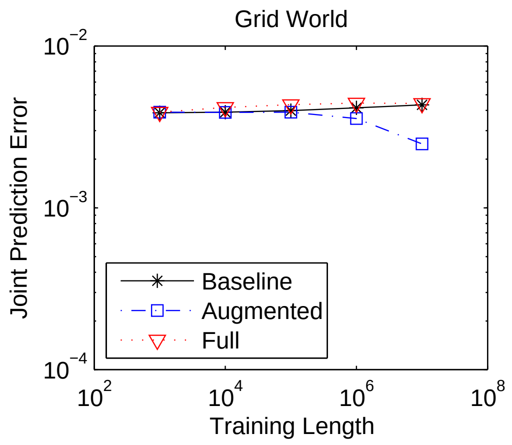


Figure 5.8: Evaluation of the factored PSR on the grid world. The median joint prediction error is plotted versus the training length. The median was taken over ten trials. The "Full" series is a linear PSR for the whole system, the "Baseline" series is a completely-factored PSR, and the "Augmented" series is a factored PSR where each component model had two observation dimensions in its scope. For the larger training lengths, the augmented model has the lowest error.

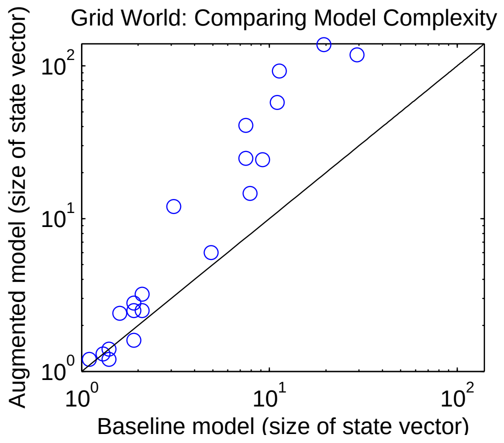


Figure 5.9: Grid world: average number of core tests for the augmented versus baseline models. One point is plotted for each $\mathcal{M}^i$ and training sequence length. The line $x = y$ illustrates that the augmented model usually has more tests.

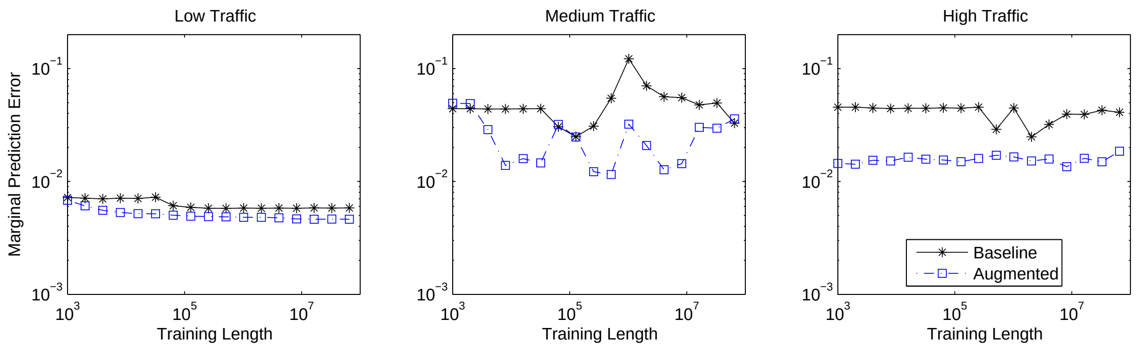


Figure 5.10: Evaluation of the factored PSR on the simulated traffic system. The median marginal prediction error is plotted versus the training length. The median is taken over fifty trials. The augmented model realizes the greatest improvement over the completely-factored (i.e., baseline) model when the most traffic is on the road.

Each car corresponds to a different observation dimension.

Cars enter the field of view stochastically, contingent upon a fixed congestion threshold θ: no new cars enter the field of view if there are already θ cars in the field of view. The low, medium, and high traffic levels discussed below correspond to θ values of two, three, and four, respectively. Each car that enters the field of view is assigned an unused observation dimension that remains unchanged until it disappears from the field of view. Each car has a default velocity that it will maintain unless the car in front of it was going too slowly at the last time step. In that case, the car will change lanes or slow down, depending on traffic in the neighboring lanes.

Learning a single PSR for the whole system was intractable because the prediction vector would have size combinatorial in the number of cars: a conservative lower bound on this size is 125,000 for the high traffic setting. The component models in the factored PSR use much smaller state vectors (cf. Figure 5.11); they automatically exploited the structure among the observation dimensions that results from the spatially localized interaction between cars.

The baseline scope f^i just selects the position of car i; the augmented scope

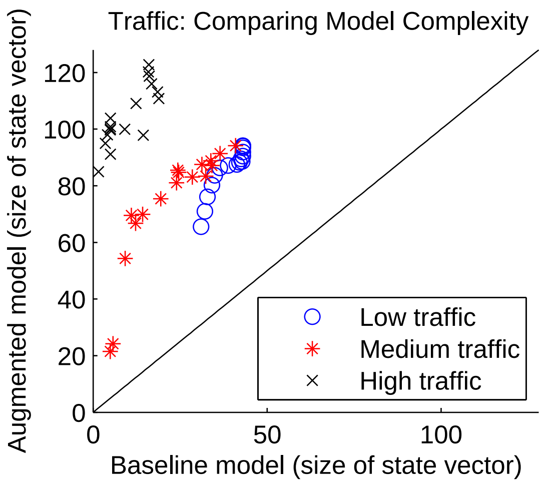


Figure 5.11: Simulated traffic system: average number of core tests for the augmented versus baseline models. One point is plotted for each training sequence length. The marker shape indicates the level of traffic. The line $x = y$ illustrates that the augmented model always has more tests.

$f^{i'}$ also selects the position of the car directly in front of i. A factored PSR for this system consists of multiple copies of one linear PSR $\mathcal{M}$, since each observation dimension corresponds to a car of initially unknown default velocity. Each time a new car i enters the field of view, a new copy $\mathcal{M}^i$ of $\mathcal{M}$ is initialized. When car i leaves the field of view, $\mathcal{M}^i$ is discarded. To train $\mathcal{M}$, I divided the agent's single training sequence into multiple overlapping trajectories: a trajectory begins when a car enters the field of view and ends the first time the car exits the field of view. I took the trajectories for car i and passed them through f^i (or $f^{i'}$), then gave them to the reset algorithm (James & Singh, 2004) to learn the linear PSR $\mathcal{M}$.

Because the joint observation space was so large, I evaluated the accuracy of the model using marginal predictions for each car: at each time τ, I used the component model $\mathcal{M}^i$ to get an estimate $\hat{p}(i, x, \tau)$ of the probability of car i moving to space x at time $\tau + 1$. The error measure is the mean over τ, i, and x of $(\delta_x - \hat{p}(i, x, \tau))^2$, where δ_x is 1.0 if x is the actual next position of car i and 0.0 otherwise. Note that 0.0 error is not always attainable, since not even the full history will always be sufficient to predict the next position of each car. Overall, as with the first example system, the augmented model made better predictions (Figure 5.10) but required more core tests (Figure 5.11).

Learning Model Scopes

In the previous experiments, the scopes of the component models within the factored PSR were manually specified, so the learning algorithm simply consisted of learning the component models with those given scopes. The experiments in this section demonstrate that one can use a simple local search algorithm (Figure 5.6) to *automatically learn the scopes* of the component models from the data.

For these experiments, the "learn_factored_PSR" function in Figure 5.6 learned each component linear PSR using the suffix-history algorithm (Section 2.1). The suffix-history algorithm was executed with several values of its free parameter (for estimating matrix ranks), each of which produced a candidate component model. Each of these candidates was evaluated on the portion of the training data that was also used for evaluating the overall factored PSRs. The candidate component model with the highest marginal likelihood for the observation dimensions in its scope was used as the component model in the factored PSR.

I used the local search algorithm to learn models of the grid world domain for several different amounts of training data. I evaluated the learned factored PSRs on a separate testing sequence (*not* the same data used for evaluating the candidate models in the local search) using the same mean squared error measure that I used in the previous experiments with the grid world. These results are shown in Figure 5.12, which also includes the results from the previous experiments with manually-specified scopes (i.e., the results in Figure 5.8). For the smaller training lengths, the local search algorithm performs comparably to the models with manually specified

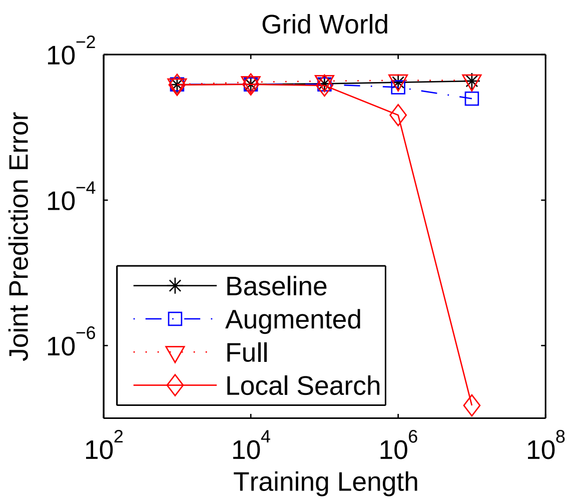


Figure 5.12: Evaluation of the factored PSR with learned component scopes in the grid world. The median joint prediction error is plotted versus the training length. This plot includes the results from Figure 5.8 (with the manually chosen scopes) and adds the results from the "Local Search" algorithm for automatically choosing the model scopes. The local search algorithm leads to significantly more accurate models for the larger training lengths.

scope. For the larger training lengths, the local search algorithm achieves error that is several orders of magnitude better than the models with manually specified scope.

To demonstrate that the local search algorithm is adjusting its scope selection to reflect the amount of training data, I measured the average number of observation dimensions included in the component models of the factored PSR (Figure 5.13). The average scope increases with the amount of training data, since that additional data supports the learning of more complex models (i.e., those with larger scope). Also, this figure shows that the algorithm does not simply exit after creating the initial model (which would have only one dimension per component model). Rather, the algorithm is searching over several candidate models, adding multiple dimensions to the component models.

While the local search algorithm can learn a more accurate model than using a manually specified scope, the process of searching over several candidate factored

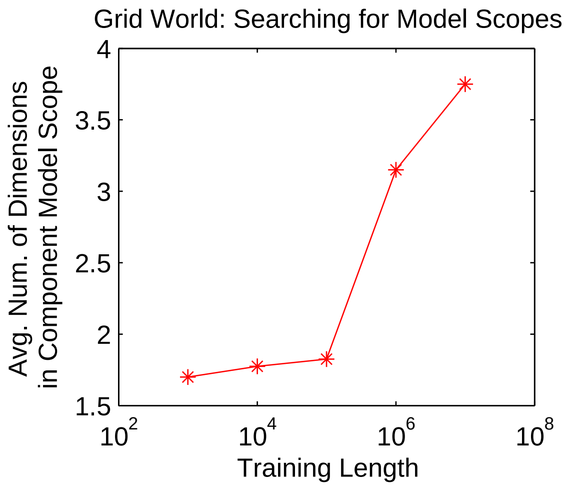


Figure 5.13: The average number of observation dimensions in the component models learned by the local search algorithm. The longer training lengths support the use of component models with larger scope.

PSRs requires additional time. Specifically, for the grid world experiments (running on a 2.2GHz dual-processor machine with 1GB of RAM), learning the baseline factored PSR model (i.e., each component model's scope was manually set to contain only a single dimension) took approximately 15 minutes. The whole local search procedure took approximately 95 minutes when using 1,000 time steps of training data and approximately 255 minutes (4.25 hours) when using 10,000,000 time steps of training data (because the local search performed more search steps with the additional training data).

I also tested the local search algorithm upon the simulated traffic system. For this system, there is only one component model to be learned; different copies of that component model are used to model different cars.[3] I evaluated candidate component models in the local search procedure according to the average likelihood they assign to the modeled car moving to its actual next position. The observation dimensions

[3]Even though the models for the different cars are initialized the same way, as the model for a car receives observations about that car, it will tailor its predictions based upon the behavior of that particular car.

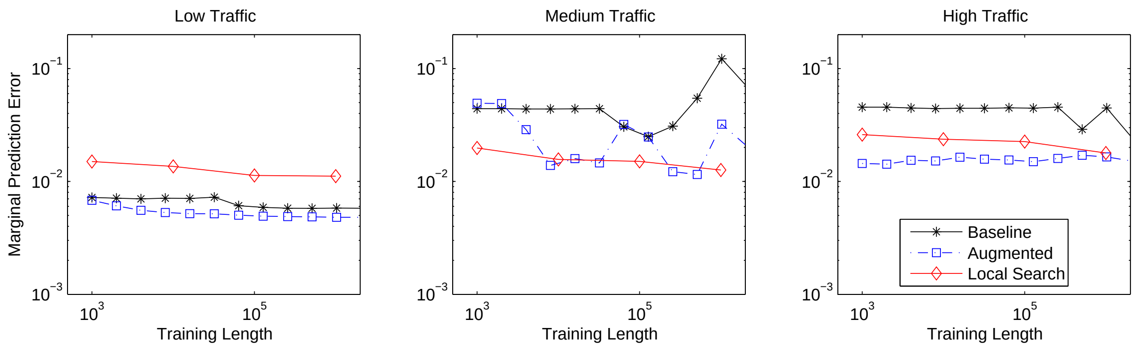


Figure 5.14: Evaluation of the factored PSR with learned component scopes on the simulated traffic system. The median marginal prediction error is plotted versus the training length. This plot includes the results from Figure 5.10 (with the manually chosen scopes) and adds the results from the "Local Search" algorithm for automatically choosing the model scopes. The local search algorithm does well in the medium and high traffic cases. For the low traffic, it may be overfitting when searching for the proper model scopes (see text for discussion).

from which the local search algorithm could select were

- the position of the modeled car (included in the initial/baseline model)
- the position of the closest car in front of the modeled car
- a bit to indicate if there is a car on the left of the modeled car
- a bit to indicate if there is a car on the right of the modeled car.

I ran the local search algorithm upon data from the traffic simulator at each of the low, medium, and high traffic levels. Figure 5.14 compares the error of factored PSRs learned using the local search procedure with the error of the factored PSRs learned in the previous experiments (with manually provided scopes). For the high traffic system, the local search model has error between that of the two models with manually specified scopes, with the local search model's error reaching the better of the comparison models' error for the higher amounts of training data. For the medium traffic system, the local search model performs as well as the better of the comparison models. For the low traffic system, the local search procedure does not

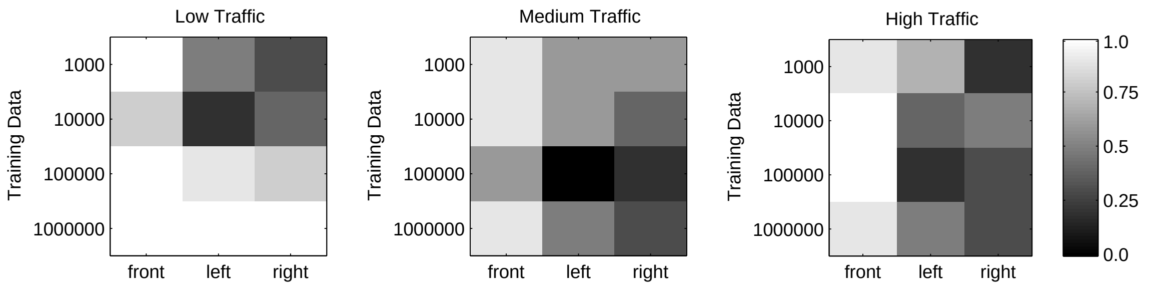


Figure 5.15: The fraction of trials on the simulated traffic domain in which each observation dimension was included in the scope of the model returned by the local search procedure. The "front," "left," and "right" are the observation dimensions for the car in front, to the left, and to the right of the modeled car (whose position is always included in the scope).

perform as well as the comparison models, possibly due to overfitting.

In particular, the training data from the low traffic system will have few examples where cars are side-by-side, so including the observation dimensions that indicate the presence of cars on the side may increase the average likelihood on the validation data but actually decrease model accuracy. Indeed, Figure 5.15 illustrates that in the low traffic case it is not unusual for the local search procedure to include as least one of the "car on left" or "car on right" observation dimensions, which may explain why the local search models did not perform as well as the models with the smaller, manually-specified scopes.

For each of the traffic levels and amounts of training data, Figure 5.15 illustrates the fraction of trials in which each observation dimension was included in the scope of the model. Overall, the most commonly selected observation dimension is the position of the car in front of the one that is being modeled, which fits with the actual decision procedure that occurs in the simulation and with one's intuition about the most important information from history.

In summary, these experiments demonstrate that one can use a basic local search procedure to learn the scopes of the component models in the factored PSR. In order to scale to large numbers of observation dimensions, the algorithm as described in Figure 5.6 would need some simple modifications. In particular, rather than trying all possible dimensions to add to a component model's scope, one could use some domain-specific knowledge (e.g., locality/neighborhood information about observation dimensions) to try only a subset of the dimensions. One could also limit the number of dimensions that are included in any component model's scope. Both of these modifications could reduce the number of component models that need to be learned and evaluated, leading to a significant reduction in running time.

5.3.2 Real-world Traffic Data

The third example domain for the factored PSR experiments is a real-world version of the simulated traffic domain presented above. The data was collected as part of the Next Generation SIMulation (NGSIM) project (U.S. Federal Highway Administration, 2006b) that captures traffic movement along approximately 500 meters of six-lane freeway in the San Francisco Bay area (U.S. Federal Highway Administration, 2006a). This is a rich data set, useful for analyzing many different aspects of traffic movement (e.g., Lu & Coifman, 2007; Brockfeld & Wagner, 2006). (A video is available at `http://www.youtube.com/watch?v=JjxNu2kbtDI`.)

The factored PSR model is built to predict traffic movements relative to a given *reference car* (i.e., the car in which the model would be employed). For these experiments, observations are only possible within some field of view that is defined relative to the front, center point of the reference car. The field of view extends 15 feet on either side, 100 feet behind, and 150 feet in front. Figure 5.16 shows the field of view of some car at one time point of the data. One can see that, even though traffic lanes are clearly present, the cars' positions in the lanes vary significantly.

The factored PSR consists of one component model for each car. The observations

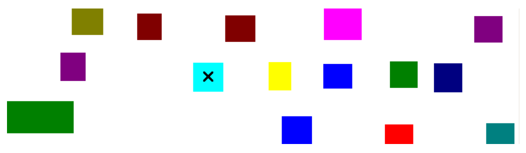
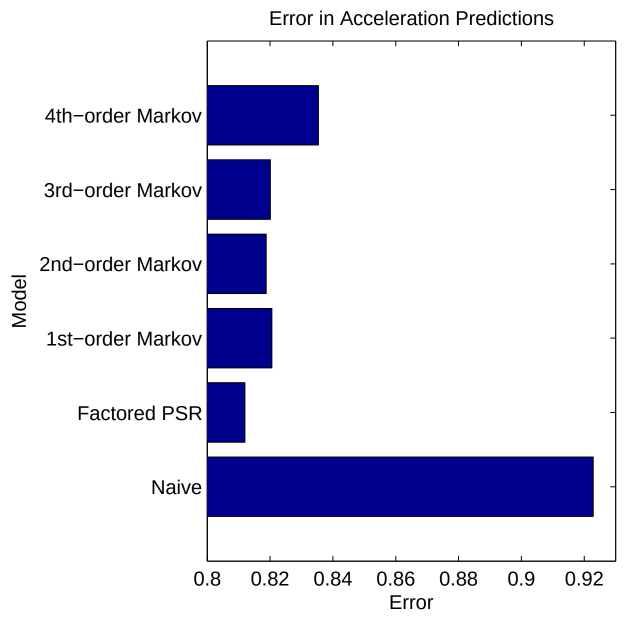


Figure 5.16: A overhead snapshot from a time step of the NGSIM traffic data. The rectangles are cars, and the direction of travel is toward the right of the page. The field of view is defined relative to the car marked by the 'x'.

of the component model were discretized, instantaneous, relative accelerations in the x (side-to-side) and y (forward-and-back) dimensions. Acceleration values were measured in feet per second squared. I discretized the y acceleration into bins of width 1; the data ranged from approximately -10 to 10. I discretized the x acceleration into 3 bins: $(-\infty, -20)$, $[-20, 20]$, and $(20, \infty)$. The threshold of 20 was selected after noting a correlation in the data between lane changes and spikes in x acceleration of magnitude 20 or greater.

The data for training the model consists of trajectories of relative accelerations: for each reference car, one obtains a trajectory of data each time a car passes through the field of view (or if a car enters the field of view and remains there until the end of the data). I used each car in the training data as a reference car, adding all the associated trajectories into the training set for the PSR. I used the NGSIM I-80 traffic data from the 4:00–4:15 time period for training. In both testing and training, I sub-sampled the original files (which have an observation every 1/10 second) to get one observation every second.

For testing, I used a subset of the 5:00–5:15 data from the NGSIM I-80 traffic data. The testing data represents approximately one minute of real time. I evaluated the factored PSR model on each trajectory for each reference car in the testing data. The evaluation uses the same error measure as in the simulated traffic system, except that in this case the model predicts the likelihood of the actual next acceleration (rather



Model	**Error**
Factored PSR	0.81198427
2^{nd}-order Markov	0.81875077
3^{rd}-order Markov	0.82011353
1^{st}-order Markov	0.82056525
4^{th}-order Markov	0.83541472
Naive	0.92289034

Figure 5.17: Comparison of the factored PSR with other models in predicting NGSIM traffic data. The error measure is detailed in the text. Note that 0.0 error is not always attainable, since not even the full history will always be sufficient to predict the next acceleration of each car. Thus, this error measure is solely used for comparing models.

than the actual next position, as used for evaluation with the simulated system). Predictions about position are easily calculated from the predictions of acceleration and the last observed position and velocity.

I compared the factored PSR with two other classes of models. The first is a "naive" model that predicts that the acceleration in one second will be the same as the last observed acceleration. The second class of models are k^{th} order Markov models for $1 \leq k \leq 4$, which were trained on the same data that I used to train the factored PSR.

Figure 5.17 compares the error for each of these models on the testing data. The naive model does much worse than the other models. The poor performance of the fourth-order Markov model is likely due to data sparsity in the training set (i.e., not all length-four histories are seen in the data). The other Markov models and the PSR obtained similar error, with the PSR achieving the lowest error. Learning a reasonably accurate PSR model of a system as complex as highway traffic is yet another step in developing PSR models and learning algorithms that an agent can use to build a model of a complex environment.

5.4 Discussion

Assuming statistical independence is an approximation technique commonly used to make computation tractable (e.g., Engelhardt, Jordan, & Brenner, 2006; Sandberg et al., 2001; McCallum & Nigam, 1998). This work on factored PSRs is the first to use such an independence assumption with predictive state models. The factored PSR model is a first step in using graphical models techniques to scale PSRs to complex systems.

5.5 Summary

This chapter presented the factored PSR model and an algorithm for learning a factored PSR from experience in a dynamical system. The factored PSR is designed to leverage conditional independencies among the observation dimensions. One may assume conditional independencies that do not strictly hold in the system in order to construct a model that is learnable but makes approximate predictions. This property allows one to trade off compactness for accuracy within the factored PSR framework. In particular, for very large systems, one can make independence assumptions that simplify the modeling task to the point of being tractable.

The results in this chapter include a proof that making stronger independence assumptions will not increase the size of the model, and an empirical demonstration that stronger independence assumptions can decrease the model size and complexity significantly. This decrease in size permitted me to learn factored PSR models of systems that were too complex to model with a single linear PSR. Thus, the factored PSR represents progress toward modeling complex dynamical systems with PSR models.

CHAPTER VI

Multi-Mode PSRs

This chapter describes a class of structured PSR models called multi-mode PSRs (MMPSRs) for modeling uncontrolled dynamical systems that switch between several modes of operation. The MMPSR makes predictions conditioned upon the current mode, allowing specialized predictions for each mode. Unlike latent-variable models like hierarchical HMMs (Fine, Singer, & Tishby, 1998), the MMPSR requires that the modes be a function of past and *future* observations. This requirement yields advantages both when learning and using an MMPSR, as I explain throughout this chapter.

The particular type of structure that the MMPSR is designed to exploit — the different modes of operation of the system — distinguishes it from the hierarchical PSR (Chapter IV) and the factored PSR (Chapter V) models described in the previous chapters. Thus, the MMPSR may effectively model systems that are not amenable to either the hierarchical PSR or the factored PSR. I present a detailed comparison of the MMPSR and the other structured PSR models in Section 7.1, after I fully describe the MMPSR model.

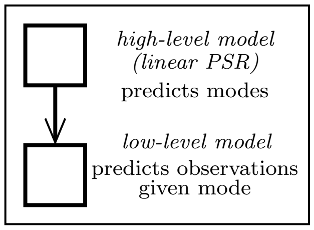


Figure 6.1: The component models in an MMPSR.

6.1 The Multi-Mode PSR (MMPSR)

The MMPSR is inspired by the problem of predicting cars' movements on a highway. One way to predict the movements of a car on the highway would be to determine what mode of behavior the car was in — e.g., a left lane change, right lane change, or going straight — and make predictions about the car's movement conditioned upon that mode of behavior. The MMPSR makes predictions in this way using two component models which form a simple, two-level hierarchy (Figure 6.1). When modeling highway traffic, the high-level model predicts the mode of behavior, and the low-level model makes predictions about the car's future positions conditioned upon the mode of behavior. The remainder of this section formalizes the MMPSR model in general terms, making it applicable to dynamical systems other than highway traffic.

6.1.1 Observations and Modes

The MMPSR can model uncontrolled, discrete-time dynamical systems, where the agent receives some observation O_i at each time step $i = 1, 2, \ldots$. The observations can be vector-valued and can be discrete or continuous. In addition to the observations that the agent receives from the dynamical system, the MMPSR requires that there exists a discrete set of *modes* the system could be in, and that there is some mode associated with each time step. The system can be in the same mode for

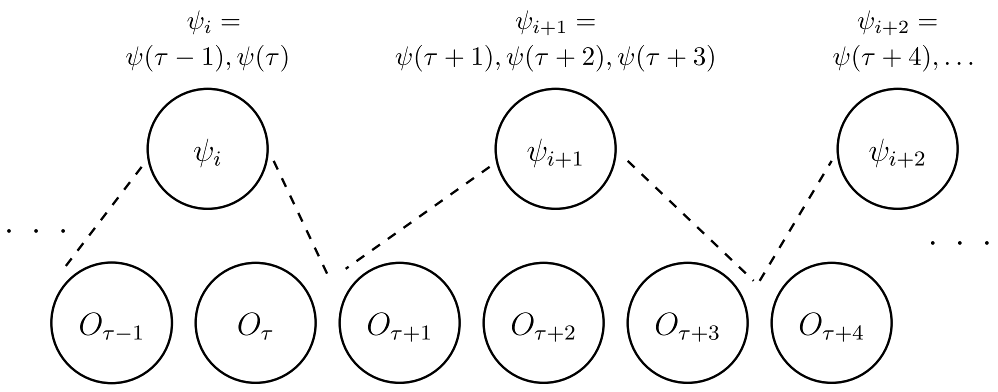


Figure 6.2: How the mode variables relate to the observation variables. The observation at time τ is O_τ, the i^{th} mode seen since the beginning of time is ψ_i, and $\psi(\tau)$ is the mode at time τ.

several time steps, so a single mode can be associated with multiple contiguous time steps (Figure 6.2). I use ψ_i to denote the i^{th} mode since the beginning of time, and I use $\psi(\tau)$ to denote the mode for the τ time step (Figure 6.2).

What distinguishes MMPSR models from hierarchical latent-variable models (e.g., hierarchical HMMs (Fine et al., 1998)) is the fact that the modes are not latent. Instead, they are defined in terms of past and possibly *future* observations. Specifically, the modes used by an MMPSR must satisfy the following *recognizability requirement*: There is some finite k such that, for any sequence of observations $O_1, \ldots, O_\tau, O_{\tau+1}, \ldots O_{\tau+k}$ (for any $\tau \geq 0$), the modes $\psi(1), \ldots, \psi(\tau-1), \psi(\tau)$ are known at time $\tau + k$ (or before).

Definition 6.1. A mode $\psi(\tau)$ is *known at time* τ' (where τ' can be greater than τ) if the definitions of the modes and the observations $O_1, \ldots, O_{\tau'}$ from the beginning of time through time τ' unambiguously determine the value of $\psi(\tau)$.

To reiterate, the recognizability requirement differentiates the MMPSR from hierarchical latent-variable models. If one were to incorporate the fact that modes were recognizable into a hierarchical latent-variable model, one would in effect get

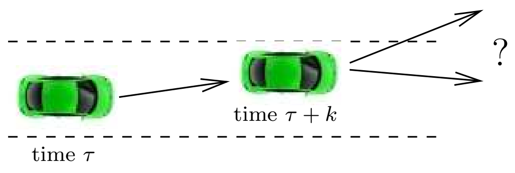


Figure 6.3: A situation in the traffic system where the mode for the last few time steps of history is unknown. After moving from the position on the left of the figure at time τ to the position towards the right at time $\tau + k$, it is unclear if the car is beginning a left lane change or is just weaving in its lane. These two possibilities will assign different modes to the time steps τ through $\tau + k$ (i.e., "left lane change" vs. "going straight").

an MMPSR. The recognizability of the modes plays a crucial role in learning an MMPSR, because the modes for the batch of training data are known. If the modes were not recognizable, one would have to use the expectation-maximization algorithm to estimate the latent modes, as is typical with hierarchical latent-variable models. The MMPSR also exploits the recognizability of the modes when maintaining its state, as described in Section 6.1.2.

Because the modes can be defined in terms of past *and future* observations, the MMPSR is not limited to using history-based modes. A model using history-based modes would only apply to systems where the current mode $\psi(\tau)$ was always known at time τ. In contrast, the MMPSR can model systems where the agent will not generally know the mode $\psi(\tau)$ until several time steps after τ. The traffic system is an example system where there are natural modes (e.g., a "left lane change" mode) that can be defined in terms of past *and future* observations, but not past observations alone. During the first few time steps of a left lane change, the mode at those times is not known from the past observations of the car (Figure 6.3): the car could be in the "going straight" mode but weaving in its lane, or it could be starting the left lane change mode. Even though the left lane change mode will not immediately be known, it can be recognized when the car crosses the lane boundary. Thus, the left lane change mode can be defined as "The car crossed a lane boundary from right to left within the most recent k time steps *or it will do so* within the next k time steps."

Defining the modes in terms of future observations provides a common characteristic with other PSR models, where the state is defined in terms of future observations. The PSR literature shows that a handful of features of the short-term future can be very powerful, capturing information from arbitrarily far back in history (Singh et al., 2004). This motivates the MMPSR's use of modes that are defined in terms of past and future observations.

Note that the recognizability requirement is a constraint on the *definitions of the modes* and not on the dynamical system itself (although the dynamical system determines if a set of modes' definitions meet the requirement). That is, requiring the modes to be recognizable does *not* limit the dynamical systems one can model with an MMPSR. This is because the modes that are defined for use by the MMPSR do not need to match the modes that the system actually used to generate the data. I expect that the MMPSR will be most useful when its modes are defined so they are closely related to the system's modes for data generation. However, this is not a strict requirement. Section 6.2.1 includes experiments with an example system that highlights this point.

In addition to the requirement that modes be known at some point in the future, the MMPSR makes the following independence assumptions that characterize the relationship between modes and observations:

1. The observation $O_{\tau+1}$ is conditionally independent of the history of modes given the mode at time $\tau + 1$ and the history of observations $O_1, \ldots, O_\tau$.

2. The future modes $\psi_{i+1}, \ldots$ are conditionally independent of the observations through the end of ψ_i, given the history of modes $\psi_1, \ldots, \psi_i$.

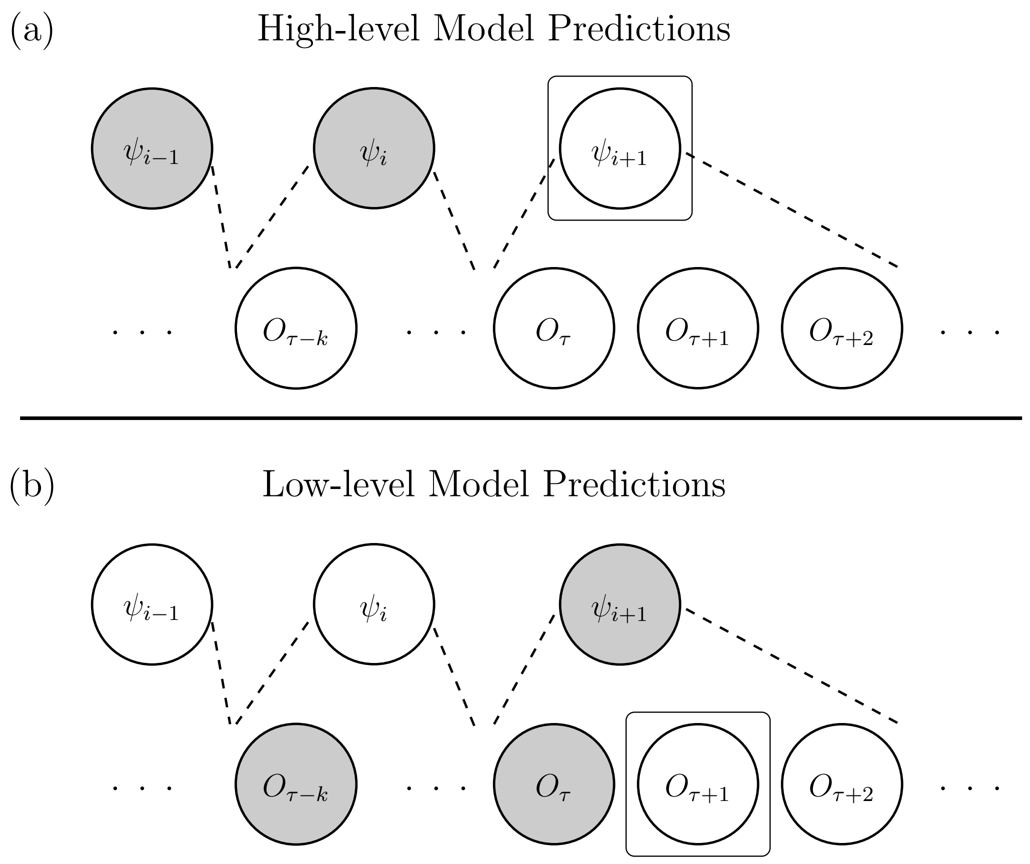


Figure 6.4: Predictions made by the component models of the MMPSR. The subscripts for the modes differ from the observations because each mode will last for several time steps (so $i << \tau$). The variable that is predicted is shown in the box, while the variables that are conditioned upon are shaded. (a) The high-level model predicts ψ_{i+1} given the history of modes (shaded). (b) The low-level model predicts $O_{\tau+1}$ given the history of observations and current mode (shaded).

Even if the independence properties do not strictly hold, the MMPSR forms a viable approximate model, as demonstrated by my empirical results (Section 6.2).

These independence properties lead to the forms of the high and low-level models within the MMPSR. The low-level model makes predictions for the next observation given the history of observations and the mode at the next time step (Figure 6.4.b). The low-level model also predicts the probability of a mode ending at time τ, given $\psi(\tau)$ and the history of observations through time τ. The high-level model makes predictions for future modes given the history of modes (Figure 6.4.a). Because of the second independence assumption, the high-level model can be learned and used independently from the low-level model; it models the sequence of modes $\psi_1, \psi_2, \ldots$ while abstracting away details of the observations.

6.1.2 Updating the MMPSR State

The MMPSR updates its state at every time step to reflect the new history. Suppose for a moment that for all τ, $\psi(\tau)$ was known at time τ. Then the high-level model would update its state whenever a mode ends, using that mode's value as its "observation." The low-level model would update its state after every time step τ, conditioned upon the most recent observation O_τ *and* the mode $\psi(\tau)$. Even though $\psi(\tau)$ will not always be known at time τ, this process defines the states of the high and low-level models under the assumption that some hypothetical values $\psi(1), \ldots, \psi(\tau)$ are the modes of history. Each sequence of hypothetical values has some posterior probability given the observations through time τ. The number of sequences $\psi(1), \ldots, \psi(\tau)$ with non-zero posterior probability will remain bounded, even as $\tau \to \infty$. Specifically, *only one sequence of hypothetical values for the known modes will have non-zero posterior probability* (i.e., the modes' true values), and only a finite (and typically small) window of past modes will be unknown, because of the recognizability requirement.

The MMPSR state at time τ consists of the posterior distribution over the modes of history $\psi(1), \ldots, \psi(\tau)$ and the high and low-level model states corresponding to each sequence $\psi(1), \ldots, \psi(\tau)$ with non-zero posterior probability. At the next time step $\tau + 1$, the MMPSR updates its state as follows. The MMPSR computes the high and low-level models' states for a given $\psi(1), \ldots, \psi(\tau), \psi(\tau + 1)$ from the high and low-level models' states at time τ using the respective model updates. The MMPSR updates the posterior distribution over modes of history using Bayes' Rule.

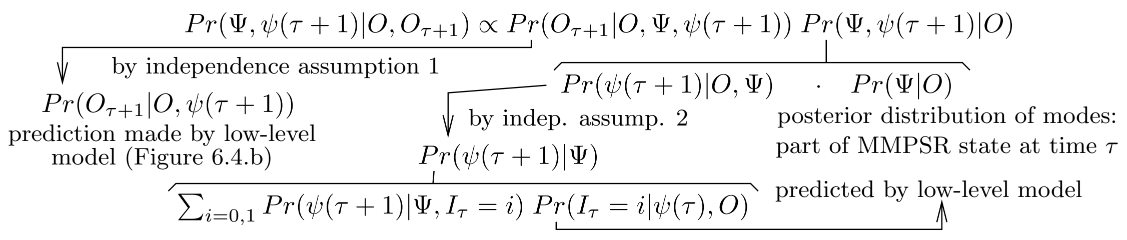

$Pr(\Psi, \psi(\tau+1)|O, O_{\tau+1}) \propto Pr(O_{\tau+1}|O, \Psi, \psi(\tau+1))\ Pr(\Psi, \psi(\tau+1)|O)$
by independence assumption 1
$Pr(O_{\tau+1}|O, \psi(\tau+1))$
prediction made by low-level model (Figure 6.4.b)
$Pr(\psi(\tau+1)|O, \Psi) \cdot Pr(\Psi|O)$
by indep. assump. 2
$Pr(\psi(\tau+1)|\Psi)$
posterior distribution of modes: part of MMPSR state at time τ
$\sum_{i=0,1} Pr(\psi(\tau+1)|\Psi, I_\tau = i)\ Pr(I_\tau = i|\psi(\tau), O)$
predicted by low-level model

Figure 6.5: Updating the MMPSR's posterior distribution over the modes of history.

Let $O \overset{\text{def}}{=} O_1, \ldots, O_\tau$ be the observations through time τ and let $\Psi \overset{\text{def}}{=} \psi(1), \ldots, \psi(\tau)$ be the modes through time τ. Let I_τ be an indicator variable that is 1 if a mode ends at time τ and 0 otherwise. The MMPSR computes the updated posterior distribution $Pr(\Psi, \psi(\tau+1)|O, O_{\tau+1})$ using the following breakdown, illustrated in Figure 6.5:

$$Pr(\Psi, \psi(\tau+1)|O, O_{\tau+1}) \propto Pr(O_{\tau+1}|O, \Psi, \psi(\tau+1))\ Pr(\Psi, \psi(\tau+1)|O).$$

- The first term on the right-hand side is equal to $Pr(O_{\tau+1}|O, \psi(\tau+1))$, by independence assumption 1. This is a prediction for the next observation $O_{\tau+1}$ given the history of observations O and the mode $\psi(\tau+1)$ at the next time step, which is made by the low-level model (Figure 6.4.b).

- The second term on the right-hand side is equal to $Pr(\Psi|O)\ Pr(\psi(\tau+1)|O, \Psi)$. The MMPSR computes these two terms as follows:

  – The $Pr(\Psi|O)$ term is the posterior distribution over history modes Ψ, which is part of the MMPSR state at time τ.

  – The last probability needed to compute the updated posterior is $Pr(\psi(\tau+1)|O, \Psi)$, which equals $Pr(\psi(\tau+1)|\Psi)$ by independence assumption 2. The MMPSR computes this by marginalizing an indicator variable I_τ that is 1 if a mode ends at time τ and 0 otherwise:

$$Pr(\psi(\tau+1)|\Psi) = \sum_{i=0,1} Pr(I_\tau = i|\psi(\tau), O) Pr(\psi(\tau+1)|\Psi, I_\tau = i).$$

The $Pr(I_\tau = i|\psi(\tau), O)$ predictions are made by the low-level model. The source of the $Pr(\psi(\tau+1)|\Psi, I_\tau = i)$ predictions depends on i.

* For $i = 0$, the mode does not end at τ, so $\psi(\tau+1)$ must equal $\psi(\tau)$. Thus, if $\psi(\tau+1)$ matches the $\psi(\tau) \in \Psi$, then $Pr(\psi(\tau+1)|\Psi, I_\tau = 0)$ is 1.0; otherwise, it is 0.0.
* For $i = 1$ (i.e, the mode ended at time τ), $Pr(\psi(\tau+1)|\Psi, I_\tau = 1)$ is the prediction for a future mode given a history of modes, which is computed by the high-level model (Figure 6.4.a).

Because the learned component models of an MMPSR will not be perfectly accurate, the Bayesian posterior update may not assign zero probability to hypothetical mode values even if they contradict the recognized value for those modes. Thus, in addition to the Bayesian posterior update, after each time step the MMPSR explicitly assigns zero probability to values of the history modes that contradict the known values, renormalizing the distribution after those changes. In addition, one can use pruning techniques (e.g., keep only the k most likely sequences of history modes) to reduce the number of history mode sequences that are maintained in the posterior distribution.

6.1.3 Making Predictions with the MMPSR Model

If a model can always make predictions about the next observation $O_{\tau+1}$ given the history of observations $O_1, \ldots, O_\tau$, then those predictions can be combined to make any prediction about the system, including predictions further than one step in the future. An MMPSR can make predictions about the next observation $O_{\tau+1}$

given the history of observations $O_1, \ldots, O_\tau$ in the following way. For any given values of the modes $\psi(1), \ldots, \psi(\tau+1)$, the low-level model can directly predict $O_{\tau+1}$ given $O_1, \ldots, O_\tau$. Since not all of $\psi(1), \ldots, \psi(\tau+1)$ will be known at time τ, the MMPSR makes predictions about $O_{\tau+1}$ given $O_1, \ldots, O_\tau$ by marginalizing the modes that are not known. This marginalization can be done relatively efficiently by using the distribution over modes that the MMPSR maintains.

To make predictions about observations several time steps in the future, it can be more efficient to make those predictions directly rather than calculating them from several next-observation predictions (cf. Singh et al., 2004). For example, in my empirical results, I include in the MMPSR several regression models to predict features of the future given the state of the low-level model and the most recent mode. As with the next-observation predictions, the overall prediction from the MMPSR marginalizes the unknown modes of history.

6.1.4 Learning an MMPSR

Learning an MMPSR consists of learning the high and low-level component models from a single sequence of observations (i.e., the training data). The high-level model is a linear PSR, so it is learned by applying the suffix-history algorithm (Wolfe et al., 2005) to the sequence of modes of the training data. Note that the recognizability of the modes ensures that the learning algorithm can correctly and automatically determine the mode for each time step of the training data (except perhaps a few time steps at the beginning and/or end of the data).

Learning the low-level model also requires the correct modes of the training data, since the low-level model makes predictions and updates its state conditioned upon the corresponding mode. One way to implement this conditional form is to have separate parameters of the low-level model for each mode. For example, the low-level model could consist of several parameterized functions from features of history (i.e., the state of the low-level model) to predictions about the next observation, with a separate function for each mode. The function for each respective mode can then be learned by applying an appropriate regression method to the time steps of training data that have that mode. This form allows the low-level model to make specialized predictions for each mode. It is the form I use in the following experiments.

6.2 Experiments

I learned MMPSR models for three systems: a simple random walk system, and both simulated and real-world highway traffic. The low-level model includes parameterized functions for each mode that map features from a finite window of history to predictions of interest. I learned each function using locally weighted linear regression (Cleveland, Devlin, & Grosse, 1988), a flexible non-linear function approximation method. I learned the high-level linear PSR using the suffix-history algorithm (Wolfe et al., 2005).

Because the modes are defined in terms of observations, the modes of the training data are known, which is critical in ensuring efficient training of the model. If the modes were not known, then an iterative estimation procedure would be needed to estimate the modes (e.g., expectation-maximization), often requiring several iterations to converge. At each iteration, the estimated modes would change, so the low-level model would have to be re-learned. In contrast, the low-level model of the MMPSR is only learned one time, using the true values of the modes because the modes are recognizable. For purposes of comparison, it would be possible to adapt hierarchical HMMs to exploit these modes, but as discussed above this would effectively yield an MMPSR. The comparison would then become a comparison between

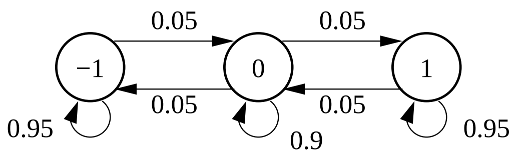


Figure 6.6: The Markov chain to generate the modes for the random walk system.

the suffix-history learning algorithm and the EM learning algorithm, a comparison which has already been done (Wolfe et al., 2005) with results showing the suffix-history algorithm to be generally superior. Therefore, I do not evaluate hierarchical HMMs here.

6.2.1 Random Walk

The empirical evaluation of the MMPSR begins with a simple random walk system, where the (scalar) observation at each time step is the change in (one-dimensional) position from the last time step. That change is given by the mode — which can take on values -1, 0, and 1 — plus mean-zero Gaussian noise. The mode is determined by a Markov chain, shown in Figure 6.6. The features that comprise the state of the low-level model are the average observations over the last k time steps for $k \in \{2, 5, 10\}$, along with a constant bias feature.

Because of the noise, the modes used to generate the data — call them the *generative modes* — cannot be defined in terms of observations. Nevertheless, one can still define a set of *recognizable modes* in terms of observations. For the random walk system, I defined the recognizable mode for time step τ as the mode (-1, 0, or 1) that is closest to the average observation over five time steps, from $\tau - 2$ through $\tau + 2$. (Note that this mode definition includes both historical and future observations.)

While there will not be perfect correspondence between the recognizable modes and the generative modes, there is enough correlation that an MMPSR learned using the recognizable modes models the system well. Figure 6.7 shows the average

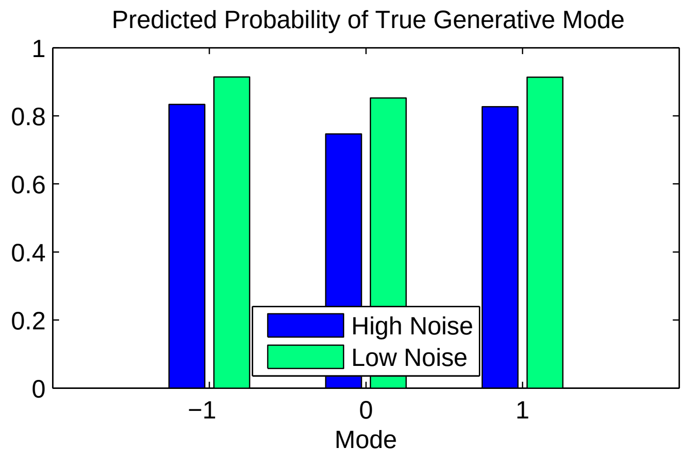


Figure 6.7: Predicting the current generative mode in the random walk system with an MMPSR.

likelihood that the MMPSR assigns (from its distribution over modes of history) to each *recognizable* mode whenever its related *generative* mode was the (latent) system mode at that time step. The high and low noise results correspond to using Gaussian noise with standard deviations 0.5 and 0.25, respectively. Though the higher noise leads to less certainty about the mode, the MMPSR attributes high probability to the generative modes for both noise levels.

In addition to tracking the modes well, the MMPSR is also able to predict the future observations well. Figure 6.8 shows the average error for predicting several time steps in the future for the high and low-noise systems. The MMPSR consistently achieves lower error than using locally-weighted linear regression on the entire data set, which does not explicitly leverage the existence of modes. When the low-level model is given the true recognized mode, lower error can be obtained, though the MMPSR already achieves close to that error, especially in the low-noise system.

6.2.2 Traffic

I also learned MMPSRs to model simulated and real-world highway traffic. The MMPSR predicts the movements of a car given the history of that car and some

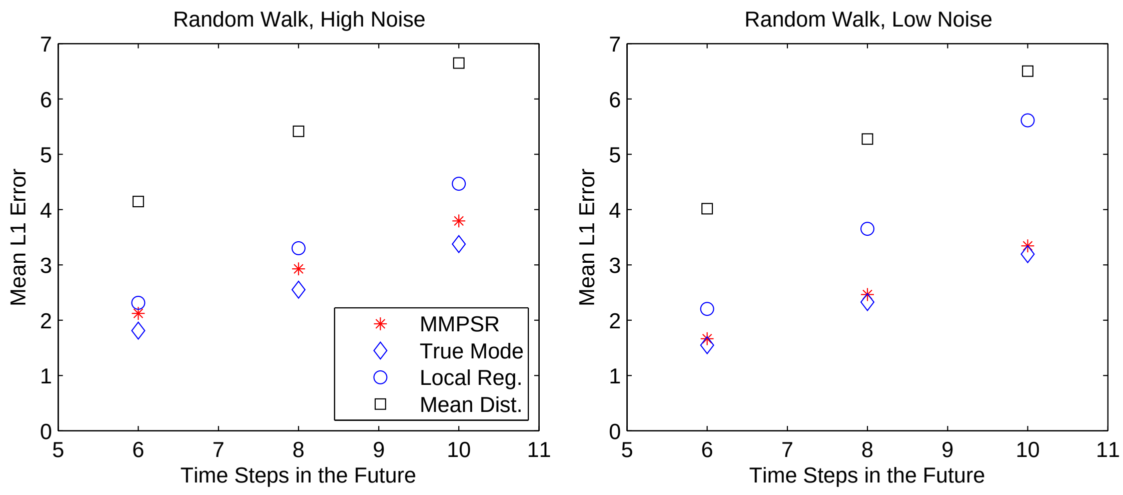


Figure 6.8: Prediction error of the MMPSR and comparison models on the random walk system. To give a qualitative sense of the error, "Mean Dist." is the average distance traversed by the random walk. "True Mode" is the error when the low-level model in the MMPSR is given the true mode. "Local Reg." uses locally-weighted linear regression over the entire data set.

neighboring cars. The observation at each time step consists of the x velocity, y velocity, and headway for the car being modeled, where headway is the time to collision with the car in front. I use "x" to refer to the lateral or side-to-side direction, while "y" refers to the longitudinal or forward direction. The features of history that composed the state of the low-level model were as follows: average y velocity over three different windows (the most recent 0.5 seconds, the 0.5 seconds before that, and the most recent 5 seconds); average x velocity over the most recent 0.5 seconds, and the 0.5 seconds before that; the x position; the distance to the closest car in front, and its average y velocity over the last 1.0 seconds; and a constant term. I used left and right lane changes as two of the modes of behavior for both simulated and real traffic. I defined a lane change as any 4.0-second window where the car center crossed a lane boundary at the midpoint of the window.

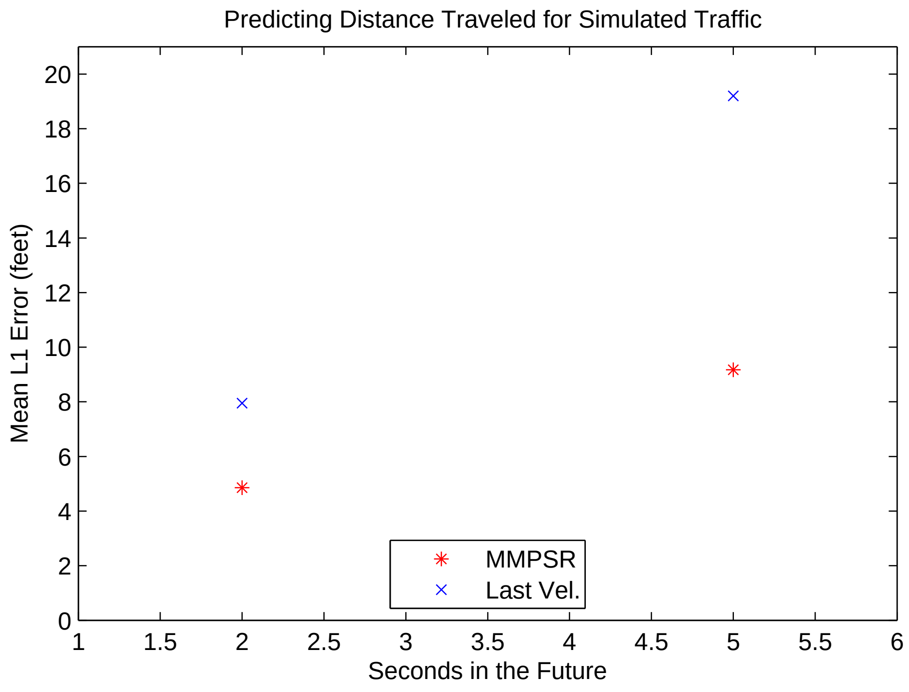
Simulated Traffic

The simulated traffic is a continuous-observation version of the simulated traffic system I used in the factored PSR experiments (Section 5.3.1). It consists of three lanes of traffic, with cars entering the field of view stochastically. (A video is available at `http://www.youtube.com/watch?v=8eHx_EzJOs0`.) Each car has a default velocity that it will maintain unless there is a car in the way. In that case, if there is room in an adjacent lane, the car will change lanes; otherwise, it will slow down. I added Gaussian noise to the observations of the cars' positions to emulate the inherent noise in physical sensors.

In addition to left and right lane change modes, I experimented with several possible sets of modes for the traffic data, including modes defined in terms of y velocity, headway, and a combination of y velocity and headway. The results I present here use left and right lane change modes and two different "stay-in-lane" modes for cars at different speeds (i.e., fast and slow).

I compare the accuracy of the learned MMPSR with two other prediction methods: a baseline method that predicts that the car will maintain its last observed velocity, and locally weighted linear regression trained upon the entire data set. I evaluated the models based upon their predictions about how far each car would travel in the y direction over the future 2 and 5 seconds. The MMPSR performed significantly better than both comparison methods at both the 2 and 5-second horizons (Figure 6.9). (The error for local regression is above the scope of the plot but is listed in the tables.) It is worth noting that, even in the presence of noisy observations, the MMPSR achieves less than two percent error, compared to the distance traveled, for both two and five seconds in the future.

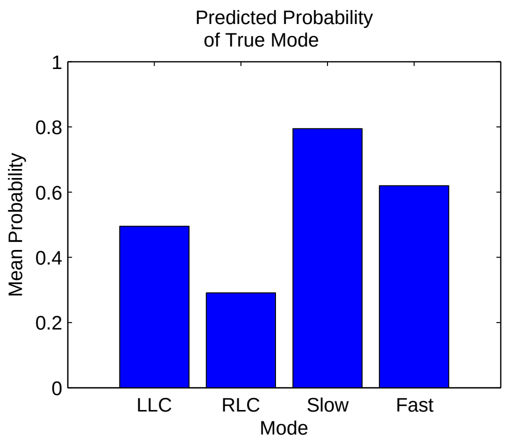
Not only does the MMPSR predict the movement of the cars accurately, it also



2 sec.	L1 Error (feet)	Percent Error
MMPSR	4.856 ± 0.722	1.615 ± 0.219
Last Vel.	7.952 ± 0.130	3.397 ± 0.063
Local Reg.	184.165 ± 16.022	94.298 ± 7.105

5 sec.	L1 Error (feet)	Percent Error
MMPSR	9.172 ± 1.950	1.151 ± 0.278
Last Vel.	19.197 ± 0.527	3.176 ± 0.058
Local Reg.	420.237 ± 62.923	86.694 ± 11.396

Figure 6.9: Error in predicting longitudinal distance traveled for simulated traffic. The MMPSR achieves the lowest error for predicting both 2 and 5 seconds in the future. The confidence intervals in the tables are two standard deviations, computed across 15 data sets. The local regression results do not appear in the plot because the error is so high.



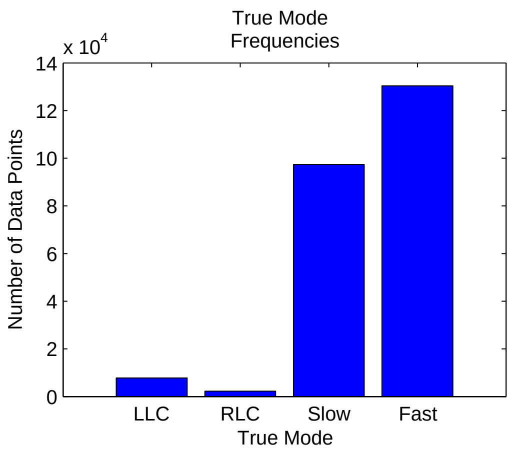


Figure 6.10: Predicting the current mode for simulated traffic. "LLC" and "RLC" are the left and right lane change modes; "Slow" and "Fast" are the two possible modes if a car is not making a lane change. Left: likelihood assigned to the true mode by the learned MMPSR. Right: the distribution of true modes, showing the low prior probability of the lane change modes.

assigns reasonably high likelihood to the true value of the current mode (Figure 6.10). The low likelihoods for the lane change modes are primarily due to the low prior probability of those modes (Figure 6.10).

Interstate 80 Traffic

I also learned MMPSRs to model real highway traffic on Interstate 80 (I-80) near San Francisco. This is the same overall data set used in the factored PSR experiments (Section 5.3.2), though the subset of the data used in these experiments differed from the previous experiments. Recall that the data comes from approximately 500 meters of six-lane freeway (U.S. Federal Highway Administration, 2006a). (A video is available at `http://www.youtube.com/watch?v=JjxNu2kbtDI`.)

As with the simulated traffic, I ran preliminary experiments with several sets of mode definitions. Based on the models' errors in those preliminary experiments, the experiments presented here use modes determined by the y velocity and the headway to the car in front, which I discretized into four and three bins, respectively. Along

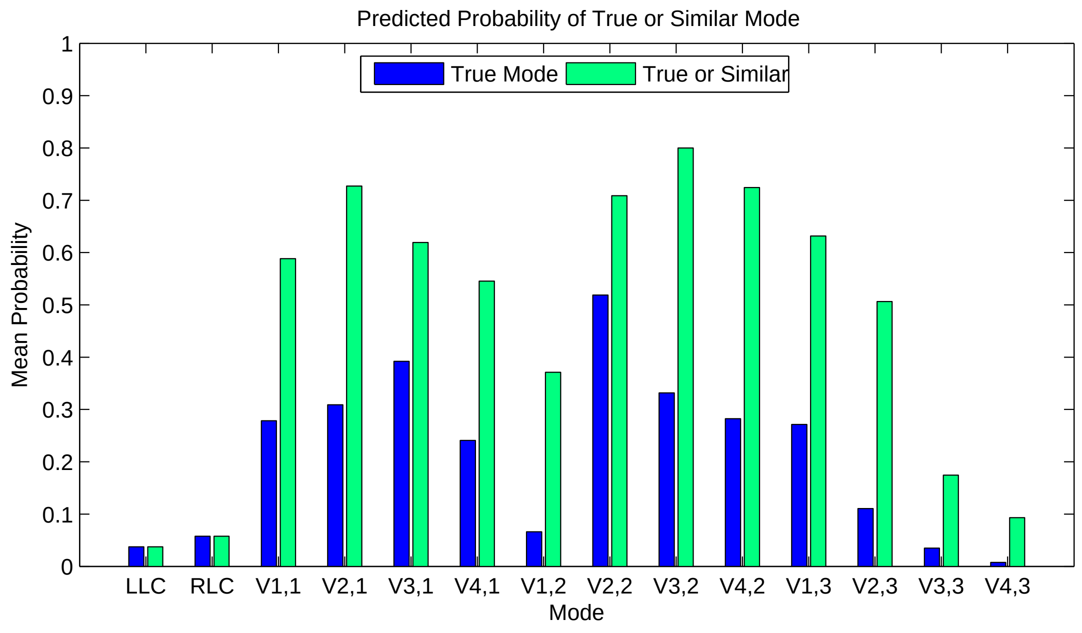


Figure 6.11: Predicting the current mode for I-80 traffic. The plot shows the likelihood assigned to the true (or similar) modes by the learned MMPSR. Regarding the mode labels, LLC and RLC are left and right lane changes; the other labels "Vk,j" indicate the mode of a car that falls in the k^{th} velocity bin (4 is fastest) and j^{th} headway bin (3 is the longest distance to the car in front).

with the two lane change modes, this gives fourteen total modes for the system.

Partly because of the fine distinctions between the modes (cf. the simulated traffic with only two velocity bins), the MMPSR has more difficulty determining the true mode than with the previous domains, as seen by the lower likelihoods assigned to the true modes in Figure 6.11. As with the simulated data, the low likelihoods assigned to the first two modes — the lane change modes — are primarily due to a very low prior probability for those modes (Figure 6.12). The likelihood that the MMPSR assigns to either the true mode or a similar mode (i.e., those that differ by one notch in velocity or headway, but not both) is significantly higher than the likelihood assigned to the true mode alone (Figure 6.11).

Even though the MMPSR does not always track the true mode, its predicted modes are close enough to enable good predictions about the cars' movements. I

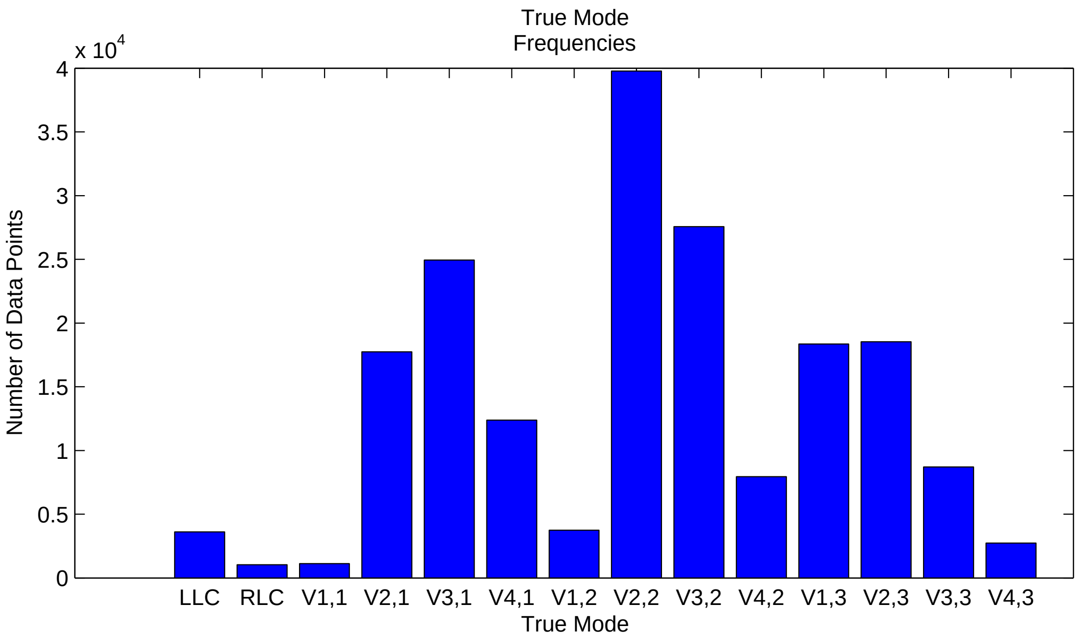
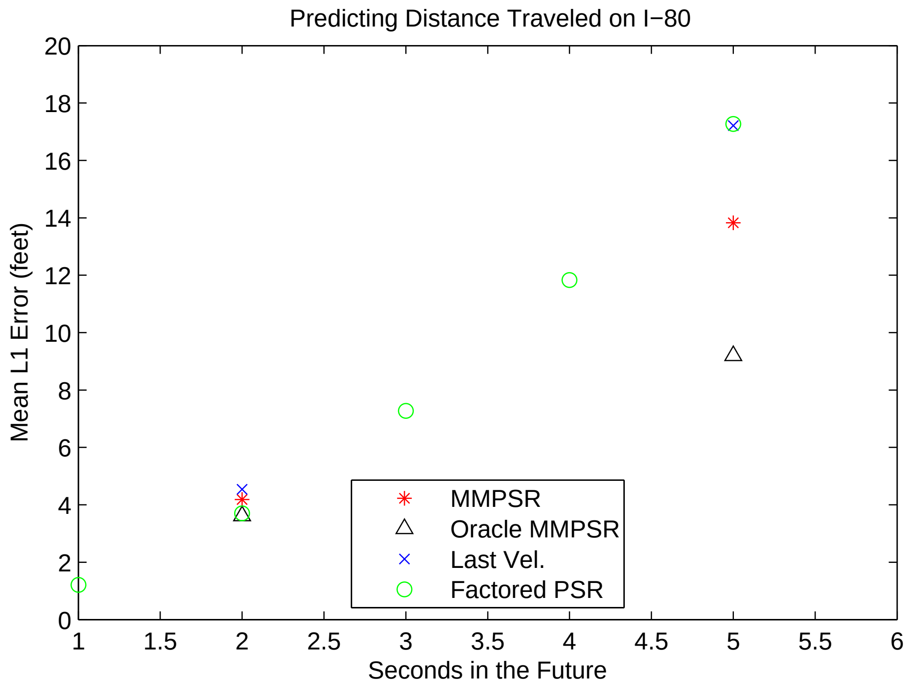


Figure 6.12: The distribution of true modes in the I-80 traffic, showing the low prior probability of the lane change modes. Regarding the mode labels, LLC and RLC are left and right lane changes; the other labels "Vk,j" indicate the mode of a car that falls in the k^{th} velocity bin (4 is fastest) and j^{th} headway bin (3 is the longest distance to the car in front).

evaluated these predictions in the same way as for the simulated traffic, including a comparison with locally-weighted regression and predicting the last velocity. I also evaluated an MMPSR that uses an "oracle" feature that peers into the future. This feature is included in the low-level state, even though it will never be available online. Nevertheless, the oracle MMPSR's error provides a sense of the best error that could be achieved. Specifically, the oracle feature is *the desired prediction* (i.e., the distance the car will travel) minus the future value of the gap between it and the car it is following.[1]

As with the simulated traffic, the MMPSR performed better than both the local-regression and the last-velocity comparison methods at both the 2 and 5-second

[1]This evaluation only considers the time points where there is a car in front of the modeled car; otherwise, the oracle feature is not defined. In the I-80 data set, there is almost always a car in front, so the vast majority of the data is evaluated.



2 sec.	L1 Error	Percent Error
MMPSR	4.185 ± 0.250	6.552 ± 0.228
Oracle MMPSR	3.615 ± 0.123	6.048 ± 0.258
Last Vel.	4.538 ± 0.161	6.601 ± 0.249
Factored PSR	3.705	–
Local Reg.	49.648 ± 2.416	99.258 ± 0.129

5 sec.	L1 Error	Percent Error
MMPSR	13.827 ± 0.501	9.092 ± 0.424
Oracle MMPSR	9.196 ± 0.263	6.265 ± 0.316
Last Vel.	17.216 ± 0.687	11.187 ± 0.511
Factored PSR	17.273	–
Local Reg.	110.548 ± 5.943	89.432 ± 0.547

Figure 6.13: Error in predicting longitudinal distance traveled for I-80 traffic. The “Oracle MMPSR” error provides a sense of the best error that could be achieved. Among the remaining models, the MMPSR achieves the lowest error for predicting 5 seconds in the future and the second-lowest error for predicting 2 seconds into the future. The confidence intervals in the tables are two standard deviations, computed across 15 data sets. The local regression results do not appear in the plot because the error is so high. The results from the factored PSR experiments in Section 5.3.2 are presented for comparison.

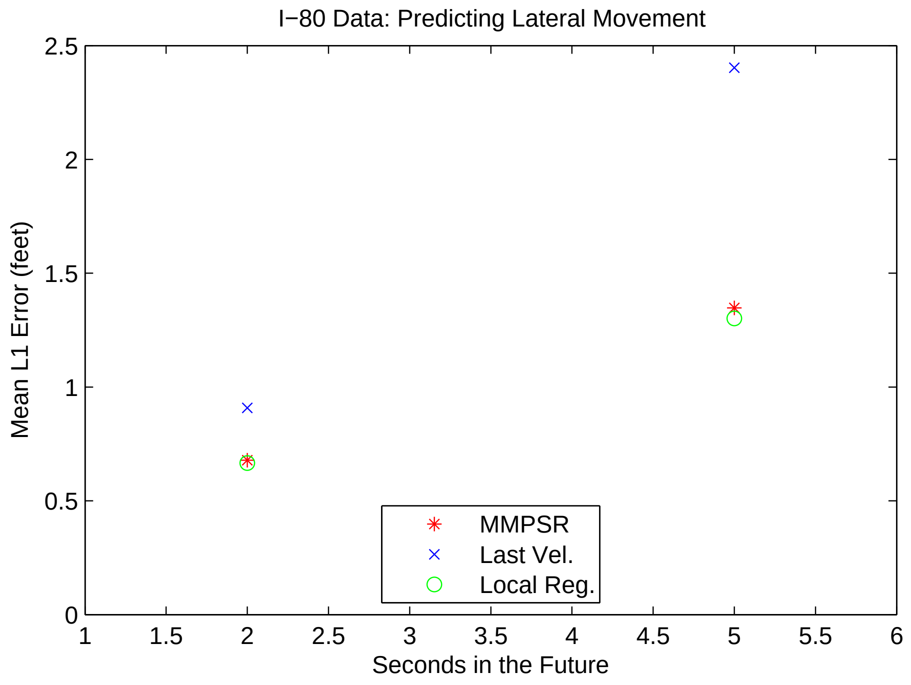
horizons (Figure 6.13), with a considerable difference at the 5-second horizon. (The error for local regression is above the scope of the plot but is listed in the tables.) Although the observations in these experiments are significantly different from those in the factored PSR experiments in Section 5.3 (which used discretized acceleration values), for the sake of completeness, Figure 6.13 presents the error of those factored PSRs in terms of L1 error in distance traveled.[2] Although the factored PSR achieves slightly lower error than the MMPSR for the 2-second predictions, the MMPSR is significantly better at making 5-second predictions. Also, the error of the MMPSR is reasonably close to that of the oracle MMPSR, despite all the information contained in the oracle feature. Finally, it is worth noting that the MMPSR achieves less than ten percent error (relative to distance traveled), even when predicting five seconds in the future for cars at highway speeds. In absolute terms, this is only 14 feet, or roughly one car length.

In addition to evaluating the MMPSR's predictions about distance traveled in the forward direction, I also evaluated its predictions about lateral movement. Figure 6.14 compares the MMPSR, the last-velocity model, and locally-weighted linear regression. The last-velocity model does the worst, while the MMPSR and local regression models are comparable. The percent error is so high because the true lateral movement is often small. Thus, small absolute errors can translate to large percent errors. However, in absolute terms, the average error of the MMPSR is quite small: less than 1.5 feet for five seconds in the future.

Detailed Analysis

This section presents a more detailed analysis of the error of the MMPSR when predicting the highway traffic on I-80, looking at the degree to which the high and

[2]The factored PSR experiments used a different subset of the Interstate 80 data.



2 sec.	L1 Error	Percent Error
MMPSR	0.678 ± 0.120	99.057 ± 2.656
Local Regression	0.666 ± 0.027	102.652 ± 3.272
Last Velocity	0.908 ± 0.039	105.316 ± 1.918

5 sec.	L1 Error	Percent Error
MMPSR	1.348 ± 0.214	101.307 ± 4.148
Local Regression	1.301 ± 0.087	99.366 ± 0.352
Last Velocity	2.403 ± 0.103	135.781 ± 4.937

Figure 6.14: Error in predicting lateral movement of I-80 traffic. The last-velocity comparison model has the highest error, while the MMPSR and local regression models are comparable. The confidence intervals in the tables are two standard deviations, computed across 15 data sets. The percent error is so high because the true lateral movement is often small, so small absolute errors can still translate to large percent errors.

low-level models in the MMPSR contribute to the overall error. For this analysis, I computed the error that would result if the MMPSR (hypothetically) always knew the true value of the most recent mode $\psi(\tau)$ at time τ. This will not actually be possible when using the model online, but for purposes of analysis I artificially altered the MMPSR's posterior distribution over the modes of history so that all the mass is on the true values of the modes, even when those modes are not known from the observations through the current time. In the following discussion, I refer to this artificial scenario as the MMPSR that "always knows the true modes." If such an MMPSR achieves error comparable to the standard MMPSR, that would indicate that the high-level model in the standard MMPSR is performing well, because it is tracking the modes well enough to make predictions that are comparable to the predictions made using the true mode values. Thus, the error of an MMPSR that always knows the true modes is directly attributable to the low-level component model.

Figure 6.15 compares the error of several models — including a standard MMPSR and an MMPSR that always knows the true modes — when making predictions about lateral movement. This figure breaks down the error according to the different modes of behavior. Recall that there are two modes for the left and right lane changes (LLC and RLC, respectively); the remaining modes are determined by the car's velocity (4 bins labeled V1 through V4) and headway to the car in front (3 bins, labeled 1 through 3). As expected, the lane change modes have the highest error, because they are the only modes with significant lateral movement. For the left lane changes, the MMPSR is comparable to the better of the last-velocity and local regression models. For the right lane changes, the MMPSR does well predicting 5 seconds in the future, but is slightly worse than the last velocity model for predicting 2 seconds in the future. The following analysis explains why this is the case.

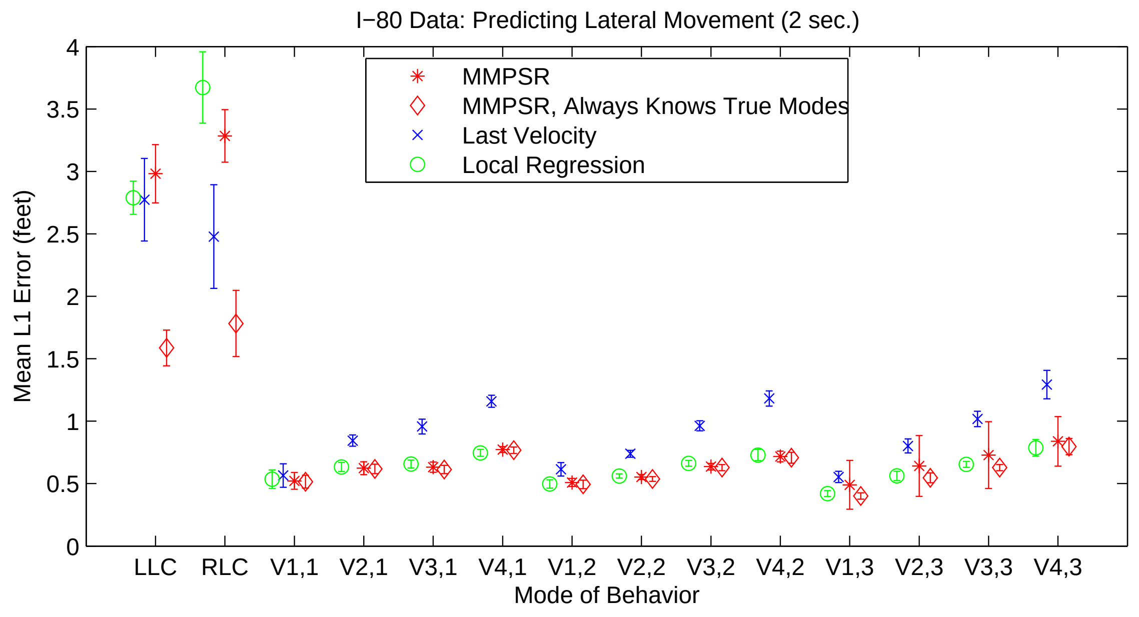
The primary factor contributing to the error in the predictions during right lane changes is the infrequent occurrence of right lane changes (cf. Figure 6.12). Because the right lane changes are so rare, the MMPSR does not assign very high probability to that mode when a right lane change is actually occurring (cf. Figure 6.11). However, the data in Figure 6.15 for the MMPSR that always knows the true modes shows that the MMPSR has a good model of both left and right lane changes.

Aside from the lane change modes, the error of the MMPSR that always knows the true modes is very close to that of the standard MMPSR, both of which are comparable to the local regression model. The last-velocity model does progressively worse at higher velocities.

As with the lateral predictions, I also broke down the longitudinal predictions according to mode, comparing the MMPSR with the other models, including an MMPSR that always knows the true modes (Figure 6.16). There are a few things to highlight about these results:

- The error of the local regression model was higher than the scope of these plots.
- Higher-velocity modes generally lead to higher error.
- Modes that occur infrequently (cf. Figure 6.12) tend to have high variance in the errors.
- The standard MMPSR error is generally close to (or within the error bars of) the error of an MMPSR that always knows the true modes. This indicates that the MMPSR is tracking the current mode well enough to make predictions that rival those made specifically for the car's actual mode.[3]

[3]In some cases, the error when using the modes' true values is actually higher. One explanation for this phenomenon is that there may not be adequate training data for some modes. Another



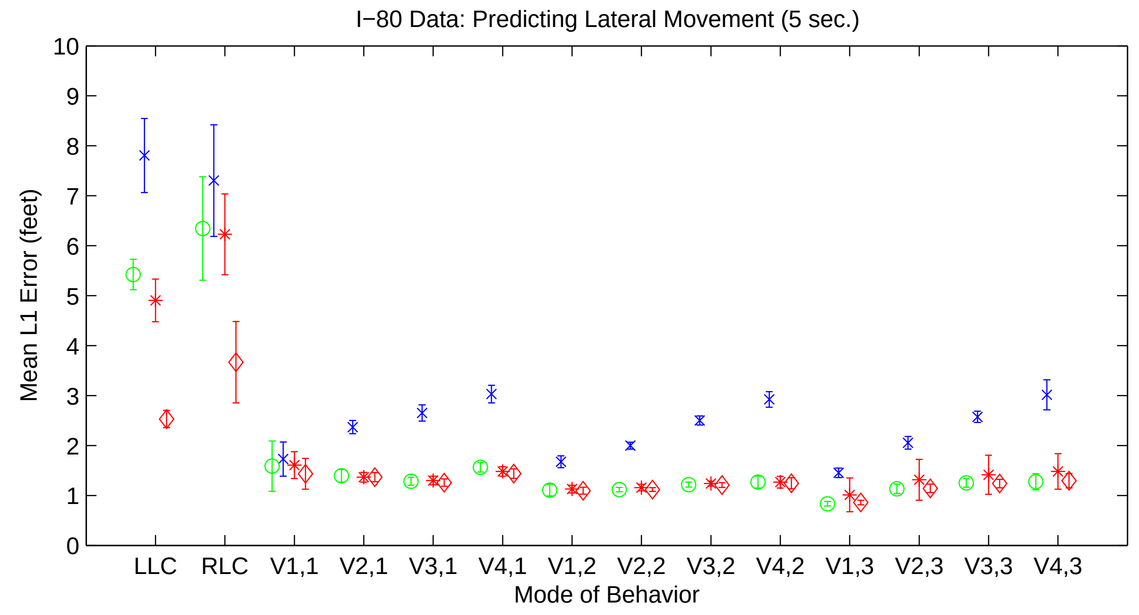
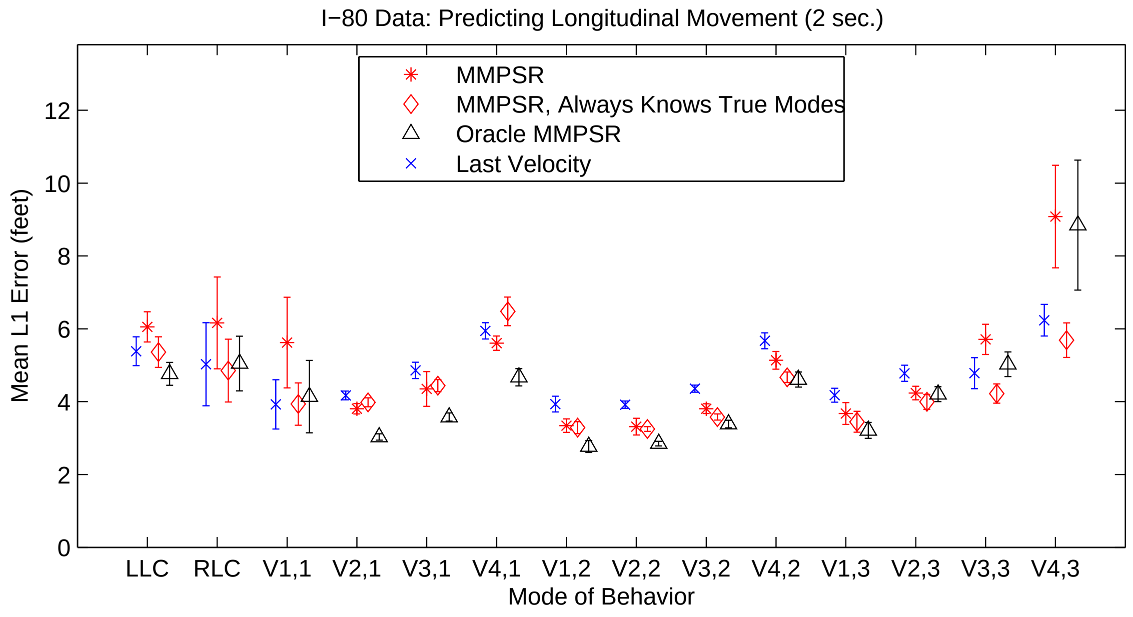


Figure 6.15: Error in predicting lateral movement of I-80 traffic, broken down by mode of behavior. For clarity, the models' errors are offset slightly along the x-axis. Regarding the mode labels, LLC and RLC are left and right lane changes; the other labels "Vk,j" indicate the mode of a car that falls in the k^{th} velocity bin (4 is fastest) and j^{th} headway bin (3 is the longest distance to the car in front). Error bars show plus/minus one standard deviation over 15 data sets. See text for further details.



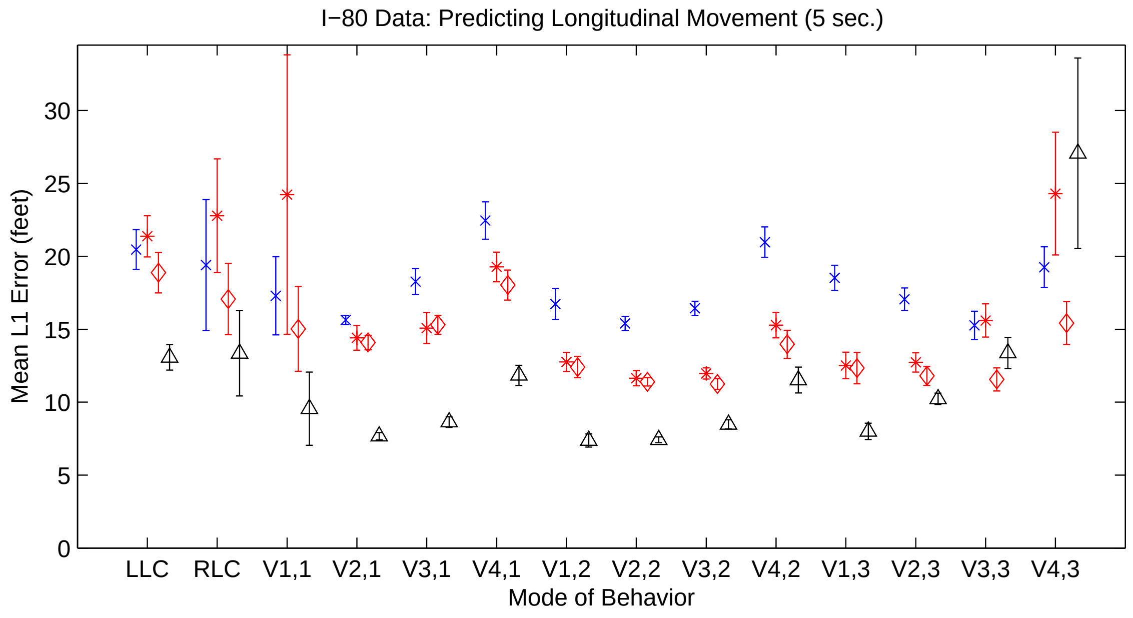


Figure 6.16: Error in predicting longitudinal movement of I-80 traffic, broken down by mode of behavior. For clarity, the models' errors are offset slightly along the x-axis. Regarding the mode labels, LLC and RLC are left and right lane changes; the other labels "Vk,j" indicate the mode of a car that falls in the k^{th} velocity bin (4 is fastest) and j^{th} headway bin (3 is the longest distance to the car in front). Error bars show plus/minus one standard deviation over 15 data sets. See text for further details.

- The oracle feature is particularly helpful when the headway is small (i.e., there is car close in front). Visual inspection of the data suggests that this is because of the traffic jams that occur on the highway. The oracle feature reveals if the traffic will start moving again (which is understandably difficult to predict).

Summary of the MMPSR Results for I-80 Traffic

To summarize, the overall error of the MMPSR for modeling the I-80 traffic data is quite small: for predicting five seconds in the future, the average lateral error is less than 1.5 feet and the average longitudinal error is less than 14 feet (roughly one car length). The lateral error is attributable to both the mode prediction and the positional predictions of the MMPSR (i.e., both the high and low-level models). Specifically, when the MMPSR does not have to predict the current mode but is given the mode's true value, the lateral error during lane changes is roughly cut in half. On the other hand, the longitudinal error is primarily due to the low-level model. Much of this error is likely due to the high variance in the data that results from noisy camera images and the variation in the way different drivers behave.

6.3 Summary

This chapter describes the MMPSR, a hierarchical model designed for uncontrolled systems that switch between modes of behavior. Inspired by PSRs, the modes are not latent variables but are defined in terms of both historical and future observations. Because the modes are defined in terms of observations, learning the MMPSR model is more efficient than if the modes were latent variables. Furthermore, when using the MMPSR model to make predictions, the MMPSR can adjust its state to reflect the true values of the modes because those true values are eventually recognized

possibility is that if a car is near the "border" between two modes of behavior, it may be better to blend the predictions for those modes than to predict based upon the true mode.

from the observations. Even though the modes are defined in terms of observations, they are not restricted to be features of history, which would limit the expressiveness of the model. Rather, the mode definitions can include *future* observations; the PSR literature has shown that features of the short-term future can be very expressive, capturing information from arbitrarily far back in history. This chapter also introduced a learning algorithm for the MMPSR, using it to learn MMPSR models of three systems. The MMPSR achieved lower error than comparison methods on all three systems, including highway traffic on Interstate 80.

CHAPTER VII

Conclusions

When beginning my work on predictive state representations, the state-of-the-art method for learning a PSR model was the reset algorithm for learning a linear PSR from multiple sequences of experience in the system (Section 1.3). The first contribution of this dissertation was the suffix-history algorithm for learning linear PSR models (Section 2.1), which extends the reset algorithm so that it is able to use a *single* sequence of experience for learning a model. This is important because many agents will not be able to reset their environment to its initial state in order to obtain multiple sequences of experience, so they will only have a single sequence of experience. I empirically demonstrated that the suffix-history algorithm is able to learn more accurate PSR models than the standard learning algorithm for POMDP models, supporting the use of predictive state models. In addition to the empirical evaluation, I provided theoretical guarantees about the running times of the suffix-history and reset algorithms, proving that they have polynomial running time. This is in spite of the fact that the set of potential core tests (from which the algorithms choose a set of core tests for the linear PSR model) is exponentially large.

While the suffix-history and reset algorithms work very well on systems for which the system-dynamics matrix has small rank, both the algorithms and the class of linear PSR models will scale poorly if applied directly to complex systems, where the rank can be combinatorially large. Therefore, the primary contributions of this dissertation center around three classes of PSR models that exploit different types of structure in complex systems in order to tractably model such systems. These three classes of structured PSR models — the hierarchical PSR (HPSR, Chapter IV), the factored PSR (Chapter V), and the multi-mode PSR (MMPSR, Chapter VI) — are the only known predictive state models that explicitly take advantage of structure in an environment. The contributions relating to these structured PSR models include the mathematical formulation of the three model classes along with the development of a learning algorithm for each class. These learning algorithms (and associated models) scale to larger systems than the suffix-history and reset algorithms, while still leveraging the advantage of predictive state for learning accurate models.

7.1 Comparing the Hierarchical PSR, Factored PSR, and Multi-Mode PSR

Each of the three classes of structured PSR models exploits a different type of structure. For a given dynamical system, one can choose the type of structured PSR model that is best suited to that system. For systems with multiple types of structure, Section 7.2 describes how techniques from the different structured PSR models can be combined into a single PSR model.

Among the three classes of structured PSR models, only the MMPSR is applicable to systems with either continuous or discrete observations.[1] This flexibility is a result of allowing for a generic low-level model within the MMPSR. For example, in the case of continuous observations the low-level model could consist of regression models, or

[1]There are several predictive state models that have been designed specifically for systems with continuous observations, mentioned in Section 1.2.1.

in the case of discrete observations the low-level model could consist of several linear PSRs (one for each mode). However, the MMPSR is only applicable to uncontrolled dynamical systems, whereas the HPSR and factored PSR can model controlled or uncontrolled systems.[2]

All three of the structured PSR classes are aggregate models, using multiple component models in different divide-and-conquer approaches to modeling the overall system. The component models of the factored PSR make predictions for different subsets of the observation dimensions in systems with multiple observation dimensions (or variables) at each time step.[3] This allows the factored PSR to exploit conditional independence relationships among the different dimensions, leading to a smaller model that is easier to learn.

Comparing the HPSR and MMPSR

Both the HPSR (Figure 7.1) and the MMPSR (Figure 7.2) include a high-level component model that is a temporally abstract model of the system, but they use different techniques to achieve temporal abstraction. This leads to a couple of fundamental differences between the models.

1. *The MMPSR and the HPSR differ in the way the component models at different levels interact.* The high-level model of the MMPSR predicts which modes will occur in the future. *The predictions that the high-level model makes about the modes affect the predictions that the MMPSR makes about low-level observations,* because the predictions about low-level observations are made by marginalizing the unknown modes of history according to the predictions made

[2]Given a set of options in an uncontrolled system, one can use an HPSR just as in a controlled system. The use of options in uncontrolled systems is discussed in Section 7.1.

[3]While one could use a factored PSR to model systems with a single observation dimension, the factored PSR would reduce to a single linear PSR.

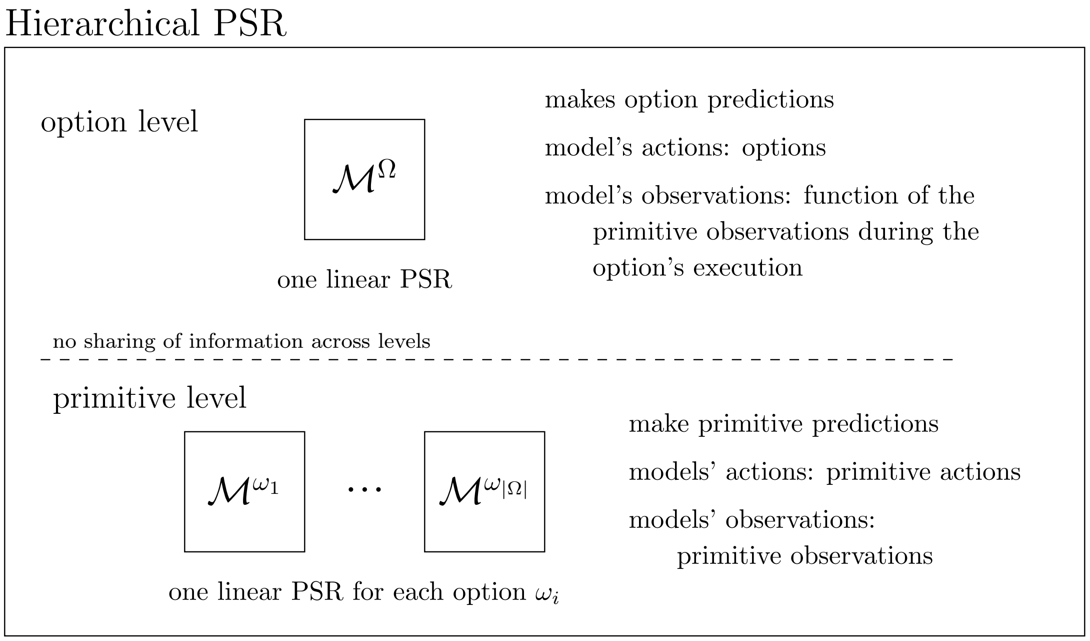

Hierarchical PSR
option level
$\mathcal{M}^{\Omega}$
one linear PSR
makes option predictions
model's actions: options
model's observations: function of the primitive observations during the option's execution
no sharing of information across levels
primitive level
$\mathcal{M}^{\omega_1}$ $\cdots$ $\mathcal{M}^{\omega_{|\Omega|}}$
one linear PSR for each option ω_i
make primitive predictions
models' actions: primitive actions
models' observations: primitive observations

Figure 7.1: A review of the component models within an HPSR. Each square represents a linear PSR component model. There is one temporally abstract, high-level linear PSR that makes option predictions, and there is one primitive, low-level linear PSR for each option that makes predictions while its option is executing.

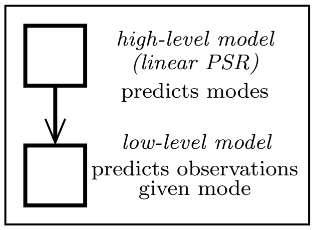

high-level model (linear PSR)
predicts modes
low-level model
predicts observations given mode

Figure 7.2: A review of the component models in an MMPSR.

by the high-level model. In contrast, *in the HPSR, the two levels of models operate completely independently.* This is because the low-level predictions of an HPSR are conditioned upon the options the agent selects, which are known to the agent and therefore do not need to be predicted.[4]

2. *The MMPSR requires the specification of a set of modes, while the HPSR requires the specification of a set of options.* The first distinction between modes and options that may come to mind is that options deal with actions, whereas modes are defined in terms of observations. It turns out that this is not really a fundamental difference between options and modes. Even though I have only discussed options for controlled systems and modes for uncontrolled systems, one can also use options in uncontrolled systems and modes in controlled systems:

   - *Using modes in controlled systems*

     I expect that MMPSRs will generalize to controlled systems, where the mode definitions may include actions as well as observations. Because the actions are observable, they can be used in the mode definitions while maintaining the recognizability of the modes (and the associated advantages for model learning and operation).[5]

   - *Using options in uncontrolled systems*

     First note than an uncontrolled system is equivalent to a controlled system with only one action. Thus, any option in an uncontrolled system will have the same policy: "take the only action with probability 1.0." However,

[4]Instead of making predictions about which options an agent will choose, the high-level model in the HPSR predicts the outcomes of options. No such predictions are made in the MMPSR, since it does not explicitly reference options.

[5]Allowing modes to depend on actions would mean that the high-level model's predictions may depend on the policy of the agent, potentially requiring a more complicated model.

one can still create multiple distinct options because their initiation and termination conditions can differ. Thus, choosing an option to execute amounts to choosing a termination condition and waiting for that condition to be met.

Now that I have addressed the issue of controlled/uncontrolled applicability, it remains to describe the actual, fundamental differences between modes and options. To begin, note that the particular *option* that is active at a given point in time is *chosen by the agent*, whereas the *mode* at a given time is not chosen by the agent but is *recognized by the agent* based upon the observations (and actions) that it has seen (or will see in the future). This fact leads to two fundamental differences between the MMPSR and the HPSR:

(a) With an HPSR, the agent is restricted to behaving according to the options in the specified set. In contrast, even in a controlled version of the MMPSR, the modes would not restrict the behavior of the agent.

(b) With an MMPSR, the modes must be recognizable from the actions and observations of the past and future. In contrast, the HPSR does not require that one be able to recognize which option was being executed at a given point in time (because the agent knows which option it is executing).

Another fundamental difference between modes and options is that modes can be defined in terms of both past *and future* observations. In contrast, the parts of an option — the initiation condition, the termination condition, and the policy — are all functions of the past alone. As discussed when describing the MMPSR (Chapter VI), allowing modes to be defined in terms of the future provides more flexibility for what constitutes a valid mode. Furthermore, PSRs have demonstrated that features of the short-term future can be very powerful, capturing information from arbitrarily far back in history.

Due to these differences between the MMPSR and HPSR, there are systems where an MMPSR would naturally capture the system's structure but an HPSR would not, and vice versa.

7.2 Combining Techniques from the Structured PSR Models

Many of the component models used in the structured PSRs were linear PSR models. However, one can use structured PSR models as the component models within other structured PSRs in order to model a system with multiple kinds of structure. For example, one could replace any linear PSR component of a factored PSR with an MMPSR or HPSR, capturing temporally extended structure in the particular observation dimensions for that component model.

One could also combine structured PSR models by using a factored PSR in place of a linear PSR component model whenever the observation space is multi-dimensional instead of single-dimensional. For example, one could modify the HPSR to handle multi-dimensional option observations, using a factored PSR instead of a linear PSR as the high-level model in the HPSR. This would facilitate the use of rich option observations, which can be any function of the actions and observations seen during the option's execution. For example, the option observation vector could consist of indicator bits for seeing the light turn off, hearing the phone ring, hitting a wall, etc., during the option's execution. Using a factored PSR as the high-level model in the HPSR would exploit independence among the option observation variables (e.g., hearing the phone ring is independent of seeing the light turn off).

7.3 Summary of Contributions

In this dissertation, I have developed predictive state models and accompanying learning algorithms that allow an agent to use its experience in a large, complex dynamical system to learn a predictive state model of that system. The previously existing work on PSRs (Chapter I) helps motivate the use of predictive state, but those linear PSR models and algorithms for learning those models from data do not scale to large systems (Section 3.1). Therefore, I have introduced new classes of PSR models and new learning algorithms (Section 2.1 and Chapters IV, V, and VI) that extend the applicability of PSR models to larger dynamical systems by exploiting different types of structure in the system. Specifically, the hierarchical PSR (Chapter IV) exploits temporal structure based upon macro-actions; the factored PSR (Chapter V) exploits conditional independence of observation variables; and the multi-mode PSR (Chapter VI) exploits temporal structure based upon recognizable modes of the system. The development of these structured PSR models makes PSRs a viable possibility for modeling structured dynamical systems, providing an alternative to using latent-state models such as DBNs.

The previously existing methods for learning PSR models were practically viable only on small dynamical systems, such as mazes that have about a dozen discrete locations. In contrast, the developments presented in this dissertation enable the learning of PSR models of complex dynamical systems such as real-world highway traffic.

APPENDICES

APPENDIX A

Predictions for Tests

A.1 Sequence Tests

The basic test is a *sequence test* or *s-test* or just *test* for short. An s-test is a possible sequence of future actions and observations: e.g., $t = a_{\tau+1}o_{\tau+1}a_{\tau+2}o_{\tau+2}\dots a_{\tau+k}o_{\tau+k}$, where τ is the current time step. As noted in Section 1.1.2, the prediction for an s-test $t = a_{\tau+1}o_{\tau+1}a_{\tau+2}o_{\tau+2}\dots a_{\tau+k}o_{\tau+k}$ from a history $h = a_1o_1a_2o_2\dots a_\tau o_\tau$ is defined as

$$p(t|h) \overset{\text{def}}{=} \prod_{i=\tau+1}^{\tau+k} Pr(o_i|a_1o_1a_2o_2\dots a_{i-1}o_{i-1}a_i). \tag{1.1}$$

Each of the conditional probabilities in this product is the prediction for a one-step test, so

$$p(t|h) = \prod_{i=\tau+1}^{\tau+k} p(a_io_i|a_1o_1a_2o_2\dots a_{i-1}o_{i-1}).$$

A.2 Set Tests

A *set test* is defined as a sequence of alternating individual actions and observation *sets*, such as $t = a_{\tau+1}\sigma_{\tau+1}a_{\tau+2}\sigma_{\tau+2}\dots a_{\tau+k}\sigma_{\tau+k}$. Such a test succeeds if the observations at each time step are within their respective sets $\sigma_{\tau+1}\dots\sigma_{\tau+k}$. Thus, the prediction for t from a history h is

$$p(t|h) \overset{\text{def}}{=} \sum_{o_{\tau+1}\in\sigma_{\tau+1}, o_{\tau+2}\in\sigma_{\tau+2}, \ldots o_{\tau+k}\in\sigma_{\tau+k}} p(a_{\tau+1}o_{\tau+1}a_{\tau+2}o_{\tau+2}\ldots a_{\tau+k}o_{\tau+k}|h). \tag{A.1}$$

The following lemma helps establish that one can decompose the prediction for a set test into a product of predictions for one-step set tests, just as one can do for s-tests (Section A.1).

Lemma A.1. *Let t' be some test, and let t be the set test $t = a_{\tau+1}\sigma_{\tau+1}t'$. Then*

$$p(t|h) = p(a_{\tau+1}\sigma_{\tau+1}t'|h) = p(a_{\tau+1}\sigma_{\tau+1}|h) \cdot p(t'|ha_{\tau+1}\sigma_{\tau+1})$$

where conditioning upon $\sigma_{\tau+1}$ in the history means that one conditions upon the fact that $o_{\tau+1}$ was a member of $\sigma_{\tau+1}$.

Proof. The set test prediction $p(a_{\tau+1}\sigma_{\tau+1}t'|h)$ is defined to be

$$\begin{aligned}
& \sum_{o_{\tau+1}\in\sigma_{\tau+1}} p(a_{\tau+1}o_{\tau+1}t'|h) \\
= & \sum_{o_{\tau+1}\in\sigma_{\tau+1}} p(a_{\tau+1}o_{\tau+1}|h)p(t'|ha_{\tau+1}o_{\tau+1}) \\
= & \; p(a_{\tau+1}\sigma_{\tau+1}|h) \sum_{o_{\tau+1}\in\sigma_{\tau+1}} \frac{p(a_{\tau+1}o_{\tau+1}|h)}{p(a_{\tau+1}\sigma_{\tau+1}|h)} p(t'|ha_{\tau+1}o_{\tau+1}).
\end{aligned}$$

The next step writes out the fraction above as a conditional probability, using $O_{\tau+1}$ to denote the random variable for the observation at time step $\tau+1$.

$$\begin{aligned}
= & \; p(a_{\tau+1}\sigma_{\tau+1}|h) \sum_{o_{\tau+1}\in\sigma_{\tau+1}} Pr(O_{\tau+1} = o_{\tau+1}|ha_{\tau+1}, O_{\tau+1}\in\sigma_{\tau+1})p(t'|ha_{\tau+1}o_{\tau+1}) \\
= & \; p(a_{\tau+1}\sigma_{\tau+1}|h)E_{O_{\tau+1}|ha_{\tau+1},O_{\tau+1}\in\sigma_{\tau+1}}[p(t'|ha_{\tau+1}o_{\tau+1})] \\
= & \; p(a_{\tau+1}\sigma_{\tau+1}|h) \cdot p(t'|ha_{\tau+1}\sigma_{\tau+1}).
\end{aligned}$$

The last step here uses the natural definition of making a prediction given a *set* history: the expected value of the prediction for the test from a primitive history, where the expectation is taken over primitive histories that are consistent with the set history (i.e., an expectation over observations of the set history). □

Using this lemma, one can decompose the prediction for a set test into a product of one-step predictions. I drop the $\tau+$ from the actions and observations of the test for brevity (e.g., a_1 is the first action of the test, not necessarily the action at the first time step).

Theorem A.2. *For a set test $t = a_1\sigma_1 a_2\sigma_2 \dots a_k\sigma_k$, the prediction $p(t|h)$ is equal to*

$$\prod_{i=1}^{k} p(a_i\sigma_i|ha_1\sigma_1 \dots a_{i-1}\sigma_{i-1}).$$

Proof. The proof of this result comes from repeatedly applying Lemma A.1.

$$\begin{aligned}
p(t|h) &= p(a_1\sigma_1|h)p(a_2\sigma_2 \dots a_k\sigma_k|ha_1\sigma_1) \\
&= p(a_1\sigma_1|h)p(a_2\sigma_2|ha_1\sigma_1)p(a_3\sigma_3 \dots a_k\sigma_k|ha_1\sigma_1a_2\sigma_2) \\
&\vdots \\
&= \prod_{i=1}^{k} p(a_i\sigma_i|ha_1\sigma_1 \dots a_{i-1}\sigma_{i-1}).
\end{aligned}$$

□

APPENDIX B

Constraints on Linear PSR Parameters

The validity of a set of linear PSR parameters depends upon the predictions that are generated by those parameters. *If the predictions satisfy the axioms of probability, then the parameters are considered valid.* Given some initial state vector $x_\emptyset$, the predictions made by the parameters $\{m_{ao}, M_{ao} : \forall a, o\}$ are

$$p(a_1 o_1 \ldots a_\tau o_\tau | \emptyset) \overset{\text{def}}{=} x_\emptyset{}^\top M_{a_1 o_1} M_{a_2 o_2} \ldots M_{a_{\tau-1} o_{\tau-1}} m_{a_\tau o_\tau}. \tag{B.1}$$

The following are necessary and sufficient conditions for the predictions to be sound:

$$\forall k \geq 1,\ \forall t = a_1 o_1 \ldots a_k o_k, \qquad p(t|\emptyset) \geq 0 \tag{B.2}$$

$$\forall k \geq 1,\ \forall a_1 a_2 \ldots a_k, \qquad \sum_{o_1 o_2 \ldots o_k} p(a_1 o_1 \ldots a_k o_k | \emptyset) = 1 \tag{B.3}$$

$$\forall k \geq 0,\ \forall t = a_1 o_1 \ldots a_k o_k,\ \forall a, \qquad p(t|\emptyset) = \sum_o p(tao|\emptyset). \tag{B.4}$$

Note that each one of these equations specifies an infinite set of conditions, since t and k can each take on an infinite number of values. The remainder of this section proves that each of Equations B.3 and B.4 are equivalent to a finite set of conditions (given Equation B.2).

Towards that end, the following theorem proves that Equation B.3 can be replaced by

(B.5) $$\forall a \quad , \quad {x_\emptyset}^\top \sum_o m_{ao} = 1$$

and the conditions remain both necessary and sufficient for the parameters to be valid.

Theorem B.1. *Equations* B.*3 and* B.*5 are equivalent when Equation* B.*4 holds.*

Proof. I start with the left hand side of Equation B.3 and show that it reduces to the left hand side of Equation B.5 when Equation B.4 holds. Thus they will equal 1 for exactly the same parameters.

For any $k \geq 1$ and any $a_1 \dots a_k$,

$$\begin{aligned}\sum_{o_1 \dots o_k} p(a_1 o_1 a_2 o_2 \dots a_k o_k | \emptyset) &= \sum_{o_1 \dots o_{k-1}} \sum_{o_k} {x_\emptyset}^\top M_{a_1 o_1} \dots M_{a_{k-1} o_{k-1}} m_{a_k o_k} \\ &= \sum_{o_1 \dots o_{k-1}} {x_\emptyset}^\top M_{a_1 o_1} \dots M_{a_{k-2} o_{k-2}} m_{a_{k-1} o_{k-1}}\end{aligned}$$

where the last step applies Equation B.4 for $t = a_1 o_1 a_2 o_2 \dots a_{k-1} o_{k-1}$. Repeatedly applying Equation B.4 for each prefix $a_1 o_1 \dots a_{k-i} o_{k-i}$ reduces the sum to

$$= \sum_{o_1} {x_\emptyset}^\top m_{a_1 o_1},$$

which is just the expression given by the left hand side of Equation B.5. □

The next theorem proves that one can replace the infinite set of constraints given by Equation B.4 with the following finite set of constraints:

(B.6) $$\forall h \in H^*, a, o, a' \quad , \quad p(hao|\emptyset) = \sum_{o'} p(haoa'o'|\emptyset),$$

where H^* is a minimal basis set of sequences. (If the parameters are valid, then H^* will be core histories for the system.) The following definition aids in defining H^*:

(B.7) $${y_h}^\top \overset{\text{def}}{=} {x_\emptyset}^\top M_{a_1 o_1} M_{a_2 o_2} \dots M_{a_k o_k}$$

for any sequence $h = a_1o_1 \dots a_ko_k$.

Then $H^* = \{h_1, \dots, h_j\}$ must be such that $y_{h_1}, \dots, y_{h_j}$ forms a minimal basis for all y_h (i.e., for all histories h). That is, $y_{h_1}, \dots, y_{h_j}$ must be linearly independent, and y_h for any h must be linearly dependent upon $y_{h_1}, \dots, y_{h_j}$. Since each y_h vector has dimension n, one can find a valid H^* with no more than n elements, which makes Equation B.6 a finite set of constraints.

Theorem B.2. *Equation* B.*6 is equivalent to Equation* B.*4 when Equation* B.*5 holds.*

Proof. The first piece of the proof shows that Equation B.6 holds for all h if it holds for all $h \in H^*$. (The other direction is trivial: if it holds for all h, then it must hold for $h \in H^*$.)

Let $h_1, \dots, h_j$ be the histories of H^*, and define ψ as the matrix with i^{th} row equal to ${y_{h_i}}^\top$. Then ${y_h}^\top = w_h^\top \psi$ for some vector w_h (because y_h is linearly dependent upon $\{y_{h_i}\}$).

Thus, for any $h = a_1o_1 \dots a_ko_k$,

$$\begin{aligned} p(hao|\emptyset) &= {x_\emptyset}^\top M_{a_1o_1} M_{a_2o_2} \dots M_{a_ko_k} m_{ao} \\ &= {y_h}^\top m_{ao} = w_h^\top \psi m_{ao} \\ &= w_h^\top \begin{bmatrix} - & {y_{h_1}}^\top & - \\ - & {y_{h_2}}^\top & - \\ & \vdots & \\ - & {y_{h_j}}^\top & - \end{bmatrix} m_{ao} = w_h^\top \begin{bmatrix} p(h_1ao|\emptyset) \\ p(h_2ao|\emptyset) \\ \vdots \\ p(h_jao|\emptyset) \end{bmatrix} \end{aligned}$$

by definition of the y's. Using the assumption that Equation B.6 holds for all $h_i \in H^*$, this is equal to

$$= \sum_{o'} w_h^\top \begin{bmatrix} p(h_1 aoa'o'|\emptyset) \\ p(h_2 aoa'o'|\emptyset) \\ \vdots \\ p(h_j aoa'o'|\emptyset) \end{bmatrix} = \sum_{o'} w_h^\top \psi M_{ao} m_{a'o'} = \sum_{o'} {y_{hao}}^\top m_{a'o'},$$

where the last step uses the fact that ${y_{hao}}^\top = {y_h}^\top M_{ao} = (w_h^\top \psi) M_{ao}$. The resulting sum is just $\sum_{o'} p(haoa'o'|\emptyset)$, which completes the proof that Equation B.6 being satisfied for all $h \in H^*$ implies that it is satisfied for all histories.

The remaining part of the proof is to show that Equation B.6 holding for all h is equivalent to Equation B.4 being satisfied, when Equation B.5 is satisfied. In particular, Equation B.6 for all histories h (including the null history) is exactly the same as Equation B.4 holding for all t of length at least 1. The empty test $t_\emptyset$ is the only test of length less than 1, and Equation B.5 ensures that Equation B.4 is satisfied for that test. □

Finally, note that Equation B.6 is equivalent to

$$\text{(B.8)} \qquad \forall h \in H^*, a, o, a' \quad , \quad {y_h}^\top m_{ao} = {y_h}^\top M_{ao} \sum_{o'} m_{a'o'}$$

by definition of y_h and $p(t|\emptyset)$. Equation B.8 just expresses the constraint more directly in terms of the parameters. Notice that one way to satisfy this equation is when

$$\text{(B.9)} \qquad \forall a, o, a' \quad , \quad m_{ao} = M_{ao} \sum_{o'} m_{a'o'}.$$

Corollary B.3. *Equation* B.*9 is equivalent to Equation* B.*8 when* $|H^*| = n$.

Proof. Let ${y_{H^*}}^\top$ be the matrix with rows equal to ${y_h}^\top$ for each $h \in H^*$. Then

Equation B.8 is equivalent to

$$\forall a, o, a' \quad , \quad {y_{H^*}}^\top m_{ao} = {y_{H^*}}^\top M_{ao} \sum_{o'} m_{a'o'}.$$

When $|H^*| = n$, ${y_{H^*}}^\top$ is an $n \times n$ matrix that has full rank: the rows are linearly independent by definition of H^*. Left-multiplying the equation above by $({y_{H^*}}^\top)^{-1}$ yields Equation B.9. □

In summary, these theorems prove that the following conditions are both necessary and sufficient for a set of parameters make predictions that satisfy the axioms of probability:

$$\forall t, a, o \quad , \quad p(tao|\emptyset) = {y_t}^\top m_{ao} \geq 0 \tag{B.10}$$

$$\forall a \quad , \quad {x_\emptyset}^\top \sum_{o} m_{ao} = 1 \tag{B.11}$$

$$\forall h \in H^*, a, o, a' \quad , \quad {y_h}^\top m_{ao} = {y_h}^\top M_{ao} \sum_{o'} m_{a'o'}. \tag{B.12}$$

I showed that they are equivalent to the conditions given by Equations B.2, B.3, and B.4. Each of Equations B.3 and B.4 specifies an infinite set of conditions, and I replaced each equation with a finite set of conditions.

APPENDIX C

Representing Tests' Predictions in TD Networks

For any test t, one can define a node in a TD network such that its value is the prediction for t. That is, the node's value at history h is defined to be equal to $p(t|h)$, for *any* history h. I prove that this is possible constructively, using induction on the length $|t|$ of t.

For the base case of $|t| = 1$, let $t = a^*o^*$ and let x be a node in the TD network that is 1 if the most recent observation is o^* and 0 otherwise. Let y be a node that is connected to x with an edge that conditions upon a^* being the most recent action taken. Then by definition of connections in the TD network, the value of node y at time k is

$$E[x(k+1)|h, A_{k+1} = a^*],$$

where h is the history through time k, $x(k+1)$ is the random variable for value of node x at time $k+1$, A_{k+1} is the random variable for the action at time $k+1$, and the expectation is taken over the value of O_{k+1} (i.e., the random variable for the observation at time $k+1$). Since $x(k+1)$ will be 1 if $O_{k+1} = o^*$ and 0 otherwise, the value of node y at time k is equal to the probability that $O_{k+1} = o^*$ given h and $A_{k+1} = a^*$. This is exactly the prediction $p(a^*o^*|h) = p(t|h)$, so the base case is completed.

For the inductive step, $|t| > 1$, so I write $t = a^*o^*t'$ for some test t' of length one or more. By the inductive hypothesis, one can construct a node x in the TD network whose value at history h is $p(t'|h)$, for all histories. Let y be a node connected to x by an edge that has both a condition and a function f_y. The condition is the same as in the base case: that a^* is the most recent action taken. Then the value of node y at time k is

$$E[f_y(x(k+1))|h, A_{k+1} = a^*]$$

where h is the history through time k. Define

$$f_y(z) = \begin{cases} z, & \text{if } O_{k+1} = o^* \\ 0, & \text{otherwise} \end{cases}.$$

Next, I expand the expectation in the value of y at time $k+1$ by conditioning upon the event $O_{k+1} = o^*$:

$$\begin{aligned} & E[f_y(x(k+1))|h, A_{k+1} = a^*] \\ =& Pr(O_{k+1} = o^*|h, A_{k+1} = a^*) \cdot (f_y(x(k+1))|h, A_{k+1} = a^*, O_{k+1} = o^*) \\ +& Pr(O_{k+1} \neq o^*|h, A_{k+1} = a^*) \cdot E[f_y(x(k+1))|h, A_{k+1} = a^*, O_{k+1} \neq o^*]. \end{aligned}$$

Since $f_y(x(k+1)) = 0$ whenever $O_{k+1} \neq o^*$, the second term is equal to zero, leaving

$$\begin{aligned} & E[f_y(x(k+1))|h, A_{k+1} = a^*] \\ =& Pr(O_{k+1} = o^*|h, A_{k+1} = a^*) \cdot (f_y(x(k+1))|h, A_{k+1} = a^*, O_{k+1} = o^*) \\ =& p(a^*o^*|h)(f_y(x(k+1))|h, A_{k+1} = a^*, O_{k+1} = o^*) \text{ (by definition of prediction)} \\ =& p(a^*o^*|h)(x(k+1)|h, A_{k+1} = a^*, O_{k+1} = o^*) \text{ (by definition of } f_y) \\ =& p(a^*o^*|h)p(t'|ha^*o^*) \text{ (by definition of } x(k+1)) \\ =& p(a^*o^*t'|h) = p(t|h) \text{ (by definition of prediction).} \end{aligned}$$

This completes the inductive step, thereby proving that one can define a node in a TD network whose value is the prediction for an arbitrary test t.

APPENDIX D

Details of the Reset Algorithm for Learning a Linear PSR Model

This section describes the details of how the reset algorithm for learning a linear PSR (Section 1.3) accounts for noise in the estimated predictions. The main ideas in this appendix are the work of James and Singh (2004), who introduced the reset algorithm.

To find core tests and histories, the reset algorithm calculates the rank of different matrices of predictions. Golub and Van Loan (1996) address the problem of estimating the rank in a noisy matrix, using the singular values of the matrix. The exact rank of a matrix is the number of non-zero singular values. However, the noise in the matrix will almost certainly make all of the singular values non-zero. The method for estimating the rank is designed to separate the singular values that are non-zero only because of noise from the singular values that are truly non-zero in the non-noisy matrix. Thus, the estimated rank is the number of singular values that are greater than some cutoff value σ. For some estimated matrix $\hat{A}$, the cutoff value is $\sigma \overset{\text{def}}{=} \epsilon \parallel \hat{A} \parallel_\infty$, where ϵ is the average error in the matrix entries. The reset algorithm calculates ϵ as the product of a user-specified parameter and the average estimated standard deviation for each entry $\hat{a}$ in the matrix (i.e., $\sqrt{\hat{a}(1-\hat{a})n^{-1}}$, where n is the number of samples used to calculate $\hat{a}$).

When searching for core tests and histories, the reset algorithm needs to select linearly independent tests T_i' and histories H_i' from respective sets T_i and H_i. The method for doing this is different when dealing with estimated predictions than when the true predictions are available. When the true predictions were known, one could choose T_i' and H_i' by finding linearly independent rows and columns of $p(T_i|H_i)$. The particular linearly independent rows and columns that were selected did not affect the algorithm because the remaining rows and columns were linearly dependent on those selected. Furthermore, the rank r_i of $p(T_i|H_i)$ was equal to the number of tests and histories in T_i' and H_i', respectively, as this number is equal to the number of linearly independent rows and columns of $p(T_i|H_i)$.

When dealing with estimated predictions $\hat{p}(T_i|H_i)$ in place of $p(T_i|H_i)$, these equalities break down. Specifically, the algorithm estimates the rank of $p(T_i|H_i)$ as some $\hat{r}_i$ (as described above). Then it chooses $\hat{r}_i$ linearly independent tests and histories to form T_i' and H_i', respectively. However, because $\hat{r}_i$ is not the actual rank of the noisy, estimated $\hat{p}(T_i|H_i)$ — the noise in $\hat{p}(T_i|H_i)$ will almost certainly make it full rank, greater than $\hat{r}_i$ — even after choosing $\hat{r}_i$ tests for T_i', there will be tests remaining in T_i that are (in the estimated matrix) linearly independent of the selected T_i'. Thus, the algorithm must choose which linearly independent tests to put in T_i', because not all of them can be included. Similarly, the algorithm must choose which linearly independent histories to put in H_i', because some histories that are linearly independent (in the estimated matrix) must be left out: only $\hat{r}_i$ histories will be included.

Thus, when using estimated predictions, the algorithm seeks to find the $\hat{r}_i$ histories and tests that are "most linearly independent"; these will form H_i' and T_i', respectively. The original version of the reset algorithm used one method for finding these H_i' and T_i'; I have since developed a significantly faster procedure.

D.1 Finding the Most Linearly Independent Rows and Columns

Each procedure has at its core a method for finding the k "most linearly independent" columns of some $m \times n$ matrix A. In the original procedure of James and Singh (2004), the columns of A were removed one by one until k columns remained. To describe the process of removing columns, let A_i be the columns of A remaining after i columns have been removed. Given A_i, the matrix A_{i+1} is constructed by temporarily removing each column of A_i and calculating the condition number of the resulting matrix. The column which yielded the lowest condition number when it was removed is selected as the actual column to remove from A_i to get A_{i+1}. Note that, when using a basic singular value decomposition to calculate the condition number,[1] it takes $O(m(n-i)(n-i+m)(n-i))$ time to figure out which column to remove from A_i to get A_{i+1}. Since $n-k$ columns must be removed from A to get the k most linearly independent columns, this process takes time $O((n-k)n^2(n+m)m)$.

In contrast, I developed a new procedure that takes time $O(km(k+n)(k+m))$, which has degree four, whereas the previous procedure has degree five. Furthermore, in the core search algorithm, one would expect $k << n$, since n will be at least the size of the joint action/observation space (because T_i and H_i each include the one-step tests), while k should be no more than the rank of the system dynamics matrix. Thus, the k terms are practically insignificant when compared with the m and n terms, making the new procedure even more efficient when compared with the old procedure.

The new procedure works by selecting the desired k columns of A one by one, as

[1]Exact algorithms exist for computing the singular value decomposition of an $m \times n$ matrix in time $O(m(m+n)n)$ (cf. Golub & Van Loan, 1996).

opposed to removing $n-k$ columns one by one. After i columns have been selected, forming a matrix A_i, the $i+1$ column is added to A_i to form A_{i+1} by choosing the column vector that is furthest from the image of A_i (measured in Euclidean distance). Since the image of A_i is the set of vectors that are linearly dependent upon the columns of A_i, the column that is furthest from the image of A_i is "most" linearly independent. Computing the distance from A_i can be done by taking the singular value decomposition of A_i, which takes $O(i(i+m)m)$ time. Then $O((n-i)(im+m^2))$ multiplications are required to compute the distance of each of the $n-i$ vectors from the image of A_i. Since k vectors are added, the total running time of finding k linearly independent columns from an $m \times n$ matrix is $O(k(k(k+m)m+n(km+m^2))) = O(km(k(k+m)+n(k+m))) = O(km(k+n)(k+m))$.

D.2 The Core Search Process with Estimated Predictions

Either of these procedures to find the "most linearly independent" columns can be used when searching for core tests and histories in the reset algorithm. Each core search iteration consists of two calls to find the most linearly independent columns of a matrix. The first call is to find the $\hat{r}_i$ most linearly independent columns of $\hat{p}^\top(T_i|H_i)$; the histories for the selected columns are chosen as H_i'. The second call is to find the $\hat{r}_i$ most linearly independent columns of $\hat{p}(T_i|H_i')$; the tests for the selected columns are chosen as T_i'. Because each of H_i and T_i have size $O(|\mathcal{A}||\mathcal{O}|n)$, where n is the rank of the system-dynamics matrix,[2] the running time of one iteration of the core search procedure is $O(n(|\mathcal{A}||\mathcal{O}|n)(n+|\mathcal{A}||\mathcal{O}|n)(n+|\mathcal{A}||\mathcal{O}|n)) = O(n^4(|\mathcal{A}||\mathcal{O}|)^3)$. Section 2.2 proves that no more than n core search iterations are required, for a total

[2]These calculations assume that the estimated rank $\hat{r}_i$ is never larger than the true rank n of the system-dynamics matrix. In practice, because $\hat{r}_i$ is only an estimate, it can be larger than n. However, if n (or an upper bound on n) is known to begin with, then $\hat{r}_i$ can be capped at that value.

running time of $O(n^5(|\mathcal{A}||\mathcal{O}|)^3)$ to find the core tests and histories (stage 2).

In practice, the noise in the estimated predictions means that the core histories K and tests Q found in stage 2 may not yield a well-conditioned matrix $\hat{p}(Q|K)$, which is needed in stage 3 to solve for the model parameters. Thus, it is helpful to iteratively remove tests and histories from Q and K until the estimated rank of $\hat{p}(Q|K)$ is equal to its size. The selection of the tests and histories to remain in Q and K is done by finding the "most linearly independent" $m-1$ rows and columns of the $m \times m$ matrix $\hat{p}(Q|K)$, using either method described above. This will result in a new Q and K, and a new $m-1 \times m-1$ matrix $\hat{p}(Q|K)$. If the estimated rank of this matrix is $m-1$, then Q and K are taken as the core tests and histories. Otherwise, another iteration commences, reducing Q and K by one and checking the rank of that matrix. When the estimated rank of $\hat{p}(Q|K)$ is equal to the sizes of Q and K, then those Q and K are the core tests and histories used by stage 3.[3]

[3]Note that the running time of this additional loop $O(n^5)$ is dominated by the time of the core search procedure itself.

APPENDIX E

Linear PSR Parameters for Different Initial Conditions

The suffix-history algorithm (Section 2.1) takes experience from a system $\mathcal{D}$ and builds a linear PSR 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 distribution of $\mathcal{D}$ under the agent's policy during its experience in $\mathcal{D}$.

The learning algorithm for a hierarchical PSR (Chapter IV) also builds a model of a system $\mathcal{D}'$ that has the same dynamics as $\mathcal{D}$ but a different initial condition.

The remainder of this section provides proofs for three 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 of the linear PSR is different. In the following proofs, I use $p(\cdot)$ and $p'(\cdot)$ to denote predictions for $\mathcal{D}$ and $\mathcal{D}'$ respectively. Similarly, Q and Q' are sets of core tests for $\mathcal{D}$ and $\mathcal{D}'$, respectively. In reading the following proofs, it is useful to recall that for a matrix $Z = XY$, $\text{rank}(Z) \leq \min(\text{rank}(X), \text{rank}(Y))$.

(**Theorem 2.1**). Let $\mathcal{D}$ be a system-dynamics matrix with finite rank n that can be modeled by a POMDP P with n hidden states. Let P' be a POMDP model that is identical to P except for a different initial belief state, and let $\mathcal{D}'$ be the system-dynamics matrix generated by the model P'. If the rank of $\mathcal{D}'$ is also n, then any set of core tests and update parameters for either $\mathcal{D}'$ or $\mathcal{D}$ are valid for both $\mathcal{D}'$ and $\mathcal{D}$.

Proof. The proof will show that minimal core tests and update parameters for $\mathcal{D}'$ are valid for $\mathcal{D}$. The other direction — the fact that minimal core tests and update parameters for $\mathcal{D}$ are valid for $\mathcal{D}'$ — follows immediately by swapping $\mathcal{D}$ with $\mathcal{D}'$ and P with P'. Lemma E.2 provides the generalization from minimal core tests to any set of core tests.

Let H be the set of all histories. Let B be an $\infty \times n$ matrix where the i^{th} row is the belief state of P in history h_i. Similarly, the i^{th} row of B' is defined as the belief state of P' in history h_i. Let $p(Q'|S) = p'(Q'|S)$ be an $n \times n$ matrix[1] where the $(i,j)^{th}$ element is the probability of core test j succeeding from POMDP state i. (S denotes the set of POMDP states, which are common between P and P'.) Note that the ranks of B, B', $p(Q'|S)$ and $p'(Q'|S)$ are upper bounded by their smallest dimension n.

Because Q and Q' are core tests for $\mathcal{D}$ and $\mathcal{D}'$, respectively, $\text{rank}(p(Q|H)) = \text{rank}(p'(Q'|H)) = n$. From the equation $p(Q|H) = Bp(Q|S)$, one can see that $\text{rank}(B) = n$. Similarly, from the equation $p'(Q'|H) = B'p(Q'|S)$ one can see that $\text{rank}(p(Q'|S)) = n$. Finally, given that $p(Q'|H) = Bp(Q'|S)$, and that $p(Q'|S)$ is a square matrix with full rank, one obtains that $B = p(Q'|H)p^{-1}(Q'|S)$. This implies that $\text{rank}(p(Q'|H)) = n$ because $\text{rank}(B) = n$ and the maximum possible rank of $p(Q'|H)$ is n. The fact that $\text{rank}(p(Q'|H)) = n$ proves that the Q' columns are linearly independent in $\mathcal{D}$. Therefore, the set of minimal core tests Q' for system $\mathcal{D}'$ must also be minimal core tests for the original system $\mathcal{D}$.

The linear PSR update parameters for $\mathcal{D}'$ can be computed from P' without reference to the initial belief state (Littman et al., 2001). Thus, the update parameters

[1]Because Q' are minimal core tests for a system with rank n, $|Q'| = n$.

computed for $\mathcal{D}'$ and $\mathcal{D}$ from the POMDPs P' and P will be identical, so parameters from $\mathcal{D}'$ will be valid for $\mathcal{D}$. □

(**Theorem 2.2**). Let $\mathcal{D}$ be a system-dynamics matrix and h^* be a history of $\mathcal{D}$ such that $p(h^*|\emptyset) > 0.0$. Let $\mathcal{D}'$ be a system-dynamics matrix such that $\mathcal{D}'$ has the same dynamics as $\mathcal{D}$, but its null history is identical to history h^* in $\mathcal{D}$. If $\text{rank}(\mathcal{D}) = \text{rank}(\mathcal{D}')$, then any set of core tests and update parameters for either $\mathcal{D}'$ or $\mathcal{D}$ are valid for both $\mathcal{D}'$ and $\mathcal{D}$.

Proof. The proof assumes that the sets of core tests under consideration are *minimal* core tests. Lemma E.2 provides the generalization from minimal core tests to any set of core tests.

First, note that all the rows of $\mathcal{D}'$ are rows of $\mathcal{D}$: the row of $\mathcal{D}'$ for history h is identical to the row of $\mathcal{D}$ for history h^*h. It immediately follows that minimal core tests Q for $\mathcal{D}$ are still core tests for $\mathcal{D}'$ (because any column is still a linear function of the Q columns, even after removing rows from $\mathcal{D}$ to form $\mathcal{D}'$). Because $\text{rank}(\mathcal{D}) = \text{rank}(\mathcal{D}')$, Q remains a minimal set of core tests. Finally, the linear relationships among the columns of $\mathcal{D}$ still hold after removing rows to form $\mathcal{D}'$, so the update parameters from $\mathcal{D}$ are valid in $\mathcal{D}'$.

The rest of the proof shows that minimal core tests Q' and update parameters from $\mathcal{D}'$ are valid in $\mathcal{D}$. The Q' columns are linearly independent in $\mathcal{D}'$, and adding rows to $\mathcal{D}'$ to form $\mathcal{D}$ cannot make them linearly dependent, so they remain a set of n linearly independent columns in $\mathcal{D}$. Thus, Q' forms a minimal set of core tests in $\mathcal{D}$.

It then remains to show that the update parameters for $\mathcal{D}'$ are valid for $\mathcal{D}$. Let K' denote minimal core histories for $\mathcal{D}'$, and let h^*K' denote the corresponding histories in $\mathcal{D}$. By assumption, for any test t and history h, $p(t|h^*h) = p(t|h)$. The m vector in $\mathcal{D}'$ for any test t is equal to $m_t = {p'}^{-1}(Q'|K')p'(t|K')$. In the other system $\mathcal{D}$, $m_t = p^{-1}(Q'|h^*K')p(t|h^*K')$, which is the same as in $\mathcal{D}'$. Since this holds for any test t, the update parameters found for Q' in $\mathcal{D}'$ are also update parameters for Q' in $\mathcal{D}$. $\square$

The following lemmas extend results from minimal sets of core tests to general sets of core tests.

Lemma E.1. *Every set of core tests contains a minimal set of core tests.*

Proof. Let T be an arbitrary set of core tests for a system whose system-dynamics matrix has rank n. For a set of core histories K, $\text{rank}(p(T|K)) = n$ (Corollary F.4). Therefore, there exist n linearly independent columns of $p(T|K)$, which correspond to some tests Q. Because $p(Q|K)$ has rank n (by construction), Q are core tests (Lemma F.3), and because $|Q| = n$, they are minimal core tests. $\square$

Lemma E.2. *If* minimal *core tests and update parameters are the same between two systems $\mathcal{D}$ and $\mathcal{D}'$, then* any *core tests and update parameters from one system are valid in the other.*

Proof. Without loss of generality, the proof shows that core tests and update parameters from $\mathcal{D}$ are valid in $\mathcal{D}'$. Let n be the rank of $\mathcal{D}$, which is also the rank of $\mathcal{D}'$ (because minimal core tests are the same between $\mathcal{D}$ and $\mathcal{D}'$). Let $T = \{t_1, t_2, \ldots, t_k\}$ be core tests in $\mathcal{D}$, with associated update parameters $m_{T,t}$ for any test t such that

$$\forall h, p(t|h) = p^\top(T|h)m_{T,t}.$$

Let Q be a subset of T that are minimal core tests for $\mathcal{D}$; Lemma E.1 proves the existence of such a Q. For any test t, there exists a vector $m_{Q,t}$ such that

$$\forall h, p(t|h) = p^\top(Q|h)m_{Q,t}.$$

Because T contains Q, which are also minimal core tests in $\mathcal{D}'$ (by assumption), T is also a set of core tests in $\mathcal{D}'$. Now it remains to show that the update parameters $m_{T,t}$ for T in $\mathcal{D}$ are also valid in $\mathcal{D}'$.

Let

$$M \overset{\text{def}}{=} \begin{bmatrix} | & | & & | \\ m_{Q,t_1} & m_{Q,t_2} & \cdots & m_{Q,t_k} \\ | & | & & | \end{bmatrix}.$$

Then

$$\begin{aligned} \forall h, p(t|h) &= p^\top(T|h)m_{T,t} \\ &= p^\top(Q|h)Mm_{T,t}. \end{aligned}$$

Therefore, $Mm_{T,t}$ satisfies the definition of $m_{Q,t}$. Since $m_{Q,t}$ maps the predictions for *minimal* core Q to the prediction for t, it is valid in $\mathcal{D}'$ as well as in $\mathcal{D}$. That is,

$$\forall h, p'(t|h) = p'^\top(Q|h)Mm_{T,t}.$$

Furthermore, each column of M is also valid in $\mathcal{D}'$, so $p'^\top(Q|h)M = p'^\top(T|h)$, which leads to

$$\forall h, p'(t|h) = p'^\top(T|h)m_{T,t}.$$

Thus, the update parameters $m_{T,t}$ from $\mathcal{D}$ are valid in $\mathcal{D}'$. $\square$

The following theorem states that if one builds a linear PSR of an MDP that has some initial state distribution, then (after the initial time step) that PSR will be an accurate model of the system that starts in any one of those possible initial states, as long as that starting state is reachable in the MDP at some point after the initial time step.

(**Theorem 2.3**). Let $\mathcal{D}$ be a system-dynamics matrix that can be modeled by an MDP M with states S. Let b_0 be the initial state distribution of M, and let M' be an MDP that is identical to M except for its initial state distribution b'_0. If b'_0 satisfies the following property — $\forall s_i \in S$ such that $b'_0(s_i) > 0$, it holds that $b_0(s_i) > 0$ and s_i is reachable at some time $t > 0$ — then core tests and update parameters for $\mathcal{D}$ are valid core tests and update parameters for the system-dynamics matrix $\mathcal{D}'$ given by M'.

Proof. For an MDP system, changing the initial state distribution does not change any row of the corresponding system-dynamics matrix 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. (A history h is unreachable if $p(h|\emptyset) = 0.0$.) The condition $\forall s_i \in S, b'_0(s_i) > 0 \Rightarrow b_0(s_i) > 0$ ensures that any reachable history in $\mathcal{D}'$ is also reachable in $\mathcal{D}$. Thus, if one ignores the null history row, $\mathcal{D}'$ is a sub-matrix 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. I 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 tests and let $p(T|s_i)$ be the vector of predictions for each 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. The 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 a linear combination of other rows of $\mathcal{D}'$. Finally, since the linear relationships among the columns of $\mathcal{D}'$ are the same as those for $\mathcal{D}$, the update parameters from $\mathcal{D}$ are valid in $\mathcal{D}'$. $\square$

APPENDIX F

Auxiliary Results for Bounding Core Test and History Length

This appendix provides lemmas and notation to support Theorems 2.4 and 2.5 from Section 2.2.1. These auxiliary results are not necessary for an intuitive understanding of Theorems 2.4 and 2.5, but are a necessary part of the proofs. Furthermore, several of the results and techniques used here are sufficiently general that they may provide insights into properties of linear PSRs or system-dynamics matrices that are unknown as of yet.

Section F.1 begins with some basic existing properties of core histories and core tests. The remaining two sections describe some of my contributions to the theory of linear PSRs: Section F.2 describes a model that is similar to the linear PSR model but uses a state defined by linearly independent rows of the system-dynamics matrix (instead of linearly independent columns, like the linear PSR). I use this model to bound the length of core histories, but it may also be an interesting model itself. Section F.3 uses the results from the previous two sections to provide the supporting lemmas for the bounds on core history and test length.

F.1 Basic Properties of Core Histories and Core Tests

The following properties of core histories and core tests are true because their rows and columns of $\mathcal{D}$ form bases for the row and column spaces of $\mathcal{D}$, respectively. The fact that the entries of $\mathcal{D}$ are predictions does not come to bear in this section. These properties are rather elementary, and several of them have been derived in the course of proving existing results about linear PSRs, but they are stated here explicitly for completeness and for later reference.

In the following proofs, I will use the following fact from linear algebra: for any matrices $AB = C$, $\text{rank}(A) \geq \text{rank}(C)$ and $\text{rank}(B) \geq \text{rank}(C)$. A few special cases that follow directly from this fact are worth noting explicitly. First, if C is the product of several matrices, the rank of C is less than or equal to the smallest rank of any of the matrices in the product. Second, if C is a sub-matrix of some B, then $\text{rank}(C) \leq \text{rank}(B)$, since one can write $C = LBR$ for some matrices L and R that effectively select the proper rows and columns from B to form C. Third, if $C = B_1B_2 \ldots B_k$ and any B_i has either $\text{rank}(C)$ rows or columns, then $\text{rank}(B_i) = \text{rank}(C)$.

Lemma F.1. *A set of tests T are core tests if and only if there exists a set of histories H such that $p(T|H)$ has rank n, where n is the rank of $\mathcal{D}$.*

Proof. If T are core tests, then the T columns of $\mathcal{D}$ have rank n, since they form a basis of the column space of $\mathcal{D}$. Therefore, one can find n linearly independent rows of the T columns; the histories H corresponding to those rows satisfy $\text{rank}(p(T|H)) = n$.

If $p(T|H)$ has rank n for some H, then the T columns of $\mathcal{D}$ have rank no less than n. Since the whole of $\mathcal{D}$ has rank n, the T columns must have rank exactly n. Since they are columns of $\mathcal{D}$ that have rank equal to n, they form a basis of the column space of $\mathcal{D}$, which makes T a set of core tests. $\square$

A similar statement can be made for core histories:

Lemma F.2. *A set of histories H are core histories if and only if there exists a set of tests T such that $p(T|H)$ has rank n, where n is the rank of $\mathcal{D}$.*

The proof is analogous to that of Lemma F.1, interchanging "tests" with "histories" and "rows" with "columns."

Lemma F.3. *The following statements are equivalent for any $\mathcal{D}$ with rank n, any set of histories H, and any set of tests T:*

1. $\operatorname{rank}(p(T|H)) = n$
2. *H are core histories and T are core tests*
3. *$\mathcal{D} = W_{H_\infty}^\top p(T|H) M_{T_\infty}$ for some matrices W_{H_∞} and M_{T_∞} such that $W_{H_\infty}^\top p(T|H)$ are the T columns of $\mathcal{D}$ and $p(T|H)M_{T_\infty}$ are the H rows of $\mathcal{D}$.*

Proof. First note that (1) $\Rightarrow$ (2) follows from Lemmas F.1 and F.2. The next step is to show that (2) $\Rightarrow$ (3). Let T_∞ be the set of all tests and let H_∞ be the set of all histories. Because T are core tests, the T columns form a basis of the column space of $\mathcal{D}$. That is, for any test t, there exists some m_t such that $p(t|h) = p^\top(T|h)m_t$ for any history h. One can define M_{T_∞} as the matrix with one column for each test, where the column for test t is m_t. Then by definition of M_{T_∞}, $p(T|H)M_{T_\infty} = p(T_\infty|H)$ are the H rows of $\mathcal{D}$.

Similarly, because H are core histories, for any history h there exists some w_h such that $p(t|h) = w_h^\top p(t|H)$. Define W_{H_∞} to have one column for each history, with the column for history h defined as w_h. Then by definition of W_{H_∞}, $W_{H_\infty}^\top p(T|H) = p(T|H_\infty)$ are the T columns of $\mathcal{D}$.

Thus, $W_{H_\infty}^\top p(T|H) M_{T_\infty} = p(T|H_\infty) M_{T_\infty} = p(T_\infty|H_\infty) = \mathcal{D}$, which completes the proof that (2) $\Rightarrow$ (3).

The final step is to show that (3) $\Rightarrow$ (1). Since $\text{rank}(\mathcal{D}) = \text{rank}(W_{H_\infty}^\top p(T|H) M_{T_\infty}) = n$, $\text{rank}(p(T|H)) \geq n$, and because $p(T|H)$ is a sub-matrix of $\mathcal{D}$, $\text{rank}(p(T|H)) \leq n$. Together, these imply $\text{rank}(p(T|H)) = n$. □

Corollary F.4. *For any set of core histories H, T is a set of core tests if and only if* $\text{rank}(p(T|H)) = n$. *Also, for any set of core tests T, H is a set of core histories if and only if* $\text{rank}(p(T|H)) = n$.

Proof. There are four brief pieces to this proof, each of which follows immediately from the equivalence of the statements in Lemma F.3:

- *Given a set of core histories H, if T is a set of core tests then* $\text{rank}(p(T|H)) = n$.

  This follows from the fact that (2) $\Rightarrow$ (1) in Lemma F.3.

- *Given a set of core histories H, if* $\text{rank}(p(T|H)) = n$, *then T is a set of core tests.*

  This follows from the fact that (1) $\Rightarrow$ (2) in Lemma F.3.

- *Given a set of core tests T, if H is a set of core histories then* $\text{rank}(p(T|H)) = n$.

  This follows from the fact that (2) $\Rightarrow$ (1) in Lemma F.3.

- *Given a set of core tests T, if* $\text{rank}(p(T|H)) = n$, *then H is a set of core histories.*

  This follows from the fact that (1) $\Rightarrow$ (2) in Lemma F.3.

□

F.2 A Model Based Upon Linearly Independent Rows of $\mathcal{D}$

This section presents a model that is similar to the linear PSR, but its state is based upon linearly independent *rows* of the system-dynamics matrix $\mathcal{D}$ (whereas the linear PSR state is based upon linearly independent *columns* of $\mathcal{D}$). The idea behind the model is that any row of the system-dynamics matrix $\mathcal{D}$ is a unique linear combination of the rows for a set of minimal core histories K. The state representation of the model at history h is that unique vector of weights w_h that specify how to combine the K rows to get the h row of $\mathcal{D}$. That is, $w_h^\top p(T|K) = p^\top(T|h)$ for any set of tests T.

The remainder of this section describes the state update process – how w_{hao} can be calculated from model parameters and w_h – and how the model can make any prediction from h using w_h and the model parameters. These procedures are derived from the linear PSR state update and prediction procedures. The state representation and update function of this new model are used in the following section for bounding the length of core histories.

To derive the state update procedure of calculating w_{hao} from w_h, let Q be a set of minimal core tests. Then $p^\top(Q|hao) = w_{hao}^\top p(Q|K)$ by definition of w_{hao}, so the desired $w_{hao}^\top$ is equal to $p^\top(Q|hao)p^{-1}(Q|K)$. The $p^\top(Q|hao)$ can be broken down via the linear PSR state update equation:

$$p^\top(Q|hao) = \frac{p^\top(Q|h)M_{ao}}{p^\top(Q|h)m_{ao}} = \frac{w_h^\top p(Q|K)M_{ao}}{w_h^\top p(Q|K)m_{ao}}$$

where the last step used the fact that $p^\top(Q|h) = w_h^\top p(Q|K)$. Then one can use the fact that $M_{ao} = p^{-1}(Q|K)p(aoQ|K)$ and $m_{ao} = p^{-1}(Q|K)p(ao|K)$ to simplify to

$$p^\top(Q|hao) = \frac{w_h^\top p(Q|K)p^{-1}(Q|K)p(aoQ|K)}{w_h^\top p(Q|K)p^{-1}(Q|K)p(ao|K)} = \frac{w_h^\top p(aoQ|K)}{w_h^\top p(ao|K)}.$$

Plugging this back into the equation for w_{hao} shows what model parameters one needs to update state:

$$w_{hao}^\top = p(Q|hao)p^{-1}(Q|K) = \frac{w_h^\top p(aoQ|K)}{w_h^\top p(ao|K)} p^{-1}(Q|K) = \frac{w_h^\top Z_{ao}}{w_h^\top z_{ao}}$$

where $Z_{ao} \overset{\text{def}}{=} p(aoQ|K)p^{-1}(Q|K)$ and $z_{ao} \overset{\text{def}}{=} p(ao|K)$ are the parameters of this new model, along with the initial state $w_\emptyset$.

Like with a linear PSR, the prediction for any test from history h is a history-independent linear function of the state w_h at history h. For a test $t = a_1o_1a_2o_2 \ldots a_ko_k$, the prediction $p(t|h)$ equals

$$\begin{array}{l}
\quad\;\; p^\top(Q|h) \qquad\qquad M_{a_1o_1} \qquad\qquad\qquad M_{a_2o_2} \qquad\quad \cdots \qquad\quad M_{a_{k-1}o_{k-1}} \qquad\qquad\qquad m_{a_ko_k} \\
= w_h^\top \; p(Q|K) \;\; p^{-1}(Q|K)p(a_1o_1Q|K)p^{-1}(Q|K)p(a_2o_2Q|K) \cdots p^{-1}(Q|K)p(a_{k-1}o_{k-1}Q|K)p^{-1}(Q|K)p(a_ko_k|K) \\
= w_h^\top \qquad\quad I_n \qquad\qquad\qquad Z_{a_1o_1} \qquad\qquad\qquad \cdots \qquad Z_{a_{k-2}o_{k-2}} \qquad\qquad Z_{a_{k-1}o_{k-1}} \qquad\qquad\quad z_{a_ko_k} \\
= w_h^\top z_{a_1o_1\ldots a_ko_k}.
\end{array}$$

F.2.1 Properties of the w_h

The predictions within the predictive state of a linear PSR are constrained to fall in the $[0, 1]$ range of valid probabilities. Since the entries of w_h are not probabilities, they do not need to fall in a specified range. However, there is another constraint that must hold for every w_h: its entries must sum to 1.

Let t^* be a test that always succeeds, i.e., $\forall h, p(t^*|h) = 1$. One can write $p(t^*|h) = \sum_{o\in\mathcal{O}} p(ao|h)$. Then $\forall h, 1 = p(t^*|h) = \sum_{o\in\mathcal{O}} p(ao|h) = \sum_{o\in\mathcal{O}} w_h^\top z_{ao} = w_h^\top \sum_{o\in\mathcal{O}} p(ao|K) = w_h^\top \mathbf{1}$, where $\mathbf{1}$ is the $n \times 1$ vector of 1's. This proves that the elements of w_h must sum to 1 for any history:

$$\forall h, w_h^\top \mathbf{1} = 1.$$

Another point worth mentioning is that the denominator in the state update equation ($w_h^\top z_{ao}$) is equal to $p(ao|h)$, just as in the linear PSR update equation. This follows from the fact that $z_{ao} = p(ao|K)$ and $w_h^\top p(ao|K) = p(ao|h)$.

F.3 Supporting Lemmas

In this section, I develop the remaining supporting lemmas for Theorems 2.4 and 2.5 from Section 2.2.1 (i.e., bounding the length of core histories and tests). The first lemmas address core tests, while the remaining lemmas mirror those results for core histories. Recall that many of the steps in these proofs rely upon the linear algebra properties listed at the beginning of Section F.1.

Lemma F.5. *Let Q be a set of n linear core tests for some $\mathcal{D}$ with rank n, and define m_t as the unique solution to $p^\top(Q|h)m_t = p(t|h)$ for all histories h. Another set of n tests $T = \{t_1, \ldots, t_n\}$ is a set of linear core tests for $\mathcal{D}$ if and only if the matrix*

$$M_T \overset{def}{=} \begin{bmatrix} | & | & & | \\ m_{t_1} & m_{t_2} & \cdots & m_{t_n} \\ | & | & & | \end{bmatrix}$$

has rank n.

Proof. The first step is to show that T are core tests if M_T has rank n. For any set of histories H, $p(T|H) = p(Q|H)M_T$. Since M_T is full rank, $p(T|H)M_T^{-1} = p(Q|H)$. Letting H be a set of core histories, then $p(Q|H)$ has rank n (Lemma F.3 in Section F.1), which implies that $p(T|H)$ has rank n, which implies that T are core tests (Corollary F.4 in Section F.1).

The rest of the proof shows that M_T has rank n if T are core tests. Letting H be a set of core histories, $p(T|H) = p(Q|H)M_T$ has rank n (Lemma F.3 in Section F.1), which implies that M_T has rank n. □

Lemma F.6. *Let $\mathcal{M}_i$ be the subspace spanned by the set of vectors $\{m_t : |t| \leq i\}$, where the m_t are defined with respect to the set of core tests Q and $|t|$ denotes the length of the test t. If $\operatorname{rank}(\mathcal{D}) = n$, then for all $x \geq n$, $\dim(\mathcal{M}_n) = \dim(\mathcal{M}_x) = n$.*

Proof. The first step is to define an operator F that maps a (possibly infinite) set of vectors V to another set of vectors $F(V)$. The operator F is defined such that $F(\mathcal{M}_i) = \mathcal{M}_{i+1}$ for all $i \geq 1$.[1]

$$F(V) \overset{\text{def}}{=} \text{span}\left(\left(\bigcup_{a,o}\{M_{ao}v : \forall v \in V\}\right) \cup V\right)$$

where span denotes the subspace spanned by a set of vectors.

To prove that $F(\mathcal{M}_i) = \mathcal{M}_{i+1}$ for all $i \geq 1$, I will first show that any vector in $F(\mathcal{M}_i)$ is also contained in $\mathcal{M}_{i+1}$. Note that any vector in $F(\mathcal{M}_i)$ can be written as a linear combination of vectors in the set $G(\mathcal{M}_i) \overset{\text{def}}{=} \left(\bigcup_{a,o}\{M_{ao}v : \forall v \in \mathcal{M}_i\}\right) \cup \mathcal{M}_i$, since $F(\mathcal{M}_i) = \text{span}(G(\mathcal{M}_i))$. Now if each $g \in G(\mathcal{M}_i)$ is in $\mathcal{M}_{i+1}$, then each $f \in F(\mathcal{M}_i)$ is also in $\mathcal{M}_{i+1}$, because $\mathcal{M}_{i+1}$ is a subspace: that is, any linear combination (e.g., f) of vectors in $\mathcal{M}_{i+1}$ (e.g., $G(\mathcal{M}_i)$) is also contained in $\mathcal{M}_{i+1}$. Thus, the problem is now reduced to showing that each $g \in G(\mathcal{M}_i)$ is in $\mathcal{M}_{i+1}$.

By definition of $G(\mathcal{M}_i)$, each $g \in G(\mathcal{M}_i)$ is either contained in $\mathcal{M}_i$ itself or is of the form $g = M_{ao}v$ for some $v \in \mathcal{M}_i$. If $g \in \mathcal{M}_i$, then $g \in \mathcal{M}_{i+1}$, because $\mathcal{M}_i \subseteq \mathcal{M}_{i+1}$ (by definition of $\mathcal{M}_i$). For the other case – $g = M_{ao}v$ for some $v \in \mathcal{M}_i$ – let α_v be a vector such that $v = M_{\leq i}\alpha_v$, where the columns of the matrix $M_{\leq i}$ are defined as the m_t vectors for all tests of length no greater than i. Then $g = M_{ao}v = M_{ao}M_{\leq i}\alpha_v$. Since $m_{aot} = M_{ao}m_t$ for any a, o, t, each column of $M_{ao}M_{\leq i}$ is an m-vector for a test of length no greater than $i + 1$. Thus g is in the span of the m-vectors for tests of length no greater than $i + 1$, which is exactly the definition of $\mathcal{M}_{i+1}$.

Now that $F(\mathcal{M}_i) \subseteq \mathcal{M}_{i+1}$ has been proven, it remains to show that $\mathcal{M}_{i+1} \subseteq F(\mathcal{M}_i)$. Any vector $\beta \in \mathcal{M}_{i+1}$ can be written as $M_{\leq i+1}\alpha_\beta$ for some vector α_β. Thus, if each column of $M_{\leq i+1}$ is in $F(\mathcal{M}_i)$, then any $\beta \in \mathcal{M}_{i+1}$ is also in $F(\mathcal{M}_i)$. (As above, I use the fact that a subspace (e.g., $F(\mathcal{M}_i)$) is closed under linear combination.)

[1] Each subspace $\mathcal{M}_i$ can be viewed as an infinite set of vectors.

Any column of $M_{\leq i+1}$ is equal to m_t for some test t of length no greater than $i+1$. If $|t| \leq i$, then m_t is also a column of $M_{\leq i}$, which implies that $m_t \in \mathcal{M}_i \subseteq F(\mathcal{M}_i)$. If $|t| = i+1$, then $m_t = m_{aot'} = M_{ao}m_{t'}$, where t' is the length-i suffix of t. Since $m_{t'} \in \mathcal{M}_i$, $m_t = M_{ao}m_{t'} \in F(\mathcal{M}_i)$.

This completes the proof that $F(\mathcal{M}_i) = \mathcal{M}_{i+1}$ for all $i \geq 1$. With that fact in hand, one can justify writing the series of subspaces $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3, \ldots$ as $F^0, F^1, F^2, \ldots$, where F^i denotes i applications of F to $\mathcal{M}_1$. The remainder of this proof shows that this series reaches a unique fixed point of $\mathbb{R}^n$ (for finite n), and that the smallest i^* at which the fixed point $F^{i^*} = \mathbb{R}^n$ is obtained is no more than n.

Because the m_t vectors for a set of core tests will span $\mathbb{R}^n$ (Lemma F.5), there must be some smallest i^* for which $F^{i^*} = \mathbb{R}^n$. Then for all $k \geq i^*$, $F^k = \mathbb{R}^n$. Specifically, $F^k \subseteq \mathbb{R}^n$ because the vectors of F^k are n-dimensional, and $F^k \supseteq \mathbb{R}^n$ because $F^k \supseteq F^{k-1} \supseteq F^{k-2} \cdots \supseteq F^{i^*} = \mathbb{R}^n$. (Recall that $F^k \supseteq F^{k-1}$ for all $k \geq 1$ by definition of F.)

It remains to show that $i^* \leq n$. First, note that $F^i \subseteq F^{i+1}$, and $F^i = F^{i+1}$ only for $i \geq i^*$ (by uniqueness of the fixed point and definition of i^*), which implies that $F^i \subset F^{i+1}$ for all $i < i^*$. Since each F^i is a subspace $F^i \subset F^{i+1}$ implies $\dim(F^i) < \dim(F^{i+1})$. Thus the series of dimensions $\dim(F^0), \dim(F^1), \dim(F^2), \ldots$ is strictly increasing until $\dim(F^{i^*})$. Since the series cannot increase greater than n and it begins with $\dim(F^0) \geq 1$, it must reach the fixed point of n at $i^* < n$. Because $\dim(F^i) = \dim(\mathcal{M}_{i+1})$, $\dim(\mathcal{M}_x) = n$ for all $x \geq n$. □

(Theorem 2.4). For any system-dynamics matrix of rank n, there exists some set T of core tests such that any $t \in T$ has length no greater than n.

Proof. Lemma F.6 proves that $\dim(\mathcal{M}_n) = n$, where the m_t vectors in $\mathcal{M}_n$ are defined with respect to some set of core tests Q. Therefore, there exists a set of n tests T of length n or shorter such that the m_t's for the tests of T are linearly independent, which makes T core tests (Lemma F.5). □

The remainder of this section proves analogous results for core histories: Lemmas F.7 and F.8 are the history analogues to Lemmas F.5 and F.6, and Theorem 2.5 gives the bound on the maximum required core history length. The following proofs for core histories exactly mirror the proofs above for core tests except for the first portion of Lemma F.8. The proofs use the w_h, Z_{ao}, and z_{ao} notation of the model presented in Section F.2.

Lemma F.7. *Let K be a set of n linear core histories for some $\mathcal{D}$ with rank n, and define w_h as the unique solution to $w_h^\top p(t|K) = p(t|h)$ for all tests t. Another set of n histories $H = \{h_1, \ldots, h_n\}$ is a set of linear core histories for $\mathcal{D}$ if and only if the matrix*

$$W_H \overset{def}{=} \begin{bmatrix} | & | & & | \\ w_{h_1} & w_{h_2} & \cdots & w_{h_n} \\ | & | & & | \end{bmatrix}$$

has rank n.

Proof. The first step is to show that H are core histories if W_H has rank n. For any set of tests T, $p(T|H) = W_H^\top p(T|K)$. Since W_H is full rank, $(W_H^\top)^{-1} p(T|H) = p(T|K)$. Letting T be a set of core tests, then $p(T|K)$ has rank n (Lemma F.3 in Section F.1), which implies that $p(T|H)$ has rank n, which implies that H are core histories (Corollary F.4 in Section F.1).

The rest of the proof shows that W_H has rank n if H are core histories. Letting T be a set of core tests, $p(T|H) = W_H^\top p(T|K)$ has rank n (Lemma F.3 in Section F.1), which implies that W_H has rank n. □

Lemma F.8. *Let $\mathcal{W}_i$ be the subspace spanned by the set of vectors $\{w_h : |h| \leq i\}$, where the w_h are defined with respect to a set of core histories K. If* $\operatorname{rank}(\mathcal{D}) = n$, *then for all $x \geq n$,* $\dim(\mathcal{W}_n) = \dim(\mathcal{W}_x) = n$.

Proof. Following the analogous proof for core tests (i.e., Lemma F.6), I will define an operator F such that $F(\mathcal{W}_i) = \mathcal{W}_{i+1}$ for all $i \geq 1$. Because w_{hao} is a non-linear function of w_h, some additional machinery is required to define F. In particular, I define a vector v_h for any h that is a scalar multiple of w_h. Furthermore, v_{hao} is a linear transformation of v_h, which will permit the construction of the operator F.

The vector v_h is defined recursively: $v_{hao} \overset{\text{def}}{=} Z_{ao}^\top v_h$ with the base case $v_\emptyset \overset{\text{def}}{=} w_\emptyset$. Note that $v_h = p(h|\emptyset)w_h$, by induction on the length of h. For the base case $h = \emptyset$, $v_h = 1w_h$ holds. Then for any h, a, o,

$$\begin{aligned}
v_{hao} &= Z_{ao}^\top v_h \\
&= p(h|\emptyset) Z_{ao}^\top w_h \\
&= p(h|\emptyset)(w_h^\top p(ao|K)) \frac{Z_{ao}^\top w_h}{w_h^\top p(ao|K)} \\
&= p(h|\emptyset) p(ao|h) w_{hao} \\
&= p(hao|\emptyset) w_{hao}
\end{aligned}$$

where the second line employs the inductive hypothesis.

Since each v_h is just a scalar multiple of w_h (i.e., $v_h = p(h|\emptyset)w_h$), the v_h's and w_h's for a given set of *reachable* histories[2] will always span the same subspace. More specifically, $\mathcal{W}_i$ is equal to the subspace spanned by the v_h's for all $|h| \leq i$. Then the following operator F satisfies $F(\mathcal{W}_i) = \mathcal{W}_{i+1}$ for all $i \geq 1$:

$$F(V) \overset{\text{def}}{=} \operatorname{span}\left(\left(\bigcup_{a,o} \{Z_{ao}^\top v : \forall v \in V\}\right) \cup V\right),$$

[2]A history h is *reachable* if and only if $p(h|\emptyset) > 0.0$.

by definition of v_h.

The remainder of the proof exactly mirrors the proof of Lemma F.6, replacing $\mathcal{M}_i$ with $\mathcal{W}_i$ and using Lemma F.7 to guarantee that the fixed point of the $\mathcal{W}_i$ series is $\mathbb{R}^n$. $\square$

(**Theorem 2.5**). For any system-dynamics matrix of rank n, there exists some set H of core histories such that any $h \in H$ has length no greater than n.

Proof. Lemma F.8 proves that $\dim(\mathcal{W}_n) = n$, where the w_h vectors in $\mathcal{W}_n$ are defined with respect to some set of core histories K. Therefore, there exists a set of n histories H of length n or shorter such that the w_h's for the histories of H are linearly independent, which makes H core histories (Lemma F.7). $\square$

These bounds on the length of core tests and histories give a finite portion of the system-dynamics matrix that is sufficient for building a linear PSR model of the system: the portion of the system-dynamics matrix for histories of length n of less and tests of length $n + 1$ or less (because one needs core test extensions to solve for the model parameters). Put another way, a learning algorithm only needs to estimate the predictions for history/test pairs of length $2n + 1$ or less.

APPENDIX G

Proofs for PSR Models of k^{th}-order Markov Systems

These results use 1^{st}-order Markov systems or MDPs as example systems. The proofs hinge upon the fact that the transition matrices of an MDP are actually transposed sub-matrices of the system-dynamics matrix $\mathcal{D}$. Recall that an MDP consists of an initial state and one *transition matrix* $\mathcal{T}_a$ for each action a. The $(i,j)^{th}$ entry of $\mathcal{T}_a$ is the probability of transitioning from latent state j to latent state i after taking action a. In an MDP, these probabilities are just predictions in $\mathcal{D}$: $p(a'o'|hao) = [\mathcal{T}_{a'}]_{o',o}$. To write this in matrix form, let H be a set of histories that contains exactly one history ending in each observation, and let T_a be the set of one-step tests with action a. Then

$$p(T_a|H) = \mathcal{T}_a^\top.$$

If T is the union of all the T_a, then define

$$Z \overset{\text{def}}{=} p(T|H) = [\mathcal{T}_{\mathcal{A}^1}^\top \;\; \mathcal{T}_{\mathcal{A}^2}^\top \;\; \cdots \;\; \mathcal{T}_{\mathcal{A}^{|\mathcal{A}|}}^\top].$$

Note that T contains core tests (Theorem 2.7) and H contains core histories because each row in $\mathcal{D}$ is identical to the row for some $h \in H$ (except the null-history row, which is a linear combination of the Z rows, according to the initial state distribution of the MDP). This means that the rank of Z is equal to the rank of $\mathcal{D}$ (Lemma F.3).

(**Theorem 2.11**). There exist MDP systems where no set of one-step core tests Q exists such that each test in Q contains the same action.

Proof. The proof is by example: an MDP with 3 observations (states) and 2 actions. Let the initial state always be state 1. Along with that initial state distribution, the following Z matrix completely defines the system (since it is composed of the transition matrices for the MDP):

$$\begin{bmatrix} 1 & 0 & 0 & \vdots & 0 & 0.5 & 0.5 \\ 0 & 0.5 & 0.5 & \vdots & 0 & 0.5 & 0.5 \\ 0 & 0.5 & 0.5 & \vdots & 1 & 0 & 0 \end{bmatrix}$$

The left half of Z is the transposed transition matrix for $\mathcal{A}^1$ and the right half is for $\mathcal{A}^2$. The whole Z matrix has rank 3, so $\mathcal{D}$ has rank 3. Therefore, 3 linear core tests would be required in a model of this system. Because any row of $\mathcal{D}$ corresponds to a row of Z, columns of Z are linearly independent if and only if the corresponding columns of $\mathcal{D}$ are linearly independent. Therefore, since each transition matrix in Z only has rank 2, there is no set of 3 one-step core tests that contain the same action. □

(**Theorem 2.12**). There exist MDP systems where, for any set of minimal core tests Q, there is some action a and observation o such that the parameter vector m_{ao} of a linear PSR contains at least one negative entry.

Proof. The proof is by example, using the same MDP as above. I will refer to tests and their columns interchangeably; context will make clear the intended meaning. Also, I will denote the i^{th} column (from left to right) of the Z matrix above as z_i.

The first case to consider is when Q is composed entirely of one-step tests. As shown above, the three core tests must contain tests from both halves of Z (i.e., tests with different actions). There are four distinct columns in Z, and any three of those columns would constitute a valid core set. There are four such possible core sets: $\{z_1, z_2, z_4\}$, $\{z_1, z_2, z_5\}$, $\{z_1, z_4, z_5\}$, and $\{z_2, z_4, z_5\}$.

- For the $\{z_1, z_2, z_4\}$ set, the unique m_{z_5} (to form the 5^{th} column) is $[0.5, 1.0, -0.5]^\top$.
- For the $\{z_1, z_2, z_5\}$ set, the unique m_{z_4} is $[1.0, 2.0, -2.0]^\top$.
- For the $\{z_1, z_4, z_5\}$ set, the unique m_{z_2} is $[-0.5, 0.5, 1.0]^\top$.
- For the $\{z_2, z_4, z_5\}$ set, the unique m_{z_1} is $[-2.0, 1.0, 2.0]^\top$.

Since each of these m vectors contains a negative entry, this completes the case where Q is composed of one-step tests.

In the general case, let $Q = \{q_1, q_2, q_3\}$. Recall that each of these core is a non-negative scalar multiple of some z_i (Lemma 2.6 in Section 2.2.2). Let $z(t)$ be the z_i of which t is a scalar multiple, and let $c(t)$ be that scalar; so $t = c(t)z(t)$. Then the fact that q_1, q_2, and q_3 are linearly independent means that $z(q_1) = (c(q_1))^{-1}q_1$, $z(q_2) = (c(q_2))^{-1}q_2$, and $z(q_3) = (c(q_3))^{-1}q_3$ are linearly independent. (Note that the inverse of each $c(q_i)$ is well-defined, because if $c(q_i)$ was 0, then q_i would be a vector of 0's and would not be a core test.) Because they are linearly independent, $Z(Q) \overset{\text{def}}{=} \{z(q_1), z(q_2), z(q_3)\}$ must be one of the sets of core columns of Z examined above.

For $Z(Q) = \{z_1, z_2, z_4\}$, the m_{z_5} vector would be $\frac{1}{c(q_1)c(q_2)c(q_3)}[0.5, 1.0, -0.5]^\top$. Since each $c(q_i)$ is positive (i.e., a probability), the resulting m_{z_5} vector contains a negative entry. Similarly,

- for $Z(Q) = \{z_1, z_2, z_5\}$, the unique m_{z_4} is $\frac{1}{c(q_1)c(q_2)c(q_3)}[1.0, 2.0, -2.0]^\top$.
- For $Z(Q) = \{z_1, z_4, z_5\}$, the unique m_{z_2} is $\frac{1}{c(q_1)c(q_2)c(q_3)}[-0.5, 0.5, 1.0]^\top$.

- For $Z(Q) = \{z_2, z_4, z_5\}$, the unique m_{z_1} is $\frac{1}{c(q_1)c(q_2)c(q_3)}[-2.0, 1.0, 2.0]^\top$.

□

(Theorem 2.13). There exist *deterministic* MDP systems where, for any set of minimal core tests Q, there is some action a and observation o such that the parameter vector m_{ao} of a linear PSR contains at least one negative entry.

Proof. The proof is by example, using the MDP described by the following Z matrix:

$$\begin{bmatrix} 1 & 0 & 0 & \vdots & 0 & 1 & 0 \\ 0 & 1 & 0 & \vdots & 0 & 1 & 0 \\ 1 & 0 & 0 & \vdots & 0 & 0 & 1 \end{bmatrix}$$

The argument is the same as that of Theorem 2.12, first showing exhaustively that when Q is all one-step tests, there is always some m_t with a negative entry. The table below shows that, for any set of one-step core, a negative entry is required in the m_t vector for another one-step test. The table lists the column numbers (in Z) of the core tests, the column number of the "target" test t, and the corresponding m_t, each of which contains a negative entry.

Core Columns	Column for t	m_t
1,2,5	6	$[1, 1, -1]^\top$
1,2,6	5	$[1, 1, -1]^\top$
1,5,6	2	$[-1, 1, 1]^\top$
2,5,6	1	$[-1, 1, 1]^\top$

The extension from one-step core test sets to any core test set is the same as in the proof above.

□

APPENDIX H

Proofs for Hierarchical PSRs

This appendix provides theoretical results related to temporal abstraction and PSRs. The notation and background are given in Chapter IV.

H.1 Defining Option Predictions in Terms of Primitive Predictions

The purpose of this section is to define the prediction for an option test t^ω from an option history h^ω in terms of primitive predictions. Before examining the prediction of an option test from an option history, it is helpful to first examine the prediction for an option test $t^\omega = \omega^1 o^1 \ldots \omega^k o^k$ from a primitive history h. To write this prediction $p(t^\omega|h)$ in terms of primitive predictions, one can condition upon the primitive sequences underlying each ω^i. Note that both the actions and observations of each primitive sequence are random variables in this context, because the option policy selects actions stochastically (and the dynamical system still emits observations stochastically).

To formally define $p(t^\omega|h)$, let τ be the length of the primitive history h. Define T^i as the random variable for the primitive sequence generated by the execution of ω^i. All of the probabilities in this discussion are conditioned upon the fact that the options $\omega^1 \omega^2 \ldots \omega^k$ are being executed in order from history h, but this is left out of the notation for simplicity.

One can condition upon T^1 to get

$$p(t^\omega|h) = \sum_{t^1} Pr(T^1 = t^1) Pr(O_{\tau+|t^1|} = o^1|T^1 = t^1) p(\omega^2 o^2 \dots \omega^k o^k|ht^1) \tag{H.1}$$

where $O_{\tau+|t^1|}$ is the random variable for the primitive observation at the last time step of ω_1. There are three terms in the product here, each of which is now examined in more detail.

- Looking at the term $Pr(T^1 = t^1)$, note that the event $T^1 = t^1$ accounts for three things:
  - the actions selected by the option ω^1 must match those of t^1;
  - the observations emitted by the system must match those of t^1;
  - and the option ω^1 must terminate after exactly $|t^1|$ time steps.

  This breakdown will be seen in the following expansion of the term $Pr(T^1 = t^1)$ using the chain rule of probability. Define $\beta^1(h')$ as the probability that ω^1 terminates in history h'. Without loss of generality, let $t^1 = a_1 o_1 \dots a_n o_n$. Then

$$\begin{aligned}
Pr(T^1 = t^1) = & Pr(a_1|h)Pr(o_1|ha_1)(1 - \beta^1(ha_1o_1)) \\
& Pr(a_2|ha_1o_1)Pr(o_2|ha_1o_1a_2)(1 - \beta^1(ha_1o_1a_2o_2)) \\
& \dots \\
& Pr(a_n|h \dots a_{n-1}o_{n-1})Pr(o_n|h \dots a_{n-1}o_{n-1}a_n)\beta^1(h \dots a_{n-1}o_{n-1}a_no_n) \\
= & (Pr(a_1|h) \cdots Pr(a_n|ha_1 \dots a_{n-1}o_{n-1})) \\
& (Pr(o_1|ha_1) \cdots Pr(o_n|ha_1 \dots o_{n-1}a_n)) \\
& \big((1 - \beta^1(ha_1o_1)) \cdots (1 - \beta^1(ha_1o_1 \dots a_{n-1}o_{n-1}))(\beta^1(ht^1))\big)
\end{aligned}$$

$$= (Pr(a_1|h) \cdots Pr(a_n|ha_1 \ldots a_{n-1}o_{n-1}))$$

$$p(t^1|h)$$

$$\big((1-\beta^1(ha_1o_1)) \cdots (1-\beta^1(ha_1o_1 \ldots a_{n-1}o_{n-1}))(\beta^1(ht^1))\big)\,.$$

Each of the $Pr(a_i|\cdot)$ terms is defined by the policy of the option ω^1, and the $\beta^1(\cdot)$ terms are defined by the termination condition of the option ω^1. The remaining term $p(t^1|h)$ is a primitive prediction; this equation establishes the link between primitive and option predictions.

- The term $Pr(O_{\tau+|t^1|} = o^1|T^1 = t^1)$ is next to be examined. Note that, once one conditions upon some sequence t^1 as the value of T^1, the value of $O_{\tau+|t^1|}$ is no longer random. Thus, the term $Pr(O_{\tau+|t^1|} = o^1|T^1 = t^1)$ is either 1 or 0, depending upon whether or not the last observation of t^1 is o^1.

  Note that one can define more general indicator functions than just checking if the last observation from ω^1 matches o^1. One can replace the $Pr(O_{\tau+|t^1|} = o^1|T^1 = t^1)$ term with an indicator function that takes t^1 and returns 1 if it constitutes a success of $\omega^1 o^1$ and 0 otherwise. However, one must ensure that the indicator functions are defined such that every primitive sequence t^1 with $Pr(T^1 = t^1) > 0.0$ will be considered a success of $\omega^1 o^1$ for exactly one value of o^1. One way to ensure this is to define a function from primitive sequences t^1 to the option-level observation space (which need not be the same as the primitive observation space). The indicator functions then correspond to whether or not that function maps t^1 to a given observation value or not.

- Finally, the term $p(\omega^2 o^2 \ldots \omega^k o^k|ht^1)$ is the prediction for a length-$(k-1)$ option test from a primitive history ht^1. One can apply Equation H.1 to write $p(\omega^2 o^2 \ldots \omega^k o^k|ht^1)$ in terms of predictions for length-$(k-2)$ option tests from primitive histories. This recursive definition of $p(t^\omega|h)$ given by Equation H.1 will reach a base case when t^ω just has one option (i.e., $k = 1$). In that case, the term $p(\omega^2 o^2 \ldots \omega^k o^k | h t^1)$ from Equation H.1 is just 1, because $\omega^2 o^2 \ldots \omega^k o^k$ would be an empty test when $k = 1$.

Exhaustively expanding Equation H.1 into its non-recursive form would result in an incomprehensible expression. However, the expanded form has only four different kinds of terms, and the terms of like kind can be grouped together to arrive at an expression that is more manageable. Three of the four kinds of terms correspond to (1) the action probabilities, (2) the observation probabilities (i.e., the primitive test predictions), and (3) the termination probabilities, each of which result from the expansion of the $Pr(T^i = t^i)$ part of Equation H.1. The fourth kind of term in the expanded form comes from the $Pr(O_{\tau+\cdot} = o^i|\cdot)$ part of Equation H.1; these terms are indicator variables that indicate whether or not a given set of primitive sequences is a success of the option test.

Grouping terms of the same type from the recursive expansion of Equation H.1, the resulting expression is

$$p(t^\omega|h) = \sum_{t^1,t^2,\ldots,t^k} \Pi(h; t^1,\ldots,t^k; \omega^1,\ldots,\omega^k)\psi_{t^\omega}(t^1,\ldots,t^k) \cdot \beta(h; t^1,\ldots,t^k; \omega^1,\ldots,\omega^k) p(t^1 t^2 \ldots t^k|h) \tag{H.2}$$

where

- $\Pi(h; t^1,\ldots,t^k; \omega^1,\ldots,\omega^k)$ is the product of all of the terms of the form $Pr(a_i|\cdot)$, accounting for the policies of the options.
- $\psi_{t^\omega}(t^1,\ldots,t^k)$ is the indicator function that tells whether or not the underlying primitive sequences $t^1,\ldots,t^k$ constitute a success of the option test t^ω. For

example, when the success of t^ω is determined by the last observation of each option's execution, the function just returns 1 if the last observations of $t^1, \ldots, t^k$ match the observations of t^ω (i.e., the product of the $Pr(O. = o^i | T^i = t^i)$ terms).

- $\beta(h; t^1, \ldots, t^k; \omega^1, \ldots, \omega^k)$ is the product of all of the terms containing $\beta^i(\cdot)$ for any $1 \leq i \leq k$, accounting for the termination condition of the options.
- and $p(t^1 t^2 \ldots t^k | h)$ is the primitive prediction which is equal to the product of all the $p(t^i | h t^1 \ldots t^{i-1})$ terms.

Now that Equation H.2 has defined the prediction of an option test from a primitive history, one can use this to write the prediction for a option test from an option history. This is done by conditioning upon the primitive history that underlies a given option history. Let $Pr(h^\omega = h | h^\omega)$ be the probability that h is the primitive history through some time τ given that h^ω is the option history through time τ. One can use Equation H.2 to define this probability in terms of primitive probabilities. First, note that

$$Pr(h^\omega = h | h^\omega) = \frac{Pr(h^\omega = h, h^\omega)}{Pr(h^\omega)}$$

where the numerator is the probability of h^ω succeeding with underlying history h from the null history, and the denominator is the probability that h^ω succeeded from the null history. The denominator is the prediction for an option test h^ω from a primitive history $\emptyset$ (i.e., the null history), which is given by Equation H.2. The numerator can be computed by a variation of Equation H.2 where the sum over the primitive sequences $t^1 \ldots t^k$ is not over all primitive sequences, but only those that form a specific sequence t when concatenated together:

$$\text{(H.3)} \quad Pr(t^\omega = t, t^\omega | h) = \sum_{t^1, t^2, \ldots, t^k : t^1 \ldots t^k = t} \Pi(h; t^1, \ldots, t^k; \omega^1, \ldots, \omega^k) \psi_{t^\omega}(t^1, \ldots, t^k) \cdot \beta(h; t^1, \ldots, t^k; \omega^1, \ldots, \omega^k) p(t^1 t^2 \ldots t^k | h).$$

Applying this equation with $h \leftarrow \emptyset$, $t^\omega \leftarrow h^\omega$, and $t \leftarrow h$ leads to an expression for $Pr(h^\omega = h, h^\omega)$ in terms of primitive predictions.

Finally, the prediction for an option test from an option history is found by conditioning upon the underlying primitive history h:

$$\text{(H.4)} \qquad p(t^\omega | h^\omega) = \sum_h \frac{p(h^\omega = h, h^\omega | \emptyset)}{p(h^\omega | \emptyset)} p(t^\omega | h).$$

When combined with the definitions of $p(t^\omega|h)$ and $p(h^\omega|\emptyset)$ from Equation H.2, and the definition of $p(h^\omega = h, h^\omega|\emptyset)$ from Equation H.3, this Equation H.4 defines an option prediction in terms of primitive predictions.

H.2 The Validity of the Option-level System-dynamics Matrix $\mathcal{D}^\Omega$

This section proves the following theorem, which is instrumental in establishing the ability of a PSR model to make accurate option predictions.

Theorem H.1. *The matrix $\mathcal{D}^\Omega$ that consists of the predictions of all option tests from all option histories is a valid system-dynamics matrix.*

Proof. In order to check the validity of a system-dynamics matrix, one can check the validity of the predictions from the null history row and then ensure that the other predictions are consistent with the predictions from the null history.

In order for the predictions from the null history to be valid, they must specify a valid multinomial probability distribution over possible observation sequences for any sequence of options. There are also marginal constraints that the predictions must satisfy in order to be valid. The following list establishes that the option predictions from the null history are indeed valid.

- All of the predictions must be greater than 0.0. This is true because all of the terms in Equation H.2 are non-negative, so any $p(t^\omega|\emptyset)$ will be non-negative.
- For any sequence of options $\omega^1 \ldots \omega^k$, let T be the set of option tests that have $\omega^1 \ldots \omega^k$ as their sequence of options. Then the predictions for all the tests in T must sum to 1.0. This ensures that the probabilities for the option tests with $\omega^1 \ldots \omega^k$ will form a valid multinomial distribution (given the non-negativity property, shown previously).

  The given definition of option predictions satisfies this because any execution of $\omega^1 \ldots \omega^k$ will cause exactly one test in T to succeed: the test with observations that match the last observations of the respective $\omega^1 \ldots \omega^k$.
- For any option test t^ω and any option ω, let T be the set of option tests of the form $t^\omega \omega o$ for all observations o. Then the predictions for all the tests in T must sum to the prediction for t^ω. One can verify this by expanding the option

predictions according to Equation H.2:

$$\begin{aligned}
&\sum_o p(t^\omega \omega o|\emptyset) \\
=&\sum_o \sum_{t^1,t^2,\ldots,t^k,t^{k+1}} \psi_{t^\omega \omega o}(t^1,\ldots,t^k,t^{k+1})\beta(\emptyset;t^1,\ldots,t^k,t^{k+1};\omega^1,\ldots,\omega^k,\omega) \\
&\Pi(\emptyset;t^1,\ldots,t^k,t^{k+1};\omega^1,\ldots,\omega^k,\omega)p(t^1t^2\ldots t^kt^{k+1}|\emptyset) \\
=&\sum_o \sum_{t^1,t^2,\ldots,t^k,t^{k+1}} \psi_{t^\omega}(t^1,\ldots,t^k)\psi_{\omega o}(t^{k+1}) \\
&\beta(\emptyset;t^1,\ldots,t^k;\omega^1,\ldots,\omega^k)\beta(t^1\ldots t^k;t^{k+1};\omega) \\
&\Pi(\emptyset;t^1,\ldots,t^k;\omega^1,\ldots,\omega^k)\Pi(t^1\ldots t^k;t^{k+1};\omega) \\
&p(t^1t^2\ldots t^k|\emptyset)p(t^{k+1}|t^1t^2\ldots t^k) \\
=&\sum_{t^1,t^2,\ldots,t^k} \psi_{t^\omega}(t^1,\ldots,t^k)\beta(\emptyset;t^1,\ldots,t^k;\omega^1,\ldots,\omega^k) \\
&\cdot\Pi(\emptyset;t^1,\ldots,t^k;\omega^1,\ldots,\omega^k)p(t^1t^2\ldots t^k|\emptyset) \\
&\sum_o \sum_{t^{k+1}} \psi_{\omega o}(t^{k+1})\beta(t^1\ldots t^k;t^{k+1};\omega)\Pi(t^1\ldots t^k;t^{k+1};\omega)p(t^{k+1}|t^1t^2\ldots t^k) \\
=&\sum_{t^1,t^2,\ldots,t^k} \psi_{t^\omega}(t^1,\ldots,t^k)\beta(\emptyset;t^1,\ldots,t^k;\omega^1,\ldots,\omega^k) \\
&\cdot\Pi(\emptyset;t^1,\ldots,t^k;\omega^1,\ldots,\omega^k)p(t^1t^2\ldots t^k|\emptyset) \\
&\sum_o p(\omega o|t^1\ldots t^k).
\end{aligned}$$

Since the sum of predictions $\sum_o p(\omega o|t^1\ldots t^k)$ equals 1.0, the above expression simplifies to $p(t^\omega|\emptyset)$, as desired.

Thus, *the null history predictions of $\mathcal{D}^\Omega$ are valid.*

It remains to show that the remaining predictions in $\mathcal{D}^\Omega$ are consistent with those null history predictions. This will be the case if and only if

$$p(h^\omega|\emptyset)p(t^\omega|h^\omega) = p(h^\omega t^\omega|\emptyset).$$

Let h^ω have length j with option sequence $\omega^1 \dots \omega^j$, and let t^ω have length k with option sequence $\omega^{j+1}\omega^{j+2} \dots \omega^{j+k}$. The derivation begins with the left-hand side of the equation above:

$$
\begin{aligned}
& p(h^\omega|\emptyset)p(t^\omega|h^\omega) \\
=& p(h^\omega|\emptyset)\sum_h \frac{p(h^\omega = h, h^\omega|\emptyset)}{p(h^\omega|\emptyset)} p(t^\omega|h) \\
=& \sum_h p(h^\omega = h, h^\omega|\emptyset)p(t^\omega|h) \\
=& \sum_h \Bigg(\sum_{t^1,\dots,t^j : t^1 \dots t^j = h} \psi_{h^\omega}(t^1,\dots,t^j)\beta(\emptyset; t^1,\dots,t^j; \omega^1,\dots,\omega^j) \\
& \cdot \Pi(\emptyset; t^1,\dots,t^j;\omega^1,\dots,\omega^j)p(t^1 \dots t^j|\emptyset) \Bigg) p(t^\omega|h) \\
=& \sum_{t^1,\dots,t^j} \psi_{h^\omega}(t^1,\dots,t^j)\beta(\emptyset; t^1,\dots,t^j; \omega^1,\dots,\omega^j) \\
& \cdot \Pi(\emptyset; t^1,\dots,t^j;\omega^1,\dots,\omega^j)p(t^1 \dots t^j|\emptyset)p(t^\omega|h) \\
=& \sum_{t^1,\dots,t^j} \psi_{h^\omega}(t^1,\dots,t^j)\beta(\emptyset; t^1,\dots,t^j; \omega^1,\dots,\omega^j) \\
& \cdot \Pi(\emptyset; t^1,\dots,t^j;\omega^1,\dots,\omega^j)p(t^1 \dots t^j|\emptyset) \\
& \sum_{t^{j+1},\dots,t^{j+k}} \psi_{t^\omega}(t^{j+1},\dots,t^{j+k})\beta(t^1,\dots,t^j; t^{j+1},\dots,t^{j+k}; \omega^{j+1},\dots,\omega^{j+k}) \\
& \cdot \Pi(t^1,\dots,t^j; t^{j+1},\dots,t^{j+k}; \omega^{j+1},\dots,\omega^{j+k})p(t^{j+1} \dots t^{j+k}|t^1 \dots t^j) \\
& = \sum_{t^1,\dots,t^j,\dots,t^{j+k}} \psi_{h^\omega t^\omega}(t^1,\dots,t^j,\dots,t^{j+k}) \\
& \cdot \beta(\emptyset; t^1,\dots,t^j,\dots,t^{j+k}; \omega^1,\dots,\omega^j,\dots,\omega^{j+k}) \\
& \cdot \Pi(\emptyset; t^1,\dots,t^j,\dots,t^{j+k}; \omega^1,\dots,\omega^j,\dots,\omega^{j+k}) \\
& \cdot p(t^1 \dots t^j \dots t^{j+k}|\emptyset) \\
& = p(h^\omega t^\omega|\emptyset).
\end{aligned}
$$

Therefore, the predictions in $\mathcal{D}^\Omega$ are consistent with the null-history predictions, which constitute a valid system. This implies that $\mathcal{D}^\Omega$ is a valid system-dynamics matrix. $\square$

H.3 Option-level System-dynamics Matrix Can Have Larger Rank

This section proves by example that the option-level system-dynamics matrix $\mathcal{D}^\Omega$ has greater rank than the original system $\mathcal{D}$ (Theorem 4.4).

One can construct a set of options Ω so that $\mathcal{D}^\Omega$ will contain $\mathcal{D}$ as a sub-matrix by allowing Ω to contain an option for each primitive action that just executes that primitive action and then terminates. Then one can construct an option test t^ω whose predictions are linearly independent of all primitive tests (i.e., all columns of $\mathcal{D}$). Since the matrix $\mathcal{D}^\Omega$ contains the columns of $\mathcal{D}$ and the column for t^ω, its rank will be larger than $\mathcal{D}$. The primary idea behind the following example is that one can construct such an t^ω by using an option policy that is a *non-linear* function of the prediction vector.

The primitive-level example system is a POMDP with 2 latent 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 latent state unchanged. Similarly, action β always produces observation o_β and leaves the latent state unchanged. The *swap* action changes the latent state, and *stay* leaves the latent state unchanged. The observation after *swap* or *stay* depends upon the ending latent 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 latent state distribution is $\{0.5, 0.5\}$.

For $p = 0.2$, the rank of $\mathcal{D}$ is 2. 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 in history h 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:

		prediction vector		
tests $\rightarrow$		$\alpha\ o_\alpha$	$stay\ o_1$	$\omega\alpha$
	$\beta\ o_\beta$	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 sub-matrix 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 $\mathcal{D}^\Omega$. $\square$

H.4 Bounding rank$(\mathcal{D}^\Omega)$

This section shows that $rank(\mathcal{D}^\Omega) \leq rank(\mathcal{D})$ when the options of Ω satisfy two criteria:

- the option policy depends only on the history since the option began executing (which still permits it to be a closed-loop policy)
- and the termination condition must only depend upon the history since the option began executing.

To begin, the following lemma establishes that the vector of predictions for any set of tests T from an option history h^ω is a linear combination of the vectors for the predictions of T from all primitive histories. When T is all primitive tests, the result shows that replacing the primitive histories of the system-dynamics matrix with option histories (and updating the contents of the matrix accordingly) will not increase the rank of the matrix.

Lemma H.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 $p^\top(T|h^\omega) = v_{h^\omega}^T p(T|H)$ for some vector v_{h^ω}.*

Proof. This can be seen by conditioning upon the the primitive history h which underlies the option history h^ω. Since $p(T|h^\omega) = \sum_{h \in H} Pr(h^\omega = h)p(T|h)$, the i^{th} entry of v_{h^ω} is the probability that h is the primitive actions/observations of h^ω. $\square$

The next lemma shows that adding a column to $\mathcal{D}$ for an option test t^ω will not increase the rank of $\mathcal{D}$, because the new column is linearly dependent upon the columns of $\mathcal{D}$.

Lemma H.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. First, consider a single entry of $p(t^\omega|H)$ for the history h. This entry is just $p(t^\omega|h)$, which one can expand using Equation H.2:

$$p(t^\omega|h) = \sum_{t^1,t^2,\ldots,t^k} \Pi(h; t^1, \ldots, t^k; \omega^1, \ldots, \omega^k)\psi_{t^\omega}(t^1, \ldots, t^k)$$
$$\cdot\, \beta(h; t^1, \ldots, t^k; \omega^1, \ldots, \omega^k)p(t^1 t^2 \ldots t^k|h),$$

where $t^\omega = \omega^1 o^1 \ldots \omega^k o^k$.

Because neither the termination condition nor the policy of any of the options in t^ω depends upon history prior to the start of the option, one can drop the h argument to the Π and β functions. Then *the only remaining reference to h is in the term $p(t^1 t^2 \ldots t^k|h)$*. This is critical, because the entries of v_{t^ω} cannot depend upon history.

To define v_{t^ω} that satisfies the theorem condition, note that there is one entry in v_{t^ω} that corresponds to any primitive test. This is because the multiplication $p(T|H)v_{t^\omega}$ can be rewritten as

$$\sum_{t_i \in T} [v_{t^\omega}]_i p(t_i|H)$$

where $[v_{t^\omega}]_i$ is the i^{th} entry of v_{t^ω}. Thus, if one defines the t entry of v_{t^ω} to be

$$\sum_{t^1, t^2, \ldots, t^k : t^1 \ldots t^k = t} \Pi(; t^1, \ldots, t^k; \omega^1, \ldots, \omega^k)\psi_{t^\omega}(t^1, \ldots, t^k)\beta(; t^1, \ldots, t^k; \omega^1, \ldots, \omega^k)$$

then $p(t^\omega|H) = p(T|H)v_{t^\omega}$ will hold.

$\square$

One can use these lemmas to complete the proof about the rank of $\mathcal{D}^\Omega$.

(**Theorem 4.5**). Let Ω be a set of options such that neither the termination conditions nor the policies of the options in Ω depend upon history prior to the execution of the option. For a primitive system-dynamics matrix $\mathcal{D}$ and an option-level system-dynamics matrix $\mathcal{D}^\Omega$ for Ω, $rank(\mathcal{D}^\Omega) \leq rank(\mathcal{D})$.

Proof. The proof follows from the fact that one can apply two linear transformations to the matrix $\mathcal{D}$ and the result will be the matrix $\mathcal{D}^\Omega$. Since these transformations are linear, the rank of the resulting matrix $\mathcal{D}^\Omega$ cannot be greater than the starting matrix $\mathcal{D}$. The first transformation, which Lemma H.3 shows to be linear, replaces the primitive tests of $\mathcal{D}$ with all option tests. The second transformation, which Lemma H.2 shows to be linear, replaces the primitive histories with all option histories. The resulting matrix consists of the predictions for all option tests from all option histories, which is the definition of $\mathcal{D}^\Omega$. $\square$

APPENDIX I

Proofs for Factored PSRs

This appendix provides theoretical results related to factored PSRs. The notation and background are given in Chapter V.

The following fact is used in the theorem about the accuracy of predictions made by a factored PSR:

Lemma I.1. *For random variables Y, B, C, the expectation $E_{B,C}[D(Y|B, C \parallel Y) - D(Y|B, C \parallel Y|B)] \geq 0$.*

Proof. First, note that the difference inside the expectation $D(Y|b, c \parallel Y) - D(Y|b, c \parallel Y|b)$ for $B = b, C = c$ is equal to $\sum_y Pr(y|b, c) \log_2 \frac{Pr(y|b)}{Pr(y)}$, using the definition of KL divergence. Taking the expectation of this quantity with respect to B, C, one gets

$$\begin{aligned}
& \sum_{b,c} Pr(b, c) \sum_{y} Pr(y|b, c) \log_2 \frac{Pr(y|b)}{Pr(y)} \\
= \; & \sum_{b,c,y} Pr(b, c, y) \log_2 \left(\frac{Pr(y|b)}{Pr(y)} \frac{Pr(b)Pr(c|y, b)}{Pr(b)Pr(c|y, b)} \right) \\
= \; & \sum_{b,c,y} Pr(b, c, y) \log_2 \frac{Pr(b, c, y)}{Pr(y)Pr(b)Pr(c|y, b)}.
\end{aligned}$$

This is just the KL divergence

$$D(Pr(B, C, Y) \parallel Pr(Y)Pr(B)Pr(C|Y, B))$$

which is always non-negative. □

(**Theorem 5.3**). For $f^i \subseteq f^{i'}$ and any subset $\mathbf{X}$ of future observations,

$$E_H[D(\mathbf{X}|H \parallel \mathbf{X}|f^i(H)) - D(\mathbf{X}|H \parallel \mathbf{X}|f^{i'}(H))] \geq 0$$

where H is the random variable for history of some length.

Proof. Let Z denote the random variable $D(\mathbf{X}|H \parallel \mathbf{X}|f^i(H)) - D(\mathbf{X}|H \parallel \mathbf{X}|f^{i'}(H))$; this makes Z a function of the random variable H. Define the following functions to denote the random variables corresponding to different parts of H: acts selects the actions, and F^i, $F^{i'}$, F^{i+}, and F^{i-} each select some observation dimensions, detailed in Figure 5.5.

Using these definitions, one can apply iterated expectations to get $E_H[Z] = E_{\text{acts},F^i}[E_{F^{i+},F^{i-}}[Z|\text{acts}, F^i]]$. The next step is to show that the inner expectation is always non-negative, so the whole expression is non-negative. To do this, let f^i and $f^{i'}$ be realizations of (acts, F^i), and $(\text{acts}, F^i, F^{i+})$, respectively, consistent with their definitions in Section 5.2.

Then for a given f^i,

$$E_{F^{i+},F^{i-}}[Z|f^i] = E_{F^{i+},F^{i-}}[D(\mathbf{X}|f^i, F^{i+}, F^{i-} \parallel \mathbf{X}|f^i) - D(\mathbf{X}|f^i, F^{i+}, F^{i-} \parallel \mathbf{X}|f^i, F^{i+})].$$

One can apply Lemma I.1 directly to this expression to show that it is non-negative, using the following mapping: $Y \leftarrow \mathbf{X}|f^i$, $B \leftarrow F^{i+}$, and $C \leftarrow F^{i-}$. $\square$

BIBLIOGRAPHY

BIBLIOGRAPHY

Astrom, K. J. (1965). Optimal control of Markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications*, *10*(1), 174–205.

Barto, A., & Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, *13*(4), 341–379.

Baum, L. E., Petrie, T., Soules, G., & Weiss, N. (1970). A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *The Annals of Mathematical Statistics*, *41*(1), 164–171. Available from `http://www.jstor.org/stable/2239727`

Bellman, R. (1957). A Markovian decision process. *Indiana University Mathematics Journal*, *6*, 679–684.

Bowling, M., McCracken, P., James, M., Neufeld, J., & Wilkinson, D. (2006). Learning predictive state representations using non-blind policies. In W. W. Cohen & A. Moore (Eds.), *Proceedings of the* 23rd *International Conference on Machine Learning* (pp. 129–136). ACM.

Boyen, X., & Koller, D. (1998). Tractable inference for complex stochastic processes. In G. F. Cooper & S. Moral (Eds.), *Proceedings of the* 14th *conference on Uncertainty in Artificial Intelligence* (pp. 33–42). Morgan Kaufmann.

Brockfeld, E., & Wagner, P. (2006). Validating microscopic traffic flow models. In *Proceedings of Intelligent Transportation Systems conference* (pp. 1604–1608). IEEE.

Cassandra, A. (1999). *Tony's POMDP File Repository Page.* Available from `http://www.cs.brown.edu/research/ai/pomdp/examples/index.html`

Cleveland, W. S., Devlin, S. J., & Grosse, E. (1988). Regression by local fitting : Methods, properties, and computational algorithms. *Journal of Econometrics*, *37*(1), 87 - 114. Available from `http://www.sciencedirect.com/science/article/B6VC0-4582FT0-1K/2/731cd0c23ef342f8d074fdd4e9c41325`

Dean, T., & Kanazawa, K. (1989). A model for reasoning about persistence and causation. *Computational Intelligence Journal*, *5*(3), 574–585.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, *39*(1), 1–38. Available from `http://www.jstor.org/stable/2984875`

Dietterich, T. G. (1998). The MAXQ method for hierarchical reinforcement learning. In J. W. Shavlik (Ed.), *Proceedings of the* 15th *International Conference on Machine Learning* (pp. 118–126). Morgan Kaufmann.

Drake, A. (1962). *Observation of a Markov Process Through a Noisy Channel.* Unpublished doctoral dissertation, Dept. of Electrical Engineering, Massachusetts Institute of Technology, Cambridge, Mass.

Engelhardt, B. E., Jordan, M. I., & Brenner, S. E. (2006). A graphical model for predicting protein molecular function. In W. W. Cohen & A. Moore (Eds.), *Proceedings of the* 23^{rd} *International Conference on Machine Learning* (pp. 297–304). ACM.

Fine, S., Singer, Y., & Tishby, N. (1998). The hierarchical hidden Markov model: Analysis and applications. *Machine Learning*, *32*(1), 41–62.

Golub, G. H., & Van Loan, C. F. (1996). *Matrix Computations* (Third ed.). The Johns Hopkins University Press.

Jaeger, H. (1998). *Discrete-time, discrete-valued observable operator models: A tutorial* (Tech. Rep. No. 42). German National Research Center for Information Technology.

James, M. R. (2005). *Using Predictions for Planning and Modeling in Stochastic Environments.* Unpublished doctoral dissertation, University of Michigan, Ann Arbor, Mich.

James, M. R., & Singh, S. (2004). Learning and discovery of predictive state representations in dynamical systems with reset. In C. E. Brodley (Ed.), *Proceedings of the* 21^{st} *International Conference on Machine Learning* (pp. 417–424). ACM.

James, M. R., Wolfe, B., & Singh, S. (2005). Combining memory and landmarks with predictive state representations. In L. P. Kaelbling & A. Saffiotti (Eds.), *Proceedings of the* 19^{th} *International Joint Conference on Artificial Intelligence* (pp. 734–739). Professional Book Center.

Littman, M. L., Sutton, R., & Singh, S. (2001). Predictive representations of state. In T. G. Dietterich, S. Becker, & Z. Ghahramani (Eds.), *Advances in Neural Information Processing Systems 14 (NIPS 2001)* (pp. 1555–1561). MIT Press.

Lu, X., & Coifman, B. (2007). *Highway traffic data sensitivity analysis* (Tech. Rep. No. UCB-ITS-PRR-2007-3). University of California, Berkeley.

McCallum, A., & Nigam, K. (1998). *A comparison of event models for naive bayes text classification.* Available from `citeseer.ist.psu.edu/mccallum98comparison.html`

Monahan, G. E. (1982). A survey of partially observable Markov decision processes: Theory, models, and algorithms. *Management Science*, *28*(1), 1–16. Available from `http://www.jstor.org/stable/2631070`

Perkins, T. J., & Precup, D. (1999). *Using options for knowledge transfer in reinforcement learning* (Tech. Rep. No. 99-34). University of Massachusetts.

Rafols, E. J., Ring, M. B., Sutton, R., & Tanner, B. (2005). Using predictive representations to improve generalization in reinforcement learning. In L. P. Kaelbling & A. Saffiotti (Eds.), *Proceedings of the* 19^{th} *International Joint Conference on Artificial Intelligence* (p. 835-840). Professional Book Center.

Rivest, R. L., & Schapire, R. E. (1994). Diversity-based inference of finite automata. *Journal of the ACM*, *41*(3), 555–589.

Rosencrantz, M., Gordon, G., & Thrun, S. (2004). Learning low dimensional predictive representations. In C. E. Brodley (Ed.), *Proceedings of the* 21^{st} *International Conference on Machine Learning* (pp. 88–95). ACM.

Rudary, M., & Singh, S. (2003). A nonlinear predictive state representation. In S. Thrun, L. K. Saul, & B. Schölkopf (Eds.), *Advances in Neural Information Processing Systems 16 (NIPS 2003)* (pp. 855–862). MIT Press.

Rudary, M., & Singh, S. (2006). Predictive linear-Gaussian models of controlled stochastic dynamical systems. In W. W. Cohen & A. Moore (Eds.), *Proceedings of the* 23^{rd} *International Conference on Machine Learning* (pp. 777–784). ACM.

Rudary, M., Singh, S., & Wingate, D. (2005). Predictive linear-Gaussian models of stochastic dynamical systems. In F. Bachus & T. Jaakkola (Eds.), *Proceedings of the* 21st *conference on Uncertainty in Artificial Intelligence* (pp. 501–508). AUAI Press.

Sandberg, R., Winberg, G., Branden, C.-I., Kaske, A., Ernberg, I., & Coster, J. (2001). Capturing whole-genome characteristics in short sequences using a naive bayesian classifier. *Genome Res.*, *11*(8), 1404–1409.

Şimşek, Ö., & Barto, A. G. (2008). Skill characterization based on betweenness. In D. Koller, D. Schuurmans, Y. Bengio, & L. Bottou (Eds.), *Advances in Neural Information Processing Systems 21 (NIPS 2008)* (pp. 1497–1504). MIT Press.

Singh, S., James, M. R., & Rudary, M. (2004). Predictive state representations: A new theory for modeling dynamical systems. In M. Chickering & J. Halpern (Eds.), *Proceedings of the* 20th *conference on Uncertainty in Artificial Intelligence* (pp. 512–519). AUAI Press.

Singh, S., Littman, M., Jong, N., Pardoe, D., & Stone, P. (2003). Learning predictive state representations. In T. Fawcett & N. Mishra (Eds.), *Proceedings of the* 20th *International Conference on Machine Learning* (pp. 712–719). AAAI Press.

Sutton, R., 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., Rafols, E. J., & Koop, A. (2005). Temporal abstraction in temporal-difference networks. In Y. Weiss, B. Schölkopf, & J. Platt (Eds.), *Advances in Neural Information Processing Systems 18 (NIPS 2005)* (pp. 1313–1320). MIT Press.

Sutton, R., & Tanner, B. (2004). Temporal-difference networks. In L. K. Saul, Y. Weiss, & L. Bottou (Eds.), *Advances in Neural Information Processing Systems 17 (NIPS 2004)* (pp. 1377–1384). MIT Press.

Tanner, B., & Sutton, R. (2005). TD(lambda) networks: Temporal-difference networks with eligibility traces. In L. D. Raedt & S. Wrobel (Eds.), *Proceedings of the* 22nd *International Conference on Machine Learning* (pp. 889–896). ACM.

Theocharous, G., Rohanimanesh, K., & Mahadevan, S. (2001). Learning hierarchical partially observable Markov decision process models for robot navigation. In *Proceedings of the 2001 IEEE International Conference on Robotics and Automation* (p. 511-516). IEEE.

U.S. Federal Highway Administration. (2006a). *Interstate 80 Freeway Dataset.* Available from `http://www.tfhrc.gov/about/06137.htm`

U.S. Federal Highway Administration. (2006b). *Next Generation Simulation Project.* Available from `http://ops.fhwa.dot.gov/trafficanalysistools/ngsim.htm`

Wackerly, D. D., Mendenhall, W., III, & Scheaffer, R. L. (2002). *Mathematical Statistics with Applications* (Sixth ed.). Duxbury.

Wiewiora, E. (2005). Learning predictive representations from a history. In L. D. Raedt & S. Wrobel (Eds.), *Proceedings of the* 22nd *International Conference on Machine Learning* (pp. 964–971). ACM.

Wingate, D., & Singh, S. (2006a). Kernel predictive linear Gaussian models for nonlinear stochastic dynamical systems. In W. W. Cohen & A. Moore (Eds.), *Proceedings of the* 23rd *International Conference on Machine Learning* (pp. 1017–1024). ACM.

Wingate, D., & Singh, S. (2006b). Mixtures of predictive linear Gaussian models for nonlinear stochastic dynamical systems. In *Proceedings of the* 21st *National Conference on Artificial Intelligence (AAAI 2006).* AAAI Press.

Wingate, D., & Singh, S. (2007). Exponential family predictive representations of state. In J. C. Platt, D. Koller, Y. Singer, & S. T. Roweis (Eds.), *Advances in Neural Information Processing Systems 20 (NIPS 2007)* (pp. 1617–1624). MIT Press.

Wingate, D., & Singh, S. (2008). Efficiently learning linear-linear exponential family predictive representations of state. In A. McCallum & S. Roweis (Eds.), *Proceedings of the* 25th *International Conference on Machine Learning* (pp. 1176–1183). Omnipress.

Wingate, D., Soni, V., Wolfe, B., & Singh, S. (2007). Relational knowledge with predictive state representations. In M. M. Veloso (Ed.), *Proceedings of the* 20th *International Joint Conference on Artificial Intelligence* (pp. 2035–2040).

Wolfe, B., James, M. R., & Singh, S. (2005). Learning predictive state representations in dynamical systems without reset. In L. D. Raedt & S. Wrobel (Eds.), *Proceedings of the* 22nd *International Conference on Machine Learning* (pp. 985–992). ACM.

Wolfe, B., James, M. R., & Singh, S. (2008). Approximate predictive state representations. In L. Padgham, D. C. Parkes, J. Müller, & S. Parsons (Eds.), *Proceedings of the* 7th *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)* (Vol. 1, pp. 363–370). IFAAMAS.

Wolfe, B., & Singh, S. (2006). Learning predictive state representations with options. In W. W. Cohen & A. Moore (Eds.), *Proceedings of the* 23rd *International Conference on Machine Learning* (pp. 1025–1032). ACM.