Simple Partial Models for Complex Dynamical Systems

by

Erik N. Talvitie

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
2010

Doctoral Committee:

Professor Satinder Singh Baveja, Chair
Professor Benjamin Kuipers
Professor John E. Laird
Assistant Professor Ryan M. Eustice

ACKNOWLEDGEMENTS

First I would like to thank my advisor, Satinder Singh. He has been an inspiring mentor and his impact both on my thesis and on me as a scientist has been profound. Under his guidance I have to learned to sharpen both my questions and my answers and I am truly grateful for all the time, effort, and support he's given me. Thanks also to the other members of my committee. Conversations and meetings with them have significantly improved not only the work itself, but my ability to communicate the work to others.

My fellow grad students have also been great friends and colleagues over the course of this work. I would particularly like to thank Nick, David, Britton, Jon, Vishal, and Matt for conversations, brainstorms, feedback, foosball, and for generally being brilliant and inspiring people.

Thank you as well to my family (in the expansive sense of the word) and particularly my parents. I am fortunate to have been surrounded for my entire life by people who love and support me in my every endeavor. They are responsible for opening the doors that led me here as well as the thirst for knowledge and the confidence that I can obtain it that allow me to do this work.

Finally, I want to thank my partner, Annie. Without her love, her support, her confidence in my strength, and her patience with my weakness, this work would simply not have been possible.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

LIST OF APPENDICES

Appendix

ABSTRACT

Simple Partial Models for Complex Dynamical Systems

by

Erik N. Talvitie

Chair: Satinder Singh Baveja

An agent behaving in an unknown environment may wish to learn a model that allows it to make predictions about future events and to anticipate the consequences of its actions. Such a model can greatly enhance the agent's ability to make good decisions. However, in environments like the physical world in which we live, which is stochastic, partially observable, and high dimensional, learning a model is a challenge.

One natural approach when faced with a difficult model learning problem is not to model the entire system. Instead, one might focus on capturing the most important aspects of the environment and give up on modeling complicated, irrelevant phenomena. This intuition can be formalized using *partial models*, which are models that make only a restricted set of (abstract) predictions in only a restricted set of circumstances. Because a partial model has limited prediction responsibilities, it may be significantly simpler than a complete model.

Partial models (and similar ideas) have been studied in many contexts, mostly under the Markov assumption, where the agent is assumed to have access to the full state of the world. In this setting, predictions can typically be learned directly as functions of state and the process of learning a partial model is often as simple as

estimating only the desired predictions and omitting the rest from the model. As such, much of the relevant work has focused on the interesting and challenging question of *which* partial models should be learned (rather than how to learn them). In the partially observable case, however, where the state of the world is (more naturally) assumed to be hidden from the agent, just the basic problem of how to learn a partial model poses significant challenges.

The goal of this thesis is to provide general results and methods for learning partial models in partially observable systems. Some of the main challenges posed by partial observability are formalized and learning methods are developed to address some of these issues. The learning methods presented are demonstrated empirically to be able to learn partial models in systems that are too complex for standard, complete model learning methods. Finally, many partial models are learned and composed to form complete models that are used for model-based planning in high dimensional arcade game examples.

CHAPTER 1

Introduction and Background

This thesis is concerned with the problem of an agent that wishes to learn about an unknown, complex, stochastic, partially observable world. As it interacts with its environment, this agent would like to use its experience to learn a model that allows it to make predictions about future events and anticipate the consequences of its actions. Such a model can help the agent plan its behavior and make better decisions. We humans, for instance, live in a rich, noisy environment where all kinds of information is hidden from us and yet somehow often manage to make reasonable predictions about a wide variety of phenomena. How can we cope so readily with such complexity? This work is inspired by two intuitive observations about the models humans build in the face of a high dimensional environment.

First, the world in which we live is vastly more complex than an individual's model could ever capture. Our models cannot possibly make *every* prediction there is to make. For instance, how many leaves will you see on your way from home to work tomorrow? A truly complete model of the world would attempt to answer this question by keeping track of how many leaves there were yesterday, perhaps taking into account the speed of the wind, the life cycle of each species of plant and how they relate to the current time of year, etc. In practice, building such a model is not only hopeless, but pointless. There are many phenomena that are simply too complicated

or too irrelevant to a person's life to bother trying to model. Rather, a person spends their modeling effort on answering more important questions about their commute like how long it will take to get to work, or whether there will likely be more traffic on one street or another street. This suggests that an agent in a complex world ought to be able to selectively learn to make some predictions, but not others, in order to focus on what is most important (and what is tractable).

Second, the world is compositional, and correspondingly, so is a human's knowledge. A person who is cooking dinner while watching a baseball game on TV does not suddenly find themselves in a strange, new "Cooking+Baseball" situation they've never encountered before. They have a model of how onions behave when sauteed and they have a model of how a ball responds when hit with a stick and they are able to compose the predictions these models make to form more detailed predictions about what will happen in the kitchen in the near future. This suggests that an agent in a world like ours, which is comprised of many interacting but relatively self-contained components, should be able to isolate those components and learn to model them separately and then re-combine those models to form a more complete, compositional model of the environment.

Inspired by these observations, this thesis focuses on the problem of learning *partial models.* A partial model is a model that makes only some particular (often abstract) predictions in only some situations. One may have a partial model that is in charge of predicting what happens to onions when they are sauteed, one in charge of predicting the trajectory of a baseball, one that is in charge of predicting whether there will be traffic on Main Street during rush hour, and so on. The central question in this thesis is: given a set of *predictions of interest*, how can one learn a model that makes just those predictions accurately? Ideally this could be done far more simply than learning a complete model (in order to learn about onions, one shouldn't have to learn about the number of leaves seen during a commute).

The general idea of learning simple models with restricted responsibilities is certainly not new. The idea of decomposing a model into many simpler sub-models is a common and powerful approach in AI that has been studied in many different contexts. A great deal of this work has been done under the Markov assumption, where it is assumed that the agent has access to the full state of the world. In that setting, the problem of actually *how* to learn a model that makes a limited set of predictions is fairly straightforward. Because the state is known, predictions can be learned directly as functions of state. One can choose to learn to make all predictions, or just some predictions and the learning procedure remains essentially the same. This is not to imply that learning Markov models is trivial. On the contrary, as with essentially any machine learning problem, if the environment is very high-dimensional it can be a serious challenge to find ways to generalize across world states, using data from one state to learn about many other states. However, the problems of learning complete models and partial models in the Markov setting are not conceptually different from each other. As a result, much of the work in this setting has focused on which partial models to learn, since there is already a fairly general conceptual understanding of *how* to learn them.

In the partially observable setting, on the other hand, the state of the world is presumed to be hidden from the agent. In this setting the generalization problem is still present, but there is the added challenge of automatically finding and maintaining some compact representation of state. The standard methods for learning models in the partially observable setting learn complete models that make all possible predictions, and are not easily adapted for the purpose of learning partial models. As such, even if one knows which partial models to learn, in the partially observable case, it is not necessarily clear how to learn them. So, this thesis will focus on the problem of learning simple partial models in partially observable systems. Some of the fundamental issues inherent in solving this problem will be formalized, methods

will be presented that at least begin to address those issues, and applications of those methods will demonstrate some of the ways in which partial models can be used in artificial agents. It is hoped that the solutions presented for learning partial models can serve as a foundation for progress on the problem of automatically determining which models to learn in partially observable systems, though this next step will not be addressed in this thesis.

Some of the points just discussed will be expanded on in the context of discussing some relevant existing representations of partial (and complete) models later in the chapter. First, however, the next section will establish some notation and terminology and formally describe the general setting addressed by the work in this thesis.

1.1 Preliminaries

This thesis focuses on discrete dynamical systems. This allows for a particularly clean exploration of the main issues posed by partial observability, without adding in the additional challenges found in continuous systems. The ideas presented here will be briefly related to continuous-valued dynamical systems in Chapter 6. The agent has a finite set $\mathcal{A}$ of actions that it can take. The environment has a finite set $\mathcal{O}$ of observations that it can emit. Though it will generally be assumed that the number of actions is small, the observations can themselves be complex, high-dimensional structures. In several of the examples in this thesis, the observations will be images represented as pixel arrays. Time proceeds in discrete steps and at every time step the agent chooses an action $a \in \mathcal{A}$ and the environment stochastically emits an observation $o \in \mathcal{O}$.

Definition 1.1. At time step i, the sequence of past actions and observations $h_i = a_1 o_1 a_2 o_2 ... a_i o_i$ is the *history* at time i.

The history at time zero, before the agent has taken any actions or seen any

observations h_0 is called the *null history* and is often written with the null symbol $\emptyset$.

1.1.1 Predictions

An agent uses its model to make conditional predictions about future events, given the history of actions and observations *and* given its own future behavior. Because the environment is assumed, in general, to be stochastic, a prediction is not a description of what *will* happen, but rather a probability of some future event. Events are described in terms of action-observation sequences. Let $\mathcal{T}$ be the set of *all possible* action observation sequences of *all lengths*. Rivest & Schapire (1994) and Littman et al. (2002) call such a sequence $t = a_1 o_1 a_2 o_2 ... a_k o_k$ a *test*. In this thesis, a sequence of actions and observations will instead be called a *primitive test* and the term "test" will be used broadly to include more abstract descriptions of future events. Primitive tests will be the basic building block for specifying predictions.

A primitive test t describes a sequence of actions the agent could conceivably take in the future and a sequence of observations the agent might observe as a result. Test t is said to have *succeeded* if the agent actually takes the action sequence in t and observes the observation sequence in t. A *prediction* $p(t|h)$ is the conditional probability that primitive test t succeeds, given that history h has already occurred and given that the agent takes the actions in t. Essentially, the prediction of a primitive test is the answer to the question "If I were to take this particular sequence of actions, with what probability would I see this particular sequence of observations, given the history so far?" Formally,

Definition 1.2. The *prediction* of a test $t = a_1 o_1 a_2 o_2 ... a_k o_k$ at history h is defined as

$$p(t|h) \stackrel{\text{def}}{=} \Pr(o_1|h, a_1)\Pr(o_2|ha_1o_1, a_2)...\Pr(o_k|ha_1o_1a_2o_2...a_{k-1}o_{k-1}, a_k). \tag{1.1}$$

Of course, there are many more sophisticated predictions one could make. For

instance a *set test* (Wingate et al. 2007) is a sequence of actions and *sets of observations*. A set test succeeds when the agent takes the specified action sequence and *any* sequence of observations occurs where each observation is contained within the corresponding sets in the test. Set tests allow the expression of abstract predictions. While primitive tests allow an agent, for instance, to express the question "If I go outside, what is the probability I will see this exact image?" a set test can express the far more useful abstract question "If I go outside, what is the probability that it will be sunny?" by grouping together all observations of a sunny day. One can also define tests with a more abstract description of the agent's behavior, for instance using *option tests* (Wolfe & Singh 2006, Soni & Singh 2007). Note that, because the predictions over the primitive tests fully capture the probability distribution over future events, the prediction for *any* kind of abstract test can be computed from the predictions for primitive tests. Any model that can make predictions for all primitive tests can make *any* conditional prediction about future observations, given history.

1.1.2 The System Dynamics Matrix and Linear Dimension

It is sometimes useful to describe a dynamical system using a conceptual object called the *system dynamics matrix* (Singh et al. 2004). Essentially, the system dynamics matrix contains the values of *all possible* predictions, and therefore fully encodes the dynamics of the system. Specifically,

Definition 1.3. The *system dynamics matrix* of a dynamical system is an infinity-by-infinity matrix. There is a column corresponding to every primitive test $t \in \mathcal{T}$. There is a row corresponding corresponding to every history $h \in \mathcal{H}$. The ijth entry of the system dynamics matrix is the prediction $p(t_j|h_i)$ of the test corresponding to column j at the history corresponding to row i and there is an entry for every history/test pair.

Though the system dynamics matrix is infinite in size, for many systems it has a

finite rank. The rank of the system dynamics matrix can be thought of as a measure of the complexity of the system (Singh et al. 2004).

Definition 1.4. The *linear dimension* of a dynamical system is the rank of the corresponding system dynamics matrix.

For some modeling representations, the linear dimension is a major factor in the complexity of representing and learning a complete model of the system. For instance, in POMDPs, the number of hidden states required to represent the system is lower-bounded by the linear dimension. In this work linear dimension will be used as the measure of the complexity of a dynamical system. If a system has smaller linear dimension than another, it is said to be "simpler."

1.1.3 Complete and Partial Models

Recall that $\mathcal{T}$ is the set of all primitive tests (sequences of actions and observations). Then the set of all possible histories $\mathcal{H}$ is the set of all action-observation sequences that could possibly occur starting at the beginning of time: $\mathcal{H} \stackrel{\text{def}}{=} \{t \in \mathcal{T} | p(t|\emptyset) > 0\}$. Note that a model that can make a prediction $p(t|h)$ for *all* $t \in \mathcal{T}$ and $h \in \mathcal{H}$ can make *any* conditional prediction about the future, given the history and the agent's proposed behavior (Littman et al. 2002). As such, a model that makes all such predictions is a *complete model*. In fact, note as an immediate consequence of Equation 1.1 that the prediction for any multi-step primitive test can be computed from the predictions of one-step tests:

$$p(a_1o_1a_2o_2...a_ko_k|h) = p(a_1o_1|h)p(a_2o_2|ha_1o_1)...p(a_ko_k|ha_1o_1a_2o_2...a_ko_k).$$

Thus, any model that can provide one-step predictions $p(ao|h)$ for all actions $a \in \mathcal{A}$, observations $o \in \mathcal{O}$ and histories $h \in \mathcal{H}$ is a complete model that is capable of making *any* prediction.

A *partial model*, then, is a model that does *not* make all predictions in all histories. There are two main ways in which a model can be partial. It might be limited in which predictions it makes and it might be limited to make predictions in only certain situations. The following definition of a partial model takes both possibilities into account:

Definition 1.5. A *partial model* has a set $\mathcal{T}^I$ of *tests of interest* and a set $\mathcal{H}^I \subseteq \mathcal{H}$ of *histories of interest*. A partial model can be used to make the *predictions of interest:* $p(t|h)$ for all $t \in \mathcal{T}^I$ and $h \in \mathcal{H}^I$.

For instance, one might have a model that just makes predictions about cooking onions, but only when cooking with a cast iron pan. Another model might make predictions about cooking onions in a non-stick pan. Still another might make predictions about a baseball game on television.

1.2 Related Notions of Complete and Partial Models

This section will discuss some of the existing literature on learning models, and most importantly partial models. First, the commonly studied special case of Markov systems will be discussed, followed by the more general partially-observable case, which is the focus of this thesis.

1.2.1 Markov Systems

A dynamical system is *Markov* if all one needs to know about history in order to make predictions about the future is the most recent observation.

Definition 1.6. A system is Markov if for any two histories h and h' (that could be the null history), any two actions a and a', any observation o, and any test t, $p(t|hao) = p(t|h'a'o)$.

A Markov system is often called a Markov decision process (MDP) and the most recent observation is often referred to as *state* because it contains all the information necessary to make any prediction about the future. In this way it essentially captures everything there is to know about the current situation, or the "state of the world." The fact that the state is so readily available significantly simplifies the model-learning problem in comparison to the partially observable setting, which will be discussed in the next section. In the Markov case, the notational shorthand $p(t|o)$ will be used to indicate the prediction of t at *any* history that ends in observation o.

1.2.1.1 Complete Models

Note that a model of an MDP is *complete* if it makes predictions $p(ao|o')$ for every action $a \in \mathcal{A}$ and every pair of observations $o, o' \in \mathcal{O}$. There are finitely many such predictions and, as such, it is often possible to represent a complete Markov model as a $|\mathcal{A}| \times |\mathcal{O}|^2$ look-up table. It is straightforward to estimate the entries of this table from data:

$$\hat{p}(ao|o') \stackrel{\text{def}}{=} \frac{\#\text{ times } ao \text{ succeeds from histories ending in } o'}{\#\text{ times action } a \text{ taken from histories ending in } o'}.$$

If the number of observations is very large, this look-up table will be too large to represent extensively and, in addition, a large number of samples will be required in order to obtain good estimates of the one-step predictions. As a result, learning methods that generalize across many states at once are necessary. In this case, learning a partial model (or several partial models) may afford more generalization, and result in a simpler learning problem.

1.2.1.2 Partial Models

Many approaches learn models that make only certain predictions. Note that, like a complete model, learning a Markov model that makes a limited set of predictions is straightforward. If the model has some set of tests of interest $\mathcal{T}^I$ to be predicted (and no others), then for every test $t \in \mathcal{T}^I$ and every observation o,

$$\hat{p}(t|o) \stackrel{\text{def}}{=} \frac{\#\text{ times } t \text{ succeeds from histories ending in } o}{\#\text{ times actions in } t \text{ taken from histories ending in } o}.$$

While simply estimating fewer predictions can have benefits (if $|\mathcal{T}^I| \ll |\mathcal{O}|$), the *main* reason to limit the predictions made by a model is that often the full detail of the observation is not necessary in order to accurately make the predictions of interest. The main learning challenge then often becomes learning what details *are* necessary. This is a process known as *state abstraction*. One can also gain compactness by limiting the situations in which the model makes predictions. In the Markov case, this is as simple as only computing estimated predictions for the states in which the model *should* make predictions. If there are many fewer such states than states in total, one can gain significant savings in computation and space.

STRIPS Models: In the well-studied STRIPS framework (Fikes & Nilsson 1971), "operators," or "update rules" associated with a particular action *only* predict the effects of taking that action. As a result, these update rules often require only a partial (or abstract) description of the state of the world (called "pre-conditions"). So, in the classic Blocks World example where the agent controls an arm and stacks blocks, if the action is to pick up block A, one need only know whether block A is covered by something in order to predict the effect of this action. One need not know the full configuration of all other blocks. A great deal of work has been done on *learning* update rules, that is learning what effects actions have and learning

which distinctions need to be made in order to predict those effects, for instance by Gil (1994) and Wang (1995) in the deterministic case and more recently by Pasula et al. (2007) in the stochastic case. Each update rule in a STRIPS model can be thought of as a partial model with histories of interest defined by the pre-conditions and tests of interest defined by its effects. Note that STRIPS models represent the state (and correspondingly the update rules) using first-order predicates. On the one hand, this expressiveness grants them compactness and a sophisticated framework for generalization and parameter-tying. On the other hand, it requires that the state be encoded in terms of objects and relations between those objects and therefore typically relies upon a human designer's knowledge to translate the low-level observations that an agent receives into high-level, first-order state descriptions. Furthermore, the first-order inference required for learning these representations presents a significant challenge itself. The methods presented in this thesis will be propositional in nature in an effort to focus on the challenges presented by partial observability without the added challenges of learning a relational representation. That said, extending this work to allow for more expressive, first-order partial models would be an interesting direction for future work, as discussed briefly in Chapter 6.

Factored MDPs: Another well-studied representation, the *factored MDP* (Boutilier et al. 1999), decomposes the state into independent variables. The problem of predicting the next state can then also be decomposed into multiple prediction problems: one for each variable. Essentially a factored MDP consists of multiple models, each one restricted to making predictions only about its assigned state variables (its tests of interest) and making no predictions about other variables. The compactness of a factored MDP comes from its *structure*, the specification of conditional independence relationships amongst the variables across time, which allows the partial model in charge of predicting a particular variable to ignore irrelevant variables in the previous

time-step. *Given* a structure, the parameters of a factored MDP are straightforward to learn. As such, much of the work on factored MDPs has focused on efficient planning in a factored MDP (e.g. Guestrin et al. 2003) and learning the correct structure (e.g. Degris et al. 2006, Strehl et al. 2007).

Option Models: *Options* (Sutton et al. 1999) are descriptions of temporally extended ways of behaving. They consist of a set of states in which the option can be initiated (the *initiation set*), a policy which determines how the agent will behave while executing the option, and a set of states in which the option terminates (the *termination set*). For instance, one might have an option for "Go to the kitchen" which encodes all the necessary muscles twitches to accomplish this high-level task. It has been well-demonstrated that planning using long-range options can be far more efficient than planning at the primitive time scale. This, of course, requires a model that can predict the outcome of executing an option. These models are called *option models*. An option model is only required to make predictions about the termination state of its associated option (its tests of interest), and it is only required to make these predictions in states where the option can be initiated (its histories of interest). Option models can be far more compact than a complete model, because they need not make any predictions involving the intermediate states encountered during the execution of an option. Again, in the Markov case, option models themselves are straightforward to learn so much of the associated work focuses on determining which option models will be most useful for planning (e.g. Stolle & Precup 2002, Menache et al. 2002, Şimşek & Barto 2008).

Qualitative State: Recent work by Stober & Kuipers (2008) and Mugan & Kuipers (2009) focuses attention on automatically determining which partial models to learn in continuous environments. Stober & Kuipers present a "bootstrapping" framework for learning models in high-dimensional pixel-based environments by which a model

learning algorithm can progressively develop more and more abstract models, based on concepts defined in terms of the earlier models. They first attempt to learn a static model of the world. Spatial clusters of high prediction error in this model are interpreted as objects moving through the world. Movement models are then learned for the objects, as algebraic expressions of various features of the objects (position, size, etc.). In this framework, the static model and the object movement models are all partial models, each in charge of predicting some reasonably independent facet of the world.

Mugan & Kuipers present a method for automatically finding useful qualitative, abstract, descriptions of state and actions (typically in the form of ranges of continuous variables). This involves both determining which qualitative predictions can be made reliably (and are useful for control) as well as which qualitative features are needed to make good predictions. Then partial models are created for making these predictions. The models themselves ultimately take the form of look-up tables of conditional predictions of qualitative features, given the values of other qualitative features.

As seen above, the question of *how* to learn a partial model in the Markov setting is not typically a research focus. Learning a partial Markov model is essentially the same as learning a complete Markov model, though learning a partial model can save both computational and representational complexity. Since there is already a general understanding of how to learn partial models in the Markov setting, most of the work discussed above has focused on how to exploit partial models and how to identify which partial models to learn. As will be discussed in the next section, removing the Markov assumption introduces substantial additional challenges both to learning a model in general, and to learning a partial model in particular.

1.2.2 Partially Observable Systems

When a dynamical system is not Markov, it is called *partially observable*. There are some cases where partial observability does not necessarily present a major challenge. For instance, if knowing the most recent observation is not enough information to make accurate predictions about the future, but knowing the most recent k observations is, the system is called *kth order Markov*. If k is very small, then methods for Markov systems can often be straightforwardly applied (simply by treating the last k observations as state, rather than just the most recent observation). Even when the system is not short-order Markov, sometimes prior knowledge of the system can allow a human to augment the observation with extra state variables that make the system Markov (or approximately so).

In general, however, predictions about the future in a partially observable system may depend arbitrarily on the history of interaction since the beginning of time, instead of just the most recent observation. History grows unboundedly with time and so cannot be conditioned on explicitly. So, while the generalization problem found in the Markov case is still certainly an issue in the partially observable case, there is an additional challenge that it is necessary to learn to maintain some compact summary of the relevant information contained in history. This summary is called *state*, in analogy to the state in a Markov system.

1.2.2.1 Complete Models

In a Markov system the state is readily available as the last observation of a given history, and as a result, predictions can be directly estimated as functions of observations. In a partially observable system, a large part of the model-learning problem is *learning* what state should be, that is learning a compact sufficient statistic of history, with respect to predictions about the future. The following briefly introduces two approaches to representing and learning complete models of partially observable

systems.

POMDPs A popular representation for complete models of partially observable systems is the partially observable Markov decision process (POMDP) (Monahan 1982). A POMDP posits an underlying MDP with a set $\mathcal{S}$ of *hidden states* that the agent never observes. At any given time-step i, the system is in some particular hidden state $s_{i-1} \in \mathcal{S}$ (unknown to the agent). The agent takes some action $a_i \in \mathcal{A}$ and the system transitions to the next state s_i according to the transition probability $\Pr(s_i|s_{i-1}, a_i)$. An observation $o_i \in \mathcal{O}$ is then emitted according to a probability distribution that in general may depend upon s_{i-1}, a_i, and s_i: $\Pr(o_i|s_{i-1}, a_i, s_i)$. Of course because the agent does not observe the hidden states, it cannot know which hidden state the system is in. The agent *can* however maintain a probability distribution that represents the agent's current *beliefs* about the hidden state. This probability distribution is called the *belief state*. If the belief state associated with history h is known, then it is straightforward to compute the prediction of any test t: $p(t|h) = \sum_{s \in \mathcal{S}} \Pr(s|h)\Pr(t|s)$, where $\Pr(t|s)$ can be computed using the transition and observation emission probabilities. The belief state is a finite summary of history from which any prediction about the future can be computed. So, the belief state is the state vector for a POMDP. Given the transition probabilities and the observation emission probabilities, it is possible to maintain the belief state using Bayes' rule. If at the current history h one knows $\Pr(s|h)$ for all hidden states s and the agent takes action a and observes observation o, then one can compute the probability of any hidden state s at the new history:

$$\Pr(s|hao) = \frac{\sum_{s' \in \mathcal{S}} \Pr(s'|h)\Pr(s|s', a_i)\Pr(o_i|s', a_i, s)}{\sum_{s'' \in \mathcal{S}} \sum_{s' \in \mathcal{S}} \Pr(s'|h)\Pr(s''|s', a_i)\Pr(o_i|s', a_i, s'')}.$$

The parameters of a POMDP that must be learned in order to be able to maintain state are the transition probabilities and the observation emission probabilities. Given

these parameters, the belief state corresponding to any given history can be recursively computed and the model can thereby make any prediction at any history. POMDP parameters are typically learned using the Expectation Maximization (EM) algorithm (Baum et al. 1970). Given some training data and the number of actions, observations, and hidden states as input, EM essentially performs gradient ascent to find transition and emission distributions that (locally) maximize the likelihood of the provided data.

PSRs Another, more recently introduced modeling representation is predictive state representations (PSRs) (Littman et al. 2002). Instead of hidden states, PSRs are defined more directly in terms of the system dynamics matrix (described in Section 1.1.2). Specifically, PSRs find a set of *core tests* Q whose corresponding columns in the system dynamics matrix form a basis. Recall that the system dynamics matrix often has finite rank (for instance, the matrix associated with any POMDP with finite hidden states has finite rank) and thus Q is finite for many systems of interest. Since the predictions of Q are a basis, the prediction for *any* other test at some history can be computed as a linear combination of the predictions of Q at that history. The vector of predictions for Q is called the *predictive state*. While the belief state was the state vector for POMDPs, the predictive state is the state vector for PSRs. It can also be maintained by application of Bayes' rule. Specifically, if at some history h, $p(q|h)$ is known for all core tests q and the agent takes some action $a \in \mathcal{A}$ and observes some observation $o \in \mathcal{O}$, then one can compute the prediction of any core test q at the new history:

$$p(q|hao) = \frac{p(aoq|h)}{p(ao|h)} = \frac{\sum_{q' \in Q} p(q'|h) m_{aoq}(q')}{\sum_{q' \in Q} p(q'|h) m_{ao}(q')},$$

where $m_{aoq}(q')$ is the coefficient of $p(q'|h)$ in the linear combination that computes the prediction $p(aoq|h)$.

So, given a set of core tests, the parameters of a PSR that must be learned in

order to maintain state are the coefficients m_{ao} for every action a and observation o and the coefficients m_{aoq} for every action a, observation o, and core tests q. Given these parameters the predictive state at any given history can be recursively computed and used to make any prediction about the future. PSRs are learned (James & Singh 2004, Wolfe et al. 2005) by directly estimating the system dynamics matrix using sample averages in the training data. The estimated matrix is used to find a set of core tests and the parameters are then estimated using linear regression.

Both of these representations, and other related representations such as, e.g., observable operator models (Jaeger 2000), are highly sensitive to the linear dimension of the system. For POMDPs, the number of hidden states needed to represent the system is lower-bounded by the linear dimension. Both computational and space complexity of a POMDP are quadratic in the number of hidden states. In addition, being a hill-climbing algorithm, EM is subject to converging to local maxima. More hidden states means searching for parameters in a higher-dimensional space, which often leads to more local extrema, and correspondingly poorer results. For PSRs, the number of core tests is precisely the linear dimension. As with POMDPs, the number of parameters to be learned in a PSR is quadratic in the linear dimension. Furthermore, the search for core tests can be extremely sensitive to linear dimension. A high linear dimension can drastically increase the amount of data needed to obtain a correct core set. In practice it is rare for POMDPs or PSRs to be learned from data in systems with linear dimension of more than a few hundred, which is tiny in comparison to most systems of interest.

1.2.2.2 Partial Models

Compared to the Markov case, the problem of learning partial models in the partially observable case is far less straightforward. If, as in the Markov case, the

compact representation of state associated with every history were available *a priori*, learning a model that makes only some predictions in only some situations would simply be a matter of estimating only the predictions of interest, as a function of state. Unfortunately, as seen above, learning to maintain state is deeply intertwined with the model-learning process itself. In order to maintain state, both POMDPS and PSRs require access, in principle, to *all* one-step predictions. Recall that any model that can make all one-step predictions can make all possible predictions. So, these models cannot maintain state without the ability to make all possible predictions, and without state, they cannot make predictions at all. As such, it is not as clear how to learn a partial model *without first learning a complete model*. Nevertheless, there are some examples of partial models in partially observable systems. The following are just a few particularly relevant examples.

Partially Observable STRIPS Models: There has been some work on extending the STRIPS framework to partially observable domains. For instance, Amir (2005) presents an algorithm for learning propositional STRIPS rules in partially observable problems. The main idea of the method is to maintain a set of all *possible* models consistent with the data seen so far in a process analogous to Bayesian model learning. However, this method applies only to deterministic systems with a very particular type of partial observability. Essentially the assumption is that the true world state is the conjunction of some propositions and that observations consist of the true values of some (possibly changing) subset of these propositions. Because the observations are so closely tied to the true state of the system, one can still enjoy some of the benefits of the Markov case. Furthermore, because of the determinism, once one obtains an update rule, the partial observability is no longer an issue: the effects of a deterministic rule are known, regardless of whether they are observed or not. As such, this work does not seem to grapple with some of the central problems associated

with learning in partially observable environments.

DBNs: A popular representation that handles stochasticity and partial observability, *dynamic Bayes nets* (DBNs) (Ghahramani & Jordan 1995), are a generalization of both factored MDPs and POMDPs. Like a factored MDP, a DBN decomposes the world state into a number of variables but, like a POMDP, some of these variables are *observed* and some are *hidden*. The structure of a DBN, much like the structure of a factored MDP, is a set of conditional independence relationships between these variables across (and within) time-steps. A DBN can be *far* more compact than its corresponding unstructured POMDP if each variable depends on very few other variables. The parameters of a DBN, like those of a POMDP, are learned using EM.

While a DBN decomposes its representation of the world into many pieces, it is not easily interpreted as a composition of partial models. The variables in a DBN are all linked together and there is typically no easily separable piece that can be identified as a model that makes some particular subset of predictions. In this sense, a DBN is still a complete model, albeit a structured one. A more thorough comparison between DBNs and partial models can be found in Section 5.2.1.

Factored PSRs: Another representation similar to factored MDPs is called the *factored PSR* (Wolfe et al. 2008). In a factored PSR, the *observation* is broken up into several component variables and, as in a factored MDP, a model is learned to make predictions about each variable. The component models of a factored PSR are made simpler than a complete model by allowing them to ignore the history of many of the other observation variables. Wolfe et al. demonstrated that if the component models ignored more observation variables they became easier to learn and if they attended to more observation variables they became more accurate. However, they offered no principled method for determining which variables each model could safely ignore while still making accurate predictions. This problem of learning what can

safely be ignored will be discussed in detail in the more general context of *observation abstraction* in Chapter 2.

Hierarchical PSRs: The idea of options has also been studied in the partially observable case. One type of model that makes use of them is the *hierarchical PSR* (Wolfe & Singh 2006). A hierarchical PSR applies in settings where the agent is given a finite set of options *a priori* and behaves by selecting an option, executing it to completion, selecting another option, executing it to completion, and so on. In this case, Wolfe & Singh learn a model that makes predictions only in histories that can be reached by executing a sequence of options. Instead of updating at every time-step with an action and an observation, the model only updates at the termination of an option, where it is told which option was executed and only the last observation that was seen (rather than the whole observation sequence experienced during execution). Correspondingly, this option-level model only makes predictions about the observation that *will* be seen at the end of execution of each option. This idea of a model that makes predictions at only some histories and is updated with an abstract description of what happens in between those histories will be explored in more generality in Chapter 4.

While exploiting particular examples of partial models, none of this work offers a general, principled foundation for learning partial models in partially observable systems. This thesis attempts to provide such a foundation by addressing the general question, "If one is given a set of tests of interest $\mathcal{T}^I$ and a set of histories of interest $\mathcal{H}^I$, how can one learn a model that makes the predictions of interest $p(t|h), \forall t \in \mathcal{T}^I, h \in \mathcal{H}^I$ more simply than a complete model?" An ideal answer to this question would be a method that learns to maintain *just enough* state information to make the predictions of interest, and no more.

This thesis will not introduce new model-learning methods, *per se*. Instead, the focus will be on developing ways to *transform* the training data so that existing methods like POMDPs can be applied to a simplified problem, and yet produce a model that can be used to make the predictions of interest. Chapters 2 and 3 will discuss two complementary strategies for addressing one aspect of the central question: limiting predictions to a set of tests of interest. Chapter 4 will incorporate the concept of histories of interest. Chapter 5 will demonstrate one of the main motivating applications of partial models: combining them to form a more complete model. Finally, Chapter 6 will summarize the contributions of this thesis and discuss some of the future directions suggested by the findings herein.

1.3 Summary

Key points from this chapter:

- Partial models are useful for complex environments.
  - They allow one to focus on important/easy predictions, avoid useless/difficult predictions.
  - One can learn separate models of independent phenomena, and combine them to form a more complete, compositional model.
- The central problem of this thesis is learning partial models in partially observable systems:
  - Given a set of tests of interest $\mathcal{T}^I$.
  - Given a set of histories of interest $\mathcal{H}^I$.
  - Learn a model that makes the predictions of interest:
    - $p(t|h), \forall t \in \mathcal{T}^I, h \in \mathcal{H}^I$.

- How to learn a partial model in the Markov setting is conceptually straightforward.
  - State is provided (most recent observation).
  - As a result, one can directly estimate the predictions as function of state.
- *How* to learn a partial model in the partially observable setting is a challenge in itself.
  - Discovery of state and model-learning are intertwined.
  - Ideally one would learn to maintain *just enough* state to make the predictions of interest, and no more.
  - This thesis attempts to provide a general foundation for learning partial models in partially observable problems.

CHAPTER 2

Observation Abstraction

The central goal of this thesis is to learn partial models that make only some particular predictions. Specifically, given a set of tests of interest $\mathcal{T}^I$, one would like to learn a model that can accurately provide the predictions for $p(t|h)$ for all $t \in \mathcal{T}^I$ and all histories $h \in \mathcal{H}$ *and* ideally one would like that model to be easier to learn than a complete model.

This chapter will focus on one strategy for learning a simple partial model: ignoring some details of the environment that are irrelevant for making the predictions of interest. A partial model for predicting the weather, for instance, can probably safely ignore the outcomes of sports games, the workings of the stock market, and many other irrelevant details. By modeling less, the model can often be far simpler than one that tries to capture all these phenomena and yet still make the predictions of interest accurately (predictions about the weather).

As a more concrete example, consider the 4x4 grid world pictured in Figure 2.1a. The agent has 4 actions that move it one square in each cardinal direction: n, e, s, and w and it observes the label (numbers 1-16) of the square it moves into. Now imagine that the agent's goal is to reach the right-hand column. As such, perhaps it is only really interested in predicting whether it will reach the right-most column in the next time step. As such, the agent has a test of *tests of interest* $\mathcal{T}^I = \{nR, sR, wR, eR\}$,

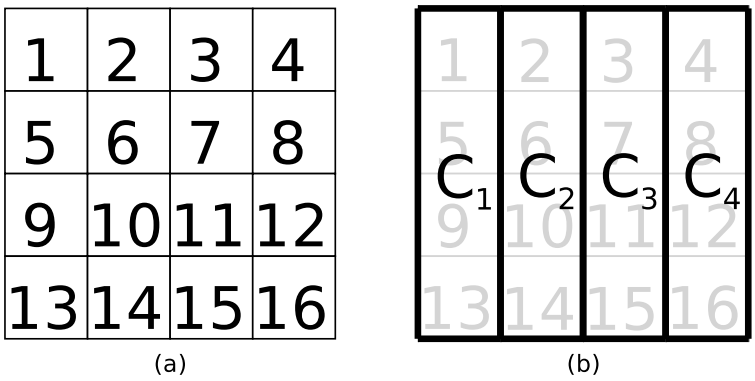


Figure 2.1: The 4x4 grid world.

where $R = \{4, 8, 12, 16\}$ is a set observation including all squares in the right column. Note that the agent's *vertical* position is entirely irrelevant to making correct predictions for the tests of interest; all that matters is what column the agent currently inhabits. One can imagine constructing a model that *only* keeps track of which column the agent is in and ignores the agent's row (pictured in Figure 2.1b). This model could still be used to make predictions for the tests of interest (will the agent enter the right-most column in the next time-step?) but is much simpler than a fully detailed model as there are only 4 columns, but there are 16 possible positions.

The "column" model of the gridworld is an example of an *abstract* model. It intentionally ignores the distinctions between many *primitive observations* (position) and instead deals only with aggregate, *abstract observations* (column). The word "abstract" here simply refers to the fact that the same label is being applied to multiple distinct concrete positions. This chapter will describe conditions on an abstraction that guarantee that the corresponding abstract model can make accurate predictions for the tests of interest (in Section 2.2). Because the abstract model ignores irrelevant details, it may be far simpler than a complete model. Section 2.3 will present theoretical results regarding the learnability of abstractions that satisfy these conditions as well as conceptual algorithms for finding such abstractions (these algorithms will be tested empirically in Section 2.3.4). Finally, in Section 2.5 some

limitations of abstraction as a means to learn partial models will be discussed, which will motivate the next chapter. First, however, the next section will carefully define observation abstractions and some related concepts.

2.1 Observation Abstraction

An *observation abstraction* is a clustering of primitive observations. An abstraction can be thought of as a partitioning of the set of observations $\mathcal{O}$ (that is, a set of abstract observations such that every primitive observation is contained within exactly one abstract observation). An *abstract observation*, then, is best thought of as a *set* of observations. For instance, in the gridworld example above, the abstract observation $C_1 = \{1, 5, 9, 13\}$.

Equivalently, an abstraction can be defined as a surjection (many-to-one mapping) η that maps primitive observations to some set of abstract observations. Viewed this way, an observation abstraction is essentially the same as a *feature* of the observation. In the gridworld example, the abstraction takes a primitive observation and provides what could be interpreted as the "column" feature. Throughout the thesis abstractions will be treated as partitions or surjections interchangeably, as best suits the discussion. In fact, it will often be notationally expedient to mix the two notions and treat $\eta(o)$ directly as the set of observations that all map to the same abstract observation as o. So, again in the gridworld example, $\eta(1) = \{1, 5, 9, 13\}$.

A key concept relating to abstractions is that of *refinement*. A refinement η' of an abstraction η is an abstraction that makes all the same distinctions η does (and potentially more). Correspondingly, η is called a *coarsening* of η'. Formally:

Definition 2.1. An abstraction η' is a *refinement* of another abstraction η if and only if for any pair of primitive observations $o_1, o_2 \in \mathcal{O}$, if $\eta'(o_1) = \eta'(o_2)$, then $\eta(o_1) = \eta(o_2)$. If η' is a refinement of η then η is called a *coarsening* of η'.

The finest possible abstraction maps primitive observations to themselves. The coarsest possible abstraction maps all primitive observations to the same abstract observation.

2.1.1 Abstract Systems

A given abstraction η can be used to define a dynamical system, called the *abstract system*, as a transformation of the original, primitive system. Specifically, the abstract system has the same set $\mathcal{A}$ of actions but its observations are the abstract observations which will be denoted $\mathcal{O}^\eta$. The dynamics of the abstract system are determined by the dynamics of the original system. Whenever the original system would emit the observation $o \in \mathcal{O}$, the abstract system instead emits the abstract observation $\eta(o) \in \mathcal{O}^\eta$.

The abstract tests $\mathcal{T}^\eta$ and abstract histories $\mathcal{H}^\eta$ are sequences of actions and *abstract* observations. Just as abstract observations can be thought of as sets of primitive observations, abstract tests and histories can be thought of as sets of primitive tests and histories, respectively. An abstract test $T \in \mathcal{T}^\eta$ contains all the primitive tests whose observations map to the abstract observations in T. It is straightforward to compute predictions in the abstract system. For any abstract test $T \in \mathcal{T}^\eta$ and primitive history $h \in \mathcal{H}^\eta$,

$$p(T|h) = \sum_{t \in T} p(t|h), \tag{2.1}$$

that is, the conditional probability of *any* of the tests in T occurring, given h. Then for any abstract test $T \in \mathcal{T}^\eta$ and abstract history $H \in \mathcal{H}^\eta$,

$$p(T|H) = \frac{p(HT|\emptyset)}{p(H|\emptyset)} = \frac{\sum_{h \in H} p(h|\emptyset) \sum_{t \in T} p(t|h)}{\sum_{h \in H} p(h|\emptyset)}, \tag{2.2}$$

which is essentially the *expected* prediction for T where the expectation is taken over

the primitive histories $h \in H$.

In the gridworld example the abstract system was simpler than the primitive system, as measured by linear dimension. Extending the example to a general $k \times k$ grid, the primitive system intuitively has a linear dimension of k^2, while the abstract system has a linear dimension of k. It is straightforward to show that the abstract system *never* has linear dimension greater than that of the primitive system. In fact, it can be shown more generally that if η' is a refinement of η, then the abstract system induced by η is never more complex than that induced by η'.

Proposition 2.2. *Consider a dynamical system, an abstraction η, and a refinement η' of η. The linear dimension of the abstract system corresponding to η, n^η, is no greater than $n^{\eta'}$, the linear dimension of the abstract system corresponding to η'.*

Proof. See Appendix A □

Because the abstract system can be far simpler than the primitive system (and, at least as measured by linear dimension, can never be more complex), it is often easier to learn, represent, and make use of abstract models than primitive models.

2.1.2 Abstract Models

Given an abstraction and training data from the original system, it is straightforward to learn a model of the abstract system. Simply take the training data and apply the abstraction. This translates trajectories of experience with the primitive system into trajectories of experience with the *abstract* system. The resulting abstract data set can then be supplied to any appropriate model-learning method (e.g. EM for POMDPs). As mentioned in Chapter 1, this idea of transforming training data from the primitive system and then applying existing model-learning methods to obtain a model of some other, ideally simpler, dynamical system is a recurring approach in this thesis.

A model of the abstract system will typically be easier to learn than a model of the primitive system. One reason is that the abstract system tends to have a lower linear dimension (and as seen above cannot have a higher linear dimension) than the primitive system. Another reason is that, because abstract events are, by nature, less specific than primitive events, they tend to occur more often. For example, one might encounter one day in a year with a specific temperature and amount of precipitation, but many days that can be described as "cold and rainy." As such, it may be easier to learn about cold and rainy days in general than days with a high of 45° and 0.7 inches of rainfall, because they occur with much higher frequency.

There are also ways in which an abstraction can make a model *harder* to learn. For instance, if the primitive system is Markov, then an observation abstraction of that system will, in general, be non-Markov. Markov systems are often much easier to model (as discussed in Chapter 1). The focus of this thesis, however, is partially observable systems, so this effect will not be of major concern.

A model of the abstract system is, in some sense, a *partial model* in that it cannot make every possible prediction. The column model in the gridworld example cannot be used to make predictions about the vertical position of the agent. It is impossible to even *express* that prediction in terms of the abstract observations! An abstract model *can* provide the conditional probability of any future abstract event, given an abstract history. That is, it provides $p(T|H)$ for any abstract test $T \in \mathcal{T}^\eta$ and any abstract history $H \in \mathcal{T}^\eta$. As such, if one has a set $\mathcal{T}^I$ of tests of interest to make, one would like an abstraction η such that the tests of interest are contained within $\mathcal{T}^\eta$, or some further abstraction thereof. Such an abstraction is *expressive* with respect to the tests of interest.

Note that while an abstract model predicts less than a primitive model (because it only makes predictions on an abstract level), it also *conditions* on less than a primitive model, because it only conditions on *abstract* histories, which are less detailed than

primitive histories. In general, there is no reason to expect that the predictions an abstract model makes for abstract tests given abstract histories to be the same as the predictions made by a primitive model for the same abstract tests given fully detailed primitive histories. An abstract model of the weather that only pays attention to whether each day was sunny or rainy will have some prediction about whether the next day will be sunny, conditioned on the history of sunny and rainy days. A more detailed model that also pays attention to temperature could *also* make a prediction about whether the next day will be sunny, conditioned now on the history of sunny and rainy days *and* the history of temperature readings. The second model will make more informed predictions because it can take into account, say, that a large change in temperature is likely to correlate with rain. Even if both models make *perfect* predictions in the sense of providing the correct conditional probabilities, one will make more *accurate* predictions because it conditions its predictions on more detail.

The next section will discuss some properties, such as *expressiveness* and *accuracy* that may be desirable in an abstraction and discuss specifically how they relate to learning partial models. Later in the chapter some theoretical results about the feasibility of automatically constructing abstractions with these properties will be presented.

2.2 Properties of Abstractions

At its heart, an abstract model ignores detail about the world. From this it gains its simplicity but also its limitations. How much detail can be safely ignored depends on what the abstract model will be used for. Many different criteria for selecting abstractions exist. Li et al. (2006) provide a description (and unifying framework) for several abstraction criteria that have been used to select abstractions in MDPs.

The central problem in *this* work is to learn a model that makes the predictions of interest. For the purposes of this chapter, this means making predictions $p(t|h)$

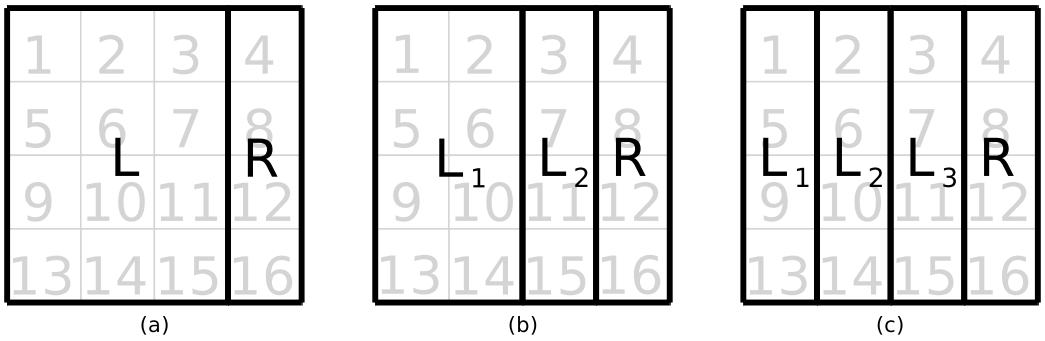


Figure 2.2: Abstractions of the 4x4 grid world: (a) expressive, (b) expressive and accurate, and (c) expressive, accurate, and homomorphic

for the *tests of interest* $t \in \mathcal{T}^I$ at every history $h \in \mathcal{H}$. This section will introduce properties required of an abstraction so that the abstract model can be used as such a partial model: *expressiveness* and *accuracy*. A third property, *homomorphism*, will be discussed as well, as it is a highly related and well studied criterion for abstractions.

2.2.1 Expressiveness

If the abstract model is going to be used to make predictions for the tests of interest, it is necessary that those tests be expressible in terms of abstract observations.

Definition 2.3. An observation abstraction η is *expressive* with respect to a set of tests of interest $\mathcal{T}^I$ if and only if for every test of interest $t \in \mathcal{T}^I$ there is a set of abstract tests $S^t \subset \mathcal{T}^\eta$ such that $p(t|h) = \sum_{T \in S^t} p(T|h)$ for all *primitive* histories $h \in \mathcal{H}$.

This property ensures that for every test of interest, there is an equivalent (abstract) test in the abstract system. Put another way, it requires that every set observation that occurs in some test of interest $t \in \mathcal{T}^I$ is either an abstract observation in $\mathcal{O}^\eta$ itself, or the union of several abstract observations. Seen in this light, it is clear that if η is expressive with respect to $\mathcal{T}^I$, then any refinement η' of η is *also* expressive with respect to $\mathcal{T}^I$.

As a simple example, recall the 4x4 grid world example in which the tests of interest $\mathcal{T}^I = \{nR, sR, wR, eR\}$, where $R = \{4, 8, 12, 16\}$ is a set observation including all squares in the right column. An expressive abstraction for this set of tests of interest is pictured in Figure 2.2a. It groups the observations into two abstract observations: R as already defined, and L, containing all other observations. Clearly the tests of interest can be expressed in terms of these abstract observations.

2.2.2 Accuracy

Expressiveness ensures that a model of the abstract system *can* be used to make predictions for the tests of interest, however it does not ensure that those predictions will be the *same* predictions a model of the primitive system would make. The ultimate goal is to learn a model that provides the predictions $p(t|h)$ for all tests of interest $t \in \mathcal{T}^I$ and all *primitive* histories $h \in \mathcal{H}$. Given an expressive abstraction η, one is still only guaranteed to be able to make the predictions $p(t|H)$ for all $t \in \mathcal{T}^I$ and all *abstract* histories $H \in \mathcal{H}^\eta$.

Even if an abstraction is expressive, it may ignore some details that are important for making good predictions for the tests of interest. An abstraction with the *accuracy* property guarantees that the predictions for the tests of interest given an abstract history are the same as they would be given a corresponding primitive history.

Definition 2.4. An abstraction η is *accurate* with respect to a set of tests of interest $\mathcal{T}^I$ if and only if for all abstract histories $H \in \mathcal{H}^\eta$, all primitive $h \in H$, and all $t \in \mathcal{T}^I$, $p(t|H) = p(t|h)$.

An accurate abstraction for the 4x4 grid world example is pictured in Figure 2.2b. It groups the observations into three abstract observations: R as already defined, L_2, containing all squares in the third column, and L_1 containing all other observations. As previously noted, the agent's horizontal position is irrelevant to predicting whether it will reach the right-most column in the next step. Also, if the agent is in any

square in the 2 left-most columns, the probability of reaching the right-most column by taking any action is zero. It is, however, important to know whether the agent is in the right-most column or in the third column for making the predictions of interest.

It is trivial to see from Equation 2.2 that the accuracy property holds if and only if the abstraction groups together only histories that have the same associated predictions for the tests of interest. It is also straightforward to see that if η is accurate, then so is any refinement of η.

If an abstraction is expressive and accurate, then a model of the corresponding abstract system can accurately make the predictions of interest in every history.

2.2.3 Homomorphism

Accuracy guarantees that a model of the abstract system makes accurate predictions for the tests of interest at every history. However, it makes no guarantees about any other predictions. A closely related property of abstractions is homomorphism, which requires that *all* abstract tests can be predicted accurately given an abstract history.

Definition 2.5. An abstraction η is a *homomorphism* if and only if for all $T \in \mathcal{T}^\eta$, $H \in \mathcal{H}^\eta$, and $h \in H$, $p(T|H) = p(T|h)$.

A homomorphism essentially promises that it can accurately predict all the information needed to make its own predictions. This is not necessarily so of an accurate abstraction. The accurate abstraction shown in Figure 2.2b makes all the distinctions necessary to predict whether the agent will be in the right-most column in the next time-step. However, it does *not* make enough distinctions to accurately predict whether the agent will see the abstract observation L_1 in the next time step. Specifically, if the agent is placed randomly in the world at the beginning of an episode, in the history nL_1, the model cannot tell whether the agent is in the first or second column. As such, it would make the prediction $p(eL_1|nL_1) = 0.5$, whereas given the

primitive history, the prediction would be deterministic (1 if in the first column, 0 if in the second). In contrast, consider the homomorphism pictured in Figure 2.2c. It groups the observations into four abstract observations: R (as previously defined) and L_1, L_2, and L_3, containing all squares in the 1st, 2nd, and 3rd column, respectively. Knowing which column the agent is in is sufficient to make accurate predictions about which column the agent will enter in the next time-step, so this abstraction is a homomorphism.

Homomorphisms can be useful if the model is required to "roll forward," or "simulate the world." At any given history, a homomorphic model can accurately predict the next abstract observation, which will yield another abstract history, which yields another accurate prediction and so on. This ability is particularly useful for planning purposes. In contrast, an abstraction that is accurate but not a homomorphism allows accurate predictions for the tests of interest, but not for *other* abstract tests. Thus, an accurate model can accurately make the predictions of interest at any history, but it cannot accurately predict how likely that history may be.

Note that the homomorphism property makes no reference to tests of interest and can hold independently of expressiveness and accuracy. That said, if the expressiveness and homomorphism properties hold, then so must accuracy (in an expressive abstraction, the tests of interest can be expressed in terms of tests in $\mathcal{T}^\eta$ and a homomorphism guarantees accurate predictions for all abstract tests). For this reason, expressiveness and homomorphism are often combined. MDP Homomorphisms (Ravindran 2004) and relatedly bisimulation (Larsen & Skou 1991) are abstractions for MDPs that are expressive with respect to the MDP's reward signal (that is, if two states have different associated reward, they must be distinguished) and homomorphic. An MDP homomorphism guarantees that the optimal policy in in the abstract system is the same as the optimal policy in the primitive system. Relatedly, Wolfe & Barto (2006) learn "reusable" homomorphisms that, rather than being expressive

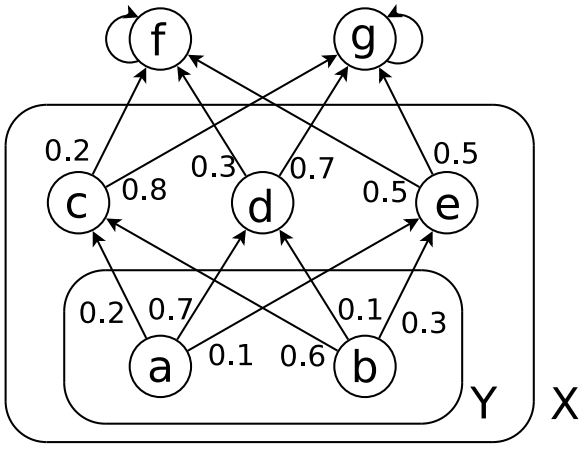


Figure 2.3: The coarsest accurate abstraction yields a simpler system than the coarsest accurate homomorphic abstraction.

with respect to reward, are expressive with respect to some particular set of state features that are likely to be useful for computing reward. This is a special case of the more general concept of tests of interest.

If all that is truly required of the model is to make accurate predictions for the tests of interest (and not additionally to be able to roll forward) a homomorphism is unnecessary, and could potentially result in a more complex model than would result from an abstraction that is accurate, but not a homomorphism. For instance, consider the uncontrolled example shown in Figure 2.3. States are labeled with their observations. Multiple arrows from a single state indicate a stochastic transition with the probabilities indicated. States a and b are the initial state with equal probability. Now say there is a single, two-step test of interest Xf, where X is a set observation: $X \stackrel{\text{def}}{=} \{a, b, c, d, e\}$. Consider the abstraction with 6 abstract observations: g, f, e, d, c, and $Y \stackrel{\text{def}}{=} \{a, b\}$. This abstraction is *not* accurate with respect to its own abstract tests (and so is not a homomorphism). For instance: $p(c|a) = 0.2$ and $p(c|b) = 0.6$ even though $a, b \in Y$. By construction, however, this refinement *is* accurate with respect to the test of interest, most notably because $p(Xf|a) = p(Xf|b) = p(Xf|Y) = 0.3$. Therefore this is an accurate abstraction that is not a homomorphism. It can be straightforwardly determined that the abstract system with respect to this abstraction

has a linear dimension of 4. On the other hand, the only abstraction that satisfies both the accuracy and homomorphism properties is the primitive system itself, which results in a linear dimension of 5.

2.2.4 Other Criteria

The properties discussed here focus mainly on the accuracy of some predictions. In some cases, accuracy may not be the primary goal. For instance, it may be that in some problems, getting the exact values of the predictions of interest is not important and only some more qualitative feature of the predictions is necessary (like identifying the most likely test at a given history, or identifying which tests are possible following a history). Accuracy is a very general criterion in this sense since an accurate model can be used to accurately compute any function of the tests of interest. Then again, a more qualitative criterion may admit a coarser abstraction (and perhaps one that is easier to obtain). Because it is both a simply-stated and general-purpose goal, the discussion in this chapter (and this thesis more generally) will tend to focus on accuracy. However, many of the results and methods discussed could be adapted to other criteria by replacing the accuracy property

$$p(t|H) = p(t|h) \; \forall t \in \mathcal{T}^I, H \in \mathcal{H}^\eta, h \in H$$

with the equality of some function of the predictions:

$$f(p(t_1|H), p(t_2|H), ...) = f(p(t_1|h), p(t_2|h), ...) \; \forall H \in \mathcal{H}^\eta, h \in H,$$

where $t_1, t_2, ...$ are the tests of interest. Much of the mathematical and algorithmic development will remain essentially unchanged.

2.3 Constructing an Abstraction

This section will present algorithms for constructing abstractions that satisfy expressiveness and accuracy, as well as homomorphism. There is already a great deal of work on constructing abstractions for MDPs, mainly focusing on the problem of finding a homomorphism that allows for accurate prediction of the reward signal. There are algorithms for finding MDP homomorphisms (and approximate homomorphisms) using exhaustive search (Ravindran 2004), optimization of bisimulation metrics (Taylor et al. 2009), and gradient descent (Sorg & Singh 2009), just to name a few. Wolfe & Barto (2006), as mentioned before, present an algorithm that uses decision tree methods to find an MDP homomorphism that is accurate with respect to predictions about some set of observation features.

This thesis is focused on the partially observable case, which is far less well-studied in the context of abstraction learning and in many ways more challenging. The main difference is that in partially observable systems predictions depend upon the entire history, and not just the most recent observation. This dramatically affects the problem of learning an accurate abstraction. For instance, in the Markov case, since histories with the same last observation are guaranteed to correspond to the same predictions, one can test the accuracy of an abstraction η by performing a finite number of comparisons: for every primitive observation $o \in \mathcal{O}$ and every test of interest $t \in \mathcal{T}^I$, simply check if $p(t|o) = p(t|\eta(o))$. In the "model-minimization" setting these predictions are assumed to be available *a priori*. Otherwise, they can be estimated from data. In the partially-observable case, however, the *full history* must be taken into account, so the analogous procedure would be to check if $p(t|h) = p(t|\eta(h))$ for every history $h \in \mathcal{H}$. Of course, since there are infinitely many histories, this procedure will never finish. It is not immediately clear if it is possible to determine whether an abstraction is accurate in the partially observable case, let alone whether it is homomorphic (which requires checking infinitely many tests, as well as histories).

Later in this chapter it will be shown that both accuracy and homomorphism can be verified in a finite amount of time related to the system's linear dimension.

Another affected issue is determining how to improve an abstraction that is not accurate. In the Markov case, if one finds a pair of observations o_1 and o_2 such that $p(t|o_1) \neq p(t|o_2)$ for some test of interest t, then it is immediately clear that o_1 and o_2 must be distinguished in any accurate abstraction. In the partially observable case, if one finds a pair of histories, say $h_1 = a_1o_1a_1o_2$ and $h_2 = a_1o_3a_1o_4$ such that $p(t|h_1) \neq p(t|h_2)$, what conclusions can one draw? Surely h_1 and h_2 must be distinguished but that could be accomplished by distinguishing o_1 from o_3 *or* by distinguishing o_2 from o_4. It is not immediately clear how to identify which observations can be safely lumped together and which must be distinguished in the partially observable case. This chapter will address this issue as well.

Compared to the Markov case, there has been relatively little work done on automatically constructing abstractions for partially observable systems. Soni & Singh (2007) defined PSR homomorphisms, which are a generalization of the class of abstractions presented here because they also allow action abstraction and history-dependent abstraction. However, they provide no algorithms for constructing an abstraction or even for determining if a given abstraction *is* a PSR homomorphism.

Wolfe (2010) defined POMDP homomorphisms which abstract not only the observations and actions, but also the hidden states in a POMDP. They present polynomial time algorithms for determining whether a given abstract POMDP is homomorphic to the primitive system. However, these algorithms only test sufficient conditions for homomorphism, not necessary conditions, and thus may generate false negatives (rejecting legitimate homomorphisms). Furthermore, these algorithms are developed for the model-minimization problem, and therefore assume access to a complete POMDP model of the primitive system *a priori*. Wolfe also provides a heuristic method that searches for the coarsest possible POMDP homomorphism. Though their algorithm

is guaranteed to provide a homomorphism, it is not guaranteed to find the coarsest homomorphism.

The results in the following sections provide the first necessary *and* sufficient conditions for accurate and homomorphic abstractions of partially observable systems that can be computed in finite time. These conditions lead to conceptual algorithms that are guaranteed to find the coarsest abstraction (or the family of coarsest abstractions) that satisfies the desired property (accuracy or homomorphism). These algorithms are computationally expensive, but they are provably sound, and can be applied to small problems with some practically-minded tweaks (as will be seen in Section 2.3.4). A more practical abstraction learning algorithm that scales to larger problems will be presented in Chapter 4.

Since the primary goal of obtaining an abstraction in this work is to simplify the model-learning problem, it would be counter-productive to assume a complete model of the primitive system *a priori* as is typically done in the model-minimization literature. That said, the algorithms in the next few sections will be developed in the idealized case that the predictions $p(t|h)$ for all primitive tests $t \in \mathcal{T}$ and all primitive histories $h \in \mathcal{H}$ are known exactly and the linear dimension of the original system is known. These assumptions are similar to assuming the existence of a perfect, primitive model. However, because there is no assumed access to the *internals* of a primitive model (such as its hidden states, or its parameters) the algorithms will be easily adapted to the more practical setting where the predictions are not known but can be estimated from data (this will be discussed further in Section 2.3.4).

The following sections describe algorithms (and supporting theoretical results) for automatically constructing coarse abstractions with the three properties described in the previous section: expressiveness, accuracy, and homomorphism.

2.3.1 Constructing an Expressive Abstraction

Compared to the other two properties, constructing an expressive abstraction is fairly straightforward. Basically, in order to satisfy expressiveness, one must ensure that η does not group together any observations that are distinguished by the tests of interest. In some cases, this is trivial. For instance, it is common to be interested only in predictions about certain observation features. In this case there may be some surjection ψ defined over $\mathcal{O}$ which gives the value of the feature for any observation and the tests of interest are *all* set tests with respect to the observation feature ψ: $\mathcal{T}^I \stackrel{\text{def}}{=} \{a_1\psi(o_1)a_2\psi(o_2)...a_k\psi(o_k) : 1 \leq k < \infty, a_1, ..., a_k \in \mathcal{A}, o_1, ..., o_k \in \mathcal{O}\}$. In this case, simply setting $\eta = \psi$ clearly satisfies expressiveness.

When the tests of interest are explicitly defined in terms of a partition, that partition can serve as the expressive observation abstraction. For arbitrary sets of tests of interest, finding an expressive abstraction is slightly more complicated. Let $\mathcal{T}^I$ be any finite set of set tests. Each test $T \in \mathcal{T}^I$ is a sequence of actions and set observations: $T = a_1O_1a_2O_2...a_kO_k$. Let $\mathcal{O}^{\mathcal{T}^I}$ be the set of *all* abstract observations that appear in the tests of interest. Note that $\mathcal{O}^{\mathcal{T}^I}$ does not itself comprise an observation abstraction as it is not necessarily a partitioning of $\mathcal{O}$: the set observations may overlap and they may not collectively cover the entire space of primitive observations. The task is to find an abstraction η that respects all distinctions made by the observations in $\mathcal{O}^{\mathcal{T}^I}$. This is actually fairly simple: for any two primitive observations o_1 and o_2, let $\eta(o_1) = \eta(o_2)$ if for all $O \in \mathcal{O}^{\mathcal{T}^I}$ either $o_1 \in O$ and $o_2 \in O$ or $o_1 \notin O$ and $o_2 \notin O$. In other words, if o_1 and o_2 are never distinguished by any set observation in $\mathcal{O}^{\mathcal{T}^I}$ they can be grouped together. The result is an abstraction in which every set observation in $\mathcal{O}^{\mathcal{T}^I}$ is the union of some set of abstract observations. So, η can be constructed in $O(|\mathcal{O}|^2|\mathcal{O}^{\mathcal{T}^I}|)$ time (it performs this check for every pair of observations) and is expressive with respect to $\mathcal{T}^I$.

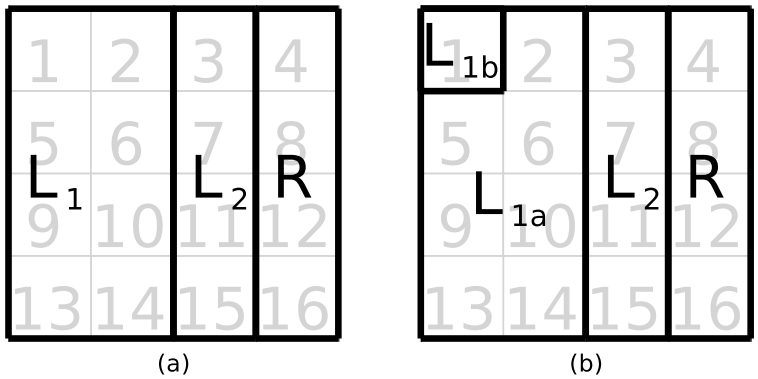


Figure 2.4: Accurate refinements in the 4x4 gridworld. (a) is a coarsest abstraction, (b) is not.

2.3.2 Constructing an Accurate Abstraction

This section will present the main contribution of this chapter: a method for automatically finding an accurate abstraction with respect to some given set of tests of interest $\mathcal{T}^I$. The algorithm will apply to any finite set of tests of interest and also to infinite sets of tests of interest under some conditions that will be described below. Note that an accurate abstraction always exists because this property is trivially true of the primitive system. Thus, it does not suffice to simply find *some* accurate abstraction. A far more sensible goal is to find the *coarsest* accurate abstraction (or one of them).

Definition 2.6. A *coarsest accurate abstraction* with respect to a set of tests of interest $\mathcal{T}^I$ is an accurate abstraction η with respect to $\mathcal{T}^I$ such that any coarsening of η is not accurate.

Note that the definition is for *a* coarsest accurate abstraction, rather than *the* coarsest accurate abstraction. In general, refinement induces only a partial order over abstractions and therefore there may be more than one coarsest accurate abstraction. In a subsequent section it will be shown that, under some conditions, the coarsest accurate abstraction is, in fact, unique. For some systems, a coarsest and a non-coarsest accurate refinement can induce drastically different linear dimensions in the

resulting abstract systems. For instance, recall the abstraction η of the 4x4 grid world pictured in Figure 2.4a that is accurate with respect to the tests of interest $\mathcal{T}^I \stackrel{\text{def}}{=} \{nR, eR, sR, wR\}$. It is easy to check that the linear dimension of the system abstracted according to η is 4. Now consider η', a refinement of η, pictured in Figure 2.4b. Clearly η' is still accurate with respect to the tests of interest (any refinement of an accurate abstraction is accurate). However, though it may not be obvious from inspection, its corresponding abstract system has a linear dimension of 16, the same as the primitive system. On a general $k \times k$ grid, this would represent a quadratic increase over the linear dimension induced by the coarsest accurate refinement.

So, given a set of tests of interest $\mathcal{T}^I$, the goal is to find a coarsest accurate abstraction with respect to $\mathcal{T}^I$. The algorithm will rely upon three theoretical results, presented in the next section.

2.3.2.1 Violations of the Accuracy Property

The procedure for finding an accurate abstraction will search for pairs of histories h and h' for which there exists some test of interest $t \in \mathcal{T}^I$ with $p(t|h) \neq p(t|h')$. Such a pair of histories violates the accuracy property, and must be distinguished by the abstraction. The main result of this section will be to show that only finitely many such pairs of histories need to be checked in order to construct an accurate refinement. The proofs for the theorems stated in this section can be found in Appendix A.

Note that, as discussed above, identifying such a pair of histories is not, in itself, very informative. Knowing that h and h' must be distinguished indicates only that at least one pair of observations at corresponding time steps in the two histories must be distinguished, but not which pair. That said, for some pairs of histories, it *is* clear. If h and h' differ only *at one time step*, then the pair of observations at that time step must be split apart. As will be demonstrated below, it turns out that it suffices to check *only* such pairs of histories.

Definition 2.7. A *violation* for observations o_1 and o_2 is a tuple $\langle t, a, h_1, h_2 \rangle$ such that $t \in \mathcal{T}^I$, $a \in \mathcal{A}$, $h_1 \in \mathcal{H}$, $h_2 \in \mathcal{T}$, and $p(t|h_1 a o_1 h_2) \neq p(t|h_1 a o_2 h_2)$.

Note that a violation can only occur between two observations if they can both occur after the same history. Two such observations are called *comparable*.

Definition 2.8. Two observations o_1, o_2, are *comparable* if there exists some history h and some action a such that $p(hao_1|\emptyset) > 0$ and $p(hao_2|\emptyset) > 0$.

The first result (stated below) implies that, in order to construct a coarsest, accurate refinement, it suffices to distinguish observations that have a violation, as defined above. If all observations are comparable, then the pair-wise relation "Have no violations" is an equivalence relation and, as such, induces a partition over the set of primitive observations $\mathcal{O}$. Using this partition as an observation abstraction distinguishes every observation that *must* be split in *any* accurate refinement, and no more:

Theorem 2.9. *If all observations are comparable and η is the abstraction induced by the relation "Have no violations," then η is the* unique *coarsest, accurate abstraction. Furthermore, there is no accurate abstraction that results in an abstract system with a smaller linear dimension than the abstract system corresponding to η.*

Proof. See Appendix A. □

Incomparable observations have only a minor impact on finding a coarsest, accurate refinement, though they can affect the linear dimension of the abstract system that results. For a brief discussion of incomparable observations, see Appendix B. For the remainder of this chapter, all observations are assumed to be comparable.

Because of Theorem 2.9, the search for pairs of histories that violate the accuracy criterion can be limited to pairs of histories that differ in only one time step. However, there are still infinitely many such history pairs. The next result will show that only finitely many pairs of histories need be considered.

Theorem 2.10. *If the linear dimension of the system is n, and a violation $\langle t, a, h_1, h_2 \rangle$ exists for observations $o_1, o_2 \in \mathcal{O}$ then a violation $\langle t, a, h_1', h_2' \rangle$ exists for o_1 and o_2 with $length(h_1' a o_1 h_2') < n^2$.*

Proof. See Appendix A. □

Theorem 2.10 implies that it can be determined whether or not any pair of observations has a violation by comparing histories of length n^2 or less, where n is the linear dimension of the system. Thus only a finite (albeit large) number of comparisons are required to determine whether an abstraction is accurate, where that number naturally depends upon the complexity of the primitive system.

Note that a comparison must be made for every test of interest. If the set of tests of interest $\mathcal{T}^I$ is infinite, there are still infinitely many comparisons to make. The final result in this section deals with a natural special case of infinite $\mathcal{T}^I$. Consider the case where there is some *abstraction of interest* η^I and the set of tests of interest $\mathcal{T}^I = \mathcal{T}^{\eta^I}$, the set of *all* abstract tests induced by η^I (that is, all sequences of primitive actions and abstract observations under η^I). In that case, it is still possible to determine whether two observations have a violation in finite time.

Theorem 2.11. *If the linear dimension of the system is n, $\mathcal{T}^I = \mathcal{T}^{\eta^I}$ for some abstraction η^I, and a violation $\langle t, a, h_1, h_2 \rangle$ exists for observations $o_1, o_2 \in \mathcal{O}$ then a violation $\langle t', a, h_1, h_2 \rangle$ exists for o_1 and o_2 with $length(t') \leq n$.*

Proof. See Appendix A. □

These results have shown that it is possible to find an accurate refinement by comparing a finite number of predictions. They also suggest a straightforward algorithm for finding an accurate abstraction: simply perform an exhaustive search for violations to find observations that must be distinguished in an accurate refinement.

2.3.2.2 One-Pass Algorithm

The one-pass algorithm is a conceptual procedure for finding an accurate abstraction. It is an exhaustive search for violations $p(t|h_1ao_1h_2) \neq p(t|h_1ao_2h_2)$ over all $o_1, o_2 \in \mathcal{O}$, $a \in \mathcal{A}$, $t \in \mathcal{T}^I$ (or, for the case where $\mathcal{T}^I = \mathcal{T}^{\eta^I}$ for some abstraction η^I, $t \in \mathcal{T}^I$ with $length(t) \leq n$), and $h_1, h_2 \in \mathcal{H}$ with $length(h_1) + length(h_2) < n^2 - 1$. Once all possible violations have been accounted for, equivalence classes are found and serve as the abstract observations. The resulting abstraction is guaranteed to be a coarsest accurate abstraction and, as long as all observations are comparable, the resulting abstract system will have the smallest possible linear dimension while still retaining accuracy.

The goal of the development of the one-pass algorithm so far has simply been to find the coarsest possible accurate abstraction. One can also adapt the algorithm to take as input an *intial abstraction* η_0 and find the coarsest accurate *refinement* of η_0. This would be useful, for instance, if one wanted an abstraction that was both expressive *and* accurate. Then one could first find an expressive abstraction, and then apply the one-pass algorithm to find an accurate refinement of that abstraction, obtaining an expressive, accurate abstraction at the end. The only change to the algorithm necessary is that for every pair of primitive observations $o_1, o_2 \in \mathcal{O}$ such that $\eta_0(o_1) \neq \eta_0(o_2)$, it must be ensured that $\eta(o_1) \neq \eta(o_2)$. This can be accomplished by asserting that o_1 and o_2 have a violation, regardless of whether or not they do. Then the one-pass algorithm is guaranteed to produce a coarsest accurate refinement of η_0.

Note that, in its pure form, the one-pass algorithm is computationally daunting. Even putting aside the fact that the linear dimension of most problems of real interest will be enormous in itself, the number of tests and histories to be checked grows *exponentially* in the length to be considered. Thus in the worst case, even for systems of moderate complexity, finding all violations is entirely impractical. An example

in Appendix A demonstrates that the worst case bound of $O(n^2)$ on the length of histories necessary to detect a violation is tight. However, in many cases one might expect to see violations involving much shorter tests and histories. The next section will describe an algorithm that finds a homomorphic abstraction and can also take advantage of the existence of such violations

2.3.3 Constructing a Homomorphic Abstraction

Because the homomorphism property is a statement about accuracy, it is intuitive to imagine that the one-pass algorithm could be adapted to produce a homomorphism. Recall that, to be a homomorphism, an abstraction must be accurate with respect to *all* of its own corresponding abstract tests. Much like accuracy, a homomorphism always exists because the primitive system is trivially homomorphic to itself. Unlike accuracy, it is not very sensible to search for the coarsest possible homomorphism, as the coarsest possible abstraction (that maps all primitive observations to the same abstract observation) is also always a homomorphism. This is why the homomorphism property is often combined with another property (such as expressiveness) when evaluating abstractions. As such, the goal of this section will be to take some initial abstraction η_0 that satisfies some other property such as expressiveness or accuracy and to find the coarsest *refinement* of η_0 that is a homomorphism.

Consider the initial abstraction η_0. One can use the one-pass algorithm to find a refinement η_1 of η_0 such that η_1 is accurate with respect to all abstract tests induced by η_0 ($\mathcal{T}^{\eta_0}$). As shown in Section 2.3.2.1, this can be done in finite time. Of course, η_1 may not be accurate with respect to its *own* set tests, so it still may not satisfy the homomorphism property. One can, however, use the one-pass algorithm to find a refinement of η_1, called η_2 such that η_2 is accurate with respect to the abstract tests induced by η_1. This iterative refinement procedure can proceed until an abstraction η_k is found for which the one-pass algorithm finds no violations. By definition, η_k is

a homomorphism. This iterative process is guaranteed to stop eventually since the completely refined abstraction that is equivalent to the primitive system trivially satisfies the homomorphism property. Because the one-pass algorithm finds the coarsest accurate refinement at every step, this iterative procedure will stop at a coarsest homomorphic refinement.

Of course, naively applying the one-pass algorithm iteratively may not be necessary. Though in order to ensure that *all* violations are found the one-pass algorithm must consider long tests and histories, one might expect in general to be able to find some violations involving short tests and histories. This is the inspiration for the following improvement over the naive iterative procedure just described that can exploit the existence of violations involving short tests and histories by being more opportunistic in its splitting.

2.3.3.1 Iterative Algorithm

The iterative algorithm is a variation of the one-pass algorithm which, rather than searching for *all* violations before refining, produces a refinement as soon as it can do so "safely." A safe refinement is any one that only distinguishes observation pairs that were previously grouped together if they have a violation. It will often be possible to make a safe refinement without finding *all* violations. In each iteration i, given abstraction η_{i-1}, the algorithm looks at increasingly long histories and tests (up to length n^2 and n, respectively) searching for a safe refinement. Once one is found, the safe refinement is named η_i and the next iteration begins. Because a safe refinement always exists (the full one-pass algorithm always produces a safe refinement), this algorithm is guaranteed to stop at a refinement that satisfies the homomorphism property. In fact, because it only makes *safe* refinements, and therefore only distinguishes observations that have a violation between them, this algorithm never distinguishes two observations that the naive iterative procedure presented ear-

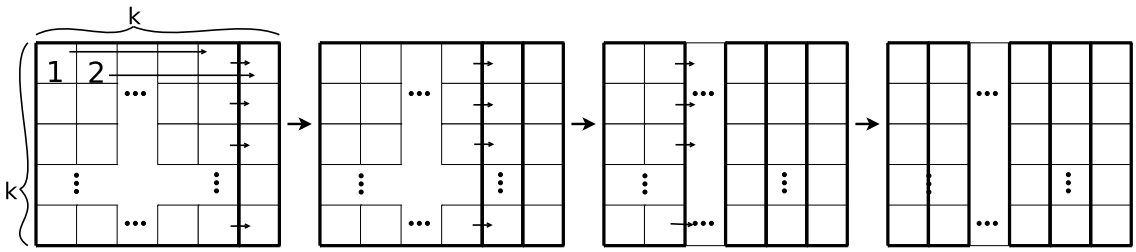


Figure 2.5: The iterative algorithm on the $k \times k$ gridworld.

lier would not. It produces precisely the same refinement in the end, though it may perform more iterations along the way.

This more opportunistic iterative algorithm can enjoy substantial improvements in the length of tests and histories that must be considered, since intermediate splits can create new violations. Consider the $k \times k$ grid world pictured in Figure 2.5, analogous to the earlier 4×4 grid world example. Let the initial abstraction be expressive with respect to the tests of interest (that is, one abstract observation for the right-most column, and one for the all the rest of the positions). In order to find a violation for observations 1 and 2, the one-pass algorithm must look at histories and tests that have a combined length of $k - 1$ (because it takes one step to see observation 1 or 2 and $k - 2$ steps to reach the right side from square 2 and not from square 1). In comparison, the iterative algorithm would find, by looking at 1-step histories and 1-step tests, that the $(k-1)$th column must be split off because it is possible to reach the kth column from those squares in one step, and not from any others. In the next iteration, again checking only one-step tests and one-step histories, it would discover that the $(k-2)$th column should be split off, and so on until the final, homomorphic refinement was found (one set observation per column).

Ultimately, of course, when the iterative algorithm reaches a stopping point, it must still check tests and histories of the same length as the one-pass algorithm in order to verify that there are indeed no more violations. So, it is appropriate to think of the iterative algorithm as a "fail-fast" style algorithm, which will tend to rule out

non-solutions quickly, but which may still take a long time to verify a solution once it is ultimately found.

In principle, since the iterative algorithm's last iteration is necessarily a full run of the one-pass algorithm, one should always prefer the one-pass algorithm if only accuracy is desired. In practice, however, the iterative algorithm can have advantages over the one-pass algorithm, even if a homomorphism is not required, due to the fact that it may only need to search for violations involving short tests and histories. This will be empirically demonstrated in the next section.

2.3.4 Experiments

While the algorithms presented here are conceptual rather than practical in nature, it is possible, with a few tweaks, to apply them to learning partial models in small problems. Assume a set of tests of interest are given, as well as a training data set of action/observation sequences of interaction with the system.

First and foremost, the true values of predictions are not known in this setting. Instead, predictions will be estimated from data, using basic sample averages[1]:

$$\hat{p}(t|h) = \frac{\#\text{ times } t \text{ succeeds from } h}{\#\text{ times } acts(t) \text{ taken from } h}. \tag{2.3}$$

Due to sampling error, no two estimates are likely to be the same, rendering the equality tests used in the abstraction learning algorithms overly strict. Instead, estimated values will be compared using a chi-square test of homogeneity to determine if the predictions are *statistically significantly* different.

Secondly, it will be impractical to perform the violation search with histories and tests of the length required to guarantee accuracy because in most systems of interest

[1]Bowling et al. (2006) note that the estimator in Equation 2.3 is biased when the behavior policy used to generate the training data depends upon past observations. These experiments use a uniform random behavior policy and therefore do not suffer from this issue. Bowling et al. provide an estimator that is unbiased for any stationary data-collection policy and adapting these algorithms to handle more general classes of exploration policies is an interesting subject for future work.

the linear dimension n will be large. Furthermore n is typically not even known *a priori*, though it could, in principle, be estimated from the data (James & Singh 2004). For many systems, however, a coarsest, accurate refinement can be found by checking only short tests and histories, especially if the iterative algorithm is employed. In practice a maximum test length l^t_{max} and a maximum history length l^h_{max} will be set as parameters to the violation search.

In the following experiments, data was collected by an agent exploring the environment using a uniform random policy in a number of distinct trajectories. Both abstraction learning algorithms were applied with various amounts of training data. The experiments compare the performance of the one-pass algorithm to that of the iterative algorithm, and explore the effect of different choices of α, the significance level for the chi-square tests used to compare table entries.

The algorithms are first applied to a 5×5 version of the running grid world example and then to a more complex problem with a linear dimension of roughly 1000. The main results can be seen in Figures 2.6 and 2.8a, which show that both algorithms were able to learn accurate abstractions, though in both cases the iterative algorithm outperformed the one-pass algorithm in terms of sample complexity. Figures 2.7 and 2.8b show that the value of α effectively controls the "agressiveness" of the refinement, and this effect is particularly pronounced in the iterative algorithm. Details about both experiments follow.

2.3.4.1 Grid World

The first set of experiments is on the grid world example seen throughout the chapter, in this case a 5×5 grid. The primitive system has 25 observations and a linear dimension of 25. Let R be an abstract observation containing all positions in the right-most column and let L contain the rest of the positions. Let η_0 be the corresponding abstraction. In the previous examples the goal was to predict whether

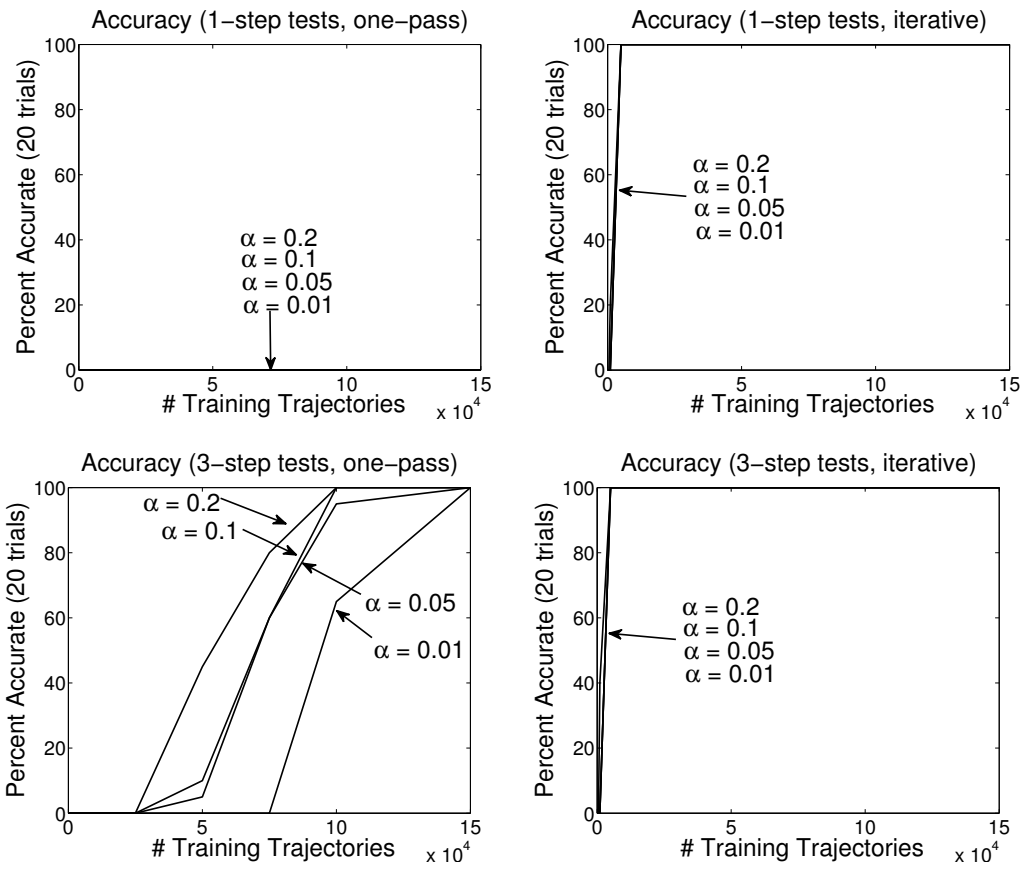


Figure 2.6: Percentage of accurate learned refinements (out of 20) for different test lengths in the 5x5 grid world domain.

the agent would be in the right-most column in the next step so the tests of interest were $\{nR, eR, sR, wR\}$. In this experiment, to exercise the algorithms' abilities to deal with infinitely many tests of interest, the tests of interest are *all* abstract tests involving R and L. Since η_0 is already expressive with respect to the tests of interest, the goal is to find the coarsest accurate (or homomorphic) refinement of η_0. In this case, the coarsest accurate refinement and the coarsest homomorphic refinement are the same. They have 5 abstract observations (one for each column) and result in an abstract system with linear dimension 5.

Figure 2.6 shows the percentage of refinements found that were accurate (out of 20 runs) compared to the number of trajectories in the training data for four choices of α (0.01, 0.05, 0.1, and 0.2), and two choices of l^t_{max} (1 and 3). In all cases $l^h_{max} = 1$.

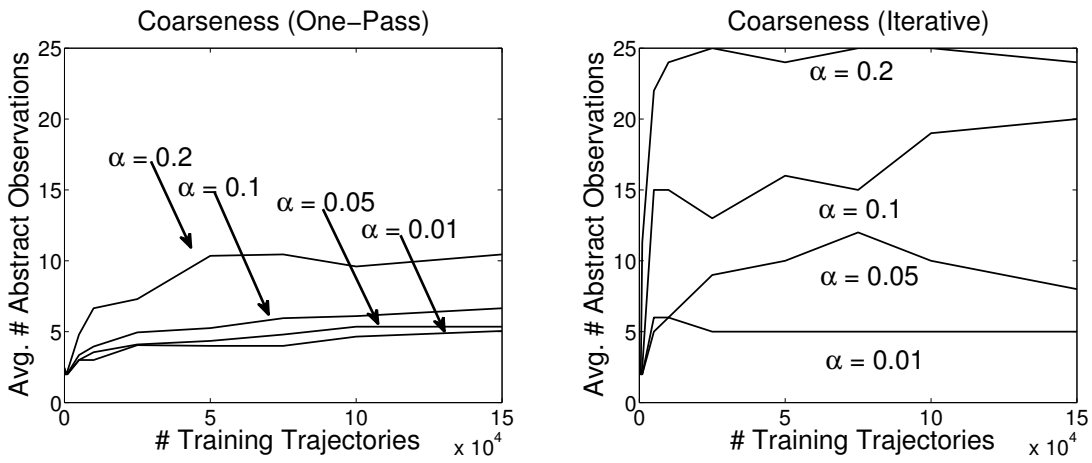


Figure 2.7: Average number of abstract observations (out of 20 runs) in the 5×5 grid world.

As predicted in section 2.3.3.1, the one-pass algorithm is unable to find an accurate refinement using only 1-step tests, while the iterative algorithm can. Even with 3-step tests, when both algorithms are capable of finding an accurate refinement, the iterative algorithm seems to be significantly more data-efficient, regardless of the choice of α. This is because short tests and short histories tend to occur more often than long tests and long histories. As a result, they will tend to have more associated data and therefore better estimates. It takes more training trajectories to obtain good enough estimates for the length 3 tests required for the one-pass algorithm to detect a violation.

Figure 2.7 shows the average number of abstract observations found (out of 20 trials) compared to the number of training trajectories for the same four choices of alpha (and 3-step tests). The effect of α is unsurprising: higher values of α tend to cause over-refinement (because the statistical tests are prone to identifying spurious differences) while lower values require more data to make the necessary distinctions. The over-splitting effect is especially pronounced with the iterative algorithm, because spurious splits in early iterations can propagate into later ones. That said, the iterative algorithm performed well with a low value of α that did not generally result

in overly-fine abstractions.

2.3.4.2 Machine Maintenance

The second set of experiments is in a slightly modified version of the Machine Maintenance domain presented by Cassandra (1998). In this domain the agent is in charge of a manufacturing machine with k components. The components can be in any of 4 states of disrepair, each associated with decreasing probabilities of working properly. At every time step, the agent can choose from four actions. If it chooses to *replace* the machine all components are reset to excellent condition. If it *repairs* the machine each component upgrades its status with some probability. In either of these cases, the agent observes only that the machine has been serviced, but not the results of the servicing. If it *inspects* the machine the current status of all components is revealed. Finally, if the agent chooses to *manufacture*, the machine produces a product, stochastically good or bad (each component independently and stochastically either works or fails and if any component fails the product is bad), and the components downgrade their status with some probability due to "wear and tear."

This experiment presumes that the agent would be primarily interested in whether the machine produces *good* or *bad* products and not in the internal workings of the machine *per se*. So, let the initial abstraction η_0 have four abstract observations: $\{good\}$, $\{bad\}$, $\{serviced\}$, and $\{excellent, fair, poor, broken\}^k$, where the last abstract observation contains all 4^k possible machine states that could be observed after an inspection. The tests of interest are all abstract tests under η_0, $\mathcal{T}^{\eta_0}$. In order to make predictions at this level, one need not actually know the full machine state. For instance, if any component is *broken*, the machine will always produce *bad* products. If no component is *broken*, it suffices to merely know *how many* components are in each state of disrepair, rather than which specific components are in which specific

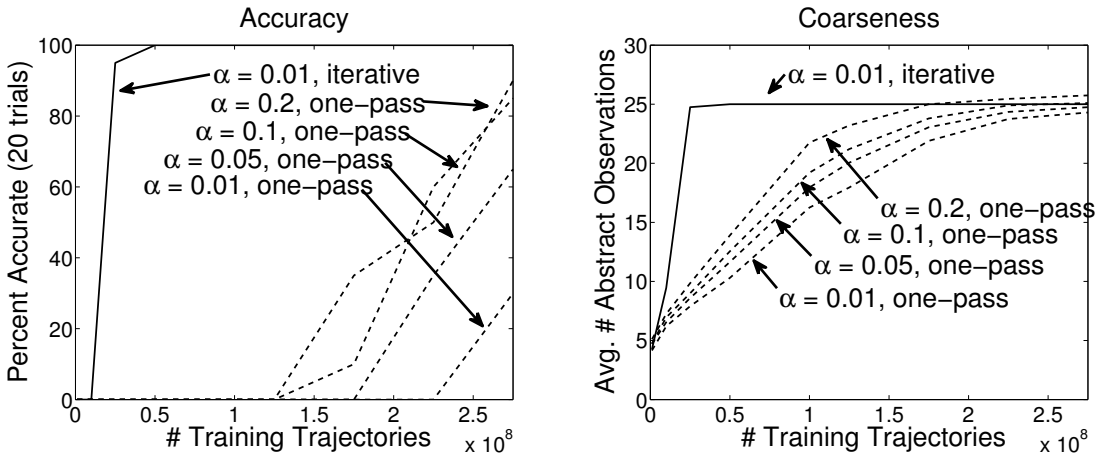


Figure 2.8: Results in the machine maintenance domain.

states.

This experiment uses a 5 component machine. Thus, the primitive system has linear dimension of $4^5 = 1024$, large enough to pose a significant challenge to standard complete model-learning methods such as POMDPs. However, the abstract system that considers only how many components are in each state and whether any component is broken has a more manageable linear dimension of 22.

In these experiments $l^t_{max} = 2$ and $l^h_{max} = 1$. At the beginning of each trajectory each component's state was chosen uniformly randomly. Figure 2.8 shows that, as with the grid world domain, the iterative splitting algorithm with a small α (solid line) significantly outperforms the one-pass algorithm (dashed lines), in this case requiring an order of magnitude fewer trajectories to find an accurate refinement.

2.4 A Note on Action Abstraction

Just as one can ignore details from the observations that are irrelevant to making the predictions of interest, one can imagine ignoring details from the agent's *actions*. For instance, MDP homomorphisms (Ravindran 2004), PSR homomorphisms (Soni & Singh 2007), and POMDP homomorphisms (Wolfe 2010) all incorporate action

abstraction into their formalisms. While this thesis will not focus explicitly on action abstraction, this section will briefly describe some specific challenges in learning an action abstraction and how they relate to the discussion in this chapter.

An action abstraction is similar to an observation abstraction: a partitioning (or surjection) over the set of possible actions. By lumping actions together, one can ignore extraneous details in the history of decisions the agent has made (which is particularly useful when the agent has a large or infinite number of available actions). Action abstraction, however, faces a challenge that observation abstraction does not. While in observation abstraction, a coarser abstraction *always* leads to a simpler (or at least no more complex) abstract system, this same monotonicity does not hold for action abstraction. As a simple example, consider a world called the "Echo Chamber." This world has k actions $a_1, a_2, ..., a_k$ and k observations $o_1, o_2, ..., o_k$. Whenever the agent takes action a_i, the environment responds deterministically by emitting o_i (the echo). This system is extremely simple; it has a linear dimension of 1. Now consider applying an action abstraction. In fact, consider the coarsest possible action abstraction that maps all primitive actions to a single abstract action A. What is, for instance, $p(Ao_1|h)$? Because the abstract test does not specify which action the agent will take, the outcome will depend upon the agent's behavior policy. Specifically, $p(Ao_1|h) \stackrel{\text{def}}{=} \sum_{a \in A} \Pr(a|h)\Pr(o_1|h,a)$. Note that the second factor is 1 if $a = a_1$ and 0 otherwise. Therefore, in this particular example, $p(Ao_1|h) = \Pr(a_1|h)$, simply the probability that the agent will choose action a_1. As such, in order to make predictions in the abstract system, one must make predictions about the *agent's own behavior*. If the agent's behavior is complex (for instance, if it is a learning agent), the abstract system will be correspondingly complex.

The echo chamber example demonstrates that a coarser action abstraction can lead to a more complex abstract model, because it may capture the agent's behavior as well as the system's dynamics. In order to avoid this, one must seek an action

abstraction that has the *policy independence* property. Formally:

Definition 2.12. Given an observation abstraction η, an action abstraction ξ is *policy independent with respect to* η if and only if for any abstract history H (sequence of abstract actions and observations), and any primitive test $t = a_1o_1a_2o_2...a_ko_k$,

$$p(\xi(a_1)\eta(o_1)\xi(a_2)\eta(o_2)...\xi(a_k)\eta(o_k)|H) = p(a_1\eta(o_1)a_2\eta(o_2)...a_k\eta(o_k)|H)$$

Essentially, an action abstraction is policy independent if the abstract actions still contain enough detail to make accurate predictions about abstract observations. Thus, once one obtains action and observation abstractions that satisfy the main objective (say, expressiveness and accuracy), one generally must refine the action abstraction even further in order to obtain policy independence.

When the observation abstraction in question is homomorphic, policy independence is fairly easy to obtain. It can be straightforwardly shown that if η is a homomorphism, then an action abstraction ξ is policy independent with respect to η if and only if $p(\xi(a)\eta(o)|H) = p(a\eta(o)|H)$ for all abstract histories H and all primitive actions a and observations o (see, for instance Soni & Singh 2007). When η is *not* homomorphic, however, verifying policy independence in the one-step tests does not suffice, making the problem of learning a policy independent action abstraction more challenging. While it is likely that bounds like the ones presented in Section 2.3 may be developed to obtain a finite-time algorithm for obtaining policy independence, this issue will not be explored in this thesis.

2.5 Limitations of Observation Abstraction

This chapter presented a particular strategy for learning a partial model. Specifically, given a set of tests of interest and some training data, one can learn an expressive, accurate abstraction (or alternatively an expressive homomorphism). This

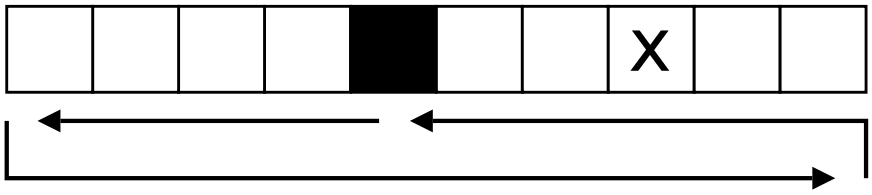


Figure 2.9: Size 10 1D Ball Bounce

abstraction can be applied to the training data and then an abstract model can be learned using any applicable model-learning method, such as EM for POMDPs. The abstract system may be far simpler than the primitive system, as seen in the examples in this chapter, and a model of the abstract system can still be used to make the predictions of interest accurately.

This approach does, however, suffer from some of the same drawbacks of learning a complete model. The abstract model one learns can be simpler because it only makes predictions at an abstract level, rather than making fully detailed predictions about the environment. On the other hand, an abstract model will still make *all abstract predictions*. If the abstraction is a homomorphism, those predictions are made accurately. If the abstraction is merely accurate, they are not guaranteed to be made accurately, but the model makes those predictions nevertheless. So, if the goal is to predict the weather, maybe an accurate abstraction can determine that what the agent sees on the weather channel is important but what the agent sees on the sports channel is not important. But even then, the result is to learn a model that makes predictions about the weather *and* about what will happen on the weather channel. The agent has no particular interest in predicting what will happen on the weather channel in the future; it only wishes to incorporate what has happened on the weather channel in the past into its predictions about the actual weather in the future.

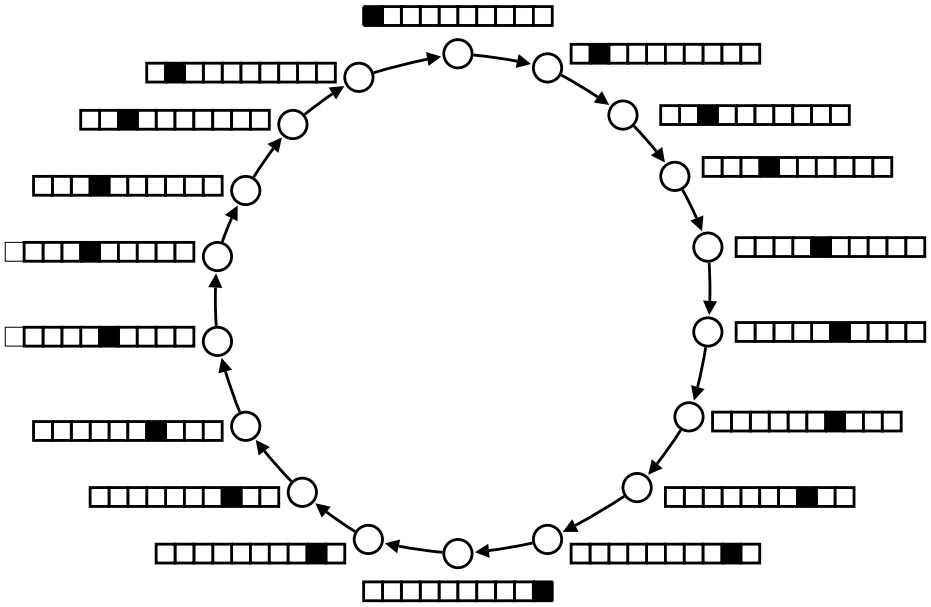


Figure 2.10: POMDP model of the size 10 1D Ball Bounce

2.5.1 The 1D Ball Bounce Example

As a more concrete example, consider the uncontrolled system pictured in Figure 2.9, called the "1D Ball Bounce" system. Variations on this system will serve as recurring motivating examples in this thesis. The agent observes a strip of pixels that can be black or white. The black pixel represents the position of a ball that moves around on the strip. The ball has a current direction and every time-step it moves one pixel in that direction. Whenever it reaches an edge pixel, its current direction changes to move away from the edge. In Figure 2.10 a complete POMDP model of a 10 pixel version of this system is pictured. If there are k pixels, the POMDP has $2k - 2$ hidden states (because the ball can have one of 2 possible directions in one of k possible positions, except the two ends, where there is only one possible direction).

Now say the agent wishes only to predict whether the ball will be in the position marked with the x in the next time step. Clearly this prediction can be made by only paying attention to the immediate neighborhood of the x. The details of what happens to the ball while it is far away do not matter for making these predictions.

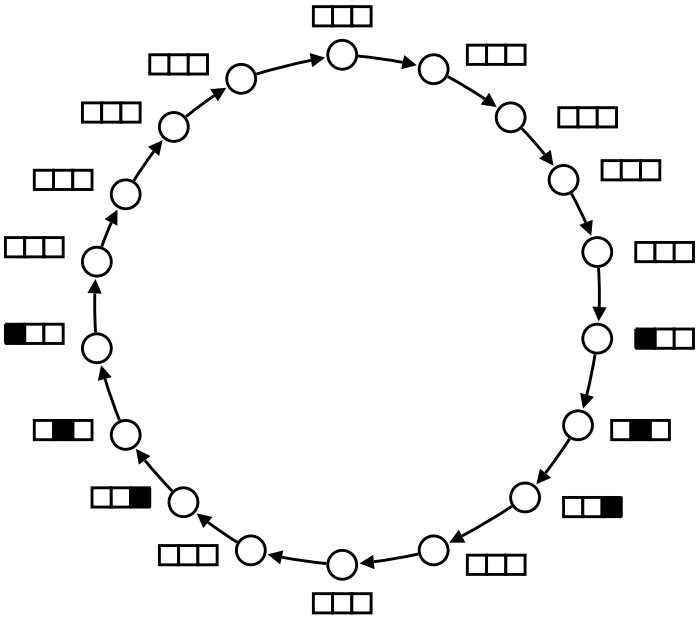


Figure 2.11: POMDP model of the *abstract* 1D Ball Bounce system

So, one could apply an abstraction that lumps together all observations in which the neighborhood about x looks the same. The problem is that an abstract model of this system makes predictions not only about x, but also about the pixels surrounding x. Specifically, the model still makes predictions about whether the ball will enter the neighborhood in the near future. This of course depends on how long it has been since the ball left the neighborhood. So, the POMDP model of the abstract system (pictured in Figure 2.11) has *exactly the same state diagram as the original system*, though its observations have changed to reflect the abstraction. The abstract system and the primitive system have the same linear dimension.

Counterintuitively, the abstract model's complexity is mainly devoted to making predictions other than the predictions of interest. While learning an abstract model can drastically simplify the learning problem by ignoring irrelevant details, an abstract model still learns to make predictions about any details that *are* relevant, even if they are not directly of interest.

A more sensible model is pictured in Figure 2.12. This is a transition diagram in which each state is associated with the predictions for whether the ball will be in x in

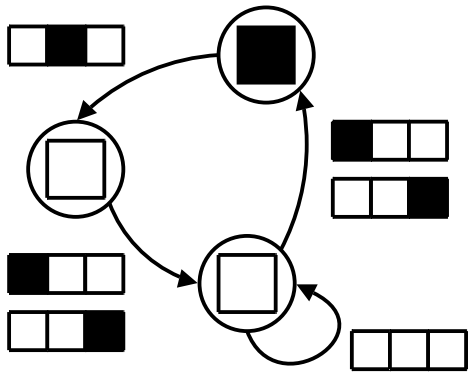


Figure 2.12: A transition diagram for the predictions of interest

the next time step (denoted by the black square or the white square inside the circle). The transitions are labeled with possible observations (limited to the neighborhood around x). When an observation is seen, one simply follows the appropriate transition to update the predictions of interest. In this case, the transitions capture the ball entering the neighborhood, entering position x (the center of the neighborhood), leaving the neighborhood, staying away for some indeterminate amount of time, and then returning. At every time step the predictions of interest can be provided *conditioned* on relevant events in the past, without also *predicting* whether those relevant events will occur in the future. Note that regardless of the number of pixels, this state diagram will *always* have 3 states. The next chapter will present a method that learns a model like that pictured in Figure 2.12.

2.6 Summary

Key points from this chapter:

- Observation abstraction is a strategy for learning a partial model
  - One can ignore details irrelevant to making the predictions of interest.
  - An abstract model is no more complex than a primitive model, and may be far simpler.

- An abstract model can be used to make the predictions of interest accurately if it has two properties:
  - *Expressiveness* – the tests of interest must be expressible in terms of abstract tests.
  - *Accuracy* – predictions for the tests of interest given abstract histories must be the same as predictions given primitive histories.
- This chapter presented conceptual algorithms for constructing abstractions.
  - The main contribution is the first theoretically sound, finite-time algorithm for finding the coarsest accurate abstraction.
  - A tight worst-case bound was presented for determining whether an abstraction is accurate that is exponential in n^2, where n is the linear dimension of the primitive system.
  - This algorithm was adapted to form the first theoretically sound, finite-time algorithm for finding the coarsest homomorphic refinement of a given abstraction.
- The approach of learning an abstract model has limitations.
  - A model of the abstract system makes *all* abstract predictions, which includes many predictions not directly of interest.

CHAPTER 3

Prediction Profile Models

The last chapter demonstrated that when an agent has a restricted set of predictions to make, there may be irrelevant details in its observations that it can ignore. By learning a model of an *abstract* system that ignores these details rather than a complete model of the primitive system, the agent can still make its predictions of interest accurately, but with a far simpler model. On the other hand, as seen in Section 2.5, this approach suffers from the fact that a model of the abstract system, though it avoids making many unnecessary predictions, still in general makes more predictions than are strictly of interest.

To further illustrate this, consider the game of Three Card Monte. The agent is shown three cards, one of which is the "special card." The dealer then flips over the cards to hide their faces and proceeds to mix them up by swapping the positions of two cards at every time step. The agent can observe which cards the dealer swaps. At the end of the game, the agent must identify which card is the special card. To perform well in this game, the agent need only answer one question: "Where is the special card?"

Via an accurate observation abstraction, an agent wishing to learn a model that will answer this question can ignore a great deal of complex and irrelevant phenomena: traffic and passersby on the street, the ambient temperature, the direction of the wind,

even the identity of the dealer and physical characteristics of the playing cards. So long as the agent observes which cards are swapped, it can accurately predict the location of the special card. On the other hand, as seen in Section 2.5, an abstract model makes *all* abstract predictions. In this case this includes predictions about the current location of the special card, but also its location in the future. Furthermore, since the abstract observations necessarily include the swaps made by the dealer, this also includes predictions about how the dealer will behave in the future. If the decision-making process the dealer uses to choose swaps is very complex, learning a model even of the abstract system may be be correspondingly difficult.

This issue arises because training a POMDP model (for instance) of the abstract system produces a model that represents the probability distribution over *all possible* abstract sequences. Such a model is called *generative* because it can be used to generate simulated trajectories of experience. Note that POMDPs (and PSRs for that matter) are *inherently* generative. The state update equations (given in Section 1.2.2.1) for these models require access to one-step predictions and any model that can make all one-step predictions can make any prediction.

Generative models are often desirable because the ability to sample possible futures is very useful for planning purposes. On the other hand, learning a generative model tightly couples what information the model uses to make predictions and what predictions the model makes. If, as in Three Card Monte, the predictions of interest are simple, but rely on the observations of some complex process, a generative model will attempt to capture that more complex process. Furthermore, a generative model is unnecessary for good decision making in Three Card Monte. The agent only needs to predict the *current* location of the special card at any given moment, with no need to predict what the dealer will do in the future or where the special card will be 10 steps from now. Intuitively, one might hope that the complexity of maintaining the predictions about the position of the special card would be independent of the

complexity of predicting what cards the dealer will swap and that therefore it might be possible to learn to make this restricted set of important predictions *even* more simply than learning a generative abstract model.

This chapter presents a type of partial model called a *prediction profile model*, which is a *non-generative* model that maintains the predictions of interest over time but makes *no other predictions*. As with observation abstraction, learning a prediction profile model will involve learning some transformation of the training data and then applying standard modeling methods to the transformed data. Once again, the transformed learning problem can be much simpler than the original problem.

3.1 Prediction Profiles

As in the previous chapter, the agent is given some set of *tests of interest*, which in this chapter is assumed to be finite: $\mathcal{T}^I = \{t_1, t_2, ..., t_m\}$. As always, the tests of interest represent what future events the model should predict at any history. Generically speaking, the goal is to learn a function $\phi : \mathcal{H} \to [0,1]^m$ where

$$\phi(h) \stackrel{\text{def}}{=} \langle p(t_1|h), p(t_2|h), ..., p(t_m|h) \rangle, \tag{3.1}$$

that is, a function from histories to the predictions of interest. Note that the output of ϕ is not necessarily a probability distribution. The tests of interest may be selected arbitrarily and therefore need not represent mutually exclusive events, nor is it generally guaranteed that at least one of the tests of interest will succeed after any particular history h. A particular vector of predictions for the tests of interest is called a *prediction profile*.

Definition 3.1. Let $\phi(h)$ be the *prediction profile* at history h.

One obvious way to learn ϕ is to learn a complete, generative model of the primitive system. Such a model makes all possible predictions in all possible histories, so it

surely can be used to compute ϕ. A second way, as seen in Chapter 2, is to learn an accurate, expressive abstraction with respect to the tests of interest, and then to learn a generative model of the *abstract* system, rather than the primitive system. The resulting model makes all possible *abstract* predictions (which include the predictions of interest) in all histories and can also be used to compute ϕ. A third approach is to more directly estimate ϕ using function approximation techniques, rather than modeling techniques. The next section briefly discusses this alternate approach. The subsequent section introduces *prediction profile models*, which marry some of the strengths of both model learning and direct function approximation.

3.1.1 Learning Predictions via Regression

Note that when the system is Markov, learning ϕ is straightforward. Recall that, by definition, predictions in a Markov system depend only on the most recent observation, rather than the entire history. Thus, rather than learning ϕ which takes histories as input, one can instead learn a function $\phi_{Markov} : \mathcal{O} \rightarrow [0,1]^m$, which maps an *observation* to the predictions for the tests of interest resulting from *all* histories that end in that observation. Note that, as an immediate consequence, in Markov systems with a finite number of observations, there is a finite number of *distinct* prediction profiles: there can be no more distinct prediction profiles than there are observations. When the number of observations and the number of tests of interest are small enough, ϕ_{Markov} can be represented as a $|\mathcal{O}| \times |\mathcal{T}^I|$ look-up table, and the entries estimated using the typical sample averages:

$$\hat{p}(t_i|o) = \frac{\#\text{ times } t \text{ succeeds from histories ending in } o}{\#\text{ times } acts(t) \text{ taken from histories ending in } o}. \tag{3.2}$$

When the number of observations is too large for a look-up table to be feasible, one may be able to exploit the fact that some observations will be associated with

the same (or similar) predictions for the tests of interest by generalizing across these observations through observation abstraction, or by some other means.

Even when the system is non-Markov, one can still attempt to learn ϕ directly, typically by performing some sort of regression over a set of features of *entire histories*. For instance, U-Tree (McCallum 1995) takes a set of history features and learns a decision tree that attempts to distinguish histories that result in different expected asymptotic return under optimal behavior. Wolfe & Barto (2006) applied a U-Tree-like algorithm but rather than restricting the model to predicting future rewards, they learn to make predictions about some pre-selected set of features of the next observation (a special case of the more general concept of tests of interest).

Note that by directly approximating ϕ, these methods have abandoned the generative property of a complete model (or an abstract model). That is, these types of models *only* make predictions for $\mathcal{T}^I$. These predictions *depend* upon the values of some set of features, but the model does not generally *predict* the future values of those features. Directly learning ϕ effectively decouples what predictions the model makes from what information it uses to make those predictions.

Though this type of approach has demonstrated promise, it also faces a clear pragmatic challenge: feature selection. In the non-Markov case, ϕ is a function of history, an ever-expanding sequence of actions and observations. Finding a reasonable set of compactly represented features that collectively capture all of the history information needed to make the predictions of interest is a significant challenge. In this sense directly learning ϕ through regression, unlike learning a generative model, takes only a small step away from the Markov case. Though the approach explicitly addresses partially observable environments it still requires a good idea *a priori* of what information should be extracted from history in order to make the predictions of interest.

The method presented in this chapter possesses the main strength of the direct

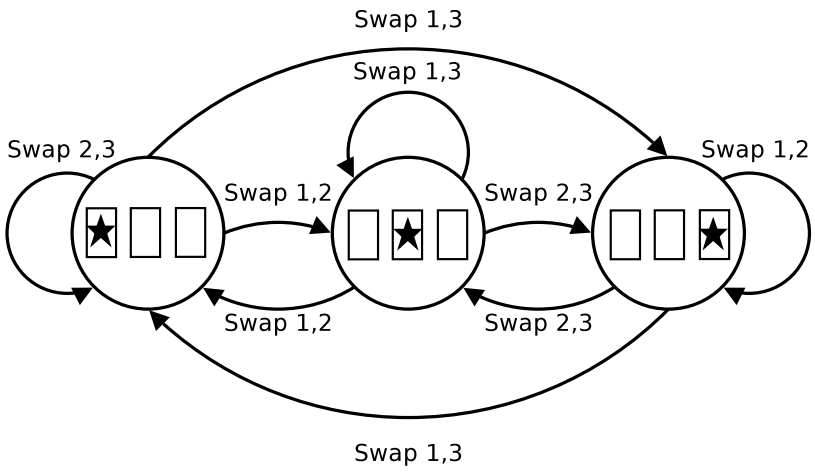


Figure 3.1: The prediction profile system for Three Card Monte. Transitions are labeled with the dealer's swaps. States are labeled with the predicted position of the special card.

function approximation approach (in that the model will *only* make the predictions of interest) and yet retains the main strength of generative model learning methods (learning to maintain state, rather than assuming it is given in the form of some history features).

3.1.2 Prediction Profile Models

Recall that, in the Three Card Monte example, all that is necessary for good performance is to keep track of the special card's position. One can do so using a prediction profile model, as shown in Figure 3.1. Each state in the diagram is labeled with a prediction about the location of the special card. Each transition is labeled with an observation of the dealer's behavior (which cards it swaps). Given the predictions in the current history and an observation, one can use this diagram to obtain the predictions associated with the next history. Quite intuitively, if the special card is predicted to be in position 1 and the agent observes the dealer swapping cards 1 and 2, then the updated prediction will be that the special card is in position 2. If the dealer swaps cards 1 and 3, then the new predicted position is 3. If the dealer swaps cards 2 and 3, then the prediction does not change. As such, predictions about the

location of the special card can be maintained, conditioning on observations of the dealer's behavior but not predicting what swaps the dealer will make in the future. Another example of a prediction profile model is shown in Figure 2.12 in Section 2.5.

While generative models learn to make predictions about future observations conditioned on the agent's action choices, the main idea behind prediction profile models is to learn to make predictions about the *future values of the predictions of interest*, conditioned on both the agent's action choices *and* the observations emitted by the environment. Because they model how the predictions themselves change over time, prediction profile models are able to take any history and provide the predictions of interest at that history. On the other hand, they cannot make any other predictions.

3.2 The Prediction Profile System

The results in this chapter make the restrictive assumption that, as in the Markov case, there is a finite number of distinct prediction profiles (that is, the predictions of interest take on only a finite number of distinct values). Certainly this is not true of all partially observable systems and all sets of tests of interest, though it is true in many interesting examples. Formally, this assumption requires that ϕ map all histories to a finite set of prediction profiles:

Assumption 3.2. *Assume there exists a finite set* $P = \{\rho_1, \rho_2, ..., \rho_k\} \subset [0,1]^m$ *such that for every history* h, $\phi(h) \in P$.

This assumption allows the definition of the *prediction profile system* (or PP for short), as a discrete dynamical system that captures the sequence of prediction profiles over time, given an action observation sequence. The prediction profile system's actions, observations, and dynamics are defined in terms of quantities associated with the original system:

Definition 3.3. The *prediction profile system* is defined by a set of observations, a set of actions, and a rule governing its dynamics.

1. Observations: The set of prediction profile observations, $\mathcal{O}_{PP}$, is defined to be the set of distinct prediction profiles. That is, $\mathcal{O}_{PP} \stackrel{\text{def}}{=} P = \{\rho_1, ..., \rho_k\}$.

2. Actions: The set of prediction profile actions, $\mathcal{A}_{PP}$, is defined to be the set of action-observation pairs in the original system. That is, $\mathcal{A}_{PP} \stackrel{\text{def}}{=} \mathcal{A} \times \mathcal{O}$.

3. Dynamics: The dynamics of the prediction profile system are deterministically governed by ϕ. At any prediction profile history, $\langle a^1, o^1\rangle\rho^1\langle a^2, o^2\rangle\rho^2...\langle a^j, o^j\rangle\rho^j$, and for any next PP-action, $\langle a^{j+1}, o^{j+1}\rangle$, the prediction profile system deterministically emits the PP-observation $\phi(a^1o^1a^2o^2...a^jo^ja^{j+1}o^{j+1})$.

The following are some important facts about the prediction profile system. Specifically, it will be demonstrated that the prediction profile system is always deterministic. Also, though the prediction profile system may be Markov (as it is in the Three Card Monte example), in general it is partially observable.

Proposition 3.4. *Even if the original system is stochastic, the prediction profile system is always deterministic.*

Proof. This follows immediately from the definition: every history corresponds to exactly one prediction profile. So a PP-history (action-observation-profile sequence) and a PP-action (action-observation pair) fully determine the next PP-observation (prediction profile). The stochastic observations in the original system have been folded into the un-modeled *actions* of the prediction profile system. □

Proposition 3.5. *If the original system is Markov, the prediction profile system is Markov.*

Proof. By definition, if the original system is Markov the prediction profile at any time step depends only on the most recent observation. So, if at time step t, the

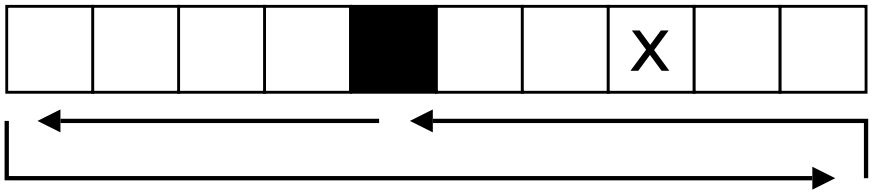


Figure 3.2: Size 10 1D Ball Bounce

current profile is ρ_t, the agent takes action a_{t+1} and observes observation o_{t+1}, the next profile is simply $\rho_{t+1} = \phi_{Markov}(o_{t+1})$. So, in fact, when the original system is Markov, the prediction profile system satisfies an even stronger condition: the next PP-observation is *fully* determined by the PP-action and has no dependence on history whatsoever (including the most recent PP-observation). □

Proposition 3.6. *Even if the original system is partially observable, the prediction profile system may be Markov.*

Proof. Consider the Three Card Monte example. The original system is clearly non-Markov (the most recent observation, that is the dealer's most recent swap, tells one very little about the location of the special card). However, the prediction profile system for the tests of interest regarding the location of the special card (pictured in Figure 3.1) *is* Markov. The next profile is fully determined by the current profile and the PP-action. □

In general, however, the PP system may be partially observable. Though in the Three Card Monte example the current prediction profile and the next action-observation pair together fully determine the next prediction profile, in general the next prediction profile is determined by the *history* of action-observation pairs (and prediction profiles).

Proposition 3.7. *The prediction profile system may be partially observable.*

Proof. Recall the 1D Ball Bounce example from Section 2.5.1, reproduced here in Figure 3.2. There is a line of pixels and a ball bounces back and forth across them,

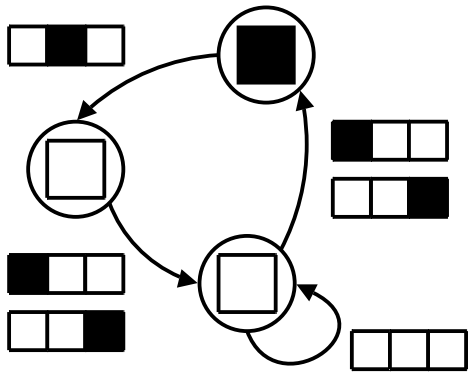


Figure 3.3: A transition diagram for the predictions of interest

moving one pixel in its current direction in each time-step and changing direction when it enters an edge pixel. The goal is to predict whether the ball will enter the pixel marked with an x in the next time step. Figure 3.3 shows the corresponding prediction profile system (where the square in each state indicates what color pixel x will be and only the neighborhood surrounding x is shown on the transitions). Note that two distinct states in the update graph are associated with the *same* prediction profile (pixel x will be white). Given only the current prediction profile (pixel x will be white) and the PP-action (the ball enters the neighborhood on the left or right), one cannot determine whether the ball is entering or leaving the neighborhood, and thus cannot uniquely determine the next profile. This prediction profile system is partially observable. □

So, in general, the prediction profile system is a deterministic, partially-observable dynamical system. A model of the prediction profile system can *only* be used to make the predictions of interest. As such, if one wishes to use a prediction profile model as a *generative* model, one must select the tests of interest carefully. For instance:

Proposition 3.8. *If the tests of interest include the set of one-step primitive tests, that is* $\{ao : a \in \mathcal{A}, o \in \mathcal{O}\} \subseteq \mathcal{T}^I$*, then a model of the prediction profile system can be used as a generative model of the original system.*

Proof. It was shown in Section 1.1.1 that any conditional prediction about the future

given history can be recursively computed using the predictions of one-step tests. Since the prediction profile model in this case provides all one-step predictions in all histories, it can be used to compute *any* prediction. It can also be used to "simulate the world" (sample possible futures) as would a generative model. At any given history and for any given action, the current prediction profile provides the distribution over next observations. As such, one can sample a next observation and use it to update the model, which gives the next profile, which can be used to sample the next observation, and so on. □

While in this special case a prediction profile model can be a complete, generative model of the system, it will be shown in Section 3.4 that if one desires a generative model, it is essentially never preferable to learn a prediction profile model over a more traditional model. A prediction profile model is *best* applied when it is relatively simple to make and maintain the predictions of interest in comparison to making *all* predictions. In general, a prediction profile model conditions on the observations, but it does not necessarily *predict* the next observation. As such, a model of the prediction profile model cannot typically be used for the purposes of model-based planning/control like a generative model could. The experiments in Section 3.6 will demonstrate that the output of prediction profile models *can*, however, be useful for model-free control methods.

The next section discusses the problem of learning a prediction profile model from data. Subsequent sections will discuss the complexity of the prediction profile model, particularly in comparison to the complexity of the original system (and relatedly when it is preferable to use a prediction profile model over a more typical generative model).

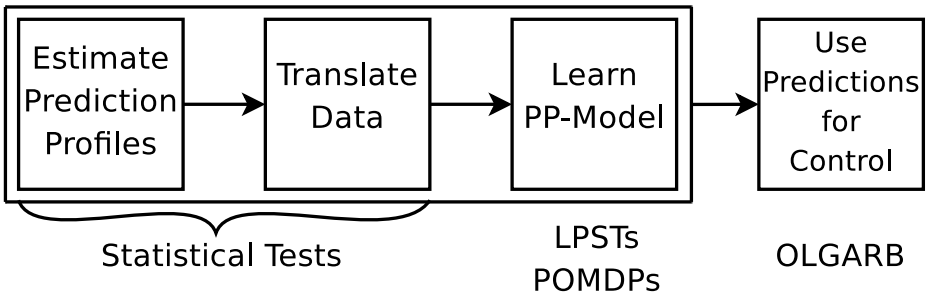


Figure 3.4: Flow of the algorithm.

3.3 Learning a Prediction Profile Model

Let S be a training data set of trajectories of experience with the original system (action-observation sequences) and let $\mathcal{T}^I = \{t_1, t_2, ..., t_k\}$ be the set of tests of interest. The main contribution of this section is a method for learning a model of the prediction profile system. The learning algorithm has three main steps (pictured in Figure 3.4). First the training data is used to estimate the prediction profiles (both the number of unique profiles and their values). Next, the learned set of prediction profiles is used to translate the training data into trajectories of experience with the prediction profile system. Finally, any applicable model learning method is used on the transformed data to learn a model of the prediction profile system. Ultimately the learned prediction profile models will be evaluated by how useful their predictions are as features for control.

3.3.1 Estimating the Prediction Profiles

Given S and $\mathcal{T}^I$, the first step of learning a prediction profile model is to determine how many distinct prediction profiles there are, as well as their values. The estimated prediction for a test of interest t at a history h is:

$$\hat{p}(t|h) = \frac{\#\text{ times } t \text{ succeeds from } h}{\#\text{ times } acts(t) \text{ taken from } h}. \tag{3.3}$$

One could, at this point, directly estimate ϕ by letting the estimated prediction profile at a history h be the vector of estimated predictions for the tests of interest, $\hat{\phi}(h) \stackrel{\text{def}}{=} \langle \hat{p}(t_1|h), \hat{p}(t_2|h), ..., \hat{p}(t_k|h) \rangle = \hat{\rho}$. Of course, due to sampling error, it is unlikely that *any* of these estimated profiles will be exactly the same, even if the true underlying prediction profiles are equal. So, to estimate the number of distinct underlying profiles, statistical tests will be used to find histories that have *significantly* different prediction profiles.

To compare the profiles of two histories, a likelihood-ratio test of homogeneity is performed on the counts for each test of interest in the two histories. If the statistical test associated with any test of interest rejects the null hypothesis that the prediction is the same in both histories, then the two histories have different prediction profiles.

In order to find the set of distinct prediction profiles, an initially empty set of exemplar histories is maintained. The algorithm searches over all histories in the agent's experience, comparing each history's estimated profile to the exemplar histories' estimated profiles. If the candidate history's profile is significantly different from the profiles of *all* exemplar histories, the candidate is added as a new exemplar. In the end, the estimated profiles corresponding to the exemplar histories are used as the set of prediction profiles. In order to obtain the best estimates possible, the search is ordered to prioritize histories with lots of data.

The prediction profile estimation procedure has two main sources of complexity. The first is the sample complexity of estimating the prediction profiles. It can take a great deal of exploration to see each history enough times to obtain good statistics, especially as the number of actions and observations increases. This issue could be addressed by adding generalization to the estimation procedure, so that data from one sample trajectory could improve the estimates of many similar histories. In one of the experiments in Section 3.6, observation abstraction will be employed as a simple form of generalization. The second bottleneck is the computational complexity of

searching for prediction profiles, as this involves exhaustively enumerating all histories in the agent's experience. It would be valuable to develop heuristics to identify the histories most likely to provide new profiles. In the experiments in Section 3.6, a simple heuristic of limiting the search to short histories is employed, as long histories will tend to have less associated data, and will therefore be less likely to provide distinguishably new profiles.

3.3.1.1 Alternative Prediction Clusterings

It is worth noting that this process of finding a finite number of distinct predictions is, at its core, simply a clustering of the estimated prediction profiles. The choice to use statistical tests to determine which prediction profiles are distinct has some advantages, most notably that it automatically combats overfitting by taking into account the amount of data used to obtain the estimates when determining whether two estimated profiles are different. However, one could imagine using other clustering methods.

In fact, as mentioned in Section 2.2.4, it will not even always be the case that obtaining high-fidelity estimates of the prediction profiles is the right goal. One might know *a priori* that estimates within some error-bound are all that are needed, or perhaps where the predictions fall in some discretization of probability space, or maybe only some qualitative feature of the profile is needed, such as which test is most likely. The main ideas behind prediction profile models can easily be adapted to these cases and the learning algorithm presented here can, in principle, be applied using essentially any clustering method that assigns discrete labels to estimated prediction profiles. Again, because it is such a basic, general-purpose goal, the remaining development will continue to focus on obtaining accurate estimates of the predictions of interest.

3.3.2 Generating Prediction Profile Trajectories

Having generated a finite set of distinct prediction profiles, the next step is to translate the agent's experience into sequences of action-observation pairs and prediction profiles. These trajectories will be used to train a model of the prediction profile system.

The process of translating a raw action-observation sequence s into a prediction profile trajectory s' is straightforward and, apart from a few practical concerns, follows directly from Definition 3.3. Recall that, for an action-observation sequence $s = a_1o_1a_2o_2...a_ko_k$, the corresponding PP-action sequence for the prediction profile system is simply $\langle a_1, o_1\rangle\langle a_2, o_2\rangle...\langle a_k, o_k\rangle$. The corresponding sequence of profiles is $\phi(a_1o_1)\phi(a_1o_1a_2o_2)...\phi(a_1o_1...a_ko_k)$. Thus, in principle, every primitive action-observation sequence can be translated into an action-observation-profile sequence.

Of course ϕ is not available to generate the sequence of prediction profiles. So, it is necessary to use an approximation $\hat{\phi}$, generated from the training data. Specifically, the estimated predictions for the tests of interest at each history h (computed using Equation 3.3) are compared, using statistical tests, to the estimated prediction profiles from Section 3.3.1. If there is only one estimated profile $\hat{\rho}$ that is not statistically significantly different from the estimated predictions at h, then let $\hat{\phi}(h) = \hat{\rho}$.

Given sufficient data, the statistical tests will uniquely identify the correct match with high probability. In practice, however, some histories will not have very much associated data. It is possible in such a case for the test of homogeneity to fail to reject the null hypothesis for two or more profiles. This indicates that there is not enough data to distinguish between multiple possible matches. This is especially common when some profiles have similar values, or when a particular history is experienced only a small number of times. In the experiments in Section 3.6, two different strategies for handling this situation are employed. The first strategy lets $\hat{\phi}(h)$ be the matching profile that has the smallest empirical KL-Divergence from the estimated

predictions (summed over all tests of interest). This is a heuristic choice that may lead to noise in the prediction profile labeling, which could in turn affect the accuracy of the learned model. The second strategy is to simply cut off any trajectory at the point where multiple matches occur, rather than risk assigning an incorrect labeling. This ensures that labels only appear in the prediction profile trajectories if there is a reasonable level of confidence in their correctness, but on the other hand, it is also quite wasteful to throw out training data that might contain useful information.

3.3.3 Learning a Prediction Profile Model

The translation step produces a set S' of trajectories of interaction with the prediction profile system. Recall that the prediction profile system is a deterministic, partially observable, discrete dynamical system and these trajectories can be used to train a model of the prediction profile system using, in principle, any applicable model-learning method.

There is one important issue faced by models of the prediction profile system that is not present in the usual discrete dynamical systems modeling setting. While the prediction profile labels are present in the training data, when actually *using* the model in the wild, of course the labels are not available. Say the current history is h, and an action a_1 is taken and an observation o_1 is emitted, which together constitute a PP-action. Being a model of the prediction profile system, a prediction profile model can provide the predictions for the next profile: $p(\langle a_1, o_1 \rangle \rho_i | h)$ for every profile ρ_i. Ideally these predictions will deterministically identify the next profile, ρ, since the underlying prediction profile system is deterministic. These predictions about profiles can be used to actually make predictions $p(t|ha_1o_1)$ for the *tests of interest* at the history ha_1o_1. Now another action a_2 and observation o_2 occur. It is now necessary to update the model's state in order to obtain the next prediction profile. A typical dynamical systems model makes predictions about the next observation,

but is then able to update its state with the *actual* observation that occurred. A prediction profile model's observations are prediction profiles themselves, which are not observable when interacting with the world. As such, the prediction profile model will update its state with the PP-action taken ($\langle a_1, o_2 \rangle$) and the prediction profile *it itself predicted* (ρ). Once it is updated, the prediction profile model can make the predictions $p(\langle a_2, o_2 \rangle \rho_i | h \langle a_1, o_1 \rangle \rho)$, which give the predictions for the tests of interest at the new history $ha_1o_1a_2o_2$.

If the prediction profile model is a *perfect* model of the prediction profile system, this is acceptable. Because the prediction profile system is deterministic, there is no need for the environment to emit the true prediction profile label, because it is fully determined by the history. In practice, of course, the model will be imperfect and different modeling representations will require different considerations when performing the two functions of providing predictions for the tests of interest, and providing a profile for the sake of updating the model.

3.3.3.1 PP-POMDPs

Since the prediction profile system is partially observable it is natural to model it using a POMDP. Unfortunately, even when the training data is from a deterministic system, POMDP training using the EM algorithm will generally not return a deterministic POMDP. Thus, at any given history, a learned POMDP model of the prediction profile system (PP-POMDP) will provide a *distribution* over prediction profiles instead of deterministically providing the one profile associated with that history. The implementation used in Section 3.6, simply takes the *most likely* profile from the distribution to be the profile associated with the history and uses it to make predictions for the tests of interest, as well as to update the POMDP model.

3.3.3.2 PP-LPSTs

Another natural choice of representation for a prediction profile model is a looping predictive suffix tree (LPST) (Holmes & Isbell 2006). Because LPSTs are specialized to deterministic partially observable systems they could not be used to model the original system (which is assumed to be stochastic in general), but they do apply to the prediction profile system, and they do not have to be determinized like a POMDP.

Briefly, an LPST captures the parts of recent history relevant to predicting the next observation. Every node in the tree corresponds to an action-observation pair. A node can either have children, or it can loop to one of its ancestors. Every leaf of the tree corresponds to a history suffix that has a deterministic prediction of an observation for every action. In order to predict the next observation from a particular history, one reads the history in reverse order, following the corresponding links on the tree until a leaf is reached, which gives the prediction. Holmes & Isbell provide a learning algorithm that, under certain conditions on the training data, is guaranteed to produce an optimal tree. The reader is referred to Holmes & Isbell (2006) for details.

A weakness of LPSTs, however, is that they fail to make a prediction for the next observation if the current history does not lead to a leaf node in the tree (or if the leaf node reached does not have a prediction for the action being queried). This typically occurs when some history suffixes do not occur in the training data but do occur while using the model. For a PP-LPST, this can mean that in some histories the model cannot uniquely determine the corresponding prediction profile. When this happens the implementation used in Section 3.6 simply finds the longest suffix of the current history that *does* occur in the data. This suffix will be associated with multiple prediction profiles (otherwise the LPST would have provided a prediction). To make predictions for the tests of interest, the model provides the average prediction over this set of profiles. The profile used to update the model is picked out of the set

uniformly randomly.

3.4 Complexity of the Prediction Profile System

This section will present some results characterizing the complexity of the prediction profile system. This will give us some indication of how difficult it is to learn a prediction profile model and thereby provide insight into when it is appropriate to apply a prediction profile model over a more typical generative model approach.

There are many factors that affect the complexity of learning a model. This section will largely focus on linear dimension as the measure of complexity, taking the view that, generally speaking, systems with smaller linear dimension are easier to learn than systems with larger linear dimension. As discussed in Section 1.1.2, this is generally true for POMDPs, for instance, where the linear dimension lower-bounds the number of hidden states. So comparing the linear dimension of the prediction profile system to that of the original system can give some idea of whether it would be easier to learn a PP-POMDP or just to learn a complete POMDP of the original system. Of course, there are other model-learning methods for which other complexity measures would be more appropriate (for instance it is not known how LPSTs interact with linear dimension). Extending some of these results to other measures of complexity may be an interesting topic of future investigation.

3.4.1 Linear Dimension Comparison

This section will discuss how the linear dimension of the prediction profile system relates to that of the original system. The first result is a "proof of concept" that simply states that there exist problems in which the prediction profile system is vastly more simple than the original system. In fact, such a problem has already been presented in the Three Card Monte example.

Proposition 3.9. *The prediction profile system can have linear dimension that is arbitrarily smaller than that of the original system.*

Proof. Recall the Three Card Monte example. Thus far the domain has been described without specifically describing the dealer's behavior. However, note that the prediction profile system for the tests of interest relating to the location of the special card (pictured in Figure 3.1) has a linear dimension of 3, *regardless of how the dealer's swaps are chosen.* If a very complex dealer is chosen, the original system will have high linear dimension, but the prediction profile system's linear dimension will remain constant. For instance, in the experiments in Section 3.6, the dealer chooses which cards to swap stochastically, but is more likely to choose the swap that has been selected the least often so far. Thus, in order to predict the dealer's next decision, one must keep track of how many times each swap has been chosen in history and as a result the system effectively has infinite linear dimension. □

On the other hand, prediction profile models are not a panacea. The following results indicate that there are problems for which learning a prediction profile model would not be advisable over learning a standard generative model, in that the linear dimension of the prediction profile system can be far greater than that of the original system. Later in the section some special cases will be characterized where prediction profile models are likely to be useful. The next result shows that the linear dimension of the prediction profile model can be infinite when the original system has finite linear dimension, via a lower bound on linear dimension that is true of all deterministic dynamical systems.

Proposition 3.10. *For any deterministic dynamical system with actions $\mathcal{A}$, and observations $\mathcal{O}$, the linear dimension, $n \geq \frac{\log(|\mathcal{A}|-1)+\log(|\mathcal{O}|+1)}{\log|\mathcal{A}|}$.*

Proof. See Appendix C. □

Because Proposition 3.10 applies to all deterministic dynamical systems, it certainly applies to the prediction profile system. Though it is a very loose bound, the basic implication is that as the number of prediction profiles (the observations of PP) increases in comparison to the number of action-observation pairs (the actions of PP), the linear dimension of the prediction profile system necessarily increases. This bound also clearly illustrates the importance of the assumption that there are a finite number of distinct prediction profiles.

Corollary 3.11. *If there are infinitely many distinct prediction profiles, the prediction profile system has infinite linear dimension.*

Proof. Clearly $|\mathcal{A}_{PP}| = |\mathcal{A} \times \mathcal{O}|$ is finite so long as there are finitely many actions and observations. So, from the last result it follows immediately that as the number of distinct prediction profiles $|\mathcal{O}_{PP}|$ approaches infinity, then so does the linear dimension of the prediction profile system. □

Hence, so long as prediction profile models are represented using methods that rely heavily on a finite linear dimension, it is critical that there be finitely many prediction profiles. Note that this is not a fundamental barrier, but a side effect of the representational choice. Model learning methods whose complexity depends mainly on other aspects of the problem, rather than linear dimension may be able to effectively capture systems with infinitely many prediction profiles.

One conclusion to be drawn from the last few results is that knowing the linear dimension of the original system does not, in itself, necessarily say much about the complexity of the prediction profile system. The prediction profile system may be far simpler or far more complex than the original system. Thus it may be more informative to turn to other factors when trying to characterize the complexity of the prediction profile system.

3.4.2 Bounding the Complexity of The Prediction Profile System

The results in the previous section fail to take into account an obviously important aspect of the prediction profile system: the predictions it is asked to make. Some predictions of interest can be made very simply by keeping track of very little information. Other predictions will rely on a great deal of history information and will therefore require a more complex model. The next result identifies the "worst case" set of tests of interest for any system, that is the tests of interest whose corresponding prediction profile model has the highest linear dimension. Ultimately this section will present some (non-exhaustive) conditions under which the prediction profile system is likely to be simpler than the original system.

Proposition 3.12. *For any system, the set of tests whose corresponding prediction profile system has the highest linear dimension is Q, the set of core tests for that system (as described in Section 1.2.2.1).*

Proof. See Appendix C. □

With this worst case identified, one can immediately obtain bounds on how complex any prediction profile system can possibly be.

Corollary 3.13. *For any system and any set of tests of interest, the corresponding prediction profile system has linear dimension no greater than the number of distinct predictive states for the original system.*

Proof. The prediction profile system for the set of core tests, Q is a deterministic MDP where the observations are prediction profiles for Q (that is, predictive states). The linear dimension of an MDP is never greater than the number of observations (Singh et al. 2004). Therefore, by the previous result the prediction profile system for any set of tests of interest can have linear dimension no greater than the number of predictive states. □

Corollary 3.14. *If the original system is a POMDP, the prediction profile system for any set of tests of interest has linear dimension no greater than the number of distinct belief states.*

Proof. This follows immediately from the previous result and the fact that the number of distinct predictive state is no greater than the number of distinct belief states (Littman et al. 2002). □

The bounds presented so far help explain why the prediction profile system can be more complex than the original system. However, because they are focused on the worst possible choice of tests of interest, they do little to illuminate when the opposite is true. A prediction profile model is at its most complex when it is asked to perform the same task as a generative model: keep track of as much information from history as is necessary to make *all possible predictions* (or equivalently, the predictive state or the belief state). These results indicate that, generally speaking, if one desires a generative model, standard approaches would be preferable to learning a prediction profile model.

On the other hand, the stated goal of this chapter is *not* to learn a generative model, but instead to focus on some particular predictions that will hopefully be far simpler to make than *all* predictions. Examples presented in this chapter make it clear that in some cases, some predictions can be made by a prediction profile model far more simple than a generative model of the original system. In general one might expect the prediction profile model to be simple when the predictions of interest rely on only a small amount of the state information required to maintain a generative model. The next bound aligns with this intuitive reasoning.

Essentially what this result will point out is that often much of the hidden state information in a POMDP will be irrelevant to the predictions of interest. The linear dimension of the prediction profile system is actually bounded only by the number of distinct beliefs over the *relevant* parts of the hidden state, rather than the number

of distinct beliefs states overall. The idea of the result is that if one can impose an abstraction over the *hidden states* of a POMDP (not the observations) that still allows the predictions of interest to be made accurately and that allows the abstract belief states to be computed accurately, then the prediction profile system's linear dimension is bounded by the number of *abstract* belief states.

Proposition 3.15. *Consider a POMDP with hidden states $\mathcal{S}$, actions $\mathcal{A}$, and observations $\mathcal{O}$. Let $\mathcal{T}^I$ be the set of tests of interest. Let a^i be the action taken at time-step i, s^i be the hidden state reached after taking action a^i, and o^i be the observation emitted by s^i. Now, consider* any *surjection $\sigma : \mathcal{S} \to \mathcal{S}^\sigma$ mapping hidden states to a set of* abstract states *with the following properties:*

1. *For any pair of primitive states $s_1, s_2 \in \mathcal{S}$, if $\sigma(s_1) = \sigma(s_2)$, then for any time-step i and any test of interest $t \in \mathcal{T}^I$, $p(t|s^i = s_1) = p(t|s^i = s_2)$.*

2. *For any pair of primitive states $s_1, s_2 \in \mathcal{S}$, if $\sigma(s_1) = \sigma(s_2)$, then for any time-step i, abstract state $S \in \mathcal{S}^\sigma$, observation $o \in \mathcal{O}$, and action $a \in \mathcal{A}$,*

$$Pr(\sigma(s^{i+1}) = S | s^i = s_1, a^{i+1} = a, o^{i+1} = o) = Pr(\sigma(s^{i+1}) = S | s^i = s_2, a^{i+1} = a, o^{i+1} = o).$$

If such a σ exists, then the prediction profile system for $\mathcal{T}^I$ has linear dimension no greater than the number of distinct beliefs over abstract states, $\mathcal{S}^\sigma$.

Proof. See Appendix C. □

There are a few things to note about this result. First, a surjection σ *always* exists that has properties 1 and 2. One can always define $\sigma : \mathcal{S} \to \mathcal{S}$ with $\sigma(s) \stackrel{\text{def}}{=} s$. This degenerate case trivially satisfies the requirements of Proposition 3.15 and recovers the bound given in Corollary 3.14. However, Proposition 3.15 applies to *all* surjections

that satisfy the conditions. There must be a surjection that satisfies the conditions *and* results in the smallest number of beliefs over abstract states. Essentially, this is the one that ignores as much state information as possible while still allowing the predictions of interest to be make accurately and it is *this* surjection that most tightly bounds the complexity of the prediction profile system (even if σ is not known).

Of course, there may still be a large or even infinite number of distinct beliefs, even over abstract states, so other factors must come into play to ensure a simple prediction profile system. Furthermore, this result does not characterize all settings in which the prediction profile system will be simple. That said, this result does support the intuition that the prediction profile system will tend to be simple when the predictions it is asked to make depend on small amounts of state information.

In order to build intuition about how this result relates to earlier examples, recall the Three Card Monte problem. In Three Card Monte there are two sources of hidden state: the special card's unobserved position and whatever hidden mechanism the dealer uses to make its decisions. Clearly the agent's predictions of interest depend only on the first part of the hidden state. So, in this case one can satisfy Property 1 with a surjection σ that maps two hidden states to the same abstract state if the special card is in the same position, regardless of the dealer's state. Under this σ there are only 3 abstract states (one for each possible position), even though there might be infinitely many true hidden states. Now, different states corresponding to the same special card position will have different distributions over the special card's next position; this distribution does, after all, depend upon the dealer's state. However, Property 2 is a statement about the distribution over the next abstract state *given* the observation that is emitted after entering the abstract state. If one knows the current abstract state and *observes* what the dealer does, the next abstract state is fully determined. So Property 2 holds as well. In fact, since the special card's position is known at the beginning of the game, this means the current abstract

state is always known with absolute certainty, even though beliefs about the dealer's state will in general be stochastic. Hence, there are only 3 distinct beliefs about the abstract states (one for each state). As such, the prediction profile model's linear dimension is upper-bounded by 3, regardless of the dealer's complexity (and in this case the bound is tight).

3.4.3 Bounding the Number of Prediction Profiles

The previous section described some conditions under which the prediction profile system may be simpler than the original system in terms of linear dimension. Also of concern is the number of prediction profiles, and particularly whether that number is finite. This section will briefly discuss some (non-exhaustive) special cases in which the number of prediction profiles can be bounded.

One case that has already been discussed is when the original system is Markov. In that case the number of prediction profiles is bounded by the number of observations (states). Of course, when the original system is Markov, there is little need to use prediction profile models. Another, similar case is when the system is partially observable, but completely deterministic (that is, the next observation is completely determined by history and the selected action). If the system is, for instance, a deterministic POMDP, then at any given history, the current hidden state is known. As such, the number of belief states is bounded by the number of hidden states. Since there obviously cannot be more prediction profiles than belief states, the number of prediction profiles are bounded as well.

One can move away from determinism in a few different ways. First, note that the important aspect of a deterministic system is really that the hidden state is fully determined by history. It is possible to satisfy this property even in stochastic systems, so long as one can uniquely determine the hidden state, *given* the observation that was emitted when arriving there. In that case, observations can be emitted stochastically,

but the number of belief states (and therefore the number of prediction profiles) is still bounded by the number of hidden states.

Another step away from determinism is a class of systems, introduced by Littman (1996), called Det-POMDPs. A Det-POMDP is a POMDP where the transition function and the observation function are both deterministic, but the initial state distribution may be stochastic. A Det-POMDP is *not* a deterministic dynamical system, as there is uncertainty about the hidden state. Because of this uncertainty regarding the initial state, the system appears to emit observations stochastically. It is only the *underlying dynamics* that are deterministic. Littman showed that a Det-POMDP with n hidden states and an initial state distribution with m states in its support has at most $(n+1)^m - 1$ distinct belief states. So, this bounds the number of prediction profiles as well.

Finally, and most importantly, if the hidden state can be abstracted as in Proposition 3.15, then these properties only really need to hold for *abstract beliefs*. That is, the environment itself may be complex and stochastic in arbitrary ways, but if the *abstract* hidden state described in Proposition 3.15 is fully determined by history, then the number of prediction profiles is bounded by the number of abstract states. Similarly, Det-POMDP-like properties can be imagined for abstract hidden states as well.

These cases by no means cover all situations where the number of prediction profiles can be bounded, but they do seem to indicate that the class of problems where the number of prediction profiles is finite is quite broad, and may contain many interesting examples.

3.5 Related Work

The idea of modeling only some aspects of the observations of a dynamical system has been raised before. In some recent examples, Rudary (2008) and Wolfe (2010)

both learn models that split the observation into two pieces, one of which is modeled while the other is treated as an action, or an "exogenous input." Such a model makes *all* conditional predictions about the modeled portion of the observation, given the exogenous inputs (as well as actual actions and the history). A prediction profile model is a slightly different idea. Instead of predicting future sequences of some piece of the next observation conditioned on another piece, prediction profile models predict the values of an arbitrary set of predictions of interest at the next time step, given the action and observation. This allows significantly more freedom in choosing which predictions the model will make.

One modeling method closely related to prediction profiles is Causal State Splitting Reconstruction (CSSR) (Shalizi & Klinker 2004). CSSR is an algorithm for learning generative models of discrete, partially observable, uncontrolled dynamical systems. The basic idea is to define an equivalence relation over histories where two histories are considered equivalent if they are associated with identical distributions over possible futures. The equivalence classes under this relation are called *causal states*. The CSSR algorithm learns the number of causal states, the distribution over next observations associated with each causal state, and the transitions from one causal state to the next, given an observation. It is straightforward to see that there is a one-to-one correspondance between causal states and the predictive states of a PSR. As such, a causal state model is precisely the prediction profile model where the set of tests of interest is Q, the set of core tests. With this correspondance in hand, the results in Section 3.4.2 show that in many cases the number of causal states will *greatly* exceed the linear dimension of the original system and that therefore CSSR may be inadvisable in many problems, in comparison to more standard modeling approaches. It is possible that the CSSR algorithm could be adapted to the more general setting of arbitrary sets of tests of interest, however the algorithm does rely heavily on the fact that a prediction profile model with Q as the tests of interest is

Markov, which is not generally the case for other sets of tests of interest.

The prediction profile system is also similar in spirit to finite state controllers for POMDPs. Sondik (1978) noted that in some cases, it is possible to represent the optimal policy for a POMDP as a finite state machine. These finite state controllers are very much like prediction profile models in that they take action-observation pairs as inputs, but instead of outputting predictions associated with the current history, they output the optimal action to take. Multiple authors (e.g. Hansen 1998, Poupart & Boutilier 2003) provide techniques for learning finite state controllers. However, these algorithms typically require access to a complete POMDP model of the world to begin with which, in the setting of this thesis, is assumed to be impractical.

3.6 Experiments

This section will empirically evaluate the prediction profile model learning procedure developed in Section 3.3. In each experiment an agent faces an environment that would be challenging to model completely due to its high linear dimension. However, in each problem the agent could make good decisions if it could only have the predictions to a small number of important tests. A prediction profile model is learned for these important tests and the accuracy of the learned predictions is evaluated.

These experiments also demonstrate one possible use of prediction profile models (and partial models in general) for control. Because they are not generative, prediction profile models cannot be used directly by model-based planning methods. However, their output may be useful for model-free methods of control. Specifically, in these experiments, the predictions made by the learned prediction profile models are provided as features to a *policy gradient* algorithm.

3.6.1 Predictive Features for Policy Gradient

Policy gradient methods (e.g. Williams 1992, Baxter & Bartlett 2000, Peters & Schaal 2008) have been very successful as a viable option for model-free control in partially observable domains. Though there are differences between various algorithms, the common thread is that they assume a parametric form for the agent's policy and then attempt to alter those parameters in the direction of the gradient with respect to expected average reward. These experiments will make use of Online GPOMDP with Average Reward Baseline (Weaver & Tao 2001), or OLGARB (readers are referred to the original paper for details). OLGARB assumes there is some set of features of history, and that the agent's policy takes the parametric form:

$$\Pr(a|h;\vec{w}) = \frac{e^{\sum_i w_{i,a} f_i(h)}}{\sum_{a'} e^{\sum_i w_{i,a'} f_i(h)}}$$

where $f_i(h)$ is the ith feature and each parameter $w_{i,a}$ is a weight specific to the feature *and* the action being considered.

Typically the features used in policy gradient are features that can be directly read from history (e.g. features of the most recent few observations or the presence/absence of some event in history). It can be difficult to know *a priori* which historical features will be important for making good control decisions. In contrast, the idea in these experiments is to provide the values of some *predictions* as features. These *predictive features* have direct consequences for control, as they provide information about the effects of possible behaviors the agent might engage in. As such, it may be easier to select a set of predictive features that are likely to be informative about the optimal action to take (e.g. "Will the agent reach the goal state when it takes this action?" or "Will taking this action damage the agent?"). Furthermore, information may be expressed compactly in terms of a prediction that would be complex to specify purely in terms of past observations. As seen in the discussion of PSRs in Section

1.2.2.1, an arbitrary-length history can be fully captured by a finite set of short-term predictions. For these reasons it seems reasonable to speculate that predictive features, as maintained by a prediction profile model, may be particularly valuable to model-free control methods like policy gradient.

3.6.2 Experimental Setup

The learning algorithm will be applied to two example problems. In each problem prediction profile models are learned (using both LPSTs and POMDPs as the representation and using both strategies for dealing with multiple matches, as described in Section 3.3.3) with various amounts of training data. The prediction accuracy of the models is evaluated, as well as how useful their predictions are as features for control. The training data is generated by executing a uniform random policy in the environment.

The free parameter of the learning algorithm is the significance value of the statistical tests, α. Given the large number of contingency tests that will be performed on the same data set, which can compound the probability of a false negative, α should be set fairly low. In these experiments $\alpha = 0.00001$, though several reasonable values were tried with similar results. As discussed in Section 3.3, there will also be a maximum length of histories to consider during the search for prediction profiles. This cutoff allows the search to avoid considering long histories, as there are many long histories to search over and they are unlikely to provide new prediction profiles.

After a prediction profile model is learned, its predictions are evaluated as features for the policy gradient algorithm OLGARB. Specifically, for each test of interest t the unit interval is split up into 10 equally-sized bins b and a binary feature $f_{t,b}$ is provided that is 1 if the prediction of t lies in bin b, and 0 otherwise. Also provided are binary features f_o, for each possible observation o. The feature $f_o = 1$ if o is the most recent observation and 0, otherwise. The parameters of OLGARB, the learning

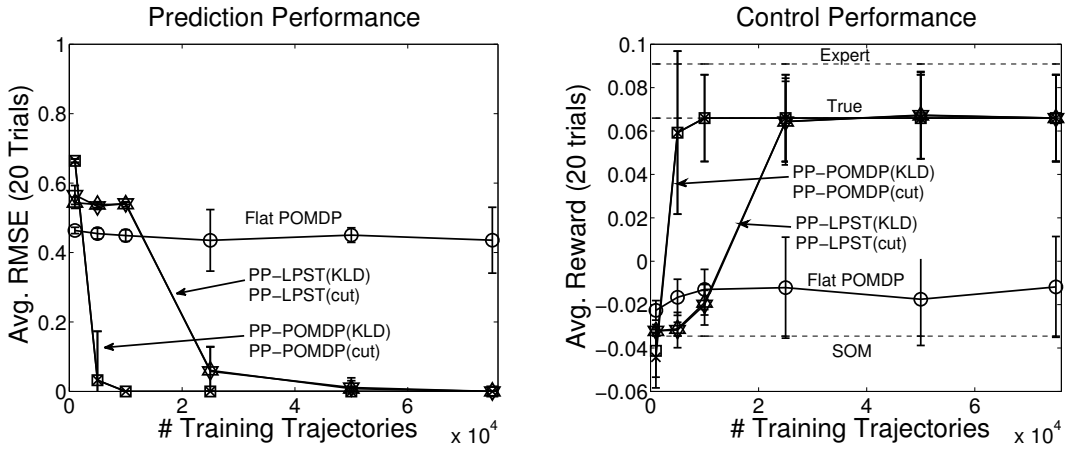


Figure 3.5: Results in the Three Card Monte domain.

rate and discount factor, are set to 0.01 and 0.95, respectively in all experiments.

To evaluate a prediction profile model OLGARB is run for 1,000,000 steps. The average reward obtained and the prediction error for the tests of interest the model accrued along the way are reported. Prediction performance is compared to that obtained by learning a POMDP on the training data and using it to make the predictions of interest. Because these problems are too complex to feasibly train a POMDP with the correct number of underlying states, 30-state POMDPs were used (stopping EM after a maximum of 50 iterations)[1]. Control performance is compared to that obtained by OLGARB using the predictions provided by a learned POMDP model as features, as well as OLGARB using the true predictions as features (the best the prediction profile model could hope to do), OLGARB using second-order Markov features (the two most recent observations, as well as the action between them) but no predictive features at all, and a hand-coded expert policy.

3.6.3 Three Card Monte

The first domain is the Three Card Monte example discussed earlier in the chapter. The agent is presented with three cards. Initially, the card in the middle (card 2) is

[1]Similar results were obtained with 5, 10, 15, 20, and 25 states.

the "special card." The agent has four actions available to it: *watch*, $flip1$, $flip2$, and $flip3$. If the agent chooses a flip action, it observes whether the card it flipped over is the special card. If the agent chooses the *watch* action, the dealer can swap the positions of two cards, in which case the agent observes which two cards were swapped, or the dealer can ask for a guess. If the dealer has not asked for a guess, then *watch* results in 0 reward and any flip action results in -1 reward. If the dealer asks for a guess and the agent flips over the special card, the agent gets reward of 1. If the agent flips over one of the other two cards, or doesn't flip a card (by selecting *watch*), it gets reward of -1. The agent has three tests of interest, and they take the form $flipX$ *special*, for each card X (that is, "If I flip card X, will I see the special card?").

As discussed previously, the complexity of this system is directly related to the complexity of the dealer's decision-making process. In this experiment, when the agent chooses "watch" the dealer swaps the pair of cards it has swapped the least so far with probability 0.5; with probability 0.4 it chooses uniformly amongst the other pairs of cards; otherwise it asks for a guess. Since the dealer is keeping a count of how many times each swap was made, the process governing its dynamics effectively has an infinite linear dimension. The prediction profile system, on the other hand, has only 3 states, regardless of the dealer's complexity (see Figure 3.1).

Training trajectories were of length 10. Figure 3.5 shows the results for various amounts of training data, averaged over 20 trials. Both PP-POMDPs and PP-LPSTs learned to make accurate predictions for the tests of interest, eventually achieving zero prediction error. In this case, PP-POMDPs did so using less data. This is likely because a POMDP model is more readily able than a LPST model to take advantage of the fact that the prediction profile system for Three Card Monte is Markov. As expected, the standard POMDP model was unable to accurately predict the tests of interest.

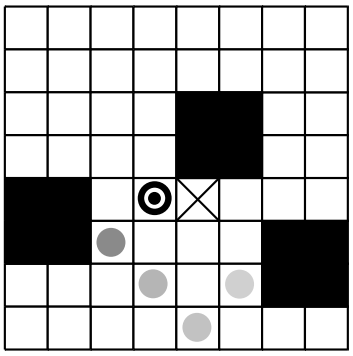


Figure 3.6: The Shooting Gallery domain.

Also compared are the two different strategies for dealing with multiple matches discussed in Section 3.3.3. Recall that the first one (marked "KLD" in the graph) picks the matching profile with the smallest empirical KL-Divergence from the estimated predictions. The second (marked "cut" in the graph) simply cuts off the trajectory at the point of a multiple match to avoid any incorrect labels. In this problem these two strategies result in almost exactly the same performance. This is likely because the profiles in 3 Card Monte are deterministic, and are therefore quite easy to distinguish (making multiple matches unlikely). The next experiment will have stochastic profiles.

The predictive features provided by the prediction profile models are clearly useful for control, as the control performance of OLGARB using their predictions approaches, and eventually exactly matches that of OLGARB using the true predictions (marked "True"). The inaccurate predictions provided by the POMDP were not very useful for control; OLGARB using the POMDP provided predictions does not even break even, meaning it loses the game more often than it wins. The POMDP features did, however, seem to contain some useful information beyond that provided by the second-order Markov features (marked "SOM") which, as one might expect, performed very poorly.

3.6.4 Shooting Gallery

The second example is called the Shooting Gallery, pictured in Figure 3.6. The agent has a gun aimed at a fixed position on an 8×8 grid (marked by the X) . A target moves diagonally, bouncing off of the edges and 2×2 obstacles (an example trajectory is pictured). The agent's task is to shoot the target. The agent has two actions: *watch* and *shoot*. When the agent chooses *watch*, it gets 0 reward. If the agent chooses *shoot* and the target is in the crosshairs in the time-step *after* the agent shoots, the agent gets reward of 10, otherwise it gets a reward of -5. Whenever the agent hits the target, the shooting range resets: the agent receives a special "reset" observation, each 2×2 square on the range is made an obstacle with probability 0.1, and the target is placed in a random position. There is also a 0.01 probability that the range will reset at every time step. The difficulty is that the target is "sticky." Every time step with probability 0.7 it moves in its current direction, but with probability 0.3 it sticks in place. Thus, looking only at recent history, the agent may not be able to determine the target's current direction. The agent needs to know the probability that the target will be in its sights in the next step, so clearly the single test of interest is: *watch target* (that is "If I choose the *watch* action, will the *target* enter the crosshairs?").

This problem has stochastic prediction profiles, so it is expected that more data will be required to differentiate them. Also, due to the number of possible configurations of obstacles and positions of the target, this system has roughly 4,000,000 observations and even more latent states. This results in a large number of possible histories, each with only a small probability of occurring. As discussed in Section 3.3, this can lead to a large sample complexity for obtaining good estimates of prediction profiles. Here this is addressed with a simple form of generalization: observation abstraction. Two observations are treated as the same if the target is in the same position and if the configuration of obstacles in the immediate vicinity of the tar-

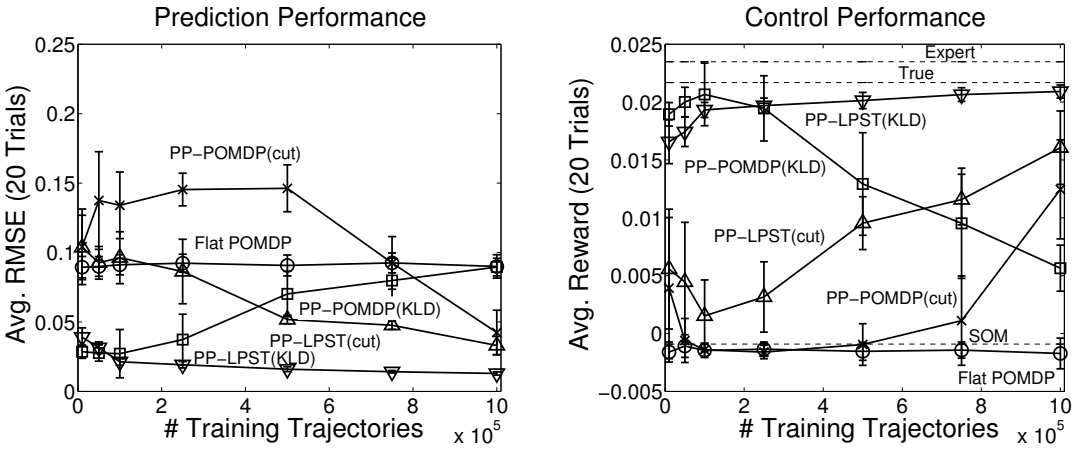


Figure 3.7: Results in the Shooting Gallery domain.

get is the same. Even with this abstraction, there are over 2000 action-observation pairs. This abstraction is accurate, so two histories have the same prediction profile if they have the same action sequence and their observation sequences correspond to the same sequence of aggregate observations. This enables one sample trajectory to improve the estimates for several histories. The same observation abstraction was applied when training the POMDP model.

Note that this demonstrates the *complimentary* relationship between the methods presented in Chapter 2 and prediction profile models. Though one can apply these ideas independently of one another, they can be effectively combined. Using both methods together to learn a non-generative model that ignores irrelevant details in the observations can simplify the learning problem even further than applying either method alone would.

Training trajectories were length 4 and the search for profiles was restricted to length 3 histories. Results are shown in Figure 3.7. Perhaps the most eye-catching feature of the results is the *upward* trending curve in the prediction error graph, corresponding to the PP-POMDP with the KL-Divergence based matching (labeled "PP-POMDP(KLD)"). Recall that the danger of the KL-divergence based matching strategy is that it may produce incorrect labels in the training data. Apparently these

errors were severe enough in this problem to drastically mislead the POMDP model. With a small amount of data it obtained very good prediction error, but with more data came more misleading labelings, and the performance suffered. The PP-POMDP trained with the other matching method ("PP-POMDP(cut)") displays a more typical learning curve (more data results in better error), though it takes a great deal of data before it begins to make reasonable predictions. This is because cutting off trajectories that have multiple matches throws away data that might have been informative to the model. The PP-LPSTs generally outperform the PP-POMDPs in this problem. With the trajectory cutting method, the PP-LPST ("PP-LPST(cut)") quickly outperforms the flat POMDP and, with enough data, outperforms both versions of PP-POMDP. The PP-LPST with the KL-divergence based matching ("PP-LPST(KLD)") is by far the best performer, quickly achieving small prediction error. Clearly the incorrect labels in the training data did not have as dramatic an effect on the LPST learning, possibly because, as a suffix tree, an LPST mostly makes its predictions based on recent steps in history, limiting the effects of labeling errors to a few time-steps.

Control performance essentially mirrors prediction performance, with some interesting exceptions. Note that even though PP-POMDP(KLD) obtains roughly the same prediction error as the flat POMDP at 1,000,000 training trajectories, the predictive features it provides still result in substantially better control performance. This indicates that, even though the PP-POMDP is making errors in the exact values of the predictions, it still has captured more the important *dynamics* of the predictions than the flat POMDP has. The flat POMDP itself provides features that are roughly as useful as second-order Markov features, which do not result in good performance. Again, OLGARB using these features does not break even, meaning it is wasting bullets on times when the target is not likely to enter the crosshairs. The best-performing prediction profile model, PP-LPST(KLD) approaches the performance of OLGARB using the true predictions with sufficient data.

3.7 Scaling Prediction Profile Models (Future Directions)

While the experiments in Section 3.6 demonstrate that it is possible to learn prediction profile models in contrived systems too complex for POMDPs, the specific learning algorithm presented in this chapter is not likely to scale to more natural domains without modification. The most critical scaling issues for prediction profile models are the sample complexity of estimating the prediction profiles, and the computational complexity of searching for prediction profiles and translating the data. In both cases, the critical source of complexity is essentially how many distinct histories there are in the training data (more distinct histories means the data is spread thin amongst them and there are more estimated profiles to search through). As such, observation abstraction is a key tool for combatting these issues because it lumps histories together. That said, observation abstraction can only generalize across histories of the same length. Because the number of distinct histories is exponential in the length of histories, it would be far more effective to be able to generalize across histories of different lengths as well. The concept of histories of interest (introduced in the next chapter) will provide one possible avenue to incorporate this type of *temporal abstraction*, and experiments in Chapter 5 will empirically demonstrate its benefits. That said, these results will rely heavily on a human designer to specify which histories have similar predictions and which are more "interesting."

So, one of the main ways the applicability of prediction profile models can be expanded is by developing methods that induce a temporal abstraction as part of the learning process. For instance, one idea would be to assume that from time-step to time-step, predictions mostly stay the same, and only occasionally change values (this is the case in a few of the examples seen in this thesis). In that case, one might attempt to discover the circumstances under which predictions change, and otherwise lump consecutive histories together, assuming they have the same prediction profiles. Another approach might start with a very simple representation for the prediction

profile model (for instance a state-based model with only a few states or a depth-limited suffix tree) and then progressively allow the representation to become more sophisticated, as is warranted by the data. This way, if a very simple model is sufficient, learning will be very fast (whereas using the current algorithm, it may take a large number of samples to get good enough estimates just to learn that the prediction profile model is very simple). This idea is similar in spirit to the state-splitting approach of Shalizi & Klinker (2004) and the incremental suffix tree building approach of McCallum (1995).

Another limitation of the prediction profile model learning method presented here is its reliance on the assumption of a finite number of prediction profiles. While this assumption does hold in many cases, an ideal method would be able to deal gracefully with a very large or infinite number of prediction profiles. One possibility is to simply cluster the predictions in other ways. For instance, one may only desire a certain level of prediction accuracy and may therefore be willing to lump some distinct prediction profiles together in exchange for a simpler prediction profile system. Another idea would be to learn a prediction profile model using continuous-valued representations (e.g. Kalman 1960, Rudary et al. 2005), which explicitly deal with systems with an infinite number of observations (prediction profiles in this case). While it is extremely unlikely that the prediction profile system is linear, there do exist methods for learning non-linear models of partially observable systems as well (e.g. Julier & Uhlmann 1997, Wingate 2008). Even when there are finitely many prediction profiles, methods for learning non-linear continuous models may still be able to (approximately) capture the discrete dynamics.

3.8 Summary

Key points from this chapter:

- Prediction profile models are *non-generative* partial models
  - A prediction profile models makes *only* the predictions of interest and no others
  - The main idea is to learn a model of the dynamics of the *predictions themselves*, rather than the dynamics of the system.
  - Prediction profile models currently rely on the assumption that the predictions of interest take on only a finite number of distinct values.
- A prediction profile model can be far simpler than a complete, generative model, and it can be far more complex.
  - A prediction profile model is best applied when the predictions of interest require relatively little state information to make (in comparison to *all* predictions).
- A procedure for learning a prediction profile model was presented that has three main steps:
  - First, estimate the number of distinct prediction profiles and their values (using statistical tests to determine which estimated predictions are significantly different).
  - Second, translate the training data into sequences of experience with the prediction profile system by assigning a profile to each history.
  - Finally, learn a model using the transformed training data.
- Experiments showed that prediction profile models can be used to learn to make some particularly important predictions in systems too complex to be modeled using standard POMDPs.

– The predictions made by the learned models were demonstrated to be useful as features for model-free control

CHAPTER 4

Histories of Interest

The previous two chapters have focused on a particular form of partial model, demonstrating that a partial model restricted to making only a small set of predictions of interest can be much simpler (and correspondingly easier to learn) than a complete model. However, making only some predictions is not the only way for a model to be partial. Another important way in which a model's prediction responsibilities can be restricted is by limiting the situations in which the model can be used. Chapter 1, for instance, discussed update rules with pre-conditions and option models, both of which benefit from learning models that apply in some specific situations, but not in others. This chapter introduces the concept of *histories of interest*, which extends the concept of partial models used in Chapters 2 and 3 to allow models that only make predictions in some particular histories. Limiting a model's responsibilities in this way can even further simplify the learning task.

As in the previous two chapters, the general approach to learning partial models that only make predictions in some histories will be to transform the training data in such a way that the transformed model-learning task is simpler and the resulting model can still be used to make the predictions of interest at the histories of interest. Abstraction will play a large role in this process and, as will be seen, the abstraction learning algorithms developed in Chapter 2 will not apply in this new setting. Thus,

this chapter will describe an abstraction learning algorithm that is far more practical and scalable than those found in Chapter 2, though it will lack theoretical performance guarantees.

4.1 Histories of Interest

Similar to the set of *tests of interest* $\mathcal{T}^I$, which specifies what predictions a partial model should make, it is possible to define a set of *histories of interest* $\mathcal{H}^I \subseteq \mathcal{H}$, which specifies in which histories a partial model should make predictions. Specifically, a partial model is responsible for providing the predictions $p(t|h)$ for all tests of interest $t \in \mathcal{T}^I$ and histories of interest $h \in \mathcal{H}^I$, but *no other tests and no other histories*. The partial models discussed in the previous two chapters have been examples of the special case where *all* histories are of interest: $\mathcal{H}^I = \mathcal{H}$.

It will simplify the discussion in this chapter to place a constraint on $\mathcal{H}^I$, which, in analogy to a similar concept in the options literature (Sutton et al. 1999), will be called the *semi-Markov* property. Essentially, the semi-Markov property requires that in order to determine if a particular history is a history of interest, one need only look at what has happened since the *last* history of interest. Formally,

Definition 4.1. A set of histories of interest $\mathcal{H}^I$ is *semi-Markov* if and only if for any two histories of interest $h, h' \in \mathcal{H}^I \cup \{\emptyset\}$, if there exists some test t such that $ht \in \mathcal{H}^I$, then either $h't \in \mathcal{H}^I$ or $p(h't|\emptyset) = 0$ (that is, $h't \notin \mathcal{H}$).

A model that makes predictions in only some histories may be useful simply because predictions are only needed in some restricted situations. However, a far more common idea is to have several partial models such that *some* partial model applies at every history and can be used to make predictions for the tests of interest. This partitioning of the possible histories can lead to individual partial models that are simpler than one that applies in all histories, but that collectively cover all situations.

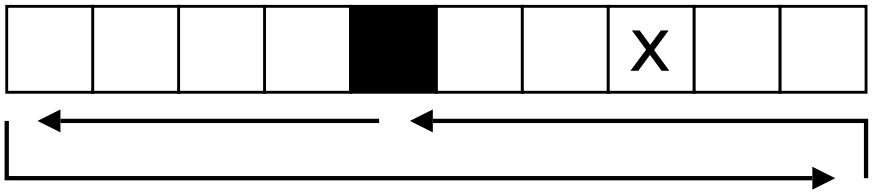


Figure 4.1: Size 10 1D Ball Bounce

This idea of switching between partial models represents one way in which multiple partial models can be combined to form a more complete model. Chapter 5 will discuss "collections of partial models" more generally. This chapter will continue the focus on how to learn a single partial model with its own set of tests of interest and now with its own set of histories of interest.

Limiting when a partial model makes predictions can significantly simply its learning problem. For instance, recall the 1D Ball Bounce example from Section 2.5.1, reproduced in Figure 4.1. In that example there was a partial model in charge of predicting whether a bouncing ball would be in position x in the next time-step. This model can accurately make this prediction if it only pays attention to the pixels in the immediate neighborhood of position x and ignores the rest of the observation. The problem, of course, was that this abstraction did not actually simplify the model; the abstract model had the same complexity as a model of the primitive observations. This is, as discussed in Chapter 3, because a generative model of the abstract system makes *all* abstract predictions, including a prediction about whether the ball will enter the neighborhood of position x in the next time-step. This extraneous prediction requires the model to keep track of the ball's position and direction when the ball is outside of the neighborhood, even though this information is irrelevant for making the prediction of interest: "Will be the ball be in position x or not in the next time step?" Prediction profile models in Chapter 3 offered a solution to this problem by making *only* the predictions of interest and no others. Histories of interest offer an alternative approach in this domain.

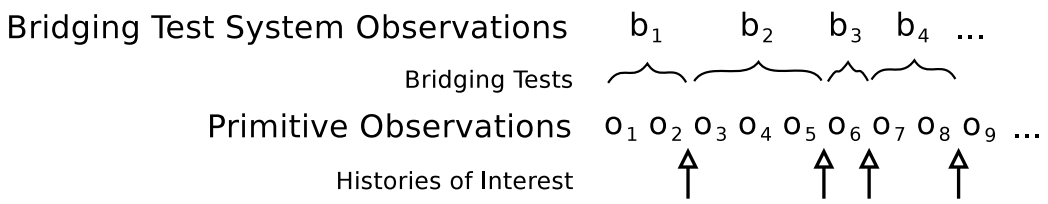


Figure 4.2: The bridging test system

Consider a similar partial model that must predict whether the ball will be in position x in the next time step but now it is only required to make that prediction in histories that end with the ball in the immediate neighborhood of x. Intuitively, its job is significantly easier. Because it is only required to make predictions when the ball is already in the neighborhood, the model clearly has no need to pay attention to the ball's position when it is outside the neighborhood. To make an accurate prediction when the ball is already in the neighborhood, the model need only keep track of the ball's current direction.

Note that, as in this example, limiting a partial model to making predictions only in some histories of interest can simplify the abstract system, making it even easier to learn a generative, abstract model. Histories of interest can *also* be helpful if one intends to learn a prediction profile model, rather than a generative model, but in a different way. This will be discussed later in the chapter. The next section will formalize the intuitive ideas illustrated by this example, demonstrating how restricting when it must make predictions can simplify a partial model.

4.2 The Bridging Test System

It will simplify the development in this chapter to limit the discussion to uncontrolled systems (where the agent has no available actions and can only observe the world's dynamics over time). Controlled systems will be discussed in Section 4.4.

As in the previous two chapters, the strategy here will be to learn a model of a transformation of the original system. In this case, the transformed model should

be able to make predictions at the histories of interest, but need not in any other histories. Intuitively, one way to accomplish this is for the model to only update its predictions at histories of interest. Essentially the model "wakes up" whenever a history of interest occurs, updates its predictions according to what has happened since the last history of interest, and then goes dormant again until the next history of interest. Sequences of observations that happen *between* histories of interest are called *bridging tests* (as illustrated in Figure 4.2).

It is straightforward to formally define the set of all bridging tests $\mathcal{T}^B$, as induced by the set of histories of interest:

Definition 4.2. A test $t \in \mathcal{T}$ is a *bridging test* if and only if

1. Either $\exists h \in \mathcal{H}^I$ such that $ht \in \mathcal{H}^I$ or t is of infinite length.

2. For any prefix t' of t, there is no history of interest $h \in \mathcal{H}^I$ such that $ht' \in \mathcal{H}^I$.

Note that as a consequence of the semi-Markov property, every history of interest can be uniquely decomposed into a sequence of bridging tests. Furthermore, every sequence of bridging tests is a history of interest (or has probability zero and is thus not a possible history at all). Lastly, because no bridging test is a prefix of any other bridging test, at any given history of interest there is a well-defined probability distribution over which bridging test will occur next (that is, some bridging test *must* occur and two bridging tests cannot simultaneously occur).

The *bridging test system*, then, is a conceptual dynamical system in which the *observations* are bridging tests. Every time-step in the bridging test system corresponds to a history of interest in the original system. The conditional probability distribution over the next "observation," given history in the bridging test system is precisely the conditional probability distribution over the next bridging tests given the corresponding history of interest in the original system.

Because the bridging test system only has time-steps at histories of interest, it can be much simpler than the original system. For instance, consider the 1D Ball Bounce system of size k. As discussed in Section 2.5.1, it has linear dimension $2k-2$ (because the state includes the ball's position and direction). Now let the histories of interest be histories that end with the ball in position x and consider the resulting bridging test system. The "observations," of course, are bridging tests: in this case all possible ways for the ball to leave position x and then come back to position x (it could go off to the left and come back, or go off to the right and come back). Note that in every time-step of the bridging test system, *the ball is in the same position* (position x). As such, which bridging test will occur next depends *only* on the ball's current direction. So, in this case, the bridging test system has linear dimension 2, regardless of k, as compared to the linear dimension of the original system, which grows linearly with k.

Not only can the bridging test system be far simpler than the original system, it is possible to show that it is *never* more complex than the original system.

Proposition 4.3. *If the linear dimension of a dynamical system is n then, given a semi-Markov set of histories of interest $\mathcal{H}^I$, the linear dimension of the induced bridging test system, $n_{BT} \leq n$.*

Proof. Recall that the linear dimension of a system is the rank of the system dynamics matrix. The matrix corresponding to the bridging test system is simply the submatrix of the system dynamics matrix of the original system with only the columns and rows corresponding to histories and tests that are sequences of bridging tests. A submatrix never has greater rank than the matrix that contains it, so the result follows immediately. □

In principle, a model of the bridging test system could be used to make *any* prediction, but only at the histories of interest. On the other hand, recall that the

observations of the bridging test system are *sequences* of observations in the original system. As such, even if the original system has a very small number of observations, the bridging test system will in general have infinitely many "observations," making it rarely (if ever) advantageous to directly learn a model of the bridging test system in practice.

That said, if the partial model has a restricted set of tests of interest *as well* as histories of interest then, just as it was possible to ignore much of the detail of the observations in Chapter 2, the model may be able to safely ignore much of the detail contained in the bridging tests and still make accurate predictions. For instance consider a variation on the 1D Ball Bounce system which behaves similarly, except now the ball's movement is noisy. At every time step it either moves in its current direction with probability 0.7 or it moves in the opposite direction with probability 0.3. Direction still changes deterministically on the end pixels. Now consider a partial model that is in charge of making predictions about whether the ball will be in position x in the next time step, but *only when* the ball is in the immediate neighborhood of x. There are infinitely many ways the ball could leave the neighborhood and return (because the ball is now performing a biased random walk). As a result, there are infinitely many bridging tests. However, all the model *really* needs to know about the ball's journey is whether it changed direction (whether it hit a side wall). Thus, an accurate abstraction could lump together all bridging tests where the ball leaves the neighborhood and hits, for instance, the left side, all the bridging tests where the ball hits the right side, and all the bridging tests where the ball hits neither side. Even though there are infinitely many bridging tests, there will be a small number of abstract observations.

So, as in Chapter 2, the main task becomes finding an accurate abstraction, only now the observation abstraction is applied to the *bridging test system*. That is, instead of abstracting single observations, the abstraction is now defined over bridging tests:

sequences of observations. Recall that the conceptual abstraction learning algorithm presented in Chapter 2 required pair-wise enumeration of all observations. Since the bridging test system can easily have infinitely many observations, this method does not apply in this setting. The next section describes an abstraction learning method that, while it lacks the theoretical guarantees of the algorithm in Chapter 2, is far more practical and can be used to find an accurate abstraction of the bridging test system. The results of applying this algorithm in example domains will be presented, but deferred to the next chapter, where abstractions will be learned in the context of learning many partial models and combining them into a *collection of partial models*.

4.3 Learning an Accurate Bridging Test Abstraction

As described in Chapter 2, an abstraction is a surjection η, now more generally defined over the set of bridging tests $\mathcal{T}^B$ rather than the set of observations $\mathcal{O}$. The abstraction maps bridging tests to abstract observations. The short-hand $\eta(h)$ where h is a history of interest will denote the *abstract history*, or the sequence of abstract observations obtained by applying η to each bridging sequence making up h. Note that, because η now maps sequences of observations to a single abstract observation, the abstract history $\eta(h)$ will, in general, be *shorter* than the primitive history h. The various properties of abstractions defined in Chapter 2 are easily transferred to this setting. Most notably an *accurate abstraction* is one such that every abstract history contains enough detail to make accurate predictions: for every $h \in \mathcal{H}^I$ and $t \in \mathcal{T}^I$, $p(t|\eta(h)) = p(t|h)$.

This section presents an incremental refinement procedure for learning an accurate abstraction, inspired by decision trees. It is similar to many existing abstraction learning algorithms, (e.g. Givan et al. 2003, Wolfe & Barto 2006) though for the most part these algorithms have only been applied in the Markov setting. The partially observable setting particularly exacerbates some scaling challenges which will need to

Algorithm 1 Abstraction Learning (basic flow)

$\eta = \eta_0$
repeat
  $R = proposeRefinements(\eta)$ {Adding each feature in $\mathcal{F}$}
  if $R \neq \emptyset$ **then**
    $\eta = \arg\max_{\eta' \in R} evaluateAbstraction(\eta')$
    repeat
      $C = proposeCoarsenings(\eta)$ {Merging each pair of abstract observations}
      if $C \neq \emptyset$ **then**
        $\eta = \arg\max_{\eta' \in C} evaluateAbstraction(\eta')$
      end if
    until $C = \emptyset$
  end if
until $R = \emptyset$

be addressed.

In addition to the histories of interest and tests of interest, the algorithm assumes as input a set $\mathcal{F}$ of potentially relevant features of bridging tests. It is from these features that the abstraction will be built. The algorithm begins with the coarsest possible abstraction η_0 that maps everything to the same abstract symbol[1] and iteratively refines the abstraction by incorporating features from $\mathcal{F}$.

The basic form of the algorithm is presented in Algorithm 1. At each iteration, a number of refinements are proposed: the cross product of each feature from $\mathcal{F}$ and the current abstraction. The refinement that most improves prediction accuracy is greedily selected, essentially adding a feature to the abstraction. Much like the algorithm provided by Givan et al. (2003), the algorithm alternates between refinement steps and coarsening steps, merging abstract observations when doing so does not impact accuracy. This keeps the number of abstract observations low and combats over-fitting. The coarsening step is itself iterative. In each iteration, each pair of abstract observations is considered for merging and, once again, the pair that retains the most accuracy is greedily selected.

[1] Alternatively, if an expressive abstraction is desired, the initial abstraction can be set to be any expressive abstraction, ideally the coarsest one.

Note that this greedy, incremental procedure is not guaranteed to find the coarsest accurate abstraction. In part, this is because $\mathcal{F}$ is not guaranteed to contain features that admit an accurate abstraction. However, even if the features in $\mathcal{F}$ *can* express an accurate abstraction, it is not guaranteed to be found. If, for instance, two features in $\mathcal{F}$ are jointly very informative for making the predictions of interest, but individually do not improve predictions, then they will never be added to the abstraction. This is an issue common to all similar greedy, incremental algorithms.

Though the basic idea of the algorithm is both simple and familiar, some of the details of the implementation used in later experiments are specific to learning an abstraction in a partially observable environment. The remainder of this section will describe and motivate some of these implementation details in the three main components of the algorithm: evaluating abstractions for accuracy, proposing and selecting refinements, and proposing and selecting coarsenings. Section 4.4 will discuss how the results in this chapter can be applied to controlled systems, and how they relate to prediction profile models.

4.3.1 Evaluating Accuracy

Because the algorithm greedily selects the "best" refinement at every iteration, it is critical to be able to evaluate a given abstraction's accuracy. The end goal, of course, is an abstraction η such that for any $t \in \mathcal{T}^I$ and any $h \in \mathcal{H}^I$ $p(t|h) = p(t|\eta(h))$. So one intuitive way to measure an abstraction's accuracy is to measure the difference between $p(t|h)$ and $p(t|\eta(h))$.

To simplify the discussion, assume that $\sum_{t \in \mathcal{T}^I} p(t|h) = 1$ for all h. That is, assume there is a well-defined probability distribution over the occurrence of the tests of interest. The algorithm described here can be adapted to handle more general sets of tests of interest. Let $\mathcal{H}_k$ be the set of length k histories. Let T be a random variable ranging over the tests of interest that could follow the current history. Then let $\epsilon_k(\eta)$

be the *inaccuracy* of η for length k histories and define it to be the expected KL Divergence:

$$\begin{aligned}\epsilon_k(\eta) &\stackrel{\text{def}}{=} \mathrm{E}_h\left[D_{KL}(T|h||T|\eta(h))\right] \\ &= \sum_{h\in\mathcal{H}_k} p(h|\emptyset) \sum_{t\in\mathcal{T}^I} p(t|h) \log\left(\frac{p(t|h)}{p(t|\eta(h))}\right).\end{aligned}$$

This measure is intuitive for several reasons. First, an abstraction η satisfies the accuracy property if and only if, for all k, $\epsilon_k(\eta) = 0$. It also penalizes abstractions more for inaccuracies in common histories than in rare histories. Finally, it is possible to show that no refinement can ever increase inaccuracy by this measure. So, ideally, ϵ could be computed to evaluate any candidate refinement, and pick the one that is the most accurate at every iteration. However, estimating ϵ is impractical at best. In a complex domain, it is unlikely that any individual history will be experienced many times, if even more than once. As a result, estimates of the necessary quantities $p(h|\emptyset)$ and $p(t|h)$ from data will be extremely noisy, because there will only be a small number of samples of the history h. To address this, consider an alternate measure $\xi_k(\eta) = \epsilon_k(\eta_0) - \epsilon_k(\eta)$. Note that the abstraction that minimizes ϵ_k maximizes ξ_k so while $\epsilon_k(\eta)$ was the *inaccuracy* of η, let $\xi_k(\eta)$ be the *accuracy*. Furthermore, ξ_k can be computed using only predictions involving *abstract* histories, as the following result demonstrates. Following the convention set in Chapter 2, let $\mathcal{H}_k^\eta$ be the set of abstract histories obtained by applying η to the set of length k histories $\mathcal{H}_k$: $\mathcal{H}_k^\eta = \{\eta(h) : h \in \mathcal{H}_k\}$.

Proposition 4.4. *For any abstraction η and any k,*

$$\begin{aligned}\xi_k(\eta) = \epsilon_k(\eta_0) - \epsilon_k(\eta) &= E_{H\in\mathcal{H}_k^\eta}[D(T|H||T|\eta_0(H))] \\ &= \sum_{H\in\mathcal{H}_k^\eta} p(H|\emptyset) \sum_{t\in\mathcal{T}^I} p(t|H) \log\left(\frac{p(t|H)}{p(t|\eta_0(H))}\right).\end{aligned}$$

Proof. See Appendix D. □

As a result of Proposition 4.4, ξ_k can be computed using quantities only involving *abstract* histories, rather than primitive histories. Abstract histories will tend to occur far more frequently than the corresponding primitive histories (since many primitive histories map to the same abstract history) and therefore it is more likely that enough samples will be available to obtain good estimates of the predictions involved in computing ξ_k.

Note that $\xi_k(\eta)$ is equivalent to the information gain of η with respect to the initial abstraction η_0. Information gain is frequently used in decision tree algorithms as a means of scoring possible expansions (notably the C4.5 algorithm by Quinlan 1992), though it is more typical to use the information gain from the current abstraction, rather than the initial one. However, this more common scoring function can only be used to compares refinements of the *current* abstraction. Using information gain with respect to η_0 ensures that *every* abstraction we consider is comparable on the same scale. This is important for the coarsening step, where candidates may not all be refinements of the most recent abstraction, but *are* guaranteed to be refinements of η_0.

In practice, an abstraction's accuracy is estimated using a finite set of training data and looking at only one particular length of history would be wasteful. To take histories of all lengths into account, the final accuracy measure will be

$$\xi(\eta) \stackrel{\text{def}}{=} \sum_{k \leq K} \xi_k(\eta),$$

where K is the maximum history length in the data.

4.3.2 Refinement step

In the refinement step, a number of candidate refinements of the current abstraction η are proposed and evaluated. Specifically, for every feature $f \in \mathcal{F}$, a new abstraction η_f is created whose abstract observations are the cross-product of the values of f and the values of the current abstraction. That is, for some bridging test b, $\eta_f(b) = \langle \eta(b), f(b) \rangle$. The proposed refinements are evaluated using ξ and the one with the highest accuracy is selected.

The main challenge in choosing a refinement is that, in the partially observable setting, the computational cost of evaluating candidate refinements is *much* higher than in the Markov case. Consider the estimate for $p(t|H)$ for some abstract history $H \in \mathcal{H}^\eta$:

$$\hat{p}(t|H) = \frac{\sum_{h \in H} \#\text{ times } t \text{ follows } h}{\sum_{h \in H} \#\text{ times } h \text{ happens}}.$$

Note that computing $\hat{p}(t|H)$ requires enumerating all primitive histories $h \in H$ that appeared in the data. Furthermore, since computing the estimate of ξ involves computing $\hat{p}(t|H)$ for all abstract histories H, ultimately it involves enumerating *all* primitive histories in the training data.

In contrast, in the Markov case, histories that end in the same observation are guaranteed to be associated with the same predictions. As such, one need only consider abstract observations, rather than abstract histories:

$$\hat{p}_{\text{Markov}}(t|O) = \frac{\sum_{o \in O} \#\text{ times } t \text{ follows } o}{\sum_{o \in O} \#\text{ times } o \text{ happens}}.$$

So, in the Markov case, where much of the work on abstraction learning has been done, evaluating the accuracy of a candidate abstraction is linear in the number of observations: $O(|\mathcal{O}|)$. In the partially observable setting, on which this thesis is

focused, evaluating a candidate abstraction is linear in the number of *histories* which, in the worst case is $O(|\mathcal{O}|^K)$ (where K, again, is the maximum history length found in the data), representing a serious scaling challenge.

One way to lower the cost of evaluating candidate refinements is to avoid performing the full computation of ξ. The strategy employed here is not to consider *every* abstract history when evaluating a refinement but instead to focus on abstract histories in most need of improvement, as measured by a heuristic. Specifically, consider the *improvement* in accuracy caused by combining the current abstraction η with feature f to produce the refinement η_f:

$$\xi(\eta_f) - \xi(\eta) = \sum_{k \leq K} \mathrm{E}_{H' \in \mathcal{H}_k^{\eta_f}}[D(T|H'||T|\eta_0(H')] - \mathrm{E}_{H \in \mathcal{H}_k^{\eta}}[D(T|H||T|\eta_0(H)].$$

Each abstract history under the current abstraction η contributes some amount to the improvement, essentially how much is gained by *just* refining *that* history. One can think of any particular abstract history H under the *current* abstraction as a set of abstract histories under the refined abstraction. In that case, the contribution of H to the improvement in accuracy can be written as

$$\delta_{\eta,\eta_f}(H) \stackrel{\text{def}}{=} \sum_{H' \in H} p(H'|\emptyset) D(T|H'||T|\eta_0(H')) - p(H|\emptyset) D(T|H||T|\eta_0(H)),$$

where H' denotes an abstract history under the refinement η_f.

By definition, $\sum_{k \leq K} \sum_{H \in \mathcal{H}_k^{\eta}} \delta_{\eta,\eta_f}(H) = \xi(\eta_f) - \xi(\eta)$. Furthermore, it is possible to place an easily computed bound on $\delta_{\eta,\eta_f}(H)$, giving an indication of which histories are in most need of refinement. For the remainder of this chapter, let Y be the symbol for entropy (instead of the more typical H) in order to avoid confusion with abstract histories H.

Proposition 4.5. *For any abstraction η, any refinement η', and any abstract history*

H under η, $\delta_{\eta,\eta'}(H) \leq p(H|\emptyset)\, Y(T|H)$ where $Y(T|H)$ is the conditional entropy *of the distribution over tests of interest, given the abstract history H.*

Proof. See Appendix D □

Proposition 4.5 is actually quite intuitive. It says that the biggest possible change in predictions is if all the randomness in the predictions were suddenly explained away. A coarse abstraction may ignore important details, making deterministic events seem random (for instance, a person who does not pay attention to federal holidays might think their trash collection service is sometimes randomly perturbed by a day). In that case, the best possible refinement is the one that completely explains the variation (the one that incorporates the dates of holidays). As such, the entropy of the predictions associated with an abstract history bounds how much those predictions *could conceivably* be improved by refining that history. Of course, in practice it may be impossible to obtain zero entropy, whether because the environment truly has stochastic events or (essentially equivalently) because the features available are not expressive enough to capture all the relevant details necessary to make the tests of interest deterministic. In that case this bound on the possible improvement will not be tight. That said, it is *always* true that very deterministic predictions will not be improved much by refining the abstraction and therefore cannot contribute much to the change in ξ.

So, when evaluating a candidate refinement, its accuracy is only measured with respect to the abstract histories in most need of improvement (those associated with the highest entropy in their predictions). In practice, a priority queue is maintained in order to sort abstract histories by their entropy. Each abstract history is associated with some number of primitive histories and it is the translation of these primitive histories using each proposed refinement that is the main computational bottleneck. As such, abstract histories are drawn from the priority queue until the number of primitive histories to translate reaches some threshold (in all experiments using this

algorithm, this threshold was set to 10,000). Let the set of selected high-entropy abstract histories be $\mathcal{H}^{\mathrm{Y}}$. The proposed refinements are then evaluated based on *just* on the accuracy of the predictions obtained when refining the histories in $\mathcal{H}^{\mathrm{Y}}$:

$$\xi_{\mathcal{H}^{\mathrm{Y}}}(\eta_f) \stackrel{\text{def}}{=} \sum_{H \in \mathcal{H}^{\mathrm{Y}}} \sum_{H' \in H} p(H'|\emptyset) \sum_{t \in \mathcal{T}^I} p(t|H') \left(log \left(\frac{p(t|H')}{p(t|\eta_0(H'))} \right) \right)$$

where the H' are that abstract histories under the refinement η_f that all map to H under the current abstraction η. The refinement with the highest $\xi_{\mathcal{H}^{\mathrm{Y}}}$ is selected. If no refinement makes any difference (that is, if $\xi_{\mathcal{H}^{\mathrm{Y}}}(\eta_f) = \xi_{\mathcal{H}^{\mathrm{Y}}}(\eta)$ for all f) then another set of abstract histories are drawn off of the priority queue and used to compute new evaluations. This continues until a refinement is picked or until the set of abstract histories is exhausted (indicating that no refinement improves the predictions in any abstract history).

Of course, picking the refinement that maximizes $\xi_{\mathcal{H}^{\mathrm{Y}}}$ is a heuristic and may result in the selection of different refinements than would have been chosen using the full ξ as the evaluation. Then again, recall that selecting the refinement with the maximum ξ was *already* a heuristic, not guaranteed to produce the optimal abstraction in the end. Furthermore, any feature that is added according to this criterion is still clearly informative, at least in some abstract histories, and refinement only stops if no feature helps in *any* history. As such, there is little reason to suspect that this heuristic would significantly harm the chances of learning a good abstraction. In exchange, the computational cost of evaluating refinements is now tunable (by setting the threshold on the number of primitive histories to re-translate), allowing for far larger sets of candidate features than would be feasible if ξ had to be fully computed for every proposal.

Apart from computational cost, there is another important concern requiring one last modification to the evaluation procedure: overfitting. Since ξ (or, more accu-

rately, $\xi_{\mathcal{H}^Y}$) is estimated empirically, it is possible for it to be artificially inflated by sampling error, especially if a refinement produces such detailed histories that each one appears very few times in the data. The predictions may appear to have changed a great deal, but in fact this could be because the *estimates* of the predictions associated with the refined histories are poor. In order to address this, the computation of $\xi_{\mathcal{H}^Y}$ is somewhat altered. Specifically, note that estimating $\xi_{\mathcal{H}^Y}$ requires estimating $p(t|H')$ for every test of interest $t \in \mathcal{T}^I$ and every abstract history under the proposed refinement η_f contained within $\mathcal{H}^Y$. To combat overfitting, the estimates of $p(t|H')$ are only used if they are *statistically significantly different* from the estimates of $p(t|H)$, the estimated predictions associated with the corresponding abstract history under the current abstraction η. For each H', if there is no significant difference (as measured by a likelihood ratio test), the estimates for $p(t|H')$ are replaced with the estimates for $p(t|H)$. As such, if adding a feature f into the abstraction does not make a *significant* difference in predictions, it will correspondingly not appear to increase accuracy over the current abstraction, and will therefore not be selected.

4.3.3 Coarsening step

After each refinement, an iterative coarsening step is performed that merges abstract observations if doing so does not affect accuracy. This step critically keeps the number of abstract observations low, which in turn means that there will be fewer abstract histories (reducing the computational cost of the next refinement step) and that each abstract history will have more associated data (improving the estimates of predictions). Finally, as seen in Chapter 2, a coarser abstraction typically results in a simpler model and should therefore be preferred.

In each iteration of the coarsening step, each pair of abstract observations is considered for merging. Ideally one would only merge observations that result in *no* change in accuracy. Of course, since accuracy is only empirically estimated, this is

overly restrictive. Instead, the proposed merging that does *not* cause a statistically significant change in predictions and that has the highest ξ is selected.

Here, the quadratic number of proposed coarsenings in each iteration is the main computational bottleneck. Unfortunately, the trick used in the refinement step of only evaluating abstractions on some histories cannot be applied here. If a coarsening were selected on the basis that it did not change accuracy on some subset of histories, it might be missed that it causes a large drop in accuracy in some other set of histories, undoing key progress. So, ξ must be computed in full to evaluate a coarsening.

Note that in the coarsening step, it can be cheaper to compute ξ than in the refinement step. When refining, the estimates of the predictions of the tests of interest given refined abstract histories $p(t|H')$ must be computed from the counts associated with primitive histories (as in Equation 5.1). In the coarsening step, they can be computed from the counts associated with the abstract histories being coarsened. That is, if one wishes to estimate $p(t|H)$ where H is an abstract history under the candidate *coarsened* abstraction, one need only compute:

$$\hat{p}(t|H) = \frac{\sum_{H' \in H} \#\text{ times } t \text{ follows } H'}{\sum_{H' \in H} \#\text{ times } H' \text{ happens}}.$$

where the histories denoted H' are abstract histories under the current abstraction being coarsened. Since there are typically many fewer abstract histories than primitive histories, computing ξ can be substantially cheaper in the coarsening step than in the refinement step. Nevertheless, computing it a quadratic number of times in each iteration is still a computational burden.

Instead of reducing the cost of evaluation, the strategy here is to reduce the *number of proposals* to be evaluated, based on the intuition that if merging two abstract observations causes a drop in accuracy in the current iteration, it will probably do so in future iterations as well. Any coarsenings that are found to cause a significant

difference in predictions in one iteration of the coarsening step are eliminated from consideration in future iterations. Such a coarsening is not proposed again until all other pairs are exhausted. Though it is in principle possible that some coarsening that would have been selected will have been removed from consideration in an earlier iteration, it will be re-considered *eventually*. So, once again, it is unlikely that this heuristic has a dramatic impact on the abstractions found by algorithm.

Finally, even with the statistical tests combating overfitting, the coarsening step was still found to occasionally undo too much of the progress made in the refinement step (causing refinement/coarsening loops). To combat this effect a threshold in the form of a percentage of the improvement in ξ made by the most recent refinement was set and only coarsenings that retained at least that much accuracy were performed. In all subsequent experiments that threshold is set to 0.9.

4.3.4 Complexity of Learning a Bridging Test Abstraction

This section will quickly summarize the complexity of applying the abstraction learning procedure just presented. At each iteration, the refinement step makes a number of proposals linear in the number of features. Each proposal can be evaluated in time linear in the number of histories being evaluated. This number is tunable, and can be as high as the number of distinct histories in the training data. The coarsening step is itself iterative. Each iteration proposes a number of coarsenings quadratic in the number of abstract observations. There cannot possibly be more iterations than there are abstract observations so in the very worst case (that will rarely, if ever, occur) the coarsening step performs a number of evaluations cubic in the number of abstract observations of the abstraction being evaluated. Each evaluation in the coarsening step is linear in the number of distinct histories in the training data.

The main factors affecting the computational complexity are:

1. The number of features necessary to construct an accurate abstraction (affects

the number of iterations)

2. The number of distinct histories in the training data (affects the cost of evaluating abstractions)

3. The number of abstract observations in the accurate abstraction (affects the number of proposals in the coarsening steps)

4. The number of features (affects the number of proposals in the refinement steps)

Thus, the algorithm is at its most efficient when a small number of very informative features are provided, when the abstraction being learned is very simple, and when the system itself has a small branching factor or when training episodes are very short (small number of distinct histories).

4.4 Controlled Systems and Prediction Profile Models

The discussion in this chapter has been limited to uncontrolled systems though it can, to some degree, be extended to address controlled systems as well. In a controlled system, a bridging test will be a sequence of observations *and* actions. One possibility is to use the machinery discussed in this chapter unchanged and still learn an abstraction that maps each bridging test to a single symbol. One could then treat those symbols as observations and learn an uncontrolled model. However, if one attempts to learn a generative model of this abstract system, it is important to note that this approach effectively folds the agent's own decisions into its observations. As discussed in Section 2.4, this can result in a drastic increase in model complexity as the model attempts to predict the agent's own behavior, as well as the world's dynamics.

To avoid this, it would be necessary to map the bridging test to both an abstract observation, that only contained information about the world's dynamics, and an

abstract action, that only contained information about the agent's decisions. The abstract actions would have to be constructed carefully to produce policy independent predictions. As seen in Section 2.4, this is difficult to achieve even when primitive actions are being abstracted, let alone *sequences* of actions. This thorny issue of finding an action abstraction that results in a model that makes accurate predictions for the tests of interest *and* makes policy independent abstract predictions will not be addressed in this thesis.

However, this does not mean abandoning hope of learning partial models that have histories of interest in controlled systems. The search for policy independence is only an issue when learning a *generative* model, that makes all abstract predictions. A prediction profile model, as described in Chapter 3, is *not* a generative model and *only* makes the predictions of interest. In fact, a prediction profile model deliberately folds together action and observation into a single symbol. As such, there is no need to map bridging tests into separate abstract actions and observations for the purposes of learning a prediction profile model. The algorithm discussed in Section 4.3 can be used to learn an accurate abstraction that maps bridging tests (now sequences of actions and observations) to single abstract symbols. Those symbols will be used as prediction profile actions, as discussed in Section 3.2. Experiments in Chapter 5 that involve histories of interest in controlled systems will use prediction profile models to represent partial models, thus side-stepping the action abstraction issue.

It is also worth noting that histories of interest and prediction profile models complement each other in other ways. The discussion in Section 4.2 centered on showing that having a restricted set of histories of interest reduces the linear dimension of the system being modeled, which is mainly relevant if one wishes to learn a generative model. Recall from Section 3.4 that the linear dimension of the system has little bearing *per se* on the complexity of a prediction profile model. That said, restricting when the model must make predictions simplifies the prediction profile model learn-

ing task in other ways. Most importantly, it introduces *temporal abstraction* to the model. Recall that one of the main performance bottlenecks for prediction profile models was the sample complexity of estimating prediction profiles themselves. The probability of seeing a particular history becomes progressively smaller and smaller as the length of the history grows. As such, it could take a large number of sampled trajectories from the world before enough experience was gained with a long history to get a reasonable estimate of its prediction profile. Observation abstraction can help with this issue by lumping some histories together but *bridging test abstraction* can have an even more dramatic effect. Collapsing entire sequences into single abstract observations can result in many more primitive histories being lumped together into the same abstract history, which will have correspondingly more associated data. Some of these intuitions will be seen empirically in the experimental section of the next chapter.

4.5 Scaling Abstraction Learning (Future Directions)

By far the tightest computational bottleneck in the abstraction learning procedure discussed in this chapter is the coarsening step. Recall that the coarsening step is critical, as learning an abstraction by only refining will rapidly spread the data thin across many overly-fine abstract histories, making it difficult to obtain good prediction estimates. In addition, the learned abstraction will make too many distinctions, likely resulting in a more complex abstract model. That said, in the worst case the coarsening procedure must evaluate a number of abstractions that is *cubic* in the number of abstract observations after the previous refinement step. Each one of those evaluations is linear in the number of distinct abstract histories. As such, this algorithm scales very poorly if the abstraction being learned has a large number of abstract observations (or if intermediate abstractions along the way do as well).

Recall that the main idea behind making the *refinement* step computationally

cheaper was to only evaluate candidate refinements using a subset of the histories in the data. The reason this strategy could not be straightforwardly applied to the coarsening step was the danger that a coarsening that doesn't reduce accuracy in some histories might undo critical refinement progress in other histories. One approach that might address this concern is to "lock" the progress made on some histories to prevent it from being undone. Essentially the idea would be to perform the *entire* abstraction procedure on a subset of the histories in the data: find the coarsest, most accurate abstraction the procedure could find for *just* those histories. Then, initializing with that learned abstraction, perform the procedure on a different subset of histories, ensuring that the coarsening steps never undo the distinctions made in the first round. This can repeat incrementally until all histories have been considered. Because only a subset of histories are used during evaluations, each iteration may be computationally cheaper. Furthermore, if the abstraction learned on one subset of the histories is useful for other histories, this procedure may find a good abstraction early on, alleviating the need to ultimately consider all histories. That said, it remains to be seen whether this incremental approach would lead to overly-fine abstractions. It will also rely upon good heuristics for selecting the history subsets used to evaluate the abstraction. This chapter has provided one such heuristic, though there is surely more investigation to be done in this space.

4.6 Summary

Key points from this chapter:

- This chapter extended the definition of partial models used in the previous two chapters to allow partial models to make predictions in only some situations.
  - A partial model is given a set of histories of interest $\mathcal{H}^I \subseteq \mathcal{H}$ which are the histories in which it must make its predictions.

  - The previous two chapters dealt with the special case where all histories are of interest (that is, $\mathcal{H}^I = \mathcal{H}$).

- Learning a partial model with limited histories of interest is done via the formalism of the *bridging test system.*
  - The bridging test system updates only at histories of interest.
  - The "observations" of the bridging test system are *sequences* of observations that occur between histories of interest (bridging tests).
  - A model of the bridging test system makes predictions only at histories of interest and can be far simpler than a model that makes predictions at all histories (and can be no more complex).
  - The bridging test system has infinitely many "observations" and thus abstraction is necessary (and the methods from Chapter 2 cannot be applied).

- An algorithm for learning an accurate bridging test abstraction was presented.
  - The algorithm (inspired by decision trees) incrementally and greedily refines the abstraction using a set of pre-defined features.
  - The problems of proposing and evaluating refinements are particularly challenging in the partially observable case.
    * Heuristic changes to the basic algorithm were presented that make it more practically applicable.
  - Experiments applying this method to example problems will be presented in the next chapter.

CHAPTER 5

Collections of Partial Models

The previous chapters have focused on the problem of learning a *single* partial model. The experiments in Section 3.6 demonstrated one possible use of a partial model: in some cases it is possible that the values of a small number of important predictions may be particularly informative for decision making. In these cases a partial model can be used to maintain the values of these important predictions and provide them as features to model-free control methods. Of course, in most interesting environments it will not be sufficient to make only a handful of predictions. Humans seem to have a model of their environment that can be used to make many predictions about a wide variety of phenomena. Humans can also use their models for generative planning, mentally trying out various courses of action and predicting possible outcomes. A single partial model would not likely suffice for these purposes.

This chapter will explore the intuitive idea of learning many partial models, each responsible for making predictions about some particular aspect of the agent's environment, and then combining their predictions to form a more complete model that can make detailed predictions and be used for planning purposes. As will be discussed in more detail, a collection of partial models is a structured representation that exploits conditional independence relationships between the predictions of its component models. This approach will primarily be contrasted with *dynamic Bayes nets*

(DBNs), a popular and well-studied structured representation that exploits conditional independence relationships between unobserved hidden variables. This chapter will also serve to bring together the methods discussed in previous chapters, learning bridging test abstractions and prediction profiles models to make up collections of partial models of high-dimensional arcade game examples, that are then used for model-based planning.

5.1 Collection of Partial Models (CPM)

A collection of partial models (CPM) consists of a set $\mathcal{M}$ of partial models. As discussed in Chapters 2 and 3, each partial model M has a set of *tests of interest*, $\mathcal{T}_M^I$ that determine *what* predictions the model will make and, as discussed in Chapter 4, a set of *histories of interest* $\mathcal{H}_M^I$ that determine *when* the model will be available to make predictions. For instance, in the recurring 1D Ball Bounce example, one might have partial models that each predict whether the ball will be in a particular position in the next time-step whenever the ball is nearby and other models that make the same prediction when the ball is not nearby. Intuitively one could imagine that, if there were such a pair of models for every position, this collection of partial models could be used as a complete, generative model, so long as the predictions of the individual models could be combined into a joint prediction. The next sections discuss the role of tests of interest and histories of interest in a CPM and then, most importantly, how to combine the predictions of the component models.

5.1.1 Tests of Interest

Though in principle the learning methods discussed in the previous chapters apply to a very general class of possible tests of interest, it will simplify the development of CPMs to constrain the tests of interest and focus on partial models that make abstract predictions about the next time-step. To that end, the predictions made by

each partial model in the collection are defined using an observation abstraction.

Definition 5.1. For a partial model $M \in \mathcal{M}$, let the surjection ω_M be the *abstraction of interest* that defines what predictions model M will make. Specifically, model M makes only one-step predictions about the abstract observations defined by ω_M. If the current time-step is k, and the *actual* observation that occurs at time $k+1$ is o_{k+1}, then call $\omega_M(o_{k+1})$ model M's *outcome*. So, each model M is only responsible for making predictions about its own outcome.

In the 1D Ball Bounce example, for a model in charge of predicting whether the ball will be in some position x, there are two *outcomes* (either pixel x is black or pixel x is white) and ω_M maps all observations appropriately to one of those two outcomes. The model provides a conditional probability distribution over these two outcomes, given history and a selected action.

In a CPM, multiple models make predictions simultaneously. The outcomes predicted by the different models capture different (abstract) aspects of the next observation (for instance, each might predict the color of a single pixel). If multiple partial models are making abstract predictions about the next observation at the same time, then, together, they may offer a more detailed prediction.

Definition 5.2. Consider a set of partial models $\{M_1, M_2, ..., M_j\}$. At the kth time-step, the corresponding *joint outcome* is a function of $k+1$th observation: $\omega_{M_1 ... M_j}(o_{k+1}) \stackrel{\text{def}}{=} \omega_{M_1}(o_{k+1}) \cap \omega_{M_2}(o_{k+1}) \cap ... \cap \omega_{M_j}(o_{k+1})$.

In the 1D Ball Bounce example, each model is responsible for predicting the color of its associated pixel. So, each model has two possible outcomes, one for when its pixel is black and one for when it is white. The *joint outcomes* for all the models are *full images* with the color of each pixel specified (so for a size 10 1D Ball Bounce there are 10 such joint outcomes, one for each possible image). A joint outcome for some

subset of the models would be a partial image with the colors of only some pixels specified.

5.1.2 Histories of Interest

Each partial model in the collection is required to make predictions only at certain histories $\mathcal{H}_M^I \subseteq \mathcal{H}$. This means that at any *particular* history, only some *subset* of the models in $\mathcal{M}$ will be able to make predictions. At history h, let $\mathcal{M}_h = \{M : h \in \mathcal{H}_M^I\}$ be the set of *active* models, those that are available to make predictions.

In the 1D Ball Bounce example, one might have models that are active when the ball is nearby their assigned position, and those that are active when the ball is not nearby. The set of active models changes over time, depending on the position of the ball.

At any given history, the predictions made by the active models determine the predictions that can be made by the CPM. Specifically, the CPM can be used to make predictions about the joint outcomes of the active models, written $\omega_{\mathcal{M}_h}(o)$ (for a given *next* observation o). Recall that any model that can make all primitive one-step predictions in all histories is a complete model that can be used to make *any* prediction. This leads straightforwardly to a condition on a CPM that, when satisfied, allow it to be used as a complete model.

Proposition 5.3. *A CPM comprises a complete model if for every history $h \in \mathcal{H}$ and every primitive observation $o \in \mathcal{O}$, the corresponding joint outcome of the active models $\omega_{\mathcal{M}_h}(o) = \bigcap_{M \in \mathcal{M}_h} \omega_M(o) = \{o\}$.*

For this condition to hold, at every history and for every distinct pair of possible next primitive observations, there must be *some* active model whose abstraction of interest distinguishes those two observations. In the 1D Ball Bounce example, for instance, it is sufficient that the color of each pixel is predicted by some model in every time-step.

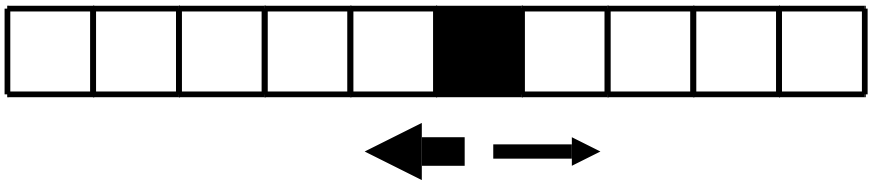


Figure 5.1: The stochastic 1D Ball Bounce example

On the other hand, if the condition does *not* hold, the CPM is effectively a partial model itself. It can make only abstract predictions about the next time step, where the abstraction is defined by the joint outcomes of the active models at each step. The remainder of the chapter will focus on CPMs that can be used as complete models, and can therefore be used by standard model-based planning methods.

5.1.3 Combining Predictions

Each partial model in a CPM makes some abstract prediction about the next observation (it predicts its outcome). In order to predict the next *primitive* observation, a CPM must efficiently combine those marginal predictions into a joint prediction. Perhaps the simplest way to combine the models' marginal predictions is to multiply them together:

$$\Pr(o|h,a) = \Pr(\omega_{\mathcal{M}_h}(o)|h,a) = \Pr(\cap_{M\in\mathcal{M}_h}\omega_M(o)|h,a) = \prod_{M\in\mathcal{M}_h} \Pr(\omega_M(o)|h,a)$$

Of course, this product form will only produce the correct joint prediction if the active models at any given history make mutually conditionally independent predictions, given history. Making this independence assumption when it does not hold can produce counter-intuitive results. For instance, it does not hold for the natural CPM discussed so far for the 1D Ball Bounce example. Recall the noisy version of the 1D Ball Bounce example presented in Section 4.2 and pictured in Figure 5.1. With probability 0.7 the ball moves in its current direction and with probability 0.3, the

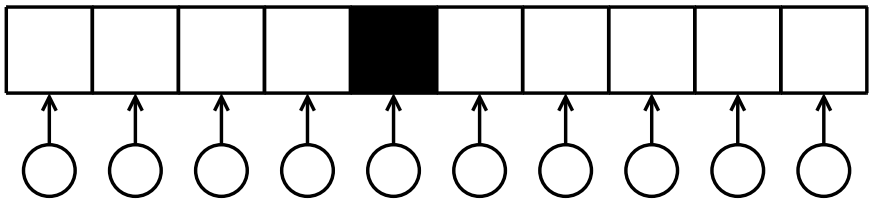


Figure 5.2: A possible CPM for 1D Ball Bounce

opposite direction. Direction changes on the edge pixels. Consider the CPM pictured in Figure 5.2 where there is a partial model (represented by a circle) for each pixel that predicts whether the ball will be in that position in the next time-step. Now imagine that the ball is in some position x and its current direction is to the right. Then with probability 0.7 it will move to position $x+1$ and with probability 0.3 it will move to position $x-1$. So, there is an active partial model for predicting whether the ball will be in position $x+1$ and one for predicting whether the ball will be in position $x-1$. If these models' predictions are treated as independent and simply multiplied together, the CPM will provide incorrect joint predictions. For instance, the CPM would assign only probability $0.7 \times 0.7 = 0.49$ to the ball landing in position $x+1$ and not in position $x-1$ (because the $x+1$ model assigns 0.7 probability to the ball moving to its position and the $x-1$ model assigns 0.7 probability to the ball *not* moving to its position). Worse, the CPM would assign positive probability $(0.7 \times 0.3 = 0.21)$ to the ball landing in *neither* position $x+1$ *nor* position $x-1$ as well as to the ball landing in *both* positions $x+1$ and $x-1$, both of which eventualities should obviously have zero probability.

The problem in this example stems from the fact that the models provide only marginal probabilities for correlated events and there is no general method for computing a joint probability from marginal probabilities. Thus, in order for the product form to be correct, the individual partial models in a CPM must be designed in a way to make it so. This section will discuss two strategies for constructing CPMs so that the product form *does* produce correct joint predictions. The first strategy is to

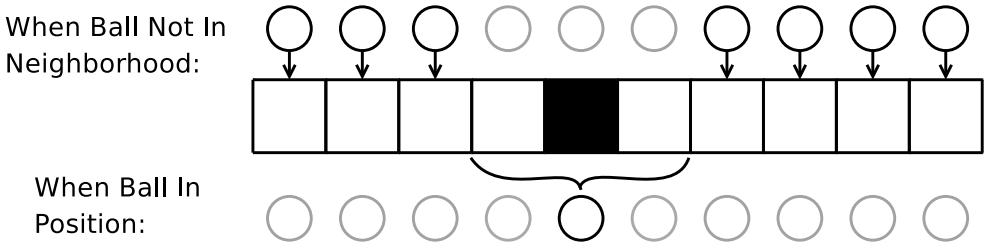


Figure 5.3: A CPM in which models make independent predictions.

ensure that any predictions that are correlated are made jointly by a single partial model, thus satisfying the mutual independence assumption. Histories of interest can limit that model's responsibilities so joint predictions are only made when necessary. The second, more general strategy is to allow partial models' predictions to depend upon the outcomes of other models, thus allowing for arbitrary joint distributions, in principle.

5.1.3.1 Making Joint Predictions When Necessary

In the CPM for 1D Ball Bounce in the previous example, there is a model for predicting whether the ball will be in each position. Because treating individual pixels independently is problematic, one might imagine instead having a model that jointly predicts positions that are correlated. Whether or not the ball will be in position x is correlated with whether or not the ball will be in position $x-2$ and whether it will be in position $x+2$. Of course, whether the ball will be in position $x-2$ is also correlated with whether it will be in position $x-4$, which is, in turn, correlated with whether it will be in position $x-6$, and so on. One partial model that jointly predicts *all* of these positions would be so complex as to essentially defeat the purpose of having partial models in the first place. That said, there is structure here that can be exploited through the use of histories of interest.

Note that predictions about position x and position $x-2$ are *only* correlated when the ball is in position $x-1$ (because it can move either left or right, but not both). In

any other situation predictions about positions x and $x-2$ are, in fact, independent. As such, it is possible to have a partial model that jointly predicts whether the ball will be in position x and whether it will be in position $x-2$, but *only* in histories that end with the ball in position $x-1$. In the new CPM (pictured in Figure 5.3) there is a pair of partial models associated with each position x. The first type of model (pictured above the line of pixels) is active in histories that end with the ball *outside* the immediate neighborhood of x (inactive models are in grey) and predicts whether the ball will be in position x in the next time step (this model has an easy job: always predict "no"). The second type of model (pictured beneath the line of pixels) is active in histories that end with the ball *in* position x. This model predicts the color of *all three* pixels in the immediate neighborhood of x.

In this CPM the mutual independence property holds because the pixels whose values are correlated are predicted jointly. However, histories of interest allow for partial models that only make joint predictions when necessary and otherwise treat pixels as independent, essentially allowing the intra-step dependency structure to change over time. As a result, the individual partial models can still have quite restricted prediction responsibilities (in comparison to a model that must jointly predict all pixels that are *ever* correlated).

This example illustrates that CPMs have a great deal of flexibility in decomposing predictions about the next observation. Because different models can be active at different times, predictions can be made jointly or independently as appropriate for the particular situation. Of course, this requires that the prediction responsibilities of the component models of the CPM be defined with the knowledge *built in* about which predictions are correlated when. While it is conceivable that this type of structural information might be available to a domain expert, or could be learned before the partial models themselves are learned, a small extension to the definition of partial models (described next) will allow for more expressiveness when

decomposing predictions. It will also suggest a straightforward method for learning the dependence relationship between the predictions of partial models that leverages methods described in earlier chapters.

5.1.3.2 Conditioned Outcomes

Instead of making the assumption that all active models make mutually conditionally independent predictions, it is far more expressive to allow models' predictions to depend upon the outcomes of other models and thus explicitly model correlation between predictions. In order to ensure that there are no circular dependencies, an ordering is imposed over the models: $M_1, M_2, ..., M_{|\mathcal{M}|}$. In principle any model M_i conditions its predictions upon the outcomes of *all* previous models that are also active: $\mathcal{M}_h^i \stackrel{\text{def}}{=} \{M_1, ..., M_{i-1}\} \cap \mathcal{M}_h$. Let the joint outcome of all active models from $M_1, ..., M_{i-1}$ for a particular next observation o be $\omega_{\mathcal{M}_h^i}(o)$. Now model M_i's predictions are required to be *conditioned* on this joint predicted abstract observation. Thus, model M_i's predictions of interest become:

$$\Pr(\omega_{M_i}(o)|h, a, \omega_{\mathcal{M}_h^i}(o)) \tag{5.1}$$

for all histories of interest $h \in \mathcal{H}_{M_i}$, actions $a \in \mathcal{A}$, and primitive observations o. Note that model M_i does *not* depend upon any internal state of the previous models, only their predicted abstract observations.

For instance, in the 1D Ball Bounce example, the models are associated with positions. One might order the models from left to right, allowing each model to condition on the values of all pixels to the left when making its own prediction. Consider a model predicting whether the ball will be in position x when the ball is in position $x-1$ and is currently moving to the right. Then, instead of predicting that the ball will move to position x with probability 0.7, the model will make *conditional*

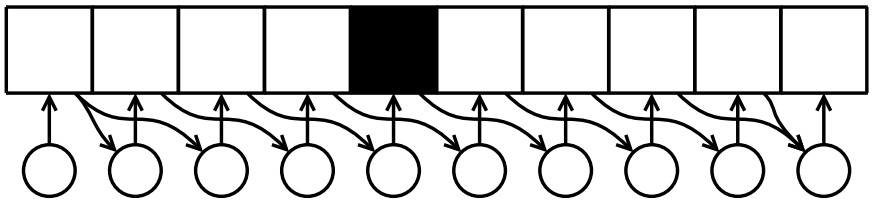


Figure 5.4: Each model conditions its predictions on the pixel two positions to the left

predictions based on the outcome of pixels to the left. So the model will predict that *if* the ball moves to position $x-2$, then it will move to position x with probability 0. If the ball does not move to position $x-2$, then it will move to position x with probability 1. As such, the previously discussed problem of assigning positive probability to impossible events has been addressed, now that the models' predictions explicitly model the dependencies amongst their outcomes.

If all partial models accurately make their predictions of interest, conditioned on the outcomes of the previous models in the order, then *any* arbitrary conditional probability distribution over the next observation o at history h can be represented as the product of the predictions of the active models, simply by the chain rule of probability:

$$\begin{aligned}\Pr(o|h,a) &= \Pr(\omega_{\mathcal{M}_h}(o)|h,a) = \Pr(\omega_{M_1}(o) \cap \omega_{M_2}(o) \cap ... \cap \omega_{|\mathcal{M}_h|}(o)|h,a) \\ &= \prod_{M_i \in \mathcal{M}_h} \Pr(\omega_{M_i}(o)|h,a,\omega_{\mathcal{M}_h^i}(o))\end{aligned}$$

This generality comes at a cost, however. Late in the order, the joint outcomes of the previous models will almost fully specify the (possibly high-dimensional) primitive observation. Thus, the compactness of late models, that must condition on this extremely detailed abstract feature of the next observation, will suffer. Taking inspiration from the graphical models literature, CPMs address this by assuming structure in the conditional independence relationships amongst models' predictions. That is, it is assumed that only *some* of the information predicted by previous models is rel-

evant for any given model; much of it can be ignored. For instance, in the 1D Ball Bounce example, a model associated with position x need only condition on the pixel in position $x-2$; the values of all other pixels to the left are independent of the value of the pixel at position x. A collection of these partial models is shown in Figure 5.4. More generally, each model will depend only on some *abstraction* of the joint predicted observation of the previous models.

Definition 5.4. For each model M_i the *conditioned abstraction* is a surjection κ_{M_i} over joint outcomes of the previous models in the order. At time-step k, when making predictions about the next observation o_{k+1}, model M_i will only condition its predictions upon $\kappa_{M_i}(o_{k+1})$, where the notational shorthand $\kappa_{M_i}(o_{k+1})$ is taken to mean $\kappa_{M_i}(\omega_{\mathcal{M}_h^i}(o_{k+1}))$. Call $\kappa_{M_i}(o_{k+1})$ model M_i's *conditioned outcome*.

The joint probability distribution as computed by the CPM will be correct if each model's prediction is conditionally independent of all earlier models' predictions, given history, the action, *and its own conditioned outcome*. That is, if:

$$\Pr(\omega_{M_i}(o)|h,a,\omega_{\mathcal{M}_h^i}(o)) = \Pr(\omega_{M_i}(o)|h,a,\kappa_{M_i}(o)).$$

In other words $\kappa_{M_i}(o)$ is required to summarize everything about the joint outcome of the previous models $\omega_{\mathcal{M}_h^i}(o)$ that is relevant to making predictions about the model M_i's outcome $\omega_{M_i}(o)$. If this property holds true of κ for every partial model (and if the partial models make accurate predictions), then the product of the predictions of the partial models is the correct prediction for the next observation:

$$\begin{aligned}\prod_{M_i\in\mathcal{M}_h}\Pr(\omega_{M_i}(o)|h,a,\kappa_{M_i}(o)) &= \prod_{M_i\in\mathcal{M}_h}\Pr(\omega_{M_i}(o)|h,a,\omega_{\mathcal{M}_h^i}(o))\\ &= \Pr(\omega_{M_1}(o)\cap\omega_{M_2}(o)\cap\ldots\cap\omega_{M_{|\mathcal{M}_h|}}(o)|h,a)\\ &= \Pr(o|h,a)\end{aligned}$$

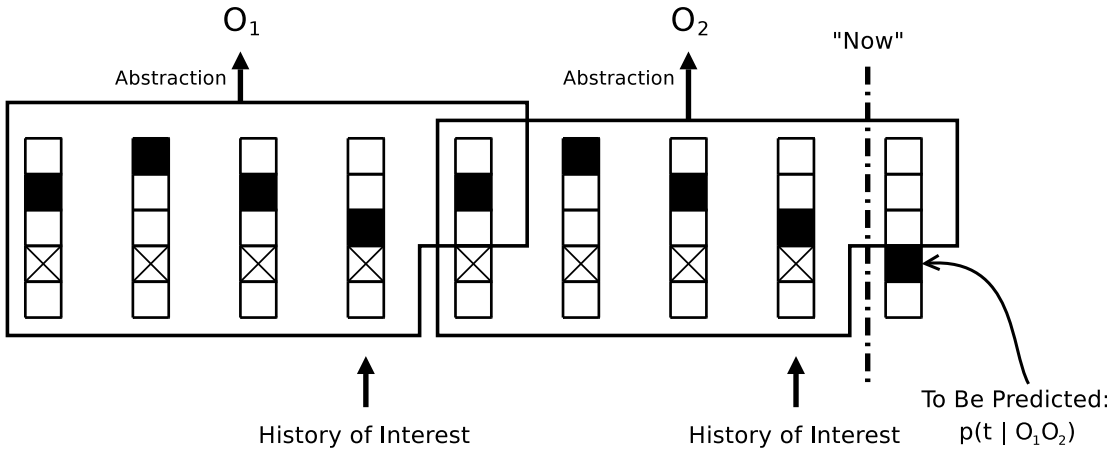


Figure 5.5: Incorporating κ into the bridging test abstraction.

5.1.3.3 Learning the Conditioned Observation Abstraction

In analogy with graphical models, the conditioned outcomes, as specified by the κ_{M_i} for each partial model M_i, essentially defines the *structure* of the CPM: the conditioned abstractions define the conditional independence relationships that will be assumed between the models' predictions during parameter learning. It is common in the graphical models literature for structure of this form to be provided by domain experts, leaving only the model parameters to be learned from data. Note that if the designer of a CPM has access to knowledge about which models are independent in various situations, as in the previous section, this knowledge can be used to design models with simple conditioned outcomes, using histories of interest to model complex dependencies only when necessary. That said, it is also straightforward to learn κ (or to adapt a given abstraction) using the abstraction learning process discussed in Chapter 4.

Recall that the algorithm in Chapter 4 assumed as given a set $\mathcal{F}$ of features of *bridging tests* (sequences of actions and observations that occur between histories of interest) and employed a decision-tree-like algorithm to construct an abstraction using these features. To fold in learning of κ_{M_i}, one simply allows the abstraction to incorporate features of the observation being predicted (the observation immediately

following the bridging test). Specifically, one can add to $\mathcal{F}$ features of the joint outcome of the previous models $\omega_{\mathcal{M}_h^i}(o)$ (where o is the next observation) and allow the abstraction learning procedure to incorporate these features as well. The abstract sequence is generated as pictured in Figure 5.5: each abstract symbol is the result of applying the abstraction to a bridging test *and* the part of the next observation that the partial model conditions on. When the model is queried for a specific prediction $\Pr(o|h,a)$, the observation being predicted is known and can easily be provided to the abstraction for the relevant features to be computed. When the model is being used to *sample* a next observation, the partial models are sampled in order, and since the abstraction only depends upon the previous models in the order, an abstract symbol can be assigned to the most recent bridging test using the sampled outcomes of the previous models. In subsequent experiments, the abstraction learning algorithm introduced in Chapter 4 will be used in this way to simultaneously learn an accurate abstraction of the bridging tests *and* a κ that results in accurate joint predictions.

5.1.4 Collections of Partial Models: Summary

This section briefly collects the discussion of the previous sections into a concise description of a CPM. A collection of partial models (CPM) consists of an ordered set $\mathcal{M} = \{M_1, M_2, ..., M_{|\mathcal{M}|}\}$ of partial models. Each model M_i has prediction responsibilities, given by three components:

1. A set of *histories of interest*, $\mathcal{H}^I_{M_i}$, which determine in which histories model M_i is *active* and able to make predictions.

2. An *abstraction of interest*, ω_{M_i}, which is a surjection defined over all primitive observations. The model makes predictions about its *outcome*, $\omega_{M_i}(o)$, where o is the *next* observation.

3. A *conditioned abstraction*, κ_{M_i}, which is a surjection defined over the *joint out-*

comes of the previous models in the order. The model conditions its predictions about its own outcome $\omega_{M_i}(o)$ on the *conditioned outcome* $\kappa_{M_i}(o)$, where o is the *next* observation.

Each partial model M_i is then responsible for making the predictions

$$\Pr(\omega_{M_i}(o)|h, a, \kappa_{M_i}(o))$$

for all histories $h \in \mathcal{H}^I_{M_i}$, all actions $a \in \mathcal{A}$, and all primitive observations $o \in \mathcal{O}$.

At any given history h, some subset of the partial models are *active* and able to make predictions: $\mathcal{M}_h \stackrel{\text{def}}{=} \{M : h \in \mathcal{H}^I_M\}$. The CPM provides predictions about the joint outcome of the models in $\mathcal{M}_h$. That is, the CPM predicts the value of $\omega_{CPM}(o, h) \stackrel{\text{def}}{=} \bigcap_{M \in \mathcal{M}_h} \omega_M(o)$ (where o is the next observation). The CPM's prediction is computed as the product of the predictions of the active partial models:

$$\Pr_{CPM}(\omega_{CPM}(o, h)|h, a) \stackrel{\text{def}}{=} \prod_{M_i \in \mathcal{M}_h} \Pr(\omega_{M_i}(o)|h, a, \kappa_{M_i}(o))$$

A CPM is a *complete model* (i.e. provides predictions for all primitive one-step tests) if for all histories $h \in \mathcal{H}$ and all primitive observations $o \in \mathcal{O}$, $\omega_{CPM}(o, h) = \{o\}$. That is, for every pair of primitive observations and every history, some active model's abstraction of interest distinguish those observations.

A CPM makes *accurate predictions*, that is, all observations $o \in \mathcal{O}$, all actions $a \in \mathcal{A}$, and all histories $h \in \mathcal{H}^I_{M_i}$, $\Pr_{CPM}(\omega_{CPM}(o, h)|h, a) = \Pr(\omega_{CPM}(o, h)|h, a)$, if for all models M_i, , $\Pr(\omega_{M_i}(o)|h, a, \kappa_{M_i}(o)) = \Pr(\omega_{M_i}(o)|h, a, \omega_{\mathcal{M}^i_h}(o))$ (and, obviously, if all partial models make accurate predictions). In words, each model M_i's outcome must be conditionally independent of the joint outcome of all previous models in the order, given history and model M_i's conditioned outcome.

With CPMs described in full, the next section will discuss some strengths and weaknesses of CPMs and relate them to existing types of structured models (primarily

dynamic Bayes nets).

5.2 Discussion and related work

A CPM is essentially a "divide and conquer" approach to modeling. Consisting of many partial models, a CPM decomposes the problems of learning, representation, and prediction. Since each partial model can itself be quite simple and learnable, the CPM can be correspondingly more compact and more learnable than an unstructured complete model. Of course, decomposition into many partial models is not the only way to introduce structure into a model. This section will discuss the relationship between CPMs and some other structured representations. Perhaps the deepest, and most interesting connection is to dynamic Bayes nets (DBNs), a popular and well-studied structured representation (described in Section 1.2.2.2). The comparison to DBNs in the next section will also serve as a useful context in which to discuss some key properties of CPMs. Comparisons between CPMs and some other relevant representations will be briefly discussed in the subsequent section.

5.2.1 DBNs and CPMs

As described in Section 1.2.2.2, DBNs are a generalization of POMDPs that decompose their representation of the hidden state and the observation into a set of variables, some of which are observable, and some of which are hidden. The value of each variable in a given time-step is stochastic, and may depend upon the values of other variables in the same step, or the values of variables in the previous step. The *structure* of a DBN is the specification of the conditional independence relationships between variables. If each variable in a DBN depends on the values of very few other variables, then it can be represented far more compactly than a flat POMDP (which implicitly assumes every variable is dependent on every other variable).

There are some clear parallels between DBNs and CPMs. In a CPM, each partial

model makes predictions about some abstraction of the primitive observation, much like an observed variable in a DBN. If the partial models in a CPM use a hidden-variable-based representation (such as a POMDP), then the hidden variables of the models roughly correspond to the hidden variables in a DBN. There are, however, several differences between the two representations that result in trade-offs for choosing to use one over the other. The *main* difference is that the structure in a DBN is primarily focused on the conditional independence amongst unobserved hidden variables while the structure in a CPM is expressed entirely in terms of conditional independence of observable events. This has two main consequences:

Localized Training: Partial models in a CPM do not share any state information they may maintain in order to make their predictions. Partial models only interact with each other insofar as the predictions of one model may depend upon the *outcomes* of other models, which are observable. Because each partial model is self-contained in this sense, each partial model can be trained independently of any others: everything needed to learn a partial model can be obtained from the observations in the training data. In contrast, the hidden variables in a DBN typically depend directly upon each others' (unobserved) values, inextricably intertwining the learning problems associated with the various hidden variables. As a result, DBN training using the EM algorithm requires a global inference step over all hidden variables that must be approximated in general (Boyen & Koller 1998). In essence, while DBNs do successfully decompose the problems of representation and prediction, *they do not decompose the learning problem.*

On the other hand, because there is no mechanism for partial models in a CPM to directly share state information, there may be duplicated effort across multiple models. For instance, recall the 1D Ball Bounce example. The CPM discussed in Section 5.1.3.2 has a partial model associated with each position that predicts whether

the ball will be in that position in the next time-step, conditioned on whether it will be in any position to the left in the next time-step. In order to make correct predictions, such a partial model must maintain information about the ball's current direction (as this affects the distribution over its next position). Because the models do not share information, each partial model in the CPM must maintain the ball's current direction for itself. A DBN, in contrast, could represent the ball's current direction once as a single hidden variable whose value can be accessed by any other variable that depends upon it.

So, one would expect the decoupling of the component models in a CPM to be an advantage mainly when the models maintain relatively disjoint state information. In this case one can independently train many simple models instead of attempting to perform global inference over the hidden state as a whole, as in DBN training. If, on the other hand, the hidden state can be expressed very concisely (as in the 1D Ball Bounce example), a DBN will likely be a more compact representation and may not suffer much from the need for global inference during learning.

Verifiable Structure: The structural assumptions made by a CPM are given by each model M_i's conditioned abstraction κ_{M_i}. The assumption is that each model's outcome is conditionally independent of the outcomes of the previous models, given its conditioned outcome (as given by κ_{M_i}). Note that this assumption involves only the conditional independence of observable events. As a result, a CPM's structural assumptions can be tested via statistical analysis of the training data and, given sufficient data, may be falsified if they are incorrect. In contrast a DBN's structure primarily makes assumptions about the conditional independence of hidden variables, whose values are never observed. As a result, a DBN's structural assumptions can never be verified or falsified by data. Though structure learning is not a focus of this dissertation, it seems reasonable to hypothesize that being able to evaluate the

objective correctness of structural assumptions would make structure learning significantly easier. Indeed, Section 5.1.3.3 has already discussed how κ_{M_i} can be straightforwardly learned from training data. Because the assumptions implied by a given κ_{M_i} are testable, one can apply an incremental generate-and-test algorithm like that presented in Chapter 4 to learn it.

The trade-off is that there may exist systems that can be compactly represented using structured hidden variables but whose observations cannot be easily decomposed. It is not difficult, for instance, to imagine an environment with complex, factored dynamics, but only a binary observation. Clearly the type of structure exploited by a CPM would be of no use in such a system. On the other hand, in a trivial sense CPMs do not rule such systems out, since an individual partial model could itself be represented as a DBN. This point is expanded on below.

Apart from the difference in the nature of their structural assumptions, there are some other ways in which CPMs differ from DBNs that are worth discussing briefly.

Histories of Interest: While histories of interest are central to a CPM, they do not have a direct analog in a DBN. As discussed in Chapter 4 and Section 5.1.2, histories of interest provide a number of benefits. Allowing models to make predictions in only some situations can simplify the modeling task, provide an opportunity for temporal abstraction, and allow the CPM to alter independence assumptions over time, as necessary. Some of these features can be obtained in a DBN in other ways. For instance, by imposing additional structure such as context-specific independence (Boutilier et al. 1996), a DBN can represent independence relationships that change over time. That said, context-specific dependence sets dependence relationships based on the values of variables, rather than explicit features of history. While this is reasonable when the DBN is being specified by an expert, when learning, it can

be difficult or impossible to know *a priori* what specific values of hidden variables will "mean" after training, making the specification of context-specific independence structure before learning difficult.

Representational Flexibility: Note that a CPM is specified by the *responsibilities* of its component models, not the models themselves. That is, a CPM's structure tells each model *what* to predict and *when* to predict, but not *how* to predict. For the CPM's purposes, the individual models are black-box predictors. This offers a great deal of representational flexibility. Dynamic Bayes nets, by their very nature make a commitment to a Bayesian network representation which is very powerful in many ways, but can also be limiting. The parameters of Bayes nets with hidden variables are typically learned using Expectation Maximization which, even apart from the necessity of approximate inference mentioned earlier, is a hill-climbing algorithm and therefore prone to converging on local maxima. In a CPM, while individual partial models *could* be represented using Bayes networks, such as POMDPs or even structured representations like a DBN, the framework makes no such commitment. As such, partial models are free to use representations whose learning algorithms have stronger convergence properties and that do not easily fit in the graphical models framework (such as PSRs or OOMs). This also allows the use of prediction profile models, and the attendant advantages discussed in Chapter 3. Finally, this flexibility offers the possibility of using different representations for different models: applying the right method to each individual phenomenon in the world. Intriguing though it is, exploration of this last possibility will be left to future work.

Some of the comparisons between CPMs and DBNs made here will be explored empirically in Section 5.3.1. First, however, it is worth briefly discussing the connections between CPMs and other related existing representations.

5.2.2 Other Relevant Work

Stochastic relational models (e.g. Pasula et al. 2007) are a structured representation of Markov environments. The state is represented as the conjunction of a number of predicates. The dynamics are driven by a set of "update rules," each with pre-conditions on the state determining when the rule applies and stochastic post-conditions specifying how the rule changes the state. Partial models are similar to update rules with histories of interest and abstract observations playing the roles of pre- and post-conditions, respectively. The main difference is that update rules *always* make the same prediction whenever they are active. That is, the pre-conditions themselves must be informative enough to make accurate predictions. A partial model, in contrast, maintains the state information necessary to make its predictions. This allows CPMs to enjoy the benefits of model decomposition in partially observable environments. A strength of relational models not reflected in CPMs is the use of variables and first-order reasoning. First order update rules can bring to bear sophisticated generalization and parameter tying, which can greatly impact compactness and learnability. While the experiments in Section 5.3.2 will make use of more *ad hoc* parameter tying schemes, first-order representations have not been incorporated into the definition of CPMs.

Product of experts models (Hinton 1999) represent a conditional probability distribution as the normalized product of the output of several "experts." However, while a partial model is defined by its prediction responsibilities, the semantics of the experts' output is not defined *a priori*. As a result, POE models are trained *globally* to ensure that different experts make different predictions. Also, with no conditional independence assumptions placed on the experts, POE models suffer from a costly re-normalization step when making predictions. A CPM does not re-normalize its predictions on the assumption that the provided structure correctly represents the true dependence relationships between models' predictions. Unlike CPMs, however,

POE models do automatically learn a decomposition of the problem into experts. It is possible that product of expert models could inspire methods for learning the structure in a CPM.

Maximum entropy models (e.g. Berger et al. 1996) represent a conditional probability distribution as a weighted, normalized product of a set of features (of both the outcome being predicted, and the values of the variables being conditioned on). The basic idea is to assume that the expected values of the features are equal to the estimated expected values from the training data. Then it can be shown that the *least commitment* model that produces a distribution with the highest possible entropy (while still retaining the expected values for the features) take this weighted product form. The learning problem is then to learn values for the weights that actually maximize the entropy. One way to see CPMs in the light of maximum entropy models is to imagine that the features are binary and take the form "Model M is active and test of interest t occurs" (remember, the features are of both the history and the future being predicted). The weights, then, are the actual predictions made by the models. Seen this way, the computation of a CPM's prediction and a maximum entropy model's prediction roughly correspond. On the other hand, in a maximum entropy model, the weights associated with each feature are fixed. In a CPM, the predictions of the models reflect the changing state of those models. A such, a CPM is in some ways more expressive.

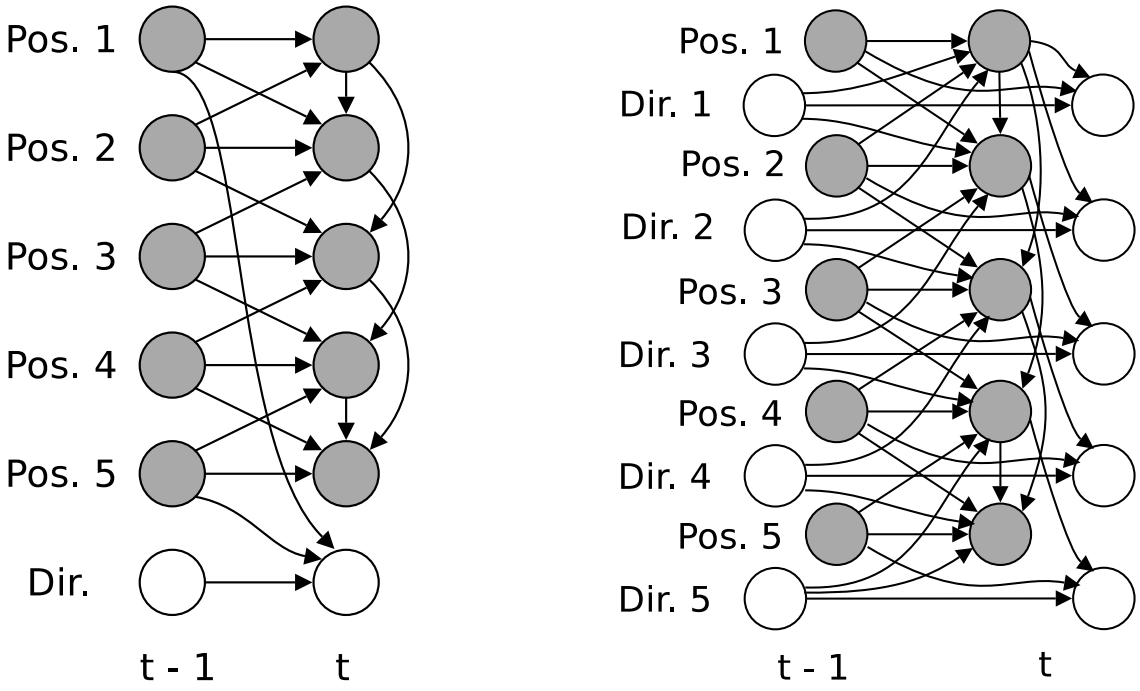
As mentioned in Chapter 1, Wolfe et al. (2008) introduced a structured representation called the *factored PSR*. In a factored PSR, the observation is split into variables and for each variable, a PSR model is learned to make predictions about that variable. The joint prediction of the model as a whole is the product of the predictions of the individual models. A factored PSR is an example of a CPM, as described here. Collections of partial models are more general in several ways but, nevertheless, the results obtained by Wolfe et al. can serve as further evidence of the

promise of this approach.

Wolfe (2009) also introduced a representation called the *multi-mode PSR*. In a multi-mode PSR, the system is assumed to have many distinct "modes" which can be labeled in the training data. A separate model is learned for each mode. Then an abstract model is learned that predicts the dynamics of the modes themselves. When the model is used for prediction in the actual environment (where mode labels are not available), this upper-level model provides a probability distribution over the current mode, and the various specific mode-models' predictions are mixed appropriately. This example is notable mostly as an interesting generalization of histories of interest. In a CPM, each model is either active, or inactive. In a multi-mode PSR, there is a soft notion of when models should make predictions, where models contribute more or less to the overall prediction, depending on how likely it is that they actually apply. It may be interesting to extend CPMs to include similar ideas.

5.3 Experiments

This section will describe empirical results of applying CPMs to some example problems. The first set of experiments will supplement the discussion comparing DBNs and CPMs by applying both methods in an illustrative example (1D Ball Bounce). These experiments will also show the effect that different decompositions can have on the difficulty of learning a CPM. The second set of experiments will apply CPMs to two arcade game examples. These experiments will serve two purposes. First, they will illustrate the application of CPMs in high dimensional problems. Second, they will integrate several facets of the discussion so far, as the individual partial models in the CPMs will be learned using the abstraction learning algorithm from Chapter 4 and the prediction profile model learning algorithm from Chapter 3.



(a) Variant 1

(b) Variant 2

Figure 5.6: DBN structures for 1D Ball Bounce

5.3.1 DBNs and CPMs

The first set of experiments will serve as an empirical companion to Section 5.2.1, which compared DBNs to CPMs. Both will be applied to the stochastic version of 1D Ball Bounce that has been discussed in several previous sections, as well as another variant meant to contrast the difference between the two approaches.

5.3.1.1 Variant 1: Simple Hidden State

First consider the stochastic 1D Ball Bounce. As described earlier, the ball has a current direction (starts to the left) and position (starts in the middle pixel) and every time-step moves one pixel in its current direction with probability 0.7 and one pixel in the opposite direction with probability 0.3. When the ball enters an edge pixel its direction is set to point away from the edge (e.g. entering the left edge pixel sets the ball's direction to the right).

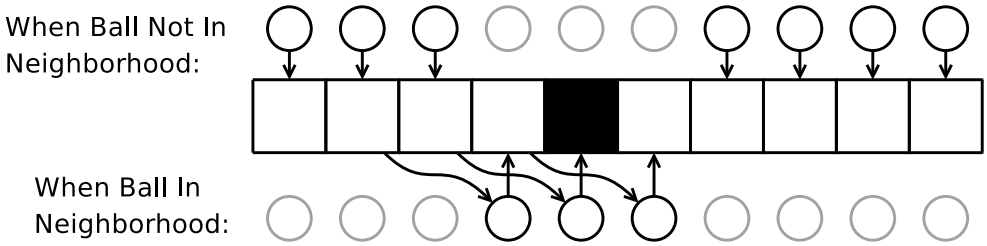


Figure 5.7: CPM-PixelHOI Structure

Figure 5.6(a) shows a DBN structure for a size 5 1D Ball Bounce. Observed variables are shaded and hidden variables are white. There is one observed variable for each pixel and a single hidden variable representing the ball's direction. The arrows denote the conditional dependence relationships (the conditional probability of a node depends on the values of the nodes pointing to it, and is conditionally independent of the rest). To avoid clutter, the arrows from "Dir" in the first time-slice to all position nodes in the second time-slice have been omitted. Note that, in addition to inter-time-slice links, there are some intra-time arrows as well, necessary to capture the correlation between locations the ball might move to.

This experiment will apply 3 different CPM structures. Note that the DBN structure specifies for each variable what it can safely ignore from the previous state and what it can safely ignore from the variables above it in the current time-step. Analogously, in this experiment the models in the CPM will be provided with an accurate bridging test abstraction η and conditioned abstraction κ to provide a fair comparison. These abstractions will be learned in later experiments. The three CPMs are:

- "CPM-Pixel" – Pictured in Figure 5.4. There is one partial model for each position x, responsible for predicting whether the ball will enter x in the next time-step, conditioned on the color of position $x - 2$. Each model is active in all histories. Thus, the bridging test abstractions are actually observation abstractions, and include the pixels in the immediate neighborhood of x and the edge pixels.

- "CPM-PixelHOI"– Pictured in Figure 5.7. There are two partial models for each position x, both responsible for predicting whether the ball will enter that position in the next time-step. The first model is active when the ball is in the immediate neighborhood of x and conditions its predictions on position $x-2$. Its bridging test abstraction includes whether the ball hit an edge pixel during the bridging test, as well as the values of the pixels in the immediate neighborhood of x at the end of the bridging test. The second model is active when the ball is *not* in the immediate neighborhood of x and does not condition its predictions on other models. Its abstraction maps all bridging tests to the same observation.

- "CPM-NeighborhoodHOI"– Pictured in Figure 5.3. There are also two partial models for each position x. One is familiar: it predicts whether the ball will be in position x when the ball is not in the neighborhood of x. The other is active in histories that end with the ball *in* x. It jointly predicts the pixels in the immediate neighborhood of x. In this CPM, all partial models make conditionally independent predictions. Each model's bridging test abstraction indicates whether the ball hit the edge pixels during the bridging test.

One of the main differences between the DBN and the various CPMs in this example is that the DBN represents the ball's current direction once as a single random variable. In contrast, each individual model in the CPMs must learn to keep track of that information for itself. Another difference is that the DBN is trained using EM, which is not guaranteed to find optimal parameters due to local maxima. The partial models in the CPMs will be represented as PP-LPSTs (as described in Chapter 3), whose learning algorithm is not subject to local extrema.

The difference between the various CPMs is essentially how much knowledge is being put into the structure. "CPM-Pixel" is essentially based on knowledge about the conditional independence relationships between pixels within and between time-

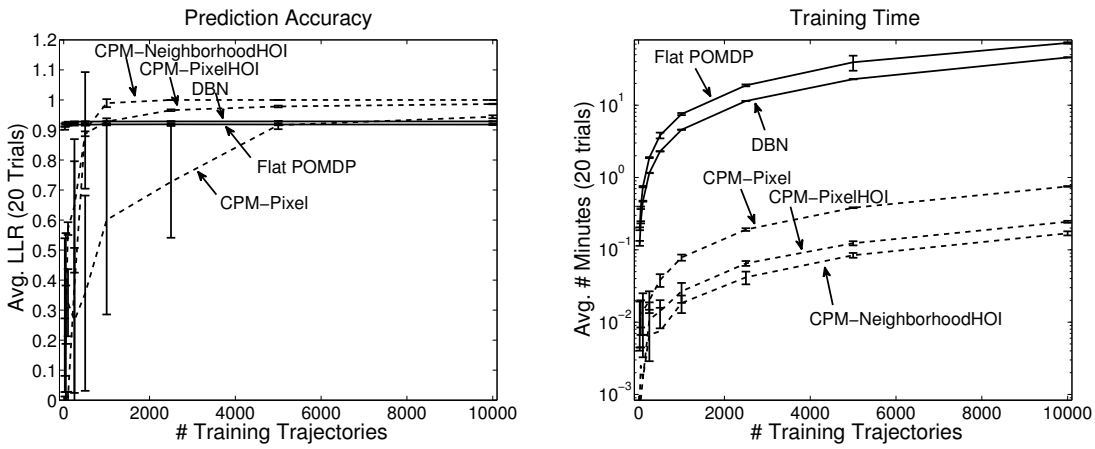


Figure 5.8: Learning results for 1D Ball Bounce Variant 1.

steps, a similar type of knowledge built into the DBN structure. "CPM-PixelHOI" adds the knowledge that for a particular position "interesting" things only happen when the ball is in the neighborhood of that position and that much of what happens at other times can be abstracted away. Finally, "CPM-NeighborhoodHOI" further adds knowledge about when pixels are correlated and when they independent, allowing for a CPM where all active models make independent predictions (and thus have simple conditioned outcomes).

Figure 5.8 shows the results of training these various models on data from 1D Ball Bounce (size 10). Also included for comparison is the result of training a 20 state flat POMDP on the same data (with flattened observations). Both the DBN and POMDP were trained using the Graphical Models Toolkit (Bilmes 2007). As mentioned earlier, the partial models in the various CPMs were PP-LPSTs. The results are averaged over 20 trials. In each trial, the models were trained with 50-step sampled trajectories. The learned models were then evaluated by measuring the likelihood they assigned to 1000 length 50 test trajectories.

The graph on the left reports the average log-likelihood ratio (LLR), which is the true log-likelihood divided by the log-likelihood of the learned model. So the optimal value, corresponding to perfect predictions, is 1. The DBN and flat POMDP (solid

lines) perform comparably well. They both arrive at good models with extremely little data (achieving an average LLR above 0.9 with only 25 trajectories) but barely improve when given more data. The structured DBN only performs marginally better than the POMDP, likely because the training for both models is converging to a local maximum. The CPMs (dashed lines) take comparatively more data to achieve the same performance as the DBN, but given sufficient data they outperform it (with CPM-NeighborhoodHOI achieving essentially perfect LLR after 5000 trajectories). The three CPM structures perform as expected. More informed structures lead to simpler individual model-learning problems, and correspondingly require less data to obtain good performance.

The graph on the right reports average training time for each of the methods in minutes (note the log scale). The experiments were run on an Intel Core2 Quad 2.66GHz processor. One cannot draw too many conclusions from the absolute difference in run-times, since the CPMs and DBNs were trained using different software with potentially different optimization priorities. However, it is true that EM for training DBNs and POMDPs is expensive. In every iteration, for every training trajectory, EM must perform probabilistic inference to obtain a distribution over the sequence of hidden variable values. This likely plays some role in the 2 orders of magnitude difference in training time between the Bayes nets and the CPMs. The difference in training time amongst the three CPMs can largely be explained by the increasing amount of temporal abstraction afforded as the structure becomes more informed. Temporal abstraction results in fewer (and shorter) abstract histories for the prediction profile model learning procedure to process.

5.3.1.2 Variant 2: Complex Hidden State

This second experiment uses another variant of the 1D Ball Bounce example. Now, instead of the *ball* having an associated direction, each *position* has an associated

direction (all start out to the left). Every time-step, with probability 0.7 the ball moves in the direction associated with its current position and with probability 0.3 it moves in the opposite direction. When the ball leaves a position, that position's direction is set to the opposite of the direction the ball went (so if the ball moves to the left, the position's direction changes to point to the right).

A DBN structure diagram of this version of 1D Ball Bounce is shown in Figure 5.6(b). Notice that there are many more hidden variables: one for each position to represent its associated direction. Recall that DBN training must perform global inference over all hidden variables so one would expect DBN performance to be worse in this domain, where the joint hidden state is so much more complex.

The same 3 CPM structures will be used with one small change: models' abstractions no longer need to include whether the ball hit an edge pixel. Unlike a DBN, a CPM is specified entirely in terms of the independence relationships amongst abstract predictions. These relationships have not changed much and so the CPM structure need not change much. The state information needed to capture the dynamics is encapsulated within the partial models themselves and so the necessary representation changes will come about through the parameter learning process, rather than through structural changes. In addition, though the joint hidden state has been made much more complex, the state information needed by each partial model remains roughly the same, so one would not expect a dramatic difference in the CPMs' performance between the two problems.

Figure 5.9 shows the results. This version of 1D Ball Bounce is too high-dimensional to feasibly train a flat POMDP (the linear dimension is greater than 1,000). As can be seen in the training time graph (on the right) the DBN took nearly an order of magnitude longer to train than it did for the first variant of 1D Ball Bounce. The DBN was not tested at 10,000 trajectories for this reason. The training time of the CPMs are essentially the same, as expected.

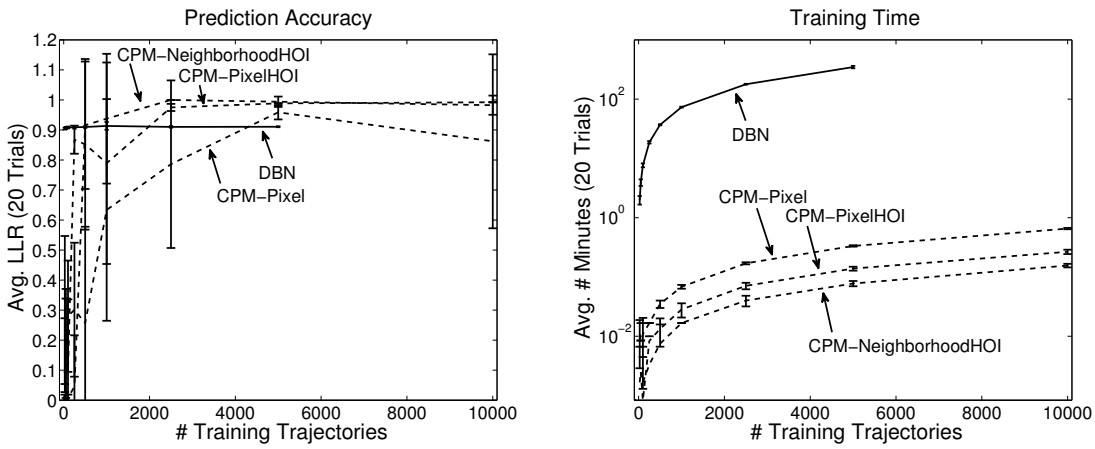


Figure 5.9: Learning results for 1D Ball Bounce Variant 2.

The prediction performance of the models did not change much between these two examples. The DBN still learned a good model with very little data, but did not improve when provided with more data. Its average LLR is marginally less than what it obtained in Variant 1. The CPMs also perform similarly. The drop in CPM-Pixel's performance at 10,000 trajectories is due to 2 trials where the learned model assigned zero probability to some test trajectory. This results in an LLR of 0 for those trials, regardless of the quality of the rest of the predictions. With those outlying trials ignored, CPM-Pixel's average LLR for 10,000 training trajectories was 0.96, which is comparable to the average obtained in Variant 1.

Overall, these results seem to align with the discussion in Section 5.2.1. Collections of partial models are best applied when the individual partial models keep track of relatively disjoint, compact information. Because a DBN does not decompose the learning problem, it is negatively affected (in this case mainly in terms of training time) by a high-dimensional hidden state, *even if* that hidden state is very structured. There are also other reasons that learning a DBN may not be desirable, namely that EM is both computationally expensive and not guaranteed to converge to optimal parameters. Nevertheless, in these experiments EM *did* produce far better models on average when very little data was available.

5.3.2 High Dimensional Examples

One of the most important driving motivations behind this work is the creation of agents that can learn to make predictions in very complex environments. At the same time, it is clear that the learning methods presented in this thesis face some serious scaling challenges. This section will demonstrate that even so they can be applied to high dimensional problems. CPMs will be learned in two example problems based on classic arcade games, with some twists to make the modeling problem more interesting. These experiments will also serve to demonstrate the full learning pipeline described in this thesis. First an abstraction will be learned for each model, then a prediction profile model will be trained, then the models will be combined into a CPM, which will be used for planning.

In these experiments the prediction responsibilities for each model in the CPM are given, as well as a set of features $\mathcal{F}$. It is also assumed that the reward function and rules for ending a trajectory are given and the focus is on learning a dynamics model. For each partial model an accurate abstraction is learned, and a prediction profile model is trained on the abstract data. Results are presented for CPMs using both PP-LPSTs and 10 state PP-POMDPs as the underlying representation for the partial models. The significance level of all the statistical tests involved in learning was set to $\alpha = 10^{-10}$.

Results are averaged over 20 trials. In each trial CPMs are learned with increasing amounts of training data. Data was generated by a model-based planning method called UCT (Kocsis & Szepesvári 2006) given access to a perfect model and given a 0.25 chance of taking a random action at every time-step (to ensure that the training data includes mistakes as well as expert play). UCT uses its model at every time-step to sample future trajectories in order to obtain estimates of the value of each action. At each step UCT was allowed 1000 sampled trajectories of length no more than 200.

The quality of the learned CPMs is evaluated both in terms of prediction accu-

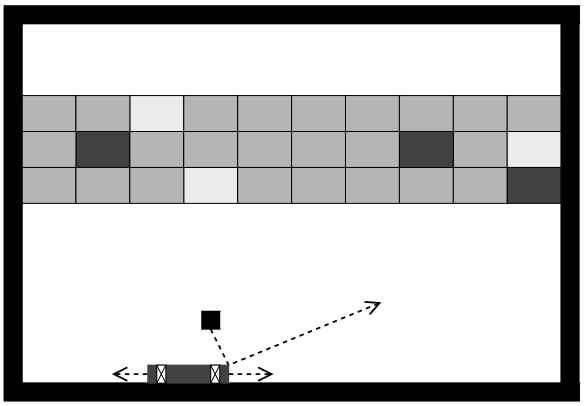


Figure 5.10: The Brick Breaker Domain

racy and how useful the model is for planning. Planning performance is evaluated by comparing the average score obtained by UCT using the learned model to the average score obtained by UCT using a perfect model on 100 test episodes. Prediction performance is measured by computing the likelihood each CPM assigns to the generated test trajectories and the root mean-square error (RMSE) of the probability it assigns to the observation that occurs at each step during the tests.

5.3.2.1 Brick Breaker

In the first arcade game example (pictured in Figure 5.10), the agent observes a 64×44 pixel image and controls a paddle at the bottom of the screen. A ball bounces around the image hitting bricks. When a brick is hit, it disappears and agent gets one point. When the ball hits the paddle, the angle at which it bounces depends on where on the paddle it hit (hitting the left side of the paddle causes the ball to bounce to the left, and like-wise the right; hitting the outside of the paddle causes a shallow angle and the middle causes a steep angle). If the ball falls past the paddle at the bottom of the screen, the game ends.

A few wrinkles have been added that make the domain more interesting (by incorporating stochastic and partially observable elements). First, when the agent moves the paddle, it moves 2 pixels in the chosen direction with probability 0.75 and one

$\omega_M(o)$: Predict	$\mathcal{H}_M$: Histories ending with
Ball color in 6×6 pixels around position p?	Ball at p, coming from direction d
Color of 6×4 pixels in brick b	Ball hitting brick b
Paddle/electric color in 14×2 pixels around position p?	Paddle at p
Color of 6×4 pixels in brick b	Ball not hitting brick b
Color of pixel p	No brick in p, no ball near p, no paddle near p

Table 5.1: CPM structure for Brick Breaker

pixel with probability 0.25. As a result, the paddle is on average slightly slower than the ball, and the agent must therefore anticipate the ball's movement rather than simply chasing it across the screen. Second, there are some special bricks scattered on the screen that cause changes in the dynamics. If the ball hits a light brick, instead of bouncing, it continues to move in its current direction (the brick still disappears). In fact, from then on, the ball plows through all bricks it encounters in the same way. This condition lasts until the ball hits a dark brick, after which the usual behavior of bouncing off of bricks returns. Thus, it is necessary to remember past events (which was hit most recently, a dark or light brick) in order to effectively anticipate the ball's trajectory. In the initial placement, bricks are regular (medium gray) with probability $\frac{5}{6}$, dark with probability $\frac{1}{9}$, and light with probability $\frac{1}{18}$. Finally, the paddle is "electrified." Spots near the left edge and right edge of the paddle flash with electricity (probability 0.4 in each time-step). If the ball touches an electric spot when it is on, the game ends immediately. Thus, one must be especially careful about where on the paddle the ball will hit. This domain is stochastic, partially-observable, and has over 10^{20} observations, and even more hidden states.

Decomposition The CPM consists of five types of partial model, specified in Table 5.1. Each row specifies what each model predicts, and when it is active. What each model conditions on will be described below.

A model in the first row predicts where the ball will be in a specific patch of pixels

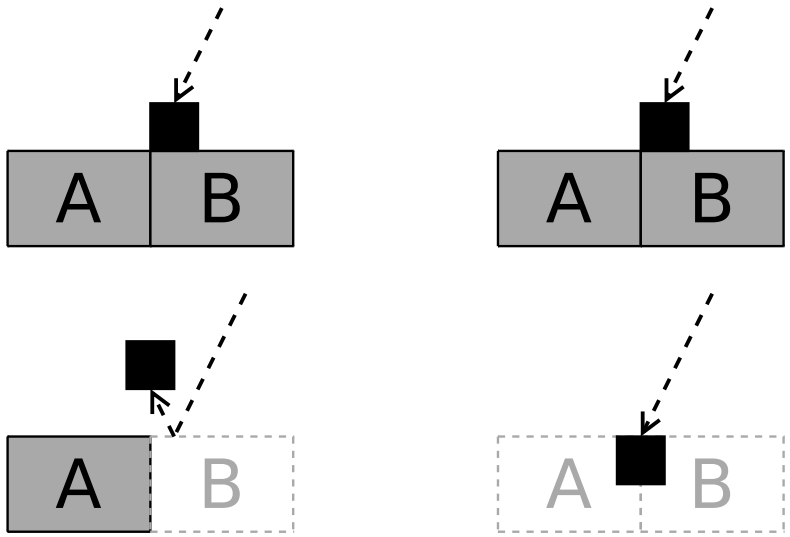
when the ball is in the middle of that patch (and came from a certain direction). This type of model will have to learn to pay attention to the configuration of bricks/paddle around the ball, as well as whether the ball is currently moving through or bouncing off of bricks in order to make accurate predictions. All other models condition on the outcomes of these models.

A model in the second row predicts what color a brick will be in histories where that brick is being hit (the ball is adjacent to the brick and moving toward the brick). This type of model actually has a surprising amount to learn as well. Whether or not the brick will disappear depends in part on the configuration of the neighboring bricks, as well as whether the ball is moving through or bouncing off of bricks. As a quick example, see Figure 5.11. In that configuration, brick A is "being hit," but will only disappear if the ball moves through brick B. That both the models in row 1 and the models in row 2 must keep track of the same state information (whether the ball will pass through a brick) is somewhat mitigated by the fact that the models in row 2 condition their predictions on the outcomes of the models in row 1. Specifically, the brick models will be able to condition their predictions on whether the ball bounces or not, thus absolving them of predicting that part of the dynamics themselves.

A model in the third row predicts where the paddle will be (and whether the electric stripes will be "on" or not) when the paddle is in a particular position. The model mainly needs to attend to the position of the paddle in order to make its predictions, since it cannot move off of the edge of the screen.

Finally, models in rows 4 and 5 make predictions for a brick or a non-brick pixel when the ball and paddle are nowhere nearby. These models are very simple; they need simply predict "no change" at every time-step.

Learning As mentioned earlier, the training data was generated with UCT using a perfect model and with a 0.25 chance of taking a random action. The training



(a) Ball Bounces (b) Ball Passes Through

Figure 5.11: Whether a brick disappears depends on the ball's behavior

episodes consisted of full games (the average length was 176 steps). Note that, as specified, there are roughly 30,000 individual partial models in the collection (associated with various pixels, bricks, etc.). In order to avoid having to train 30,000 individual models, each of which having very little associated data, a parameter tying scheme was employed. Specifically, training data was pooled for all models of a certain type, across positions. This allowed the models to exploit the spatial invariance in this domain (the dynamics of the ball in one position are the same as in another position). That said, the parameter tying was not simply imposed on the models. Included in the feature set available to the abstraction learning algorithm were features that provide the x and y coordinates associated with the bridging test. So, the abstraction learning algorithm could attend to or ignore position as it sees fit. In fact, these position features *were* used in the learned abstractions. The x coordinate is informative for the paddle dynamics (to determine if it is on the edge of the screen) and the y coordinate is informative for the ball (there are many positions where the ball cannot possibly bounce).

The abstraction learning algorithm was supplied with over 900 multi-valued fea-

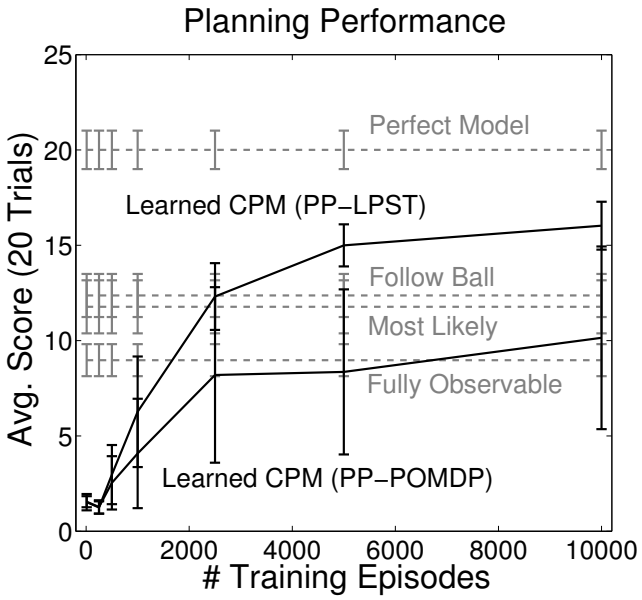


Figure 5.12: Results of planning using learned CPMs in Brick Breaker.

tures (including the position features). There were three types of features. The first type is straightforward: the color of each pixel in the 11×11 neighborhood centered on the position associated with the model at the end of the bridging test. For the second type of feature, a number of salient events were defined (a brick of each color being broken, the ball hitting the outside wall, and the ball hitting the paddle). For each event there was a feature indicating whether that event ocurred during the bridging test and for each pair of events there was a feature indicating the relative order in which they ocurred during the bridging test. Finally, the third type of feature (provided to all models *except* the ball models) indicated whether each pixel in the 11×11 neighborhood centered on the position associated with the model will be ball-colored *in the observation being predicted.* These last features facilitated the models learning to condition their predictions on the ball's position, if necessary.

Results: Planning Performance Figure 5.12 shows the average performance of UCT using the learned models (solid black lines). Clearly, as the amount of training data increases, the model improves and correspondingly so does planning perfor-

mance. In this example, PP-LPSTs seem to outperform PP-POMDPs by a wide margin. The large standard deviation on the performance obtained using PP-POMDPs indicates that the learned models' usefulness for planning is very inconsistent. This is most easily attributable to POMDP training's tendency to converge to local extrema. Also shown are a few performance baselines for comparison (grey dashed lines).

The line marked "Perfect Model" is the average performance of UCT using a perfect model. This is the best performance the learned models could hope to achieve. With enough training data, the learned CPMs using PP-LPSTs produce comparable levels of performance (though not quite as good).

The line marked "Most Likely" is the performance of UCT using a determinized model that always predicts the most likely outcome (rather than providing the probability distribution), but is perfect in every other way. The poor performance obtained using this model indicates that it is indeed important to model the stochasticity in the problem and the fact that the learned CPMs using PP-LPSTs perform better means that they are capturing that stochasticity.

The line marked "Fully Observable"is the performance of UCT using a perfect model that has been hobbled in a different way, this time to test whether capturing the partially observable aspects of the problem is important. Specifically, the "Fully Observable" model does not keep track of which special bricks have been hit, and therefore thinks that whether the ball bounces off of a brick or moves through it is random. As one might expect, this results in poor performance, as the model cannot correctly anticipate the path of the ball. The fact that the learned models perform better than this baseline means that they must be capturing this key partially observable element of the domain, at least to some degree.

Finally, the line marked "Follow Ball" is the performance of a very simple control strategy in Brick Breaker: always move the paddle towards the ball. The poor performance of this strategy simply demonstrates that to perform well in this problem,

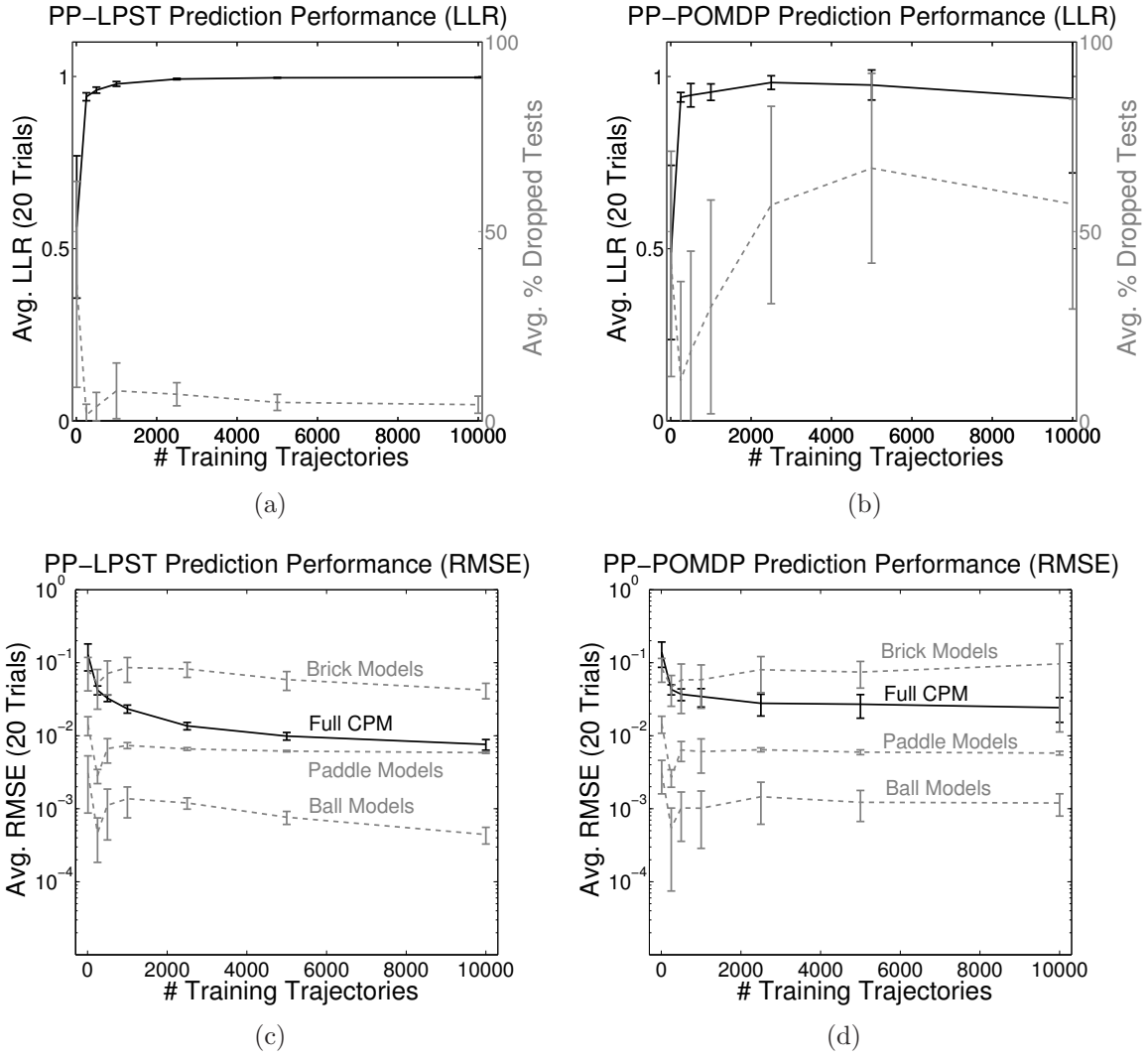


Figure 5.13: Prediction accuracy results for Brick Breaker.

one must anticipate the ball's movement. Clearly at least the learned models using PP-LPSTs have this capability.

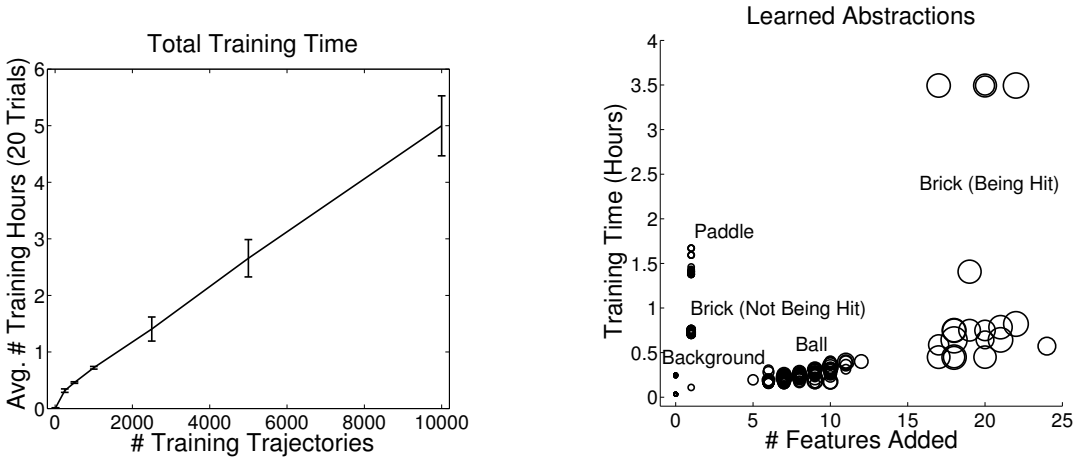
Results: Prediction Performance Figure 5.13 shows the prediction performance of the learned CPMs. Figures 5.13(a) and 5.13(b) report the average log-likelihood ratio (LLR) the learned models (solid black lines) assigned to the test trajectories generated by UCT using the learned models. In some test episodes, the models assigned zero probability to some time-steps. This results in an infinite log-likelihood, regardless of the quality of the remainder of the predictions. As a result, these test

episodes were dropped from the computation of the log-likelihood and the percentage of test trajectories that were dropped is also reported (grey dashed lines). The learned CPMs using PP-LPSTs, shown in Figure 5.13(a) made accurate predictions (LLR close to 1, and very few dropped test trajectories) with very little training data. Note that the prediction performance climbs significantly faster than the planning performance. This is likely because the prediction performance is measured on the test episodes that *actually occur*. With a small amount of training data, the model is clearly making errors during planning that detrimentally affect performance. As a result, the planning agent loses the game very quickly. So, when there is very little training data, the model obtains good prediction error, but only on poorly played games. The CPMs using PP-POMDPs (Figure 5.13(b)) are not as accurate (lower LLR, higher number of test trajectories dropped). Assigning zero probability to an event that could actually happen will clearly affect planning. The fact that the PP-POMDPs do this so often may be an important contributing factor to their comparitively poor planning performance.

Figures 5.13(c) and 5.13(d) show a different measure of the prediction performance of the models, the RMSE of the probabilities they assign to each step of the test trajectory (solid black line). This measure is less sensitive to erroneously small predictions and so it can take into account the test trajectories dropped when measuring the LLR. As with the LLR, the prediction error obtained by the CPM composed of PP-LPSTs drops quickly and remains low (note the log scale). The error obtained by the PP-POMDPs is higher, consistent with the other results. Also pictured is the average RMSE of the predictions made by various component models in the CPM. The line marked "Ball Models" averages the RMSE of the predictions made by all the models in charge of predicting the ball's position (row 1 in Table 5.1). The line marked "Brick Models" averages the RMSE of the models in charge of predicting the color of bricks, when they are hit (row 2 in the table). Finally, the line marked "Pad-

dle Models" averages the RMSE of the models in charge of predicting the paddle's position (row 3). The "easy models" from rows 4 and 5 obtained zero error. Both representations learn fairly accurate ball models (near zero RMSE), though the PP-LPSTs seem to consistently improve with more training data while the PP-POMDPs have a flatter learning curve. This is yet another indication that the PP-POMDPs are likely converging to local maxima. For both types of representation, the paddle models obtain a fairly small, but steady amount of prediction error. Since these models are stochastic, this is to be expected. In order to obtain zero error, the models must report the exact true probabilities at every time-step. This is unlikely to ever occur due to sampling error. The results indicate that the paddle models generally report probabilities that are off by less than 0.01. The brick models are the main source of error. Recall that predicting whether a brick will disappear is actually a fairly complex endeavor. It requires that one pay attention to the position and velocity of the ball relative to the brick, the color of the surrounding bricks, and which type of special brick was hit most recently. The errors made by these models may go some distance toward explaining the performance disparity between the learned models using PP-LPSTs and the perfect model as well as between the PP-LPST and the PP-POMDP-based CPM. Incorrectly predicting whether bricks will disappear could lead to incorrect predictions of the ball's path which could in turn lead to incorrect decisions.

Results: Training Time Figure 5.14(a) shows the average total time taken to learn abstractions and prediction profile models for all model types in the CPM for Brick Breaker. These experiments were run on an Intel Core2 Quad 2.66GHz processor. The training time appears to be linear in the amount of training data. This is not surprising. By far the majority of the training time is spent on abstraction learning (prediction profile models are trained within minutes once the abstraction



(a) (b)

Figure 5.14: Training time results in Brick Breaker

is learned) and the primary operation in the abstraction learning process (evaluating abstractions) is linear in the amount of training data.

Figure 5.14(b) plots the training time for individual model types across all trials (with 10,000 training trajectories) against the number of features ultimately used in the abstraction. The radius of the circles corresponds to the number of abstract observations in the learned abstraction (the smallest is 1 and the largest is 52). The main message to be taken from this graph is that the complexity of abstraction learning depends on many factors. For instance, the abstractions for the brick models (when being hit by the ball) clearly present the most involved learning problem. These abstractions have many abstract observations and incorporate many features. Correspondingly, they take the longest to train. Counterintuitively, however, the abstraction for the paddle models also takes a long time to learn, even though it only involves one feature and is ultimately a very coarse abstraction with few abstract observations. The reason for this is that the one feature that is used is the x position of the paddle. This feature has many values, and so the intermediate abstractions have many abstract observations, even if the end result is a coarse abstraction with very few abstract observations. Furthermore, the paddle's movement is stochastic. For partial

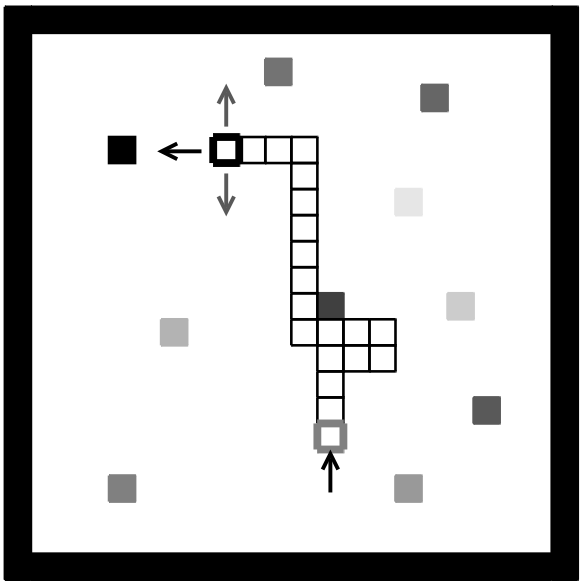


Figure 5.15: The Snake Domain

models predicting deterministic aspects of the environment, an accurate abstraction means predictions at all histories are deterministic, and can therefore obviously not be improved any further. For a stochastic model, even once an accurate abstraction is found, it must be verified that further refinement does not improve the predictions at any history, which can significantly add to training time.

5.3.2.2 Snake

The second arcade game example is pictured in Figure 5.15. Here the agent controls the head of a long snake (initially 20 pixels long) and tries to eat 10 dots of "food" scattered on the screen. The agent observes the 20×20 image and has 5 actions corresponding to the cardinal directions and "do nothing." The snake's head has a current direction and moves in that direction every time-step (its body and tail follow behind). The agent's actions change the direction of the head (but cannot reverse it). The snake must eat the food in a particular order (indicated by a grey-scale in the image). If the snake collides with itself, the wrong dot, or the edge of the screen, the agent receives a small penalty and the game ends. If the agent eats the

$\omega_M(o)$: Predict	$\mathcal{H}_M$: Active in histories ending with		
Head color in pixel p?	Head in immediate neighborhood of p		No head in neighborhood of p
Tail color in pixel p?	Tail in p	Head and/or tail in neighborhood of p (but tail not in p)	No head or tail in neighborhood of p
Color of pixel p	Head within 2 steps of p and current color is c	Tail within 2 steps of p (but not head) and current color is c	Neither head nor tail within 2 steps of p and current color is c

Table 5.2: CPM structure for Snake. Rows contain multiple model types that make the same predictions in different situations.

correct dot, it receives reward, but its tail stays still for 5 time-steps (lengthening the snake by 5 units, thereby making the control problem harder).

Again, a few tweaks were added to make the domain more interesting. First, except for the first dot, the location of the dot the agent is *currently* supposed to eat is not displayed on the screen unless the head is in its immediate neighborhood. Note that all dots that the agent will *eventually* have to eat *are* displayed through the entire game, but when a dot becomes the next dot in the order, it disappears. Thus, it is necessary to remember the location of the dot in order to find it and eat it. Second, recall that the tail stops moving briefly when the head has eaten a dot. When the tail is not stopped for that reason, the tail only moves with probability 0.8 and remains still with probability 0.2. This adds some stochasticity into the domain and also provides time pressure to the agent, since the snake is always growing longer, whether the agent eats dots or not. Like Brick-Breaker, this domain is stochastic, partially observable, and it is high-dimensional with over 10^{30} observations and even more hidden states.

Decomposition Table 5.2 shows the decomposition into partial models used in this experiment (note: there are multiple model types per row in the table with different

histories of interest). Each model is associated with a particular pixel and makes predictions about the color of that pixel in the next time-step in certain situations.

The models in the first row of the table predict whether their associated pixel will contain the snake's head when the head is nearby and when it is not, respectively. The head models were allowed to condition on other head models associated with positions above and to the left in the image. The first type of head model must learn how the head responds to the agent's actions, and must pay attention to the head's direction in order to make accurate predictions. The second type is easy: the head will not be in the pixel.

The models in the second row predict whether their pixel will contain the snake's tail in various situations. All of these models condition on the predictions of the head models (that is, whether each pixel will be head-colored) as well as the predictions of tail models above and to the left in the image. The first type is active when the tail is in its pixel already. Essentially, this model predicts *whether the tail will move* and must therefore keep track of how long it has been since a dot was eaten. The other tail models condition on this model's outcome, thus alleviating them of also learning whether the tail should move. The second type predicts whether its pixel will be tail-colored, when the tail (or the head) is in the immediate neighborhood of its pixel. In order to make accurate predictions, this model must remember what the head does in each neighboring position, in order to determine where the tail will move. The last model is simple: the tail will not be in the pixel.

Finally, the models in the third row predict the color of a particular pixel in various situations, when the pixel is already a particular color. These models depend on the outcomes of both the head and the tail models, and thus can condition their predictions on whether their associated pixel will be head-colored or tail-colored. Most notable is the model that makes predictions when the head is nearby and the pixel's current color is white. This model must predict whether the pixel will contain

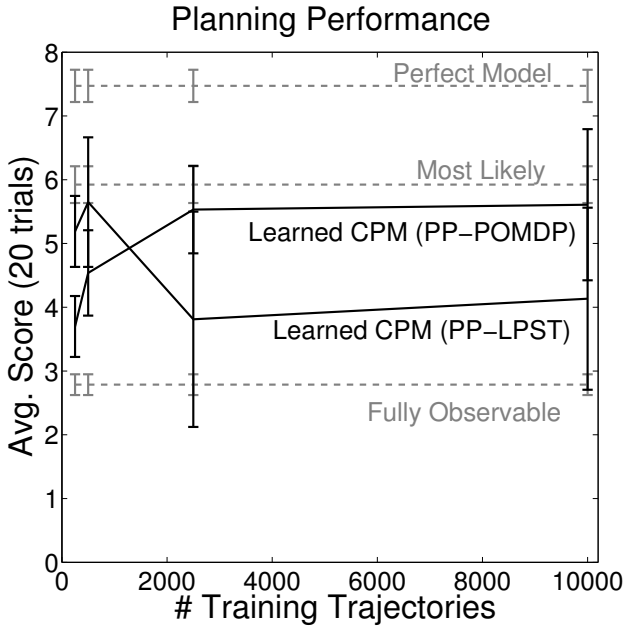


Figure 5.16: Planning results for Snake.

the target dot in the next time-step if the head comes near. As such, this model must learn to pay attention to where the head will move in the next step, whether a dot was in this position in the past, as well as whether that dot has already been eaten.

Learning As in Brick Breaker, training episodes were full games (the average length was 93 steps). The same parameter tying scheme used in Brick Breaker was employed and the abstraction learning algorithm was provided with roughly 2500 binary features (as well as the position features). The features included the color of the pixels in the 5×5 square centered on the model's position 0, 1, and 2 time-steps from the beginning and end of the bridging test. There were also features indicating whether any pixel changed from one color to another color (for all pairs of colors) at the end of the bridging sequence. Finally, similar features of the outcomes of other models were provided to the appropriate models (whether nearby pixels in the *will* contain the head or tail and whether any pixel *will* change from some particular color to the head or the tail).

Results: Planning Performance Figure 5.16 shows the average performance of UCT using the learned models (solid black lines). In this example, the CPM comprised of PP-POMDPs significantly outperforms the one comprised of PP-LPSTs. The reason for this large qualitative difference lies in the particular implementation of PP-LPSTs. Specifically, recall that LPSTs can fail to make a prediction, especially when a history suffix is encountered that is not represented in the training data. The implementation choice made in these cases was to find the longest matching suffix in the data and average together all prediction profiles associated with that suffix in order to make a prediction. Now recall the partial model that predicts the color of a white pixel when the head is nearby. This model is important because it predicts whether the target dot will appear in the next step or not. If *this* model encounters an unfamiliar history suffix, averaging the appropriate prediction profile models together will produce a prediction that assigns positive probability to a dot appearing, even when this is impossible. This has an extremely undesirable effect on the planning process. Specifically, this means that, during the planning process, if the agent can *produce* an unfamiliar history suffix, then it will think that there is a positive probability that a dot will appear. This causes the agent to actively seek these histories out by performing complicated maneuvers that do not in reality produce dots and often lead to its death. This effect is exacerbated with more data because the learned abstractions for these models tend to be finer with more training data. This leads to more accurate abstractions, but also makes it more likely that history suffixes will be missing from the training data. PP-POMDPs do not suffer from this particular issue, and result in far better performance.

The performance baselines shown (dashed grey lines) are similar to those from the Brick Breaker example. The line marked "Perfect Model" is the performance of UCT using a perfect model of Snake. The line marked "Most Likely" is the performance of UCT using a model that always predicts the most likely outcome (but is perfect

in every other way). The line marked "Fully Observable" is the performance of UCT using a model that fails to capture the important partially observable components of the domain. Specifically, the model does not remember where the target dot was when it disappeared, and therefore thinks it might appear at random in any given time-step. The learned CPMs using both representations outperform the "Fully Observable" model and so must be capturing the partially observable dynamics in the domain to some degree. The learned CPM using PP-POMDPs obtains performance comparable to that of the determinized perfect model (within a standard deviation). Both CPMs obtain performance greater than the "Fully Observable", indicating that they are learning at least to some degree to keep track of where the invisible dot is. The prediction performance results will provide some insight into the reasons behind the performance gap between the PP-POMDP CPM and the perfect model.

Results: Prediction Performance As in the Brick Breaker domain, the average log-likelihood ratio, the average percentage of training trajectories dropped from the likelihood calculation, and the RMSE of the one-step predictions made by the learned models are reported in Figure 5.17. Figures 5.17(c) and 5.17(d) also show the average prediction error of some notable model types: the models that predict whether their associated pixel will be the head when the head is nearby, the models that predict whether their associated pixel will be the tail when the tail is nearby, and the models that predict the color of a white pixel when the head is nearby. These are the most difficult partial models to learn in this domain.

Neither type of CPM obtains particularly good prediction performance overall. Both clearly make substantial prediction errors resulting in low LLR and relatively high RMSE. However, they do obtain good prediction error for some particularly important partial models. The head models in particular make very accurate predictions (even reaching an average RMSE of zero with sufficient training data). The

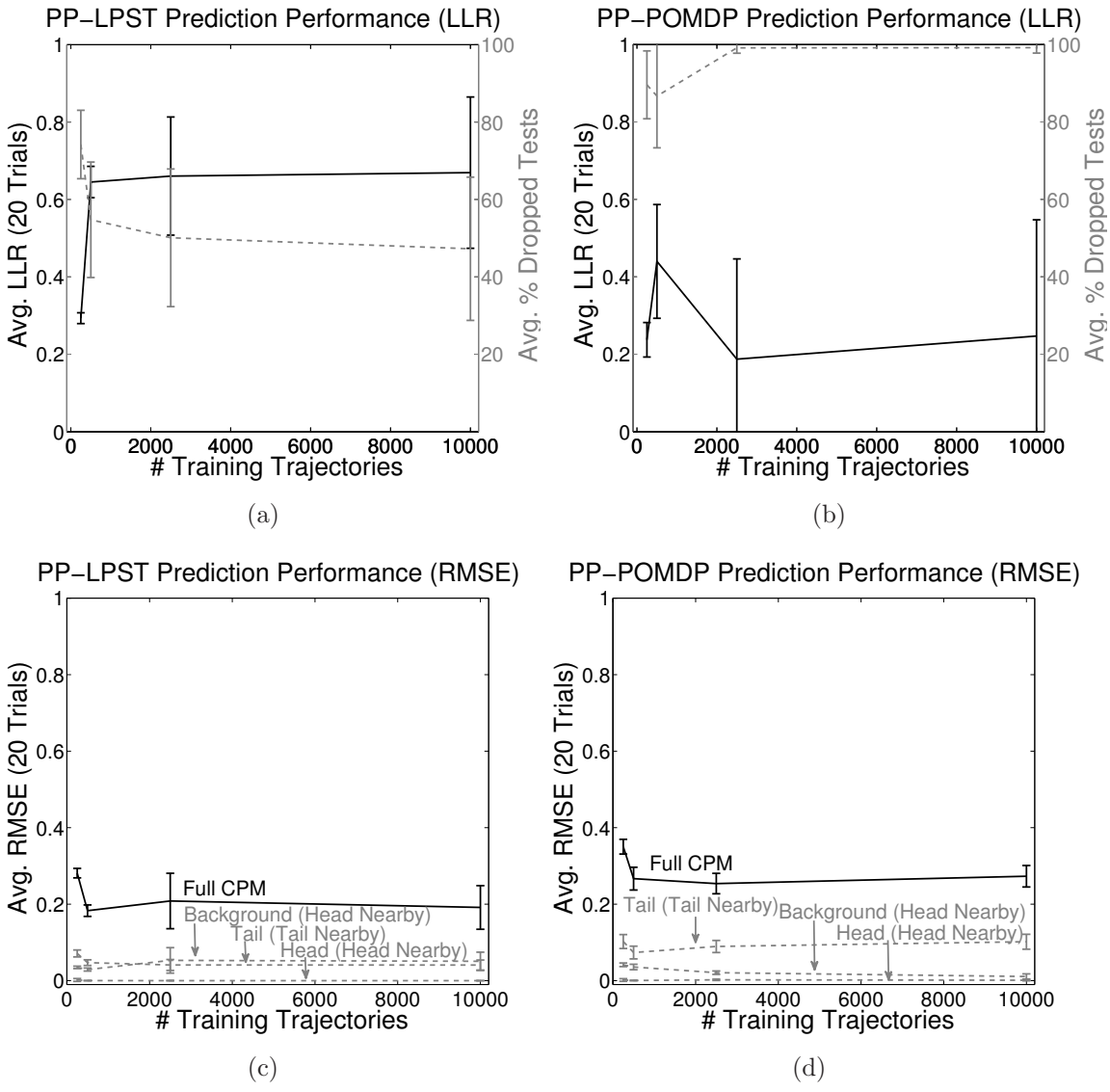
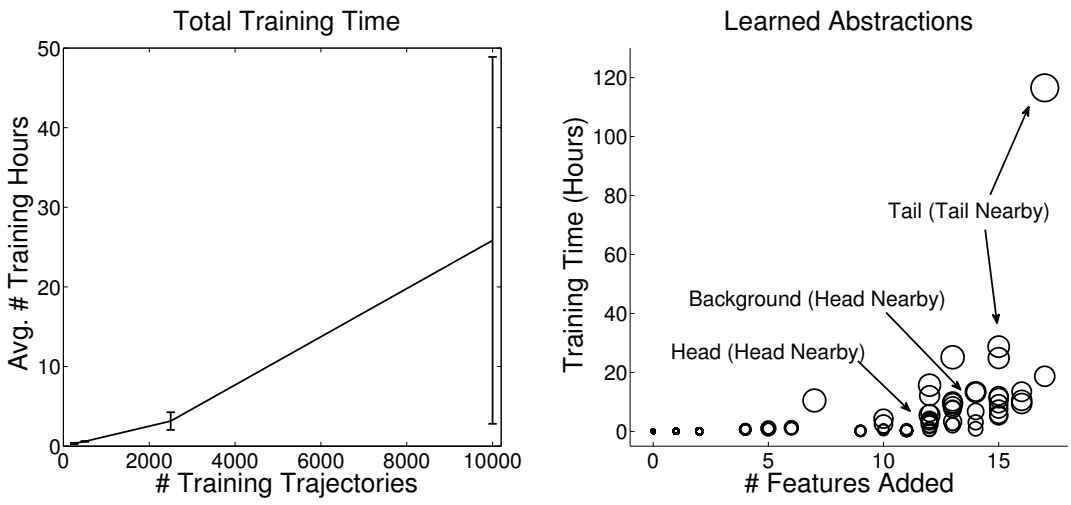


Figure 5.17: Prediction performance results for Snake.



(a) (b)

Figure 5.18: Training time results for Snake.

PP-LPSTs and the PP-POMDPs have the most trouble with different partial models. The PP-LPSTs learn to make significantly better predictions about the movement of the tail than the PP-POMDPs. Incorrectly predicting the tail's movement is not likely to drastically impact planning performance, however, since the head must already avoid any pixels where the tail might go (i.e. pixels that contain the snake's body). That said, the PP-POMDP's highly inaccurate predictions about the tail may at least partly explain the difference in planning performance between UCT using the perfect model and UCT using the PP-POMDP-based CPM. In contrast, the PP-POMDPs do a much better job than the PP-LPSTs of learning the partial model that predicts whether a dot will appear when the head is nearby. The errors made by the PP-LPSTs regarding when dots will or will not appear lead directly to the poor planning performance seen above.

Results: Training Time The average total training time in Snake is shown in Figure 5.18(a). It takes substantially longer to learn a CPM of Snake than of Brick Breaker. In part this is because there are simply more distinct types of models to learn. However, Figure 5.18(b) shows that the bulk of training time is taken up by

one particular type of model: the model that predicts where the tail will move. As in the Brick Breaker results, this graph plots all learned abstractions in all trials with 10,000 training trajectories. The training time of each abstraction is plotted against the number of features used in the abstraction. The diameter of each circle represents the number of abstract observations in each abstraction (the range was from 1 to 59). The tail models are particularly expensive to learn because they require a large number of features (they must attend to the movement of the head and the tail) and they are active in many time-steps (whenever the head *or* the tail is nearby). Notably, making good predictions about the tail's movement does not seem to be necessarily critical to performance, as evidenced by the PP-POMDP-based CPM's planning results. As such, automatic methods for determining which predictions are important and which can be made very approximately could save the learning algorithm a great deal of effort in this domain.

5.4 Summary

Key points from this chapter:

- Partial models can be combined to form a more complete, compositional model of the environment.
- This chapter introduced *collections of partial models* (CPM).
  - Each partial model in a CPM makes abstract predictions about the next observation in some particular histories of interest, *conditioned* on the outcomes of some other models.
  - Allowing models' predictions to condition on other models' outcomes explicitly models correlation between models' predictions.

- Under some conditions on the component partial models, a CPM can be a complete, accurate model of the primitive system.
  - If the various abstract features predicted by the partial models *jointly* specify a unique primitive observation, then the CPM is a complete model.
  - If each model's predictions are conditionally independent of other models' outcomes, given its conditioned outcome, the CPM makes accurate predictions (assuming each partial model makes accurate predictions).
- Collections of partial models were compared to dynamic Bayes nets (DBNs) as they are both structured models of partially observable systems.
  - The main difference: CPMs express their structure in terms of the conditional independence of observable events, while DBNs express their structure in terms of the conditional independence relationships amongst hidden variables.
    - Each partial model in a CPM can be trained independently (DBN training is global).
    - CPM structure is verifiable (DBN structure is not).
    - The models in a CPM may repeat effort because each must maintain its own state information (in a DBN state information is shared by all variables).
  - CPMs are best applied when the environment can be decomposed into many simple models, each relying on relatively disjoint state features.
- CPMs were trained on two high dimensional arcade game environments, using the methods presented in the previous chapters.
  - The learned models were shown to make accurate predictions and to be useful for model-based planning.

CHAPTER 6

Concluding Remarks

The idea of learning, and composing, many models associated with distinct facets of the world is both natural and arguably necessary in complex, compositional environments. As such, methods for learning *partial models* that are responsible for making only some predictions about the future, in only some particular situations have been developed in many different contexts, especially under the Markov assumption. As has been demonstrated in the preceding chapters, the problem of learning partial models in the partially observable setting presents unique challenges. This thesis represents an attempt to lay some general foundations for solving this problem.

6.1 Summary of Contributions

The first contribution, from which all the others spring, is the formalization of a partial model using the concepts of *tests of interest* and *histories of interest* (Section 1.1.3). This establishes a language for specifying a broad class of partial models that incorporates the main intuitions surrounding what a partial model ought to be (something that makes only certain predictions in certain situations), not to mention specifically generalizing many existing examples of partial models. Most importantly, this formal description of a partial model provides a concrete learning problem for study: learn a model that makes predictions for the tests of interest in the histories

of interest, ideally as simply as possible.

The main way one can exploit having only a limited set of predictions to make in the Markov case is by ignoring irrelevant details in the state via an abstraction. Chapter 2 presented several results regarding the generalization of this idea into the partially observable case: observation abstraction. Specifically, formal properties of an abstraction were defined that, if satisfied, allow an *abstract* model to be used to make accurate predictions for the tests of interest (*expressiveness* in Section 2.2.1 and *accuracy* in Section 2.2.2). The main contribution of this chapter was the first provably sound, finite-time algorithm for constructing such an abstraction in the partially observable case (Section 2.3.2), as well as a tight worse-case bound on the complexity of determining whether an abstraction is accurate, in terms of the complexity of the primitive system (Theorem 2.10). This algorithm was also adapted to provide the first provably sound, finite-time algorithm for finding a homomorphic abstraction of a partially observable system.

The end of Chapter 2 pointed out an issue with learning an abstract model that does not pose a challenge in the Markov case. Because standard methods for learning partially observable models learn inherently *generative* models, even an *abstract* model will generally make more predictions than are directly of interest. A generative model using an accurate abstraction will make *all abstract predictions*, including predictions for the tests of interest, but also including predictions about any details the abstraction has deemed necessary to make accurate predictions (but that are not themselves of interest). In fact, it can be the case that the abstract model devotes most of its complexity to making predictions the model will never be asked to make. The main contribution of Chapter 3 is the development of prediction profile models, which address this issue by allowing one to learn a *non-generative* model that makes *only* the predictions of interest, and no others. Chapter 3 also introduced a learning algorithm for prediction profile models under the assumption that the predictions of interest

take on only a finite number of distinct values. Like an abstraction, the method for learning prediction profile models is to learn a transformation of the training data, and then learn a model by applying an existing method to the new data.

The main contributions of Chapter 4 were two-fold. First, the formalism of *histories of interest* and the *bridging test system* were introduced, which allowed results from the previous chapters to be extended to the case where a model only needs to make predictions in certain histories. Second, an algorithm for learning an accurate abstraction was presented that, while it lacks the theoretical guarantees of the algorithm presented in Chapter 2, is significantly more practical. Though the general form of this algorithm is a natural application of existing ideas, a number of practical considerations specific to the partially observable case were addressed.

Chapter 5 discussed one of the main motivations behind learning partial models in the first place: composing partial models to form a more complete model. A structured representation called the *collection of partial models* (CPM) was introduced. Section 5.1.4 provides conditions on the component models that result in a CPM that can be used as a complete, accurate model. A CPM's structure is entirely expressed in terms of conditional independence relationships between observable events (as opposed to a DBN, which expresses its structural assumptions in terms of conditional independence between hidden variables). As a direct result, the component models of a CPM can be trained independently (as opposed to a DBN which requires global training). Furthermore, the correctness of the structure of a CPM can be tested via statistical analysis of the data (the structure of a DBN cannot be verified or falsified). This fact leads to a straightforward extension of the abstraction learning algorithm presented in Chapter 4 to additionally learn some of the structure of a CPM, and may have further implications for structure learning. The weakness of a CPM is that the partial models do not share state information. So, if multiple models make use of the same state information, they must all maintain it for themselves, possibly duplicating

effort. Thus, a CPM is best applied when the world can be decomposed into components that depend on relatively disjoint aspects of the state. Collections of partial models were learned in two high-dimensional arcade game examples. The learned models were shown to make accurate predictions and to be useful for model-based planning techniques.

6.2 Discussion and Future Directions

The methods presented in this thesis have been demonstrated to learn partial models in partially observable problems that are far too complex to be modeled completely (at least in an unstructured way). The experiments in Section 5.3 in particular illustrated their application to problems with over 10^{20} observations and underlying states. That said, there are many possible extensions to this work that could significantly expand its applicability in more natural domains (most importantly, the physical world in which we live). This section will briefly discuss some of the limitations the work presented here, and some possible extensions that may begin to address those limitations.

6.2.1 Relational Predictions

The experiments in Section 5.3 illustrated the limitations of propositionally defined partial models. Recall, for instance, in Brick Breaker, that there was a separate partial model for *every single brick*. In Snake, there were several models for each individual pixel that made predictions about the head, the tail, and other parts of the system. In both cases, the CPM consisted of tens of thousands of partial models. While parameter tying did alleviate some of the challenges inherent in training so many models, this is nevertheless a counter-intuitive way to structure a representation of knowledge about the world. It would be far more natural, and more in keeping with the inspiration behind this work, to have a model that represents "how bricks

behave" or "how the snake's head behaves" in general. This requires models that can make relative, rather than absolute predictions. Rather than, for instance, making a prediction whether a specific pixel will be a specific color when the snake's head is nearby, what is required is a model that makes predictions about the colors of some pixels *relative* to the head's location. Relational representations (like STRIPS models, discussed in Chapter 1) are able to express predictions like these through the use of variables and predicates. Extending the definition of tests of interest to allow for relational predictions could significantly impact the compactness of a CPM and allow for more sophisticated generalization and parameter tying.

6.2.2 Continuous Systems

This thesis has focused on discrete dynamical systems, mainly because doing so allowed a particularly clean exploration of the central issues of learning partial models. In practice, many systems have both continuous observations and continuous state, which imposes a number of additional challenges. The methods presented here *do* straightforwardly apply to continuous systems under some special circumstances. If the predictions being made are of some discrete features of the continuous observations ("Will I need a sweater?" as opposed to "What will the temperature be?") and if the abstractions are also composed of discrete features, then the methods presented in this thesis apply without alteration. That is, one might be able to use the methods presented here to learn discrete partial models of continuous dynamical systems. Mugan & Kuipers (2009), for example, take this approach in the Markov case, both learning which discrete features should be predicted in order to make good decisions, and learning discrete partial models to make those predictions.

The problem of learning *continuous* partial models presents more open questions. The essential ideas of abstraction and prediction profile models still likely apply, though the specific procedures presented here may not. Learning an accurate ab-

straction with continuous features may be particularly challenging because even an *abstract* history is unlikely to ever be experienced twice. As such, it may be impossible to decouple abstraction and model learning by first learning an abstraction and then learning an abstract as has been done in this work. Instead, it may be necessary to evaluate proposed abstractions by learning an abstract model and evaluating *its* accuracy.

The main motivation behind prediction profile models also has purchase in the continuous setting. Typical methods for learning models of partially observable systems in continuous systems, much like their discrete valued counterparts, learn *generative* models. As such, the non-generative approach of prediction profile models may provide similar benefits in the continuous setting. That said, in a continuous system, there will typically be infinitely many tests of interest, making the extensive vector representation of prediction profiles used in Chapter 3 impossible. Instead, prediction profiles might be represented in a parametric form (for instance, the mean and variance of a Gaussian). The main idea of Chapter 3 (though not the specific method) could still then be applied: learn a model of the dynamics of these distribution parameters, rather than the dynamics of the system itself.

6.2.3 Learning the Structure of a CPM

When learning a collection of partial models in Chapter 5, the structure (that is, the prediction responsibilities of each model) was assumed to be given. As seen in Section 5.3.1, different structural choices can significantly affect learning performance so it would be advantageous to be able to *learn* a good structure from data (or at least adapt a provided structure). One aspect of structure learning was, at least in part, addressed: learning for each partial model which other partial models' abstract observations it should condition its predictions upon. That still does not address the main question, "Which partial models should be learned in the first place?"

Ultimately, the work presented here on *how* to learn a partial model is meant to provide a foundation on which to build answers to this challenging question.

One of the clearest criteria for selecting which models to learn (or equivalently which predictions to make) is that an agent should make predictions that help it make good decisions. So, for instance, an agent might begin by attempting to learn a model that makes predictions only about whether it will be rewarded. This model will almost certainly have to attend to observation features beyond the reward signal itself. So, it may then make sense to learn models that make predictions about those features...and then learn models that make predictions about features that help to predict *those* features, and so on. Another interesting approach is to learn to make predictions that *can* be made, given the constraints of the agent. In any complex environment, there are bound to be predictions that are easy to make and predictions that are difficult to make. Furthermore, a given finite training data set might provide enough experience to learn to reliably make some predictions, but not others. As such, a possible direction for future research is identifying predictions that can be made simply and reliably, in order to focus modeling efforts appropriately. The work of Mugan & Kuipers (2009) shows promise in this direction in the Markov case. They start by making very abstract predictions and refine their predictions of interest over time, seeking abstract predictions that can be made deterministically. It is possible that similar ideas may be adapted to the partially observable case.

6.2.4 Planning with CPMs

While the experiments in Section 5.3 demonstrate a planning algorithm using learned CPMs to obtain good performance, that algorithm does not exploit the specific advantages of having a CPM. Many planning algorithms like the one used in Chapter 5.3 expect a generic generative model, specifically one that can simulate the world in full detail one step at a time. In complex worlds, like the one in which we

live, this is an unreasonable approach for a few reasons.

First, one simply cannot expect to have a complete and accurate model, even if one applies a structured representation. While the CPMs learned in Section 5.3 *were* complete models, in more naturalistic domains, obtaining a complete, accurate model is not a reasonable goal. As discussed in Chapter 1, there are many phenomena in the world that are very complex and any reasonable agent will be forced to prioritize its resources on capturing phenomena that can be understood simply and that are most important for making good decisions. As such, an interesting direction for future research is the development of planning algorithms that work well with collections of partial models that simply do not make predictions (or at least make very innaccurate predictions) for some complicated aspects of the world. Such a planning algorithm would need to be robust to missing or innacurate predictions while still exploiting the predictions the model is confident about.

Second, a planning algorithm that reasons on the most primitive level (one primitive observation at a time) fails to take into account the rich structure present in many problems of interest. Many qualitatively different phenomena occur on multiple time scales and levels of abstraction at once. Even arcade games, like the ones shown in Section 5.3 possess this quality. A planning algorithm should be able to simultaneously deal with the low level reasoning necessary to take actions from time-step to time-step, but also more abstract planning like achieving some sub-goal or setting up a more favorable future situation, and even more abstract plans like obtaining an end-of-level bonus, or beating a high score. One particularly exciting possibility that was not explored in this thesis is the collection of partial models as a *heterogeneous* representation that might be able to capture some of this richness. One can imagine a CPM with many models making predictions on different levels of abstraction and each employing different underlying modeling representations, as best suits their prediction responsibilities. Such a model could provide a planning algorithm with

many qualitatively different types of knowledge all at once. Though significant work remains in determining how best to compose such different models and how to exploit such a model during planning, the results in this thesis may provide the basic building blocks of this type of rich knowledge representation.

APPENDICES

APPENDIX A

Proofs from Chapter 2

This appendix contains proofs of the main theoretical results from Chapter 2.

A.1 Proof of Proposition 2.2

Proposition 2.2. *Consider a dynamical system, an abstraction η, and a refinement η' of η. The linear dimension of the abstract system corresponding to η, n^η, is no greater than $n^{\eta'}$, the linear dimension of the abstract system corresponding to η'.*

Proof. Recall that the linear dimension is the rank of the system dynamics matrix (defined in Section 1.1.2). Let n be the linear dimension of the primitive system. Using Equations 2.1 and 2.2 one can construct the system dynamics matrix of the abstract system, $D^{\eta'}$ (whose columns correspond to abstract tests and whose rows correspond to abstract histories) from the system dynamics matrix of the original system, D (whose columns and rows correspond to primitive tests and histories). First, consider an intermediate matrix $D^{\eta'}_{test}$ whose columns correspond to abstract tests but whose rows correspond to *primitive* histories. Note that, by Equation 2.1, each column in $D^{\eta'}_{test}$ is simply the sum of several columns in D. As such, the columns of $D^{\eta'}_{test}$ lie in the span of the columns of D so $rank(D^{\eta'}_{test}) \leq rank(D)$. Now note

that, by Equation 2.2, each row in the abstract system dynamics matrix $D^{\eta'}$ can be computed as a linear combination of rows in $D^{\eta'}_{test}$. Since the rows of $D^{\eta'}$ lie in the span of the rows of $D^{\eta'}_{test}$, it must be that $rank(D^{\eta'}) \leq rank(D^{\eta'}_{test}) \leq rank(D)$. As such, $n^{\eta'} \leq n$. Now, note that, because η' is a refinement of η, abstract tests under η correspond to sets of tests under η' (and similarly histories). As such, D^{η} can be constructed in precisely the same way from $D^{\eta'}$. Thus, the final relationship is $n^{\eta} \leq n^{\eta'} \leq n$. □

A.2 Proof of Theorem 2.9

This result will require the use of some supporting lemmas. The first lemma shows that any accurate refinement *must* split any pair of observations that have a violation.

Lemma A.1. *Consider two primitive observations $o_1, o_2 \in \mathcal{O}$. If there exists $a \in \mathcal{A}$, $t \in \mathcal{T}^I$ and primitive sequences $h_1 \in \mathcal{H}$, $h_2 \in \mathcal{T}$ such that $p(t|h_1ao_1h_2) \neq p(t|h_1ao_2h_2)$ then in any accurate abstraction η, $\eta(o_1) \neq \eta(o_2)$.*

Proof. In any abstraction that does not split o_1 and o_2, $h_1ao_1h_2$ and $h_1ao_2h_2$ will belong to the same abstract history. Since they have different predictions for t, such a refinement could not possibly be accurate. □

The following result will be used to show that, in fact, it is sufficient to compare *only* pairs of histories that differ in one time step.

Lemma A.2. *Consider two primitive histories with the same action sequence $h = a_1o_1...a_ko_k$ and $h' = a_1o'_1...a_ko'_k$. Let $h_i = h^{[1...i]}h'^{[i+1...k]}$ be the concatenation of the length i prefix of h and the length $k-i$ suffix of h'. If for some test t and all $i \in \{0, 1, ..., k\}$ $p(t|h_i) = p(t|h_{i+1})$, then $p(t|h) = p(t|h')$.*

Proof. The result is easy to see by transforming h into h' by swapping only one observation at a time. For any t, $p(t|h) = p(t|h_k) = p(t|h_{k-1}) = ... = p(t|h_0) = p(t|h')$. □

Note that for any i, h_i and h_{i-1} differ in only one time-step (the ith time-step). As such, by the contrapositive of Lemma A.2, it can inferred that for any pair of histories h and h' of length k with $p(t|h) \neq p(t|h')$ for some test t, there exists *another* pair of histories $h^{[1\ldots i-1]}a_i o_i h'^{[i+1\ldots k]}$ and $h^{[1\ldots i-1]}a_i o_i' h'^{[i+1\ldots k]}$ that also disagree on the prediction of t. These two histories differ on only one time step. So, in searching for a split, it suffices to consider every pair of primitive observations o_1 and o_2 and check for all $h_1 \in \mathcal{H}$, $h_2 \in \mathcal{T}$, $a \in \mathcal{A}$, and $t \in \mathcal{T}^I$, whether $p(t|h_1 a o_1 h_2) = p(t|h_1 a o_2 h_2)$. If any one of these equalities does not hold, o_1 and o_2 must be distinguished.

Theorem 2.9 follows closely from these two results.

Theorem 2.9. *If all observations are comparable and η is the abstraction induced by the relation "Have no violations," then η is* the unique *coarsest, accurate abstraction. Furthermore, there is no accurate abstraction that results in an abstract system with a smaller linear dimension than the abstract system corresponding to η.*

Proof. Let η be the abstraction induced by the equivalence classes of "Have no violations." Lemma A.2 shows that if all pairs of observations with a violation are distinguished, then η is accurate. Lemma A.1 shows that η is, in fact, a coarsest accurate abstraction, since all pairs of observations with a violation *must* be split in any accurate refinement. As such, merging any such pair would result in an inaccurate abstraction. Furthermore, *any* accurate refinement must be a refinement of η, since it must make at least the distinctions made by η (due to Lemma A.1). As a result, due to Proposition 2.2, any other accurate abstraction η' must induce an abstraction system with linear dimension no less than that induced by η. □

A.3 Proof of Theorem 2.10

For the purposes of this argument, it will be useful to introduce a transformation of the system dynamics matrix D, called D^2. Each row in D^2 corresponds

to *a pair* of histories $\langle h_1, h_2 \rangle$, and each column a pair of tests $\langle t_1, t_2 \rangle$. Each entry $D^2_{\langle h_1,h_2\rangle,\langle t_1,t_2\rangle} \stackrel{\text{def}}{=} p(t_1|h_1)p(t_2|h_2)$. It is straightforward to bound the rank of D^2:

Lemma A.3. *If* $rank(D) = n$*, then* $rank(D^2) \leq n^2$*.*

Proof. Let Q be a set of n tests corresponding to columns that form a basis for D. Then for any test t and history h, $p(t|h) = \sum_{q \in Q} p(q|h) m_t(q)$, where $m_t(q)$ is some scalar weight. Then

$$\begin{aligned} D^2_{\langle h_1,h_2\rangle,\langle t_1,t_2\rangle} &= p(t_1|h_1)p(t_2,h_2) \\ &= \sum_{q \in Q} \sum_{q' \in Q} p(q|h_1)p(q'|h_2)m_{t_1}(q)m_{t_2}(q') \\ &= \sum_{\langle q,q'\rangle \in Q \times Q} D^2_{\langle q,q'\rangle,\langle h_1,h_2\rangle} m_{\langle t_1,t_2\rangle}(\langle q,q'\rangle). \end{aligned}$$

So the columns corresponding to $Q \times Q$ are a column basis of $D^2$and $rank(D^2) \leq |Q \times Q| = n^2$. □

Using this fact, one can prove a general result about the system dynamics matrix which will lead directly to the bound in Theorem 2.10.

Theorem A.4. *Let the linear dimension of a system be n and consider two primitive histories with the same action sequence:* $h_1 = a_1 o^1_1 ... a_k o^1_k$ *and* $h_2 = a_1 o^2_1 ... a_k o^2_k$*. If for some test t* $p(t|h_1) \neq p(t|h_2)$*, then there exists a (non-consecutive) subsequence* $\{i_1, i_2, ...i_j\}$ *of* $\{1, ..., k\}$ *such that* $j < n^2$ *and:*

$$p(t|a_{i_1} o^1_{i_1} a_{i_2} o^1_{i_2} ... a_{i_j} o^1_{i_j}) \neq p(t|a_{i_1} o^2_{i_1} a_{i_2} o^2_{i_2} ... a_{i_j} o^2_{i_j}).$$

Proof. If $k < n^2$, this result holds trivially, so assume $k \geq n^2$. This result will use a $(k+1) \times (k+1)$ matrix, X, constructed from entries of D^2, defined entry-wise for all

$0 \leq i, j \leq k$:

$$X_{ij} \stackrel{\text{def}}{=} p(h_1^{[j\ldots k]}t|h_1^{[1\ldots i]})p(h_2^{[j\ldots k]}|h_2^{[1\ldots i]}) - p(h_2^{[j\ldots k]}t|h_2^{[1\ldots i]})p(h_1^{[j\ldots k]}|h_1^{[1\ldots i]}),$$

where $h^{[j\ldots k]}$ is the sequence of actions and observations in h starting from the jth time-step and ending with the kth time-step. So, the entries in X correspond to histories that are prefixes of h_1 and h_2 and tests that are suffixes of h_1 and h_2 concatenated with the test t.

Note that the rows of X are a subset of the rows of D^2 and the columns are weighted sums of the columns of D^2. Therefore, $rank(X) \leq rank(D^2) \leq n^2$.

If an entry X_{ij} is zero, then by simple algebraic manipulation, it must be that

$$\frac{p(h_1^{[j\ldots k]}t|h_1^{[1\ldots i]})}{p(h_1^{[j\ldots k]}|h_1^{[1\ldots i]})} = \frac{p(h_2^{[j\ldots k]}t|h_2^{[1\ldots i]})}{p(h_2^{[j\ldots k]}|h_2^{[1\ldots i]})}$$

and, by definition of conditional probability,

$$p(t|h_1^{[1\ldots i]}h_1^{[j\ldots k]}) = p(t|h_2^{[1\ldots i]}h_2^{[j\ldots k]}).$$

These predictions are predictions of t given a history that is formed by "erasing" some time-steps in the middle of h_1 (and h_2). Contrapositively, if $p(t|h_1^{[1\ldots i]}h_1^{[j\ldots k]}) \neq p(t|h_2^{[1\ldots i]}h_2^{[j\ldots k]})$ then $X_{ij} \neq 0$.

Since $p(t|h_1) \neq p(t|h_2)$, it must be that $X_{i(k-i)} \neq 0$ for any i, because these entries correspond to not removing any time-steps from the histories. These entries lie on the diagonal of X. If $X_{ij} = 0$ for all i and all $j < k - i$, then X would be triangular with non-zero entries on the diagonal, and therefore have full rank: $k + 1 > n^2$, a contradiction. As such, there *must be* some i and $j < k - i$ such that $X_{ij} \neq 0$. That is, if $p(t|h_1) \neq p(t|h_2)$ and $k \geq n^2$, then another, shorter pair of histories can be found

that disagree on t by removing the same time-steps from both h_1 and h_2. Specifically:

$$p(t|a_1o_1^1...a_io_i^1a_{k-j+1}o_{k-j+1}^1...a_ko_k^1) \neq$$
$$p(t|a_1o_1^2...a_io_i^2a_{k-j+1}o_{k-j+1}^2...a_ko_k^2).$$

The resulting subsequence of indices has length $k' = i + j$. If $k' \geq n^2$, simply repeat the argument to remove more substrings until a subsequence with length less than n^2 is found. □

Theorem 2.10 follows immediately from Theorem A.4.

Theorem 2.10. *If the linear dimension of the system is n, and a violation $\langle t, a, h_1, h_2\rangle$ exists for observations $o_1, o_2 \in \mathcal{O}$ then a violation $\langle t, a, h_1', h_2'\rangle$ exists for o_1 and o_2 with $length(h_1'ao_1h_2') < n^2$.*

Proof. Note that, since $h_1ao_1h_2$ and $h_1ao_2h_2$ only differ in one time-step, the subsequences from Theorem A.4 that disagree on t must still contain that step, otherwise they would be equal. As such, the subsequences comprise a violation for o_1 and o_2 using histories with length less than n^2. □

This completes the proof of Theorem 2.10. It may be possible to tighten these bounds in terms of other parameters of the primitive system. The quadratic dependence on n, however, is necessary in the worst case, as the following example demonstrates.

Consider the family of uncontrolled POMDPs (or HMMs), indexed by k, of the form given in Figure A.1. The distribution over observations is shown next to each state. State 0 is the initial state. The linear dimension is equal to the number of hidden states, $n = 2k$. Imagine that η distinguishes a from b and c, but aggregates b and c into the set observation X and there is a single test of interest: $a^{k-1}X$, which

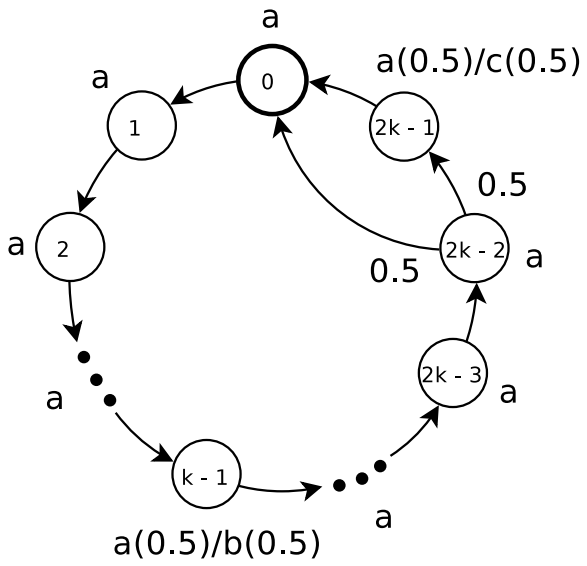


Figure A.1: A family of abstract systems where the only violations occur at histories of length quadratic in the linear dimension of the system

corresponds to seeing the observation a for the next $k-1$ steps, followed by either b or c.

This abstraction is *not* accurate because $p(a^{k-1}X|hb) = 0.25$ and $p(a^{k-1}X|hc) = 0.5$ for *any* history h. That is, if one sees c, one knows that after $k-1$ steps, one will be in state $k-2$ and will therefore reach state $k-1$ and have a 0.5 probability of seeing b in the next time-step. If one sees b, then after $k-1$ steps one will be in state $2k-2$ and will have a 0.5 chance of reaching state $2k-1$ and then a 0.5 chance of seeing c in the next time-step for total of 0.25. As such, b and c must be distinguished in any accurate abstraction.

Furthermore, there *do* exist histories h such that $p(b|h) > 0$ and $p(c|h) > 0$, so there are violations between b and c. The shortest such history is $a^{(n+1)k-1}$, as will be argued next. Clearly, if it weren't for the stochastic transition in state $2k-2$, or if the bs and cs were emitted deterministically there would be no history after which b and c both have positive probability. However, if all one sees are as, the stochastic transition adds uncertainty about the state. After seeing $k-1$ as, one is definitely

in state $k-2$ and could only possibly observe a or b in the next time-step. However, after going around the loop and seeing $k-1+2k$ as in total, one cannot tell if one is in state $k-2$ or state $k-1$ (because one might have "slipped" in state $2k-2$). After $k-1+2(2k)$ as, one might be in state $k-2$, state $k-1$, or state k. It is only after $k-1+k(2k)$ as that the uncertainty grows enough that one might be in any state from $k-2$ to $2k-2$ (and thus one might see an a b, *or* a c in the next time-step). Substituting $n=2k$, one can see that $k-1+k(2k)=(n+1)k-1=\frac{1}{2}(n^2+n-2)$. Since this is the shortest history after which b and c both have positive probability, it is also the shortest history that can be involved in a violation between b and c. Thus, in this family of systems, the shortest history at which a violation occurs has length quadratic in the linear dimension. This demonstrates that the bound in Theorem 2.10 is tight in the worst case.

A.4 Proof of Theorem 2.11

This analysis will require the concept of the *set test matrix* for an abstraction of interest η^I. This is an infinity-by-infinity matrix like the system dynamics matrix, and each entry is a prediction. In the set test matrix, however, columns correspond to the abstract tests under η^I, $\mathcal{T}^{\eta^I}$ only (the rows still correspond to primitive histories). The rank of this matrix, $\bar{n}$, can be at most n and must be at least n^{η^I}, the rank of the abstract system dynamics matrix induced by η^I (which has abstract tests for columns and abstract histories for rows).

The set test matrix also has the useful property that there always exists a linear column basis corresponding to abstract tests of length at most $\bar{n}$. This can be shown as a straightforward extension of the result by Wolfe (2009), which showed that there always exists a basis of the system dynamics matrix corresponding to tests of length less than the rank of the system dynamics matrix. Using this fact, one can show that it is only necessary to consider finitely many tests in the violation search.

Theorem 2.11. *If the linear dimension of the system is n, $\mathcal{T}^I = \mathcal{T}^{\eta^I}$ for some abstraction η^I, and a violation $\langle t, a, h_1, h_2 \rangle$ exists for observations $o_1, o_2 \in \mathcal{O}$ then a violation $\langle t', a, h_1, h_2 \rangle$ exists for o_1 and o_2 with length$(t') \leq n$.*

Proof. Note that if $\bar{Q}$ is any set of abstract tests whose columns form a basis for the set test matrix and for some pair of histories $p(\bar{q}|h) = p(\bar{q}|h')$ for all $\bar{q} \in \bar{Q}$, then $p(T|h) = p(T|h')$ for any abstract test $T \in \mathcal{T}^{\eta^I}$. By contrapositive, then, if a violation exists involving some set test T, there must also be a violation involving one of the tests in $\bar{Q}$. Since there exists a $\bar{Q}$ consisting of tests length $\bar{n}$ or less, the result immediately follows. □

APPENDIX B

Incomparable Observations

Recall that a pair of observations o_1, o_2 are *comparable* if there exists some history h and action a such that $p(hao_1|\emptyset) > 0$ and $p(hao_2|\emptyset) > 0$. Observations that are *incomparable* are notable because, by definition, they cannot have a violation and can, as a result, cause intransitivities in the "Have no violations" relation.

First note that, by Lemma A.2, whether two incomparable observations are distinguished cannot affect accuracy, since they cannot cause a violation. Furthermore, it is possible to identify incomparable observations in the course of the violation search by the following result, which is presented without proof (the argument is essentially the same as that of Theorem A.4).

Lemma B.5. *If* $p(ao_1|h)p(ao_2|h) = 0$ *for all histories* h *with* $length(h) < n^2$*, then* $p(ao_1|h)p(ao_2|h) = 0$ *for all histories* h *of any length.*

Thus, it is possible to construct a minimal, accurate refinement by respecting all comparisons that can be made and otherwise grouping incomparable observations arbitrarily. One procedure for achieving this would first group together all pairs of observations that *are* comparable and have no violations. This will produce an accurate abstraction, but not a *coarsest* accurate abstraction. Two abstract observations can

still be safely merged if all primitive observations contained in one abstract observation are incomparable with all members of the other abstract observation (otherwise, by construction, there is a violation between primitive observations contained in the two abstract observations). Thus, to find a coarsest accurate abstraction, one can simply merge pairs of abstract observations until no more safe merges are available. The result will be a coarsest accurate abstraction since no two observations with a violation will be grouped together and every pair of sets has at least one violation and thus, cannot be merged while retaining accuracy.

This simple procedure could result in any of several abstractions, depending on how incomparable sets are merged. All of them will be coarsest, accurate refinements. However, they *may* differ in the linear dimension of the aggregate system they induce, which is, in some sense, the true decision criterion. Heuristics for grouping incomparable observations in order to produce a compact abstract model could be an avenue for future research.

APPENDIX C

Proofs from Chapter 3

C.1 Proof of Proposition 3.10

This result will follow straightforwardly from a general fact about dynamical systems. Let $h^{[i\ldots j]}$ be the sequence of actions and observations from h starting with the ith time-step and ending with the jth time-step. The following two results will show that if some test t *ever* has positive probability, then it must have positive probability at some history with length less than the linear dimension of the system.

Lemma C.6. *For any test t and history h, if $p(t|h) > 0$ and the linear dimension of the system is n and $length(h) = k \geq n$, then $\exists i, j < k - i$ such that $p(t|h^{[1\ldots i]}h^{[j\ldots k]}) > 0$.*

Proof. Note that because $p(t|h) > 0$, $p(h^{[(k-i)\ldots k]}t|h^{[1\ldots i]}) = p(t|h)p(h^{[(k-i)\ldots k]}|h^{[1\ldots i]}) > 0$ for all $0 \leq i \leq k$. Now assume that for all i and all $0 \leq j < k-i$, $p(h^{[j\ldots k]}t|h^{[1\ldots i]}) = 0$ and seek a contradiction. Imagine the submatrix of the system dynamics matrix in which rows correspond to $h^{[1\ldots i]}$ for all $0 \leq i \leq k$ and columns correspond to $h^{[i\ldots k]}t$ for all $0 \leq i \leq k$. This is a $k+1 \times k+1$ matrix. Under the above assumption, this matrix is triangular with positive entries along the diagonal. As such, this matrix is

full rank (rank $k+1$). This is a contradiction since $k+1 > n$ and a submatrix can never have higher rank than the matrix that contains it. □

The next result follows immediately from Lemma C.6.

Corollary C.7. *If the system has linear dimension n and for some test t and history h $p(t|h) > 0$, then there exists a (possibly non-consecutive) subsequence h' of h such that $length(h') < n$ with $p(t|h') > 0$.*

Proof. By Lemma C.6, every history h with length $k \geq n$ such that $p(t|h) > 0$ must have a subsequence h_1 with length $k_1 < k$ such that $p(t|h) > 0$. If $k_1 \geq n$, then h_1 must have a subsequence h_2 with length $k_2 < k_1$. This argument can be repeated until the subsequence has length less than n. □

The consequence of Corollary C.7 is that *every* test that ever has positive probability, must have positive probability following some history of length less than n. With this fact in hand, Proposition 3.10 can now be proven.

Proposition 3.10. *For any deterministic dynamical system with actions $\mathcal{A}$, and observations $\mathcal{O}$, the linear dimension, $n \geq \frac{\log(|\mathcal{A}|-1)+\log(|\mathcal{O}|+1)}{\log|\mathcal{A}|}$.*

Proof. Since the system is deterministic, each history and action correspond to exactly one resulting observation. A history is a sequence of actions and observations. However, since the sequence of observations is uniquely determined by the sequence of actions in a deterministic system, the number of distinct histories of length k is simply $|\mathcal{A}|^k$. At each history there are $|\mathcal{A}|$ action choices that could each result in a different observation. So, the number of observations that could possibly occur after histories of length k is simply $|\mathcal{A}|^{k+1}$. By Corollary C.7, if the linear dimension is n, *all* observations must occur after some history h with $length(h) \leq n-1$. Thus, the

number of observations that can possibly follow histories of length less than n:

$$|\mathcal{O}| \leq \sum_{i=0}^{n-1} |\mathcal{A}|^{i+1} = \frac{|\mathcal{A}|^{n+1} - 1}{|\mathcal{A}| - 1} - 1.$$

Solving for n yields the bound on linear dimension in terms of the number of actions and the number of observations. □

C.2 Proof of Proposition 3.12

Proposition 3.12. *For any system, the set of tests whose corresponding prediction profile system has the highest linear dimension is Q, the set of core tests for that system (as described in Section 1.2.2.1).*

Proof. Recall from the discussion of PSRs in Section 1.2.2.1 that the set of core tests, Q, is a set of tests whose corresponding columns in the system dynamics matrix constitute a basis. The predictions for the core tests at a given history form the *predictive state* at that history. So, the predictive state is precisely the prediction profile for the core tests Q. The prediction for *any* other test can be computed as a linear function of the prediction profile for Q. Note that the prediction profile system for Q is itself an MDP. It was shown in Section 1.2.2.1 how to compute the next predictive state given the current predictive state and an action-observation pair.

Now consider some other set of tests of interest $\mathcal{T}^I$. Because the predictions for Q can be used to compute the prediction for any other test, it must be that there is some function ζ that maps the prediction profiles for Q to the prediction profiles for $\mathcal{T}^I$. In general, multiple predictive states may map to the same prediction profile for $\mathcal{T}^I$ so ζ is a surjection. Now it is easy to see that the prediction profile system for $\mathcal{T}^I$ is the result of applying the observation abstraction ζ to the prediction profile system for Q. Performing observation abstraction on an MDP generally produces a POMDP, but *never* increases the linear dimension, as shown in Proposition 2.2. Hence, the

prediction profile system for any set of tests of interest $\mathcal{T}^I$ has linear dimension no greater than that of the prediction profile system for Q. □

C.3 Proof of Proposition 3.15

Proposition 3.15. *Consider a POMDP with hidden states $\mathcal{S}$, actions $\mathcal{A}$, and observations $\mathcal{O}$. Let $\mathcal{T}^I$ be the set of tests of interest. Let a^i be the action taken at time-step i, s^i be the hidden state reached after taking action a^i, and o^i be the observation emitted by s^i. Now, consider* any *surjection $\sigma : \mathcal{S} \to \mathcal{S}^\sigma$ mapping hidden states to a set of* abstract states *with the following properties:*

1. *For any pair of primitive states $s_1, s_2 \in \mathcal{S}$, if $\sigma(s_1) = \sigma(s_2)$, then for any time-step i and any test of interest $t \in \mathcal{T}^I$, $p(t|s^i = s_1) = p(t|s^i = s_2)$.*

2. *For any pair of primitive states $s_1, s_2 \in \mathcal{S}$, if $\sigma(s_1) = \sigma(s_2)$, then for any time-step i, abstract state $S \in \mathcal{S}^\sigma$, observation $o \in \mathcal{O}$, and action $a \in \mathcal{A}$,*

$$Pr(\sigma(s^{i+1}) = S|s^i = s_1, a^{i+1} = a, o^{i+1} = o) = Pr(\sigma(s^{i+1}) = S|s^i = s_2, a^{i+1} = a, o^{i+1} = o).$$

If such a σ exists, then the prediction profile system for $\mathcal{T}^I$ has linear dimension no greater than the number of distinct beliefs over abstract states, $\mathcal{S}^\sigma$.

Proof. The proof follows similar reasoning to the proof of Proposition 3.12. Note that, because of Property 1 the belief over abstract states at a given history is sufficient to compute the prediction profile. For any history h and any test of interest $t \in \mathcal{T}^I$:

$$\begin{aligned} p(t|h) &= \sum_{s \in \mathcal{S}} \Pr(s|h) p(t|s) = \sum_{S \in \mathcal{S}^\sigma} \sum_{s \in S} \Pr(s|h) p(t|s) \\ &= \sum_{S \in \mathcal{S}^\sigma} p(t|S) \sum_{s \in S} \Pr(s|h) = \sum_{S \in \mathcal{S}^\sigma} p(t|S) \Pr(S|h), \end{aligned}$$

where the third equality follows from property 1: for any $S \in \mathcal{S}^\sigma$, all hidden states $s \in S$ have the same associated probabilities for the tests of interest.

Now, consider the dynamical system with beliefs over abstract states as "observations" and action-observation pairs as "actions." Call this the *abstract belief system*. Just as with the predictive state, because it is possible to compute the prediction profile from the abstract beliefs, the prediction profile model for $\mathcal{T}^I$ can be seen as the result of an observation aggregation of the abstract belief system. As a result, the prediction profile system has linear dimension no greater than that of the abstract belief system.

The rest of the proof shows that, because of Property 2, the abstract belief system is an MDP, and therefore has linear dimension no greater than the number of distinct beliefs over abstract states.

Given the probability distribution over abstract states at a given history h, and the agent takes an action a and observes and observation o, it is possible to compute the probability of an abstract state $S \in \mathcal{S}^\sigma$ at the new history:

$$\begin{aligned}\Pr(S|hao) &= \sum_{s\in\mathcal{S}} \Pr(s|h)\Pr(S|s,a,o) = \sum_{S'\in\mathcal{S}^\sigma}\sum_{s\in S'} \Pr(s|h)\Pr(S|s,a,o) \\ &= \sum_{S'\in\mathcal{S}^\sigma} \Pr(S|S',a,o)\sum_{s\in S'}\Pr(s|h) = \sum_{S'\in\mathcal{S}^\sigma} \Pr(S|S',a,o)\Pr(S'|h),\end{aligned}$$

where the third equality follows from Property 2: for any $S \in \mathcal{S}^\sigma$, all hidden states $s \in S$ have the same associated conditional distribution over next abstract states, given the action and observation.

So, because one can compute the next abstract beliefs from the previous abstract beliefs, the abstract belief system is an MDP, and therefore has linear dimension no greater than the number of distinct abstract beliefs. Because one can compute the prediction profile from the abstract beliefs, the prediction profile system can be constructed by applying an observation abstraction to the abstract belief system.

Thus, the prediction profile system has linear dimension no greater than the number of distinct abstract beliefs. □

APPENDIX D

Proofs from Chapter 4

D.1 Proof of Proposition 4.4

Proposition 4.4. *For any abstraction η and any k,*

$$\begin{aligned}\xi_k(\eta) = \epsilon_k(\eta_0) - \epsilon_k(\eta) &= E_{H\in\mathcal{H}_k^\eta}[D(T|H||T|\eta_0(H))] \\ &= \sum_{H\in\mathcal{H}_k^\eta} p(H|\emptyset) \sum_{t\in\mathcal{T}^I} p(t|H) \log\left(\frac{p(t|H)}{p(t|\eta_0(H))}\right).\end{aligned}$$

Proof. This fact can be shown algebraically. First, by the definition of KL-Divergence,

$$\begin{aligned}\xi_k(\eta) =& \mathrm{E}_{h\in\mathcal{H}_k}[D(T|h||T|\eta_0(h)) - D(T|h||T|\eta(h))] \\ &= \sum_{h\in\mathcal{H}_k} p(h|\emptyset) \sum_{t\in\mathcal{T}^I} p(t|h) \left(\log\left(\frac{p(t|h)}{p(t|\eta_0(h))}\right) - \log\left(\frac{p(t|h)}{p(t|\eta(h))}\right)\right) \\ &= \sum_{h\in\mathcal{H}_k} p(h|\emptyset) \sum_{t\in\mathcal{T}^I} p(t|h) \log\left(\frac{p(t|\eta(h))}{p(t|\eta_0(h))}\right)\end{aligned}$$

Now, the summation ranges over all primitive histories. Since every primitive

history belongs to exactly one abstract history, one can re-arrange the summation:

$$
\begin{aligned}
&= \sum_{H \in \mathcal{H}_k^\eta} \sum_{h \in H} p(h|\emptyset) \sum_{t \in \mathcal{T}^I} p(t|h) \log \left(\frac{p(t|H)}{p(t|\eta_0(H))} \right) \\
&= \sum_{H \in \mathcal{H}_k^\eta} \sum_{t \in \mathcal{T}^I} \log \left(\frac{p(t|H)}{p(t|\eta_0(H))} \right) \sum_{h \in H} p(h|\emptyset) p(t|h) \\
&= \sum_{H \in \mathcal{H}_k^\eta} \sum_{t \in \mathcal{T}^I} \log \left(\frac{p(t|H)}{p(t|\eta_0(H))} \right) \sum_{h \in H} p(ht|\emptyset) \\
&= \sum_{H \in \mathcal{H}_k^\eta} \sum_{t \in \mathcal{T}^I} \log \left(\frac{p(t|H)}{p(t|\eta_0(H))} \right) p(Ht|\emptyset) \\
&= \sum_{H \in \mathcal{H}_k^\eta} p(H|\emptyset) \sum_{t \in \mathcal{T}^I} p(t|H) \log \left(\frac{p(t|H)}{p(t|\eta_0(H))} \right) \\
&= \mathrm{E}_{H \in \mathcal{H}_k^\eta} [D(T|H||T|\eta_0(H)],
\end{aligned}
$$

which yields the result. □

D.2 Proof of Proposition 4.5

Proposition 4.5. *For any abstraction η, any refinement η', and any abstract history H under η, $\delta_{\eta,\eta'}(H) \leq p(H|\emptyset)\, Y(T|H)$ where $Y(T|H)$ is the* conditional entropy *of the distribution over tests of interest, given the abstract history H.*

Proof. Recall, $\delta_{\eta,\eta_f}(H) \stackrel{\text{def}}{=} \sum_{H' \in H} p(H'|\emptyset) D(T|H'||T|\eta_0(H')) - p(H|\emptyset) D(T|H||T|\eta_0(H))$ where H is an abstract history under η and H' is an abstract history under η' that is

contained in H. Then,

$$\begin{aligned}
&\delta_{\eta,\eta_f}(H) \sum_{H' \in H} p(H'|\emptyset) D(T|H'||T|\eta_0(H')) - p(H|\emptyset) D(T|H||T|\eta_0(H)) \\
&\quad = \sum_{H' \in H} p(H'|\emptyset) \sum_{t \in \mathcal{T}^I} p(t|H') \log\left(\frac{p(t|H')}{p(t|\eta_0(H')}\right) \\
&\qquad\qquad - p(H|\emptyset) \sum_{t \in \mathcal{T}^I} p(t|H) \log\left(\frac{p(t|H)}{p(t|\eta_0(H)}\right) \\
&\quad = \sum_{H' \in H} p(H'|\emptyset) \sum_{t \in \mathcal{T}^I} p(t|H') \log\left(\frac{p(t|H')}{p(t|\eta_0(H')}\right) \\
&\qquad\qquad - \sum_{H' \in H} p(H'|\emptyset) \sum_{t \in \mathcal{T}^I} p(t|H') \log\left(\frac{p(t|H)}{p(t|\eta_0(H)}\right) \\
&\quad = \sum_{H' \in H} p(H'|\emptyset) \sum_{t \in \mathcal{T}^I} p(t|H') \log\left(\frac{p(t|H')}{p(t|H}\right),
\end{aligned}$$

where the first equality is by the definition of KL-Divergence, the second follows from the fact that the H's are all the abstract histories under η' contained within H, and the third follows from the fact that $\eta_0(H) = \eta_0(H')$ because η' is a refinement of η. Now by the definition of entropy,

$$\begin{aligned}
&\sum_{H' \in H} p(H'|\emptyset) \sum_{t \in \mathcal{T}^I} p(t|H') \log\left(\frac{p(t|H')}{p(t|H}\right) \\
&\quad = \sum_{H' \in H} p(H'|\emptyset) \left(\sum_{t \in \mathcal{T}^I} p(t|H') \log(p(t|H')) - \sum_{t \in \mathcal{T}^I} p(t|H') \log(p(t|H))\right) \\
&\quad = -\sum_{H' \in H} p(H'|\emptyset) \sum_{t \in \mathcal{T}^I} p(t|H') \log(p(t|H)) - \sum_{H' \in H} p(H'|\emptyset) Y(T|H')
\end{aligned}$$

Noting that, in the first term, $\log(p(t|H))$ does not change with the summation

index, the summations can be re-arranged.

$$
\begin{aligned}
&- \sum_{H' \in H} p(H'|\emptyset) \sum_{t \in \mathcal{T}^I} p(t|H') \log(p(t|H)) - \sum_{H' \in H} p(H'|\emptyset) Y(T|H') \\
&\quad = - \sum_{t \in \mathcal{T}^I} \log(p(t|H)) \sum_{H' \in H} p(H'|\emptyset) p(t|H') - \sum_{H' \in H} p(H'|\emptyset) Y(T|H') \\
&\quad = - \sum_{t \in \mathcal{T}^I} \log(p(t|H)) p(H|\emptyset) p(t|H) - \sum_{H' \in H} p(H'|\emptyset) Y(T|H') \\
&\quad = -p(H|\emptyset) \sum_{t \in \mathcal{T}^I} \log(p(t|H)) p(t|H) - \sum_{H' \in H} p(H'|\emptyset) Y(T|H') \\
&\quad = p(H|\emptyset) Y(T|H) - \sum_{H' \in H} p(H'|\emptyset) Y(T|H')
\end{aligned}
$$

Since the second term is necessarily positive, this yields the result: $\delta_{\eta,\eta'}(H) \leq p(H|\emptyset)\mathrm{Y}(T|H)$. □

BIBLIOGRAPHY

BIBLIOGRAPHY

Amir, E. (2005), Learning partially observable deterministic action models, *in* 'Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)', pp. 1433–1439.

Baum, L. E., Petrie, T., Soules, G. & Weiss, N. (1970), 'A maximization technique occuring in the statistical analysis of probabalistic functions of markov chains', *The Annals of Mathematical Statistics* **41**(1), 164–171.

Baxter, J. & Bartlett, P. L. (2000), Reinforcement learning in pomdps via direct gradient ascent, *in* 'Proceedings of the Eighteenth International Conference on Machine Learning (ICML)', pp. 41–48.

Berger, A., Pietra, S. D. & Pietra, V. D. (1996), 'A maximum entropy approach to natural language processing', *Computational Linguistics* **22**(1), 39–71.

Bilmes, J. (2007), 'The graphical models toolkit (gmtk)'. `http://ssli.ee.washington.edu/~bilmes/gmtk`.

Boutilier, C., Dean, T. & Hanks, S. (1999), 'Decision-theoretic planning: Structural assumptions and computational leverage', *Journal of Artificial Intelligence Research* **11**, 1–94.

Boutilier, C., Friedman, N., Goldszmidt, M. & Koller, D. (1996), Context-specific independence in bayesian networks, *in* 'Uncertainty in Artificial Intelligence 12 (UAI)', pp. 115–123.

Bowling, M., McCracken, P., James, M., Neufeld, J. & Wilkinson, D. (2006), Learning predictive state representations using non-blind policies, *in* 'Proceedings of the Twenty-Third International Conference on Machine Learning (ICML)', pp. 129–136.

Boyen, X. & Koller, D. (1998), Tractable inference for complex stochastic processes, *in* 'Uncertainty in Artificial Intelligence 14 (UAI)', pp. 33–42.

Cassandra, A. R. (1998), Exact and Approximate Algorithms for Partially Observable Markov Decision Processes, PhD thesis, Brown University.

Degris, T., Sigaud, O. & Wuillemin, P.-H. (2006), Learning the structure of factored markov decision processes in reinforcement learning problems, *in* 'Proceedings of the 23rd International Conference on Machine Learning (ICML)', pp. 257–264.

Fikes, R. E. & Nilsson, N. J. (1971), Strips: a new approach to the application of theorem proving to problem solving, *in* 'Proceedings of the 2nd International Joint Conference on Artificial intelligence (IJCAI)', pp. 608–620.

Ghahramani, Z. & Jordan, M. I. (1995), Factorial hidden Markov models, *in* 'Advances in Neural Information Processing Systems 8 (NIPS)', pp. 472–478.

Gil, Y. (1994), Learning by experimentation: Incremental refinement of incomplete planning domains, *in* 'Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI)', pp. 10–13.

Givan, R., Dean, T. & Greig, M. (2003), 'Equivalence notions and model minimization in markov decision processes', *Artificial Intelligence* **147**, 163–223.

Guestrin, C., Koller, D., Parr, R. & Venkataraman, S. (2003), 'Efficient solution algorithms for factored mdps', *Journal of Artificial Intelligence Research* **19**, 399–468.

Hansen, E. (1998), Finite-Memory Control of Partially Observable Systems, PhD thesis, University of Massachussetts, Amherst, MA.

Hinton, G. E. (1999), Products of experts, *in* 'Proceedings of the 9th International Conference on Artificial Neural Networks (ICANN)', pp. 1–6.

Holmes, M. & Isbell, C. (2006), Looping suffix tree-based inference of partially observable hidden state, *in* 'Proceedings of the Twenty-Third International Conference on Machine Learning (ICML)', pp. 409–416.

Jaeger, H. (2000), 'Observable operator models for discrete stochastic time series', *Neural Computation* **12**(6), 1371–1398.

James, M. & Singh, S. (2004), Learning and discovery of predictive state representations in dynamical systems with reset, *in* 'International Conference on Machine Learning 21 (ICML)', pp. 417–424.

Julier, S. J. & Uhlmann, J. K. (1997), A new extension of the kalman filter to nonlinear systems, *in* 'Proceedings of AeroSense: the 11th International Symposium on Aerospace/Defense Sensing, Simulation and Controls', pp. 182–193.

Kalman, R. E. (1960), 'A new approach to linear filtering and prediction problems', *Transactions of the ASME – Journal of Basic Engineering* **82**, 35–45.

Kocsis, L. & Szepesvári, C. (2006), Bandit based monte-carlo planning, *in* 'Proceedings of the 17th European Conference on Machine Learning (ECML)', pp. 282–293.

Larsen, K. G. & Skou, A. (1991), 'Bisimulation through probabalistic testing', *Information and Computation* **94**(1), 1–28.

Li, L., Walsh, T. J. & Littman, M. L. (2006), Towards a unified theory of state abstraction for mdps, *in* 'Proceedings of the 9th International Symposium on Artificial Intelligence and Mathematics (ISAIM)'.

Littman, M. L. (1996), Algorithms for Sequential Decision Making, PhD thesis, Brown University, Providence, RI.

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

McCallum, A. K. (1995), Reinforcement Learning with Selective Perception and Hidden State, PhD thesis, Rutgers University.

Menache, I., Mannor, S. & Shimkin, N. (2002), Q-cut - dynamic discovery of sub-goals in reinforcement learning, *in* 'Proceedings of the 13th European Conference on Machine Learning (ECML)', pp. 295–306.

Monahan, G. E. (1982), 'A survey of partially observable markov decisions processes: Theory, models, and algorithms', *Management Science* **28**(1), 1–16.

Mugan, J. & Kuipers, B. (2009), Autonomously learning an action hierarchy using a learned qualitative state representation, *in* 'Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)', pp. 1175–1180.

Pasula, H. M., Zettlemoyer, L. S. & Kaelbling, L. P. (2007), 'Learning symbolic models of stochastic domains', *Journal of Artificial Intelligence* **29**, 309–352.

Peters, J. & Schaal, S. (2008), 'Natural actor-critic', *Neurocomputing* **71**, 1180–1190.

Poupart, P. & Boutilier, C. (2003), Bounded finite state controllers, *in* 'Advances in Neural Information Processing Systems 16 (NIPS)'.

Quinlan, R. J. (1992), *C4.5: Programs for Machine Learning*, Vol. 8 of *Morgan Kaufmann Series in Machine Learning*, Morgan Kaufmann Publishers.

Ravindran, B. (2004), An Algebraic Approach to Abstraction in Reinforcement Learning, PhD thesis, University of Massachusetts, Amherst, MA.

Rivest, R. L. & Schapire, R. E. (1994), 'Diversity-based inference of finite automata', *Journal of the Association for Computing Machinery* **41**(3), 555–589.

Rudary, M. (2008), On Predictive Linear Gaussian Models, PhD thesis, University of Michigan.

Rudary, M., Singh, S. & Wingate, D. (2005), Predictive linear-gaussian models of stochastic dynamical systems, *in* 'Uncertainty in Artificial Intelligence 21 (UAI)', pp. 501–508.

Shalizi, C. R. & Klinker, K. L. (2004), Blind construction of optimal nonlinear recursive predictors for discrete sequences, *in* 'Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence (2004)', pp. 504–511.

Şimşek, Ö. & Barto, A. G. (2008), Skill characterization based on betweeness, *in* 'Proceedings of the 22nd Conference on Neural Information Processing Systems (NIPS)', pp. 1497–1504.

Singh, S., James, M. R. & Rudary, M. R. (2004), Predictive state representations: A new theory for modeling dynamical systems, *in* 'Uncertainty in Artificial Intelligence: Proceedings of the Twentieth Conference (UAI)', pp. 512–519.

Sondik, E. J. (1978), 'The optimal control of partially observable markov processes over the infinite horizon: Discounted costs', *Operations Research* **26**, 282–304.

Soni, V. & Singh, S. (2007), Abstraction in predictive state representations, *in* 'Proceedings of the Twenty-Second National Conference on Artificial Intelligence, (AAAI)'. To appear.

Sorg, J. & Singh, S. (2009), Transfer via soft homomorphisms, *in* 'Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)', pp. 741–748.

Stober, J. & Kuipers, B. (2008), From pixels to policies: a bootstrapping agent, *in* 'Proceedings of the IEEE International Conference on Development and Learning (ICDL)', pp. 103–108.

Stolle, M. & Precup, D. (2002), Learning options in reinforcement learning, *in* 'Proceedings of the 5th International Conference on Abstraction, Reformulation, and Approximation', pp. 212–223.

Strehl, A. L., Diuk, C. & Littman, M. L. (2007), Efficient structure learning in factored-state mdps, *in* 'Proceddings of the 22nd Natioanl Conference on Artificial Intelligence (AAAI)', pp. 645–650.

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.

Taylor, J., Precup, D. & Panagaden, P. (2009), Bounding performance loss in approximate mdp homomorphisms, *in* 'Advances in Neural Information Processing Systems 21 (NIPS)', pp. 1649–1656.

Wang, X. (1995), Learning by observation and practice: an incremental approach for planning operator acquisition, *in* 'Proceedings of the 12th International Conference on Machine Learning (ICML)', pp. 549–557.

Weaver, L. & Tao, N. (2001), The optimal reward baseline for gradient-based reinforcement learning, *in* 'Uncertainty in Artificial Intelligence 17 (UAI)', pp. 538–545.

Williams, R. (1992), 'Simple statistical gradient-following algorithms for connectionist reinforcement learning', *Machine Learning* **8**, 229–256.

Wingate, D. (2008), Exponential Family Predictive Representations of State, PhD thesis, University of Michigan.

Wingate, D., Soni, V., Wolfe, B. & Singh, S. (2007), Relational knowledge with predictive state representations, *in* 'Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)', pp. 2035–2040.

Wolfe, A. P. (2010), Paying Attention to What Matters: Observation Abstraction in Partially Observable Environments, PhD thesis, University of Massachussetts, Amherst, MA.

Wolfe, A. P. & Barto, A. G. (2006), Decision tree methods for finding reusable MDP homomorphisms, *in* 'Proceedings of the Twenty-First National Conference on Artificial Intelligence, (AAAI)'.

Wolfe, B. D. (2009), Modeling Dynamical Systems with Structured Predictive State Representations, PhD thesis, University of Michigan.

Wolfe, B., James, M. R. & Singh, S. (2005), Learning predictive state representations in dynamical systems without reset, *in* 'Proceedings of the 22nd International Conference on Machine Learning (ICML)', pp. 985–992.

Wolfe, B., James, M. & Singh, S. (2008), Approximate predictive state representations, *in* 'Autonomous Agents and Multiagent Systems 7 (AAMAS)'.

Wolfe, B. & Singh, S. (2006), Predictive state representations with options, *in* 'Proceedings of the Twenty-Third International Conference on Machine Learning (ICML)', pp. 1025–1032.