INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI®

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

NOTE TO USERS

This reproduction is the best copy available.

UMI®

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

University of Alberta

An Online Algorithm for Discovery and Learning of Predictive State Representations

by

Peter McCracken

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science.**

Department of Computing Science

Edmonton, Alberta
Fall 2005

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

0-494-09240-8

Your file Votre référence
ISBN:
Our file Notre référence
ISBN:

NOTICE:
The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:
L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Abstract

Predictive state representations (PSRs) are a recently proposed method of modelling discrete dynamical systems using predictions about future observations. The strength of PSRs comes from their ability to represent system state using only observable data, such as actions and observations. Current techniques for learning PSRs use Monte Carlo methods to estimate prediction probabilities, but do not take advantage of the structure of the data to extrapolate information. In this work, we present the constrained gradient algorithm, a new technique for discovery and learning of PSRs that constrains its estimated predictions to augment a gradient descent approach. This algorithm is also the first online algorithm for PSRs capable of discovering core tests. We test the algorithm on a variety of standard domains, and show that it is able to build models competitive with current techniques. This work is an extension and elaboration of published work [McCracken and Bowling, 2006].

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Acknowledgements

Foremost, I would like to express my appreciation of all the help provided by my supervisor, Michael Bowling. Michael encouraged, corrected and advised me as I tried several different projects, eventually settling on the topic of predictive representations. He has always been a source of confidence in me and my work. I would also like to thank the members of the Reinforcement Learning and Artificial Intelligence group for introducing me to the topic of predictive representations and being a sounding board for my research. Of special importance is my friend and colleague Brian Tanner, with whom I've had many clarifying discussions. Finally, I would like to thank the members of my thesis committee for reading this thesis and providing critical feedback.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table of Contents

1 Introduction 1

2 Dynamical Systems and State Representations 4
2.1 Dynamical Systems 4
2.2 State Representations 7
2.3 Predictive State Representations 10
2.3.1 Tests and Histories 10
2.3.2 Core Tests 12
2.3.3 State Update 12
2.4 Related Work 13
2.4.1 Current Learning Methods for PSRs 13
2.4.2 Other Predictive Representations of State 16

3 The System Dynamics Matrix 19
3.1 Infinite Tests and Histories 19
3.2 From Matrix to PSR 21
3.3 Constraints on the System Dynamics Matrix 22

4 Constrained Gradient Learning 24
4.1 The Constrained Gradient Algorithm 25
4.1.1 Approach 25
4.1.2 Tests and Histories 26
4.1.3 Constructing the Prediction Matrix 29
4.1.4 Extracting the PSR Parameters 32
4.1.5 The Complete Algorithm 33
4.2 Experiments 36
4.2.1 Experimental Setup 36
4.2.2 Parameter Selection Experiments 37
4.2.3 Offline Experiments 42
4.2.4 Online Experiments 44
4.2.5 Summary of Learning Results 44

5 Discovery of Core Tests 47
5.1 Core Test Discovery 47
5.1.1 Selecting Core Tests 47
5.1.2 Discovery in the Constrained Gradient Algorithm 52
5.2 Experimental Results 54
5.2.1 Condition Threshold Tests 54
5.2.2 Non-Cumulative Selection of Core Tests 57
5.2.3 Summary of Discovery Results 60

6 Additional Investigation 62
6.1 Discovery and Learning 62
6.2 Investigative Experiments 65
6.2.1 Examining Sources of Error 65
6.2.2 Momentum in Learning 68

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

6.2.3 Sets of Core Tests 70
6.2.4 Summary of Investigative Tests 71

7 Conclusion **72**
7.1 Contributions 72
7.2 Future Work 73
7.2.1 Discovery Threshold 73
7.2.2 Selection of History Set 73
7.2.3 Reformulation as Optimization Problem 74
7.2.4 Enforcing Constraints 74
7.2.5 Theoretical Convergence 75
7.3 Summary 75

Bibliography **76**

A Defining Test Predictions **79**
A.1 Test Predictions 79
A.2 Problems with $p(t|h)$ 80
A.3 Redefining Test Predictions 82

B Test Domains **84**
B.1 Float-Reset 84
B.2 Tiger 85
B.3 Paint 86
B.4 Shuttle 86
B.5 Network 87
B.6 4x3 Maze 88
B.7 Cheese Maze 89
B.8 Bridge Repair 90

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

List of Tables

4.1 Size comparison for different selections of test set T. 27

5.1 Summary and comparison of discovery results. 56

B.1 The state transitions for the Network domain. 88
B.2 State transitions for the Bridge domain. 90
B.3 Observation distributions for the Bridge domain. 91

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

List of Figures

2.1 The Float-Reset dynamical system. 6

3.1 An example system dynamics matrix. 20
3.2 The Float-Reset system dynamics matrix. 20

4.1 The effect of changing the learning parameter, α. 38
4.2 The effect of changing the number of rows in H. 41
4.3 PSR error in offline tests. 43
4.4 PSR error in online tests. 45

5.1 Effect of the condition threshold on discovery. 55
5.2 Non-cumulative discovery performance. 59
5.3 The condition of $\widehat{y}(Q|H)$ over time. 61

6.1 PSR error using discovered core tests. 63
6.2 Performance when sources of gradient error are eliminated. 67
6.3 Effect of momentum on constrained gradient learning. 69
6.4 Two different core test sets for the Shuttle domain. 70

B.1 The Float-Reset domain. (Repeated from Figure 2.1) 85
B.2 The Shuttle domain. 86
B.3 The 4x3 Maze domain. 89
B.4 The Cheese domain. 89

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 1

Introduction

Suppose one is learning how to use a DVD player. Essentially, this involves figuring out which buttons to press on the player in order to produce the desired result, like powering it off or playing a DVD. Of course, which buttons to press depends on current properties of the player: Is it powered on or off? Is the DVD tray in or out? Is there a disc inside it? Properties like these constitute the state of the DVD player. Some of these properties are directly observable, like whether the tray is out; others are not directly observable, like whether a disc is inside the machine or not.

A reasonable way to learn how to use a DVD player is to press the buttons and observe what happens. There are many different buttons on the DVD player, and they can be pressed in any order, which means there are many sequences one can try: What happens if the 'power' button is pressed? What happens if it is pressed again? What happens if 'play' and then 'eject' and then 'power' and then 'pause' and then 'stop' are all pressed in sequence? In fact, there are an infinite number of such tests one could perform. But in reality, only a relatively small number of these tests actually provide unique information; after all, a DVD player is a relatively simple system, and certainly does not have an infinite amount of complexity. Intuitively, there is a small number of button combinations on a DVD player that are capable of summarizing the effect of all possible combinations of button presses.

In more technical terms, a DVD player is an example of a *dynamical system.* There has been a significant amount of work in the field of dynamical systems. They are relevant to computer science, engineering, mathematics, and even biology and psychology. Later in this work, in Section 2.1, we further explain the concept of dynamical systems, different types of dynamical systems, and various ways to

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

represent the state of a system.

An important problem is how one represents the dynamical system. In this work we focus on the concept of representing the state of a system based on predictions about the outcomes of tests. In the DVD player example, one can know the state of a DVD player by knowing how it will react to a series of button presses. This is the fundamental idea behind *predictive state representations*: the state of a system can be represented by a small number of questions about how the system will react to inputs. Predictive state representations (PSRs) are a relatively recent method of representing the state of a system, originating with work done by Littman, Sutton, and Singh [2002]. In Section 2.3 and Section 2.4, we more formally describe predictive state representations and explain previous algorithms for learning these representations.

Our main contribution in this work is the presentation of a new algorithm to build PSRs from experience gained through interaction with the system. The algorithm is also described in a recent paper [McCracken and Bowling, 2006]. This new algorithm, the *constrained gradient algorithm*, possesses several advantages over existing algorithms. First, it is capable of learning a model of a system based on a single long interaction with the system, without having to make multiple passes over the data. Furthermore, at every time step the algorithm has an estimate of the current state of the system. Together, these properties are known as *online* learning. Online learning algorithms are preferable because they allow a constantly improving model to be generated in real time. The second advantage of the constrained gradient algorithm is that it takes advantage of the large amount of structure inherent in sequential data obtained from a dynamical system. This structure is described in Chapter 3. We expect that making use of the structure of the data will lead to more efficient learning and more accurate models.

In the DVD player example, we stated that there must be a finite set of predictions about button combinations that summarize the state of the system. In order to learn to use the DVD player, one must decide which predictions are included in that special set. This process is known as *discovery*. Chapter 5 describes the approach to the discovery problem taken by the constrained gradient algorithm, and contains empirical evidence that the algorithm is capable of finding appropriate tests with far less data than is required by other algorithms. The second aspect of learning how a system operates is knowing how the system responds to interaction;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

how should the predictions about the DVD player change as its buttons are pushed? This is the process of *learning* the PSR parameters. In Chapter 4, the constrained gradient algorithm's approach to learning is described, and empirical tests compare its performance with existing methods.

Further investigation of the constrained gradient algorithm is done in Chapter 6, including final performance results and additional tests to explore interesting results found during testing. Finally, in Chapter 7 we conclude this work by summarizing our findings and describing avenues for future work.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 2

Dynamical Systems and State Representations

The purpose of this chapter is to introduce the concept of dynamical systems and methods of modelling the state of such systems. We will show that there exists many classes of dynamical systems, and that state representation is a complicated problem. In Section 2.1, we describe dynamical systems and a selection of properties a system might have. Section 2.2 overviews various ways of representing state in dynamical systems. In Section 2.3, we delve more deeply into the topic of predictive state representations, which will become the focus of the rest of this thesis. Finally, in Section 2.4 we describe existing work in PSRs and other predictive representations of state.

2.1 Dynamical Systems

In the most general sense, a *dynamical system* is any system that generates a sequence of observations, taken from a set $\mathcal{O}$, that is perceived by an agent. The agent may, or may not, control the output of the system by taking actions from a set, $\mathcal{A}$. A dynamical system can be in various 'states' that can change over time and in response to actions. The state of a system affects the impact of actions and the likelihood of observations; formally, the *state* of a system is any sufficient statistic for predicting the future of a system [Littman et al., 2002]. Research in dynamical systems describes classes of systems which meet certain restrictions. Depending on the class of system, different methods of representing its state are available. In this section, we list and briefly describe some important variations of dynamical systems.

Discrete/Continuous Time. In dynamical systems, time measures the duration

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

of the agent's interaction with the system. In continuous time systems, time is measured, in units, as a real value, and system dynamics are generally governed by continuous functions. Other systems, such as discrete semi-Markov systems, measure time in discrete units, but state transitions may take a variable number of time units [Howard, 1971]. In discrete time systems, time is measured in indivisible units called *time steps.* Each time step is long enough for the agent to perform a single action and to perceive a single observation. The agent takes its first action and perceives its first observation at time step 1, and in general time step i is the time when the agent takes its ith action and experiences its ith observation.

Discrete/Continuous Observations. This property describes whether the possible observations in the system, $\mathcal{O}$, are discretely valued or form a continuum. In a discrete system, possible observations could be things like 'black' and 'white' or 0 and 1; a continuous system's observations could include the full spectrum of grey or an interval of real numbers.

Discrete/Continuous Actions. This property describes whether the actions in $\mathcal{A}$ are discrete or continuous. Continuous actions take a continuous parameter, like 'turn $x°$ left' or 'move forward for t seconds'. Discrete actions are parameterless actions, like 'reset', or parameterized actions with pre-defined parameters, such as 'go forward one step' or 'turn 90° left'.

If a dynamical system is discrete in time, observations and actions, we call it a *discrete* dynamical system. Furthermore, if the number of discrete observations and actions is finite, we call the system a *discrete, finite* dynamical system. In this work, we consider only this class of systems, and all future references to dynamical systems will implicitly assume *discrete, finite* dynamical systems. Because the sets $\mathcal{A}$ and $\mathcal{O}$ are discrete, we can list their elements using the notation $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$ and $\mathcal{O} = \{o_1, o_2, \ldots, o_m\}$, where a subscript indicates a particular element of the set.

Some additional properties of dynamical systems include:

Uncontrolled/Controlled. This property is also known as output-only/input-output [Jaeger, 1998]. In an uncontrolled (output-only) system, the size of the set $\mathcal{A}$ is one; *i.e.*, the agent's interaction with the system consists only of a sequence of observations. In a controlled (input-output) system, $|\mathcal{A}| > 1$;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

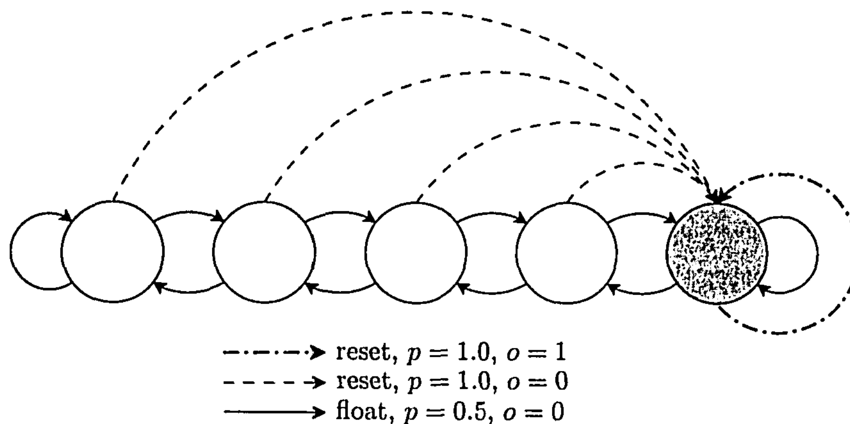


Figure 2.1: The Float-Reset dynamical system. For each line type, the corresponding action, the transition probability p and the observation o are given.

the agent is able to affect the output of the system by performing actions from $\mathcal{A}$, and the agent's interaction with the system consists of a sequence of action-observation pairs. The process by which actions are chosen is known as a *policy*.

Completely/Partially Observable. In a completely observable system, there is a one-to-one mapping between states and observations; *i.e.*, each observation is a sufficient statistic for representing the state of the system. In a partially observable system, the observations produced at each state do not uniquely identify the state.

Deterministic/Stochastic. In a deterministic system, performing the same action sequence, starting at time step 1, will always generate the same observation sequence. In a stochastic system, observations are generated according to some probability distribution, and performing the same action sequence from the first time step always generates observations according to the same probability distribution.

In general, methods that are capable of handling controlled, partially observable or stochastic dynamical systems are more powerful than methods that restrict the system, respectfully, to uncontrolled, completely observable or deterministic systems.

Figure 2.1 shows an example dynamical system, called Float-Reset, that is controlled, stochastic, and partially observable. This system was introduced by Littman

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

et al. [2002] as an example system, and has been used in other PSR work [Singh, Littman, Jong, Pardoe, and Stone, 2003; Singh, James, and Rudary, 2004]. Float-Reset, as depicted in Figure 2.1, has five positions, including a special 'reset' position. The action set in Float-Reset is $\mathcal{A} = \{f, r\}$, where f stands for 'float' and r stands for 'reset'. The observation set is $\mathcal{O} = \{0, 1\}$. The reset action takes the system back to the reset position. The agent observes a 1 when resetting if the action is taken when the system is already in the reset position, otherwise the agent observes a 0. The float action randomly moves the system to the left or right with uniform probability, except on the end positions where the system either stays in the same position or goes to the adjacent position. The agent always observes a 0 when floating. The Float-Reset system is relatively simple, but provides a useful example system for explaining concepts related to dynamical systems and predictive state representations.

The overarching problem in dynamical systems is determining a policy for optimal control of the agent. In this problem, states in the system are associated with some scalar reward, and the goal of the agent is to maximize its reward during its interaction with the system. This is a very large topic, and in this work we will focus on a crucial subproblem: How can an agent represent the state of the system, and update its state when it takes actions and perceives observations? The next chapter discusses existing methods of state representation in dynamical systems.

2.2 State Representations

Looking at the Float-Reset diagram in Figure 2.1, one can easily label each of the five positions as a state of the system, because each of these positions is sufficient to predict future observations. However, the concept of state is not so simple. Consider an agent interacting with the Float-Reset system, and assume the agent knows the current state of the system is the reset position. If the agent takes the float action, it no longer knows the exact position in which the system resides. The state of the system can be described as "the state that has a 50% chance of being in either the reset position or the the position adjacent to it" or "the state that has a 50% chance of generating a 1 observation when a reset action is taken." These types of descriptions are known as *information states*, since they are the most accurate description of the system possible, given the information that is available to the agent. In contrast, we refer to a specific set of states used to describe a system, such

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the five positions of the Float-Reset system, as *nominal states.* Note that nominal states are not necessarily a subset of reachable information states; in Float-Reset, for example, the system can never reach an information state that corresponds to any of the four non-reset positions. Unless otherwise specified, we use the term *state* to mean information state. A state only has meaning in the context of other states, and how they interact with each other using the dynamics of the system. We call the combination of the sufficient statistic and the description of dynamics a *state representation.*

One important property of a state representation is whether it requires more knowledge about the underlying system than is available from the history, or whether it is based entirely on the observable quantities contained in the history. Any system which expresses its state using only elements of the action and observation sets, $\mathcal{A}$ and $\mathcal{O}$, is said to be *grounded* in observable quantities. We prefer grounded representations because they can be learned entirely from experience [Littman et al., 2002].

The most general state representation possible is a complete recording of the agent's interaction with the system since the first time step; *i.e.*, the list of all actions and corresponding observations the agent has experienced. This is known as a *history*, and it is always a sufficient statistic. Of course, as a state representation for online learning, histories are impractical. There are an infinite number of possible histories for a given system, they can be infinite in length, and they do not allow for generalization between time steps, since each time step has a unique history.

The goal of state representation algorithms is to map these histories to more useful representations. There are many methods of representing the state of a system, depending on the properties of the system as outlined in the previous section. Below, we briefly describe some existing methods of state representation.

Markov Methods

Markov processes, or Markov chains, are a method of representing uncontrolled, fully observable dynamical systems [Russell and Norvig, 2003]. They map histories to states by using the previous observation as the state, and they have functions to map states to probability distributions over next states.

Markov decision processes [Puterman, 1994] generalize Markov processes to the controlled case. Transition functions use state and action pairs to map to distribu-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

tions over subsequent states. MDPs are based entirely on observable quantities, and thus grounded, but that property is trivial when the system is completely observable.

k-Order Markov Methods

The k-order Markov assumption is a generalization of the standard Markov assumption to include a history of k observations [Russell and Norvig, 2003]. k-order Markov models are a generalization of Markov processes for modelling systems in which the state is not purely dependent on the single previous observation. These methods consider that the previous k observations form a sufficient statistic to represent the system. When $k = 1$, a k-order Markov model is the same as a Markov process.

k-order Markov methods are typical of history-based methods in general: methods that consider state to be representable by a finite amount of history [Littman et al., 2002]. History-based methods can be both controlled and uncontrolled. History-based methods have the advantage that they are based entirely on observable quantities. However, for a given k, history-based methods are unable to represent systems that depend on information in the history older than k steps. For instance, a k-order Markov model would be unable to represent the Float-Reset system, because it cannot distinguish among states in which the previous k actions were all float actions. For large values of k, though, a k-order Markov model can often make a good approximation of such systems.

HMMs and POMDPs

Hidden Markov models [Rabiner, 1989], or HMMs, extend Markov processes to the partially observable case. Likewise, partially observable Markov decision processes [Aström, 1965], or POMDPs, extend Markov decision processes to the partially observable case. HMMs and POMDPs use a given set of postulated nominal states, and assume that the underlying system is in exactly one of these nominal states. However, the current history may not contain enough information to determine in which of these nominal states the system is, so HMMs and POMDPs represent state as a probability distribution over nominal states. This distribution is known as a belief state.

POMDPs are the most general of the state representations listed so far, and they are the first representation we have described that can fully represent the Float-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reset system. However, they still have the disadvantage that they can only represent systems with a finite number of nominal states. Furthermore, these representations are not grounded in observable data, and the nominal states must be provided to the algorithm prior to its experience with the system. In the next section, we describe predictive state representations, which have been shown to be theoretically more powerful than POMDPs and are completely based upon observable quantities.

2.3 Predictive State Representations

A relatively new paradigm for representing state is to use predictions of the future to summarize the current state. Such state representations are called *predictive representations of state*, which are based on ideas from diversity-based methods [Rivest and Schapire, 1994] and observable operator models [Jaeger, 1998]. More recently, predictive state representations (PSRs) [Littman et al., 2002] were introduced as another predictive representation of state. PSRs are grounded entirely in data observable by the agent, and require only a prior knowledge of the set of actions, $\mathcal{A}$, and observations, $\mathcal{O}$, present in the system. It has been shown that PSRs are capable of compactly modelling any system that can be modelled by a POMDP [Littman et al., 2002], and that there exist systems that cannot be modelled by any POMDP that can be modelled by an OOM [Jaeger, 1998], and therefore can also be modelled by a PSR [Singh et al., 2004]. Predictive state representations will be the subject of the rest of this thesis.

2.3.1 Tests and Histories

In this section, we describe histories and introduce the concept of tests and predictions. The notation used for describing these concepts has not been entirely consistent across the PSR literature. We use the notation from Wolfe, James, and Singh [2005], with some modifications.

As mentioned previously, a history is the sequence of action-observation pairs, or *ao* pairs, that an agent in a dynamical system has experienced beginning at the first time step. For instance, the history $h^n = a^1o^1a^2o^2 \ldots a^no^n$ of length n means that the agent chose action a^1 and perceived observation o^1 at the first time step, after which the agent chose a^2 and perceived o^2, and so on. A superscript on an action or observation indicates the relevant time step. A special history, known as the null history, is the history at the beginning of time before the agent has taken

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

any actions or seen any observations. We use ϕ to denote the null history.

A *test* is a sequence of ao pairs that begins immediately after a history; it is a potential future. The action sequence of a test t is represented by $\overline{a}_t$ and the observation sequence is represented by $\overline{o}_t$. A test is said to succeed if the observations in the sequence are observed in order, given that the actions in the sequence are taken in order. For instance, the test $t = a_1o_1a_2o_2 \ldots a_no_n$ succeeds if the agent observes o_1 followed by o_2, *etc.*, given that it performs actions a_1 followed by a_2, *etc.*. A test fails if the action sequence is taken but the observation sequence is not observed. Thus, the outcome of a test is a binary success or failure.

A *prediction* of a test is the probability that the test will be successful. The outcome of a test t depends on the history h that preceded it, so we write predictions as $y(t|h)$, to represent the probability of test t succeeding after history h. For test t of length n, the value of $y(t|h)$ is defined:

$$y(t|h) = \prod_{i=1}^{n} \Pr(o_i|h, a_1o_1 \ldots a_{i-1}o_{i-1}a_i)$$

A test prediction, therefore, is the product of the probabilities of observing each observation in $\overline{o}_t$, given the entire history that preceded the observation. Note that the above definition of a test prediction differs from the definition used in the PSR literature to date. An explanation of this difference and the associated implications can be found in Appendix A. Essentially, our definition makes test predictions independent of policy, while previous work uses a policy-dependent definition. The definitions coincide when the policy used during learning chooses actions independent of past observations. When discussing previous work, we will make this strong assumption on the policy and so simply refer to test predictions using our definition of $y(t|h)$. Our work, as we will show, does not require such an assumption on the policy used during learning.[1]

A special test, known as the null test, is the test of length zero. We use ε to denote the null test. The outcome of the null test is defined as:

$$y(\varepsilon|h) = \begin{cases} 1, & y(h|\phi) > 0 \\ 0, & y(h|\phi) = 0 \end{cases}$$

Thus, the null test is successful for any history that can possibly be generated by the system.

[1]Our work does require a different, less strict assumption on the policy used during learning. We require the policy to have the capability of discovering the full complexity of the system. Thus, we require $\Pr(a|h) > 0$, for all $a \in \mathcal{A}$ and for all histories h.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

If T is a set of tests and H is a set of histories, $y(t|h)$ is a single value, $y(T|h)$ is a row vector containing $y(t_i|h)$ for all tests $t_i \in T$, $y(t|H)$ is a column vector containing $y(t|h_j)$ for all histories $h_j \in H$, and $y(T|H)$ is a matrix containing $y(t_i|h_j)$ for all $t_i \in T$ and $h_j \in H$.

2.3.2 Core Tests

In any dynamical system there exists a (possibly infinite) set of tests, Q, whose predictions at any history are a sufficient statistic for computing predictions for all possible tests at that history [Singh et al., 2004]. This means that for any test t there exists a function f_t such that $y(t|h) = f_t(y(Q|h))$. If the size of Q is finite, then the system can be represented by a PSR. Furthermore, if the function f_t is a linear function of the tests in Q, the system can be represented by a *linear* PSR. A linear PSR computes the outcome of tests using $y(t|h) = y(Q|h)m_t$, for some column vector of weights m_t. As in most of the literature to date, we henceforth restrict our discussion of PSRs to the linear PSR case, although there has been some discussion of non-linear PSRs [Singh et al., 2004; Rudary and Singh, 2004]. The set of tests Q is called the *core tests*, and determining which tests are core tests is known as the *discovery* problem. In addition to Q, it will be convenient to discuss the set of one-step extensions of Q. A one-step extension of a test t is a test aot, which prefixes the original test with a single ao pair.

$$X = \{aot \,|\, \forall a \in \mathcal{A}, o \in \mathcal{O}, t \in Q \cup \{\varepsilon\}\}$$

The set of all one-step extensions of Q, plus all of the length one tests (*i.e.*, the one-step extensions of the null test) will be called X.

2.3.3 State Update

Previously, we defined a state representation as a sufficient statistic, combined with a description of the system dynamics. We already know that at any time i, the set of predictions $y(Q|h^i)$ is a sufficient statistic for the state of the system. In this section, we describe how the system dynamics are represented and used to update the state vector of the PSR.

At time $i-1$, the state vector of the PSR is $y(Q|h^{i-1})$. After the agent takes action a^i and sees observation o^i, the state vector must be updated to be $y(Q|h^i)$, where $h^i = h^{i-1}a^io^i$. A simple application of conditional probability is used to

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

update the state vector's predictions for time $i-1$ to predictions for time i:

$$y(q|h^i) = y(q|h^{i-1}a^io^i) = \frac{y(a^io^iq|h^{i-1})}{y(a^io^i|h^{i-1})} \quad \forall q \in Q$$

In a linear PSR, we know $y(t|h) = y(Q|h)m_t$ for any test t. Thus, we can rewrite the above equality as:

$$y(q|h^i) = \frac{y(Q|h^{i-1})m_{a^io^iq}}{y(Q|h^{i-1})m_{a^io^i}} \quad \forall q \in Q$$

Note that the fraction is written in terms of the known state vector $y(Q|h^{i-1})$, plus a set of weight vectors m_t. In order to update the PSR at each time step, the vector m_t must be known for all length one tests, a^io^i, and all one-step extensions of the core tests, a^io^iq. All of these tests are in the set of extension tests, X. The set of all of these update vectors, which we will call m_X, are the parameters of the PSR. The vectors m_X are the PSR's representation of the system dynamics. Estimation of these parameters is known as the *learning* problem.

2.4 Related Work

In this section we describe other work related to predictive representations of state. In Section 2.4.1, we discuss current learning and discovery algorithms for PSRs. In Section 2.4.2, we briefly describe other types of predictive state representations.

2.4.1 Current Learning Methods for PSRs

To date, there have been three main learning algorithms published for PSRs: a myopic gradient-based algorithm [Singh et al., 2003], a Monte Carlo algorithm that requires the presence of a reset action in the system [James and Singh, 2004], and a modification to the reset-based algorithm that removes the need for reset actions [Wolfe et al., 2005]. A fourth algorithm, which applies temporal difference methods to learning PSRs, has also been presented [Wolfe et al., 2005].

Myopic Gradient Descent Method

The myopic gradient descent learning algorithm [Singh et al., 2003] was the first algorithm for learning the parameters of a PSR. The algorithm learns a model *online*, which means that it makes only a single pass over the data, and at every time step it has a best estimate of the current state vector and parameters for the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

system. The algorithm has no method of discovering core tests; the set Q is provided to the algorithm prior to learning.

The gradient descent algorithm attempts to minimize error on the observed action-observation data by moving the update parameters m_X according to the gradient of the error. It uses Monte Carlo updating; tests t for which the full action sequence is observed have their parameters m_t adjusted to make them more likely to predict success or failure based on whether the full observation sequence was observed or not. The *myopic* gradient is used, which refers to the algorithm's approximation of the gradient by the single success or failure observed in the data stream. Computationally, the myopic gradient algorithm is efficient, since it performs only $O(|Q|^2)$ computations per time step, and Q can generally be assumed to be small.

Singh et al. show that the myopic gradient algorithm was successful at learning reasonable models on a test suite of POMDP systems, using in the range of several million action-observation pairs.

Reset Method

The reset-based Monte Carlo algorithm [James and Singh, 2004] was the first algorithm to do both parameter learning and core test discovery for PSRs. The algorithm works on dynamical systems which have a special reset action that returns the system to its initial state. This means that any history whose final action is the reset action is equivalent to the initial history of the system, ϕ. The reset-based Monte Carlo algorithm is a batch algorithm, which means it processes a finite collection of data, can make multiple passes over the data, and does not maintain a current estimate of the state vector.

The algorithm explicitly estimates a matrix of predictions $\widehat{y}(T|H)$, for a set of tests T and histories H. It computes maximum likelihood estimates of each $\widehat{y}(t|h)$, by counting the number of times the sequence of actions in t was taken after h was observed, and also by counting the number of those times that the exact observation sequence from t was observed. Thus, the samples used to generate prediction estimates are all Monte Carlo samples; if the full action sequence of t is not observed, no change is made to the prediction for t. In order to observe multiple samples for test t at history h, the history h must be observed multiple times. To do this, the algorithm makes use of the reset action to restore the system to its original

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

state, before executing history h.

The reset algorithm uses an iterative process to estimate the matrix $\widehat{y}(T|H)$ from observed data, and then re-chooses the set of tests T. From $\widehat{y}(T|H)$, the reset-based Monte Carlo algorithm computes an estimate of the number of core tests in the system, by computing from $\widehat{y}(T|H)$ an approximation of the rank k of $y(T|H)$. In Section 3.2, we explain more about the relevance of rank when selecting core tests. The algorithm then chooses a set of k core tests from T by choosing the most linearly unrelated columns in $\widehat{y}(T|H)$. It augments the set T with the one-step extensions of the selected core tests, and then re-estimates $\widehat{y}(T|H)$. This iterative process continues until the detected number of core tests does not increase between two iterations.

Suffix-History Method

Assuming the existence of a labelled reset action is a large and generally untrue assumption. The suffix-history algorithm [Wolfe et al., 2005] is a modification of the reset-based Monte Carlo algorithm that removes the need for a reset action in the system. Like the reset-based algorithm, the suffix-history method is a batch algorithm for discovery and learning of PSRs.

In order to estimate the predictions $\widehat{y}(T|H)$ without experiencing any history in H multiple times, the suffix-history method groups histories with identical suffixes. Thus, the prediction $\widehat{y}(t|h)$ is the maximum likelihood estimate of the number of times test t succeeded at any time step with a history of the form h^*h, where h^* matches any history. The effect of grouping histories in this manner is that $\widehat{y}(T|H)$ contains predictions for a modified system. In the modified system, the history ϕ is equivalent to the stationary distribution of the original system, if the original system has a stationary distribution. It was shown that a complete set of core tests in this modified system is also a complete set of core tests in the original system, as long as the original system can be modelled by a POMDP. No guarantees are made if the system is not representable by a POMDP. Furthermore, the PSR parameters in the modified system are the same as the PSR parameters for the original system, because these parameters are not dependent on the initial state of the system.

Once the $\widehat{y}(T|H)$ matrix is estimated using the suffix-history method, discovery and learning is performed in the same manner as in the reset-based Monte Carlo method.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Temporal Difference PSR Learning

The above three algorithms all use Monte Carlo sampling; predictions and parameters related to a test t are not modified unless a sample for the *entire* test t is available. Wolfe et al. [2005] present an algorithm that incorporates temporal difference methods, and therefore is able to learn even if only parts of the test t are executed. However, in their experiments they found that the TD algorithm performs very poorly, even when it was provided with a correct set of core tests. Thus, we do not currently consider the TD algorithm a viable learning algorithm for PSRs.

2.4.2 Other Predictive Representations of State

Here, we discuss some predictive representations of state other than pure PSRs.

PSRs with Memory

Memory-PSRs [James, Wolfe, and Singh, 2005], or mPSRs, represent state using a combination of a PSR state vector and a short memory of recent actions and observations. They use the general idea that the set of all possible histories H can be partitioned into subsets H_i according to some set of suffixes. Each H_i induces a system, and the complexity of this system cannot be greater than the complexity of the full system, and is often smaller. Instead of maintaining a single PSR for the complete system, an mPSR maintains a separate smaller PSR for each of the induced systems. Each smaller PSR has a separate set of core tests and update parameters.

The goal of mPSRs is to represent systems more efficiently than PSRs. James et al. show that the number of parameters necessary to represent a system can be substantially reduced by choosing a proper set of memories, although they do not approach the problem of selecting memories. Fewer parameters means that it may be easier to learn mPSRs, although some experimental results do not show a large improvement in learning efficiency [James et al., 2005]. This is likely due to the test domains used, and the fact that the induced subsystems generally had the same size as the original system. James and Singh [2005] investigated planning with mPSRs using incremental pruning.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Temporal-Difference Networks

Temporal-difference networks, or TD networks, were introduced by Sutton and Tanner [2005]. A TD network consists of two networks: a question network and an answer network. The question network contains predictive nodes. Together, the set of current values in the nodes of the question network form the state vector of the TD network. These nodes generally predict the outcome of other nodes, conditioned on an action sequence, or they directly predict the next observation. Using temporal difference methods [Sutton, 1988], the networks are able to learn without requiring complete Monte Carlo tests. The answer network contains the TD network's representation of the system dynamics; it updates the values in the predictive nodes after each action-observation pair is seen. For each node, the answer network linearly combines the values in the current state vector to generate the new value of the node.

Much of the power of TD networks comes from the question network. The simplest TD networks use trees of action-conditioned nodes to predict observations. However, the structure of the question network can be modified to include a variety of questions, such as combinations of other predictions, nodes conditioned on both actions and observations, and recursive nodes.

TD networks and PSRs currently are the two most fertile research areas in predictive representations. TD networks have been claimed to be generalizations of linear PSRs [Tanner and Sutton, 2005b]; to date, though, it is unclear how to represent a typical PSR-type test such as $y(a_1o_1a_2o_2|h)$ in a TD network. It is unknown whether non-linear PSRs are equivalent to TD networks. A major area of research for PSRs has been the discovery problem [Rosencrantz, Gordon, and Thrun, 2004; James and Singh, 2004; Wolfe et al., 2005]. TD network research, on the other hand, has delayed the problem of discovering networks in favour of augmentations to the basic TD network learning algorithm. Aside from discovery, PSRs and TD networks have had some parallel research paths: both have been augmented with history, with history-based TD networks [Tanner and Sutton, 2005b] and mPSRs, and TD networks have incorporated eligibility traces [Tanner and Sutton, 2005a] while PSRs have incorporated some TD learning [Wolfe et al., 2005]. TD networks with options [Sutton, Rafols, and Koop, 2005] have demonstrated temporal abstraction; it was also suggested that PSRs could use options for temporal abstraction [Littman et al.,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2002]. To date, however, there has been no research that directly compares TD networks and PSRs in similar settings, so it is unknown which representation performs better in practice.

Transformed Predictive State Representations

Transformed predictive state representations [Rosencrantz et al., 2004], or TPSRs, are a variant of PSRs that use linear combinations of predictions to represent state. Like other approaches to PSRs, learning a TPSR involves estimating a matrix $\widehat{y}(T|H)$, for some set of tests T and histories H. The principle components of this matrix are then found using singular value decomposition. An exact TPSR will use all non-zero principle components; however, low dimensional approximate representations can be created by selecting only the k most important components. The parameters of the TPSR are generated by using linear regression on the principle components to create weight vectors for updating the state vector for each ao pair.

The main benefit of using TPSRs over PSRs is that the former simplifies the discovery problem by making it a matter of choosing the k most important principle components of the system. These components are immediately available after singular value decomposition of the matrix. However, a disadvantage is that the state vector no longer has any meaningful interpretation; it is simply a collection of linear combinations of tests. Also, like the reset-based Monte Carlo algorithm, a TPSR cannot model controlled systems that do not have a reset action.

TPSRs were tested on a mapping task for a robot, which is the first example of using a predictive representation for learning in a real-world system. The mapping task was an uncontrolled system, and the matrix $\widehat{y}(T|H)$ was created by setting each $\widehat{y}(t|h)$ to the binary sample value for t at history h. Even with such coarse-grained approximations of test predictions, the SVD process was able to create reasonable maps of a single room setting.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 3

The System Dynamics Matrix

The *system dynamics matrix* was introduced by Singh et al. in 2004 as an intuitive method for explaining linear PSRs. The system dynamics matrix is a theoretical construct, but approximating portions of the matrix is the key principle behind most current discovery and learning algorithms for PSRs. This chapter explains the concept of the system dynamics matrix in Section 3.1, and how it can be used to generate a predictive state representation in Section 3.2. We also describe a set of properties of the system dynamics matrix, formulated as constraints on the matrix, in Section 3.3. Except where noted, this chapter is a re-explanation of material presented by Singh et al. [2004], in order to make clear the necessary details required for understanding the PSR discovery and learning algorithm described in subsequent chapters.

3.1 Infinite Tests and Histories

In a discrete, finite dynamical system, there is an infinite but countable number of tests and histories: a history and test for every combination of actions and observations, of any length. The set of all tests is T^* and the set of all histories is H^*. An ordering can be imposed on these infinite sets of sequences by sorting them first by length, and then sorting sequences of equal length lexicographically. We can now refer to members of the sets as t_i or h_i, to indicate the ith element in the set of all tests or histories, respectively. The first test is ε, and the first history is ϕ; both have length zero.

Consider the infinite matrix, $D = y(T^*|H^*)$, that has a column corresponding to each test in T^* and a row corresponding to each history in H^*. [1] The entry $d_{(i,j)}$

[1] In previous work [Singh et al., 2004], the definition of test predictions differed from our definition

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

	t_0	t_1	t_2	$\cdots$	t_n	$\cdots$
h_0	$y(t_0\|h_0)$	$y(t_1\|h_0)$	$y(t_2\|h_0)$	$\cdots$	$y(t_n\|h_0)$	$\cdots$
h_1	$y(t_0\|h_1)$	$y(t_1\|h_1)$	$y(t_2\|h_1)$	$\cdots$	$y(t_n\|h_1)$	$\cdots$
h_2	$y(t_0\|h_2)$	$y(t_1\|h_2)$	$y(t_2\|h_2)$	$\cdots$	$y(t_n\|h_2)$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$
h_m	$y(t_0\|h_m)$	$y(t_1\|h_m)$	$y(t_2\|h_m)$	$\cdots$	$y(t_n\|h_m)$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$

Figure 3.1: An example system dynamics matrix.

	ε	$f0$	$f1$	$r0$	$r1$	$f0f0$	$f0f1$	$f0r0$	$f0r1$	$\cdots$
ϕ	1.0	1.0	0.0	0.0	1.0	1.0	0.0	0.5	0.5	$\cdots$
$f0$	1.0	1.0	0.0	0.5	0.5	1.0	0.0	0.5	0.5	$\cdots$
$f1$	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	$\cdots$
$r0$	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	$\cdots$
$r1$	1.0	1.0	0.0	0.0	1.0	1.0	0.0	0.5	0.5	$\cdots$
$f0f0$	1.0	1.0	0.0	0.5	0.5	1.0	0.0	0.375	0.375	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Figure 3.2: The Float-Reset system dynamics matrix.

in the matrix is $y(t_j|h_i)$, *i.e.*, the probability of the test t_j being successful when executed immediately following history h_i. Since D contains separate entries for the probability of any test following any history, it is capable of fully describing an arbitrary discrete, finite dynamical system. See Figure 3.1 for an example of how a system dynamics matrix is structured, and see Figure 3.2 for an example of the system dynamics matrix for the Float-Reset problem.

Note that, in some dynamical systems, some histories will never be reached, because the system cannot generate some sequences of actions and observations. We will call such histories *unreachable*, and any history with a non-zero probability of occurring is *reachable*. In Float-Reset, for example, any sequence containing $f1$ is unreachable, because a 0 is *always* observed after the float action is taken. Recall that in the previous chapter, $y(\varepsilon|h) = 0$, when the history h is unreachable. The logical extension of this is that, in the matrix D, rows corresponding to unreachable histories contain only zeros. Unreachable histories are not mentioned in the literature; this definition of unreachable histories and corresponding zero-filled rows are introduced here and are helpful for defining constraints on a valid system dynamics

of $y(t|h)$. As a result of this, the system dynamics matrix described in previous work is defined differently. See Appendix A for further explanation.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

matrix.

3.2 From Matrix to PSR

Even though the system dynamics matrix that describes a dynamical system is infinite in size, we assume most systems have finite linear complexity. For matrices, linear complexity is measured by *rank*. The linear rank r of D is the number of linearly independent columns (or rows) in D, or equivalently, all columns in D can be computed with some combination of r independent columns. Note that this corresponds exactly to the description of PSR core tests given in Chapter 2: there exists a finite set of tests (corresponding to columns in the matrix) capable of describing the entire system. In fact, any system dynamics matrix of rank r can be described by a PSR with r core tests, and tests corresponding to linearly independent columns in D can be used as core tests in the PSR. Thus, given a system dynamics matrix, discovering a set of core tests for a PSR is simply a matter of choosing enough columns to span the space of the matrix.

Given a system dynamics matrix and Q, a set of r linearly independent columns, the parameters of a PSR can be computed using linear regression. Using the sub-matrix $y(Q|H)$, and the column $y(t|H)$, the parameters m_t for all tests $t \in X$ can be computed by:

$$m_t = \left(y(Q|H)^T y(Q|H)\right)^{-1} y(Q|H)^T y(t|H)$$

If $|H| = |Q|$, this can be reduced to:

$$m_t = y^{-1}(Q|H)y(t|H)$$

The former version is used by the TPSR algorithm [Rosencrantz et al., 2004], and the latter form is used in the Monte Carlo algorithms [James and Singh, 2004; Wolfe et al., 2005].

One fact that falls out of viewing core tests as linearly independent columns is that the set of core tests for a dynamical system is not unique. The rank of a matrix does not specify a particular set of columns; it only specifies the number of columns. In truth, *any* set of columns that are linearly independent and span the space of the matrix are sufficient to compute the rest of the matrix.

A second fact is that the null test, which has a value of 1 for every reachable history in the system, can be a core test for *every* PSR. In the POMDP to PSR

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

conversion algorithm [Littman et al., 2002], the null test is used as the starting point for finding linearly independent tests.

3.3 Constraints on the System Dynamics Matrix

A system dynamics matrix possesses a lot of structure. In this section, we describe four constraints on the structure of a valid system dynamics matrix. Although some of the constraints are very simple, this list shows the requirements that a matrix must meet to be a valid system dynamics matrix or a submatrix thereof. Each of the constraints below must be true for all tests $t \in T^*$ and all histories $h \in H^*$.

Range Constraint.

$$0 \leq y(t|h) \leq 1$$

This simple constraint restricts the test prediction values in the matrix to be valid probabilities, between 0 and 1. In combination with the other constraints, this constraint can actually be reduced to $y(t|h) \geq 0$, since the upper bound is taken care of by the null test constraint combined with the internal consistency constraint.

Null Test Constraint.

$$y(\varepsilon|h) = \begin{cases} 1, & y(h|\phi) > 0 \\ 0, & y(h|\phi) = 0 \end{cases}$$

This constraint is used as a base case for the normalization constraint, below. In [Singh et al., 2004], the null test was not considered, and this constraint was listed as:

$$\sum_{\overline{o} \in \mathcal{O}^k} y(\overline{o}|h, \overline{a}) = 1, \quad \forall k, \forall \overline{a} \in \mathcal{A}^k$$

We prefer our notation, since it is simpler and is not redundant with the internal consistency constraint, below. In fact, the null test constraint could be reduced even further to to $y(\varepsilon|\phi) = 1$, because for histories with length greater than zero the equality is enforced by the conditional probability constraint, below.

Internal Consistency Constraint.

$$y(t|h) = \sum_{o \in \mathcal{O}} y(tao|h) \quad \forall a \in \mathcal{A}$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

This constraint guarantees consistency within a single row of the matrix. It ensures that in all cases, the probability distribution over potential observations is valid. It also ensures that no test is more probable than a prefix of itself. See Appendix A for a short proof that all system dynamics matrices satisfy this constraint.

Conditional Probability Constraint.

$$y(t|hao) = \frac{y(aot|h)}{y(ao|h)} \quad \forall a \in \mathcal{A}, o \in \mathcal{O}$$

This constraint ensures consistency among the prediction probabilities between related rows. In cases where $y(ao|h) = y(aot|h) = 0$, we define $\frac{0}{0} = 0$. It is this constraint that is key to most of the structure in a system dynamics matrix. In fact, using this constraint, the entire matrix $y(T^*|H^*)$ can be generated from only the row $y(T^*|\phi)$, as long as this row satisfies the first three constraints [Singh et al., 2004]. See Appendix A for a short proof that all system dynamics matrices satisfy this constraint.

It is interesting to note that all of the constraints on the system dynamics matrix are local; they apply within a single row and each row depends on only a single other row, with a shorter history.

A system dynamics matrix that is generated directly from observed data using a Monte Carlo approach (as in the reset-based and suffix-history algorithms [James and Singh, 2004; Wolfe et al., 2005]) will automatically meet the above constraints. However, such approximations of the system dynamics matrix will rarely have a low linear dimension, since finite sample sizes lead only to approximations of the true probabilities.

For any method that attempts to extrapolate extra information from the observed data and fill in more of the matrix than exactly what is observed, these constraints provide a useful guideline for judging what constitutes a valid system dynamics matrix. We take advantage of these constraints in developing a constrained gradient discovery and learning algorithm in the following chapters.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 4

Constrained Gradient Learning

In Chapter 2, the problems of discovering core tests and learning PSR parameters were both defined. In this chapter, we describe the *constrained gradient algorithm* [McCracken and Bowling, 2006], a new approach to learning the update parameters of a PSR. For now, the focus will be on learning the PSR parameters; the discovery portion of the algorithm will be explained in Chapter 5.

Our goal in creating a new algorithm for discovery and learning of PSRs is to address some of the problems inherent to existing methods. Primarily, the objective of this new algorithm is to create an *online* algorithm for both discovery and learning. An online algorithm is one that can process a stream of action-observation pairs, without making multiple passes over the stream, and at any point in the stream can provide a best estimate of the current PSR state vector and parameters. Online algorithms are desirable because they do not require an explicit learning phase; the model of the system can be constantly updated throughout the algorithm's entire experience with the system. Essentially, an online algorithm never stops learning. Of the current algorithms for learning PSRs, described in Section 2.4.1, only the myopic gradient descent algorithm maintains a current state vector. However, the myopic algorithm is not capable of discovery.

A second goal for our new algorithm is to avoid making assumptions about the properties of representable systems. In particular, the algorithm should not require the presence of a labelled reset action. Both the reset-based Monte Carlo algorithm [James and Singh, 2004] and the TPSR algorithm [Rosencrantz et al., 2004] require reset actions in order to represent controlled systems. Currently, only the suffix-history algorithm [Wolfe et al., 2005] is capable of both discovery and learning without a reset action.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Finally, the third goal of our new algorithm is to attempt to leverage the structure inherent in a system dynamics matrix to create a better representation of the system. Wolfe et al. [2005] state that current algorithms for discovery and learning of PSRs are in "an early stage of development". They do not attempt to extrapolate any information beyond what is present in the stream when learning the prediction probabilities. By using extra knowledge about the structure of a system dynamics matrix, we hope to be able to improve on the speed of learning, and the performance of the learned parameters.

At the moment, there are no algorithms for discovery and learning of PSRs that meet all three of these goals. In this chapter, we describe an algorithm that does. Section 4.1 describes the constrained gradient algorithm for learning a PSR, and Section 4.2 describes experiments on the algorithm and results.

4.1 The Constrained Gradient Algorithm

In this section, we describe the constrained gradient learning algorithm. Throughout this section, we assume prior knowledge of Q, the set of core tests for the system. Section 4.1.1 explains the general approach taken by the algorithm, Section 4.1.2 explains which tests and histories are considered, and Sections 4.1.3 and 4.1.4 describe how the prediction probabilities and PSR parameters are actually computed. The entire algorithm is put together in Section 4.1.5, along with some implementation details.

4.1.1 Approach

The Monte Carlo algorithms [James and Singh, 2004; Wolfe et al., 2005] and the TPSR algorithm [Rosencrantz et al., 2004] can all be summarized by the following description: choose a submatrix of the system dynamics matrix by selecting a set of tests T and a set of histories H, estimate the prediction values in the matrix $y(T|H)$, and use linear regression to compute the parameters, m_X, of the PSR. The constrained gradient algorithm also follows this generic formula, although it uses a very different method than the previously mentioned algorithms.

We will use $\widehat{y}(T|H)$ to refer to the estimate of the true submatrix, $y(T|H)$. One of the major hurdles in creating an estimate of the values in a system dynamics matrix is that, without a reset action, each history in the matrix is experienced at most once. Thus, each prediction has at most a single binary sample, and the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

majority of the test predictions in the matrix have no samples. To generate accurate estimates in $\hat{y}(T|H)$, algorithms need to find some way to combine samples from different histories. The constrained gradient approach uses the properties of the system dynamics matrix to estimate the entire prediction row $\hat{y}(T|h^i)$ at each time i, even though the number of data samples at history h^i is extremely limited. This is possible because most of the information needed to generate those values is contained in the information already seen.

Overall, the general approach taken by the constrained gradient algorithm is to compute an estimate $\hat{y}(T|h^i)$ at each time i, using the known structure of the system dynamics matrix. Together, these row estimates form the submatrix $\hat{y}(T|H)$ that can be used to estimate the parameters, m_X, of the PSR.

4.1.2 Tests and Histories

In this section, we describe the subset of the system dynamics matrix used by the constrained gradient algorithm. We explain the selection of tests, T, and histories, H, considered by the algorithm.

Selecting Tests

The minimal set of tests required for T is the union of the core tests and the extension tests, $Q \cup X$. These are the tests necessary to compute the parameters m_X using linear regression. This minimal set of tests is used by the Monte Carlo algorithms [James and Singh, 2004; Wolfe et al., 2005]. The size of this set of tests is $O(|Q||\mathcal{A}||\mathcal{O}|)$, since there is a test for each combination of action, observation, and core test.

This minimal set of tests is not sufficient for the constrained gradient algorithm, however. The constrained gradient algorithm performs a normalization procedure on the prediction probabilities (explained in Section 4.1.3), which requires the following two properties on the set of tests, T:

1. $tao \in T \Rightarrow t \in T$

   We refer to t as the *parent* test of tao. We use the notation $\pi(tao) = t$. If a test is in T, its parent must also be in T.

2. $tao_i \in T \Rightarrow tao_j \in T \quad \forall o_{j \neq i} \in \mathcal{O}$

   We refer to tests which differ only in the final observation as *sibling* tests. If a test is in T, all of its siblings must also be in T.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

	Properties of Domain				Selection of T		
	$\|\mathcal{A}\|$	$\|\mathcal{O}\|$	$\|Q\|$	d	Minimal	Norm	Full Set
Float-Reset	2	2	5	5	23	61	1365
Tiger	3	2	2	2	13	19	43
Paint	4	2	2	2	17	25	73
Shuttle	3	5	7	2	106	241	241
4x3 Maze	4	6	10	3	241	457	14425
Cheese Maze	4	7	11	3	309	813	22765
Bridge Repair	12	5	5	2	301	961	3661
Network	4	2	7	3	57	105	585

Table 4.1: The sizes of various selections of test sets, T. 'Minimal' is the set $Q \cup X$, 'Norm' is the normalizable set of tests, and 'Full Set' is the exhaustive set of all tests of size d or less. Some properties of each test domain are also shown.

The minimal set of tests is not guaranteed to possess either of these properties. The simplest way to ensure that the tests in T have these properties is to include the exhaustive set of tests that have size less than or equal to the longest test in X. However, even with a small number of actions and observations, this set can become very large if the tests in X are of moderate length. The size of the set is $O\left(|\mathcal{A}|^d|\mathcal{O}|^d\right)$, where d is the length of the longest test in X.

Instead of using the exhaustive set, we select T to include the minimal set of tests necessary to still be able to perform the normalization step. T is initialized to $Q \cup X$, and then iteratively augmented to satisfy the above properties by adding the parent and sibling tests for each test in T. We refer to this set as the *normalizable* set of tests.

In general, the size of this normalizable set of tests is $O(d|Q||\mathcal{A}||\mathcal{O}|^2)$, or no more than a factor of $d|\mathcal{O}|$ larger than the minimal set. This is because adding all parents increases the size of T by a factor no more than d, and adding all of the siblings increases the size by a factor no more than $|\mathcal{O}|$. In practice, the normalizable set is generally much lower than this bound. Table 4.1 shows a comparison of the sizes of T using these three different types of sets, for the test domains used later in this chapter.

Selecting Histories

The set of histories, H, required for the constrained gradient algorithm must meet two conditions:

1. At time i, H must contain h^{i-1}.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2. H must contain 'sufficient' histories to represent the system.

A 'sufficient' set of histories is one that contains enough diverse rows that the full rank of the system dynamics matrix is represented. In general, since the complexity of the system is unknown, it is never possible to guarantee that the second condition is met. The best approximation would be for H to include *all* histories experienced so far. However, this approach would become computationally infeasible after a large number of time steps. Currently, we select H to be the previous n histories encountered in the system, where n is a parameter to the algorithm. In Section 4.2.2, we investigate the choice of n. This finite window method guarantees that the first condition is satisfied, as long as $n \geq 1$. This method does not necessarily guarantee the second condition, but for larger values of n, the probability that H is sufficient increases.

Using a finite window of histories has several benefits. A main benefit of using the most recent histories in H is that $\widehat{y}(T|H)$ always contains the most recent data. Because we expect that our row estimates become more accurate as more data is seen, $\widehat{y}(T|H)$ will therefore contain the most accurate rows. A second benefit of using a block of consecutive histories is that the states represented by histories in H occur proportional to the frequency that they are encountered by the system. Thus, in the regression step, more frequently encountered states will be proportionally represented and have a greater impact in computing the parameters than infrequent states. A third benefit of the finite window approach is that keeping H a constant size keeps the per-time step computation constant.

The main drawback of using a finite window is that n must be large enough such that, at all times, it contains enough histories to fully represent the system. The problem is essentially the same as that suffered by history-based models; it is possible that all the states of the system are not represented in the previous n time steps, and therefore the representation can lose track of some states of the system. This problem does not affect the constrained gradient algorithm as much as it affects history-based methods, though. The number of possible histories grows exponentially with the size of n, so when using history-based methods, choosing a large value for n is intractable. Using the constrained gradient algorithm, choosing a larger n results only in a linear increase in the memory and computation requirements of the algorithm. For example, it would be be infeasible to use $n = 1,000$ in a history-based method, because even in a system with two observations and no ac-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

tions, 2^{1000} is too many histories to represent; however, in the constrained gradient algorithm, $|H| = 1,000$ is a reasonable size.

One can imagine alternatives to a simple finite window approach. For instance, H could contain the histories whose rows are the most orthogonal, and thus are most likely to contain information about different states. This approach is similar to the one used by the Monte Carlo algorithms, and has the benefit that H can be very small and the algorithm does not suffer from the history-based method drawback. However, a small H means that the linear regression is less-constrained and could overfit. Also, choosing linearly independent rows could suffer from preferring rows with higher error, since these rows may appear more orthogonal to other rows. A hybrid method could also be possible, which combines the advantages of storing a large number of recent histories with the advantages of keeping linearly independent rows.

4.1.3 Constructing the Prediction Matrix

In this section, we describe how each row $\hat{y}(T|h^i)$ is estimated, after each new data point a^io^i is observed. This estimation process uses the constraints listed in Section 3.3 to compute an estimate of each prediction probability, even though most of these probabilities are never sampled. Computing an estimate $\hat{y}(T|h^i)$ involves three steps. In the first step, some of the predictions are computed directly from the previous row, $\hat{y}(T|h^{i-1})$. In the second step, the remaining predictions are computed using linear regression. These first two steps are the constrained part of the constrained gradient algorithm. In the third step, the predictions are adjusted in the direction of the observed data; this is the gradient step of the algorithm.

When a new action-observation pair a^io^i is observed, the tests in T can be divided into two sets: T_1 contains all the tests $t \in T$ such that $a^io^it \in T$, and T_2 contains all of the remaining tests, *i.e.* $T - T_1$. Although exactly which tests are included in each set vary, it is always true that $Q \subseteq T_1$. This is true by definition, because $X \subseteq T$, and X contains a^io^iq for all $q \in Q$.

The first step in computing $\hat{y}(T|h^i)$ is to compute $\hat{y}(T_1|h^i)$. All of the information required for this step is already contained in the previous row, $\hat{y}(T|h^{i-1})$. Using the conditional probability property of the matrix entries, we know that

$$y(t|h^i) = y(t|h^{i-1}a^io^i) = \frac{y(a^io^it|h^{i-1})}{y(a^io^i|h^{i-1})}$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Estimates of $y(a^i o^i|h^{i-1})$ are always available, and T_1 was constructed such that estimates of $y(a^i o^i t|h^{i-1})$ are also available for all $t \in T_1$. Thus, estimates of the values in $y(T_1|h^i)$ can be easily computed from available data.

In the second step, estimates for $y(T_2|h^i)$ are computed. To do so, the algorithm uses the fact that $\widehat{y}(Q|h^i)$ was computed in the previous step. Each prediction $\widehat{y}(t|h^i)$ can be computed from $\widehat{y}(Q|h^i)m_t$, for some weight vector m_t. The weight vector can be found by using linear regression to find the m_t that minimizes $|\widehat{y}(Q|H)m_t - \widehat{y}(t|H)|^2$.

Computing $\widehat{y}(T_2|h^i)$ using regression can create values that violate the range and internal consistency properties of the system dynamics matrix. The range constraint is enforced by setting any negative entries to a small positive value. The internal consistency constraint is enforced by a normalization step on the probability values. For each test tao_j,

$$\widehat{y}(tao_j|h^i) \leftarrow \widehat{y}(t|h^i)\frac{\widehat{y}(tao_j|h^i)}{\sum_{o \in \mathcal{O}} \widehat{y}(tao|h^i)}$$

This has the effect of maintaining the ratios between siblings tests, while ensuring that they sum to the value of their parent. The normalization is performed first on length one tests, since in that case $t = \varepsilon$ and $y(\varepsilon|h^i)$ is always 1, then on length two tests, since t is length one and thus $\widehat{y}(t|h^i)$ has already been normalized, *etc.*, until all tests in T have been normalized. If for any reason the value $\sum_{o \in \mathcal{O}} \widehat{y}(tao|h^i)$ is zero, then each test tao_j is set to $\widehat{y}(t|h^i)/|\mathcal{O}|$. By construction, the set T ensures that all tests required to perform this normalization step are present in T.

Note that it is this normalization step that ensures that each entry in the matrix approximates $y(t|h)$, instead of $\Pr(\overline{o}_t|h, \overline{a}_t)$. The normalization update equation, above, is a simplified view of the actual normalization process. The samples from the data stream occur according to the probability $\Pr(\overline{o}_t|h, \overline{a}_t)$, which takes into account the policy generating the actions. Thus, before the values $y(tao_j|h^i)$ are normalized, they contain values sampled from $\Pr(\overline{o}_t o_j|h^i, \overline{a}_t a)$. During normalization, probabilities are divided by the sum of their siblings.

$$\begin{aligned}
\frac{\widehat{y}(tao_j|h^i)}{\sum_{o \in \mathcal{O}} \widehat{y}(tao|h^i)} &\approx \frac{\Pr(\overline{o}_t o_j|h^i, \overline{a}_t a)}{\sum_{o \in \mathcal{O}} \Pr(\overline{o}_t o|h^i, \overline{a}_t a)} \\
&= \frac{\Pr(\overline{o}_t|h^i, \overline{a}_t a)\Pr(o_j|h^i, ta)}{\Pr(\overline{o}_t|h^i, \overline{a}_t a)\sum_{o \in \mathcal{O}} \Pr(o|h^i, ta)} \\
&= \frac{\Pr(o_j|h^i, ta)}{\sum_{o \in \mathcal{O}} \Pr(o|h^i, ta)} \\
&= \Pr(o_j|h^i, ta)
\end{aligned}$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Thus, the normalization procedure removes the effect of the policy on the sampled values. Using the above to derive the normalization step:

$$\begin{aligned}
\widehat{y}(tao_j|h^i) &\leftarrow \widehat{y}(t|h^i)\frac{\widehat{y}(tao_j|h^i)}{\sum_{o\in O}\widehat{y}(tao|h^i)} \\
&\approx \widehat{y}(t|h^i)\frac{\Pr(\overline{o}_t o_j|h^i, \overline{a}_t a)}{\sum_{o\in O}\Pr(\overline{o}_t o|h^i, \overline{a}_t a)} \\
&= \widehat{y}(t|h^i)\Pr(o_j|h^i, ta) \\
&= \widehat{y}(tao_j|h^i)
\end{aligned}$$

Thus, entry in the matrix is being changed to approximate $y(t|h)$.

At this stage, an estimate for all of the predictions in $y(T|h^i)$ has been computed. The final step in computing $\widehat{y}(T|h^i)$ is the gradient step; the observations that are actually encountered in the data stream are used to adjust the estimated predictions. The constrained gradient algorithm uses a Monte Carlo update for this step. For instance, the prediction $\widehat{y}(a^{i+1}o^{i+1}\ldots a^{i+k}o^{i+k}|h^i)$ should have its probability increased, since the observation sequence $o^{i+1}\ldots o^{i+k}$ is actually observed after taking the action sequence $a^{i+1}\ldots a^{i+k}$. The predictions $\widehat{y}(a^{i+1}o_j\ldots a^{i+k}o_l|h^i)$, where $o_j\ldots o_l \neq o^{i+1}\ldots o^{i+k}$, should have their probability decreased, since those tests were executed and their observations were not seen. The gradient step is as follows: for each test $t = a^{i+1}o^{i+1}\ldots a^{i+k}o^{i+k}$, we adjust the value of $\widehat{y}(t|h^i)$ towards the probability of its parent, using:

$$\widehat{y}(t|h^i) \leftarrow (1-\alpha)\widehat{y}(t|h^i) + \alpha\widehat{y}(\pi(t)|h^i)$$

The probability of the parent is used as the target because it is the maximum value of $\widehat{y}(t|h^i)$. The probabilities of the unobserved sibling tests of t are decreased, while maintaining the ratio of their probabilities. In practise, the adjustment of t and its siblings can be done by adding a positive value to $\widehat{y}(t|h^i)$, and then re-running the normalization step on the row. The positive value, x, can be found by solving:

$$\frac{\widehat{y}(t|h^i)+x}{\widehat{y}(\pi(t)|h^i)+x} = (1-\alpha)\widehat{y}(t|h^i) + \alpha\widehat{y}(\pi(t)|h^i)$$

This states that the ratio of the current prediction plus x and the parent prediction plus x must be equal to the desired value of $\widehat{y}(t|h^i)$. Solving for x yields:

$$x = \frac{\widehat{y}(\pi(t)|h^i) - \alpha}{1-\alpha}$$

The value of α controls the learning rate; a high value moves the prediction very close to its maximum value, while a small value tweaks the prediction only slightly. The

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

value of α should be decayed during learning, so that the algorithm will eventually converge on a solution. Decay policies for α are investigated in Section 4.2.2.

When learning online, the information to adjust the predictions in row $y(T|h^i)$ is not available at time i, since the action-observation data is available from a stream. In practise, the algorithm maintains a buffer of action-observation pairs of length d, the length of the longest test in T. When $a^i o^i$ is observed, the row $\hat{y}(T|h^{i-d})$ can be computed and the buffer of actions and observations $a^{i-d+1}o^{i-d+1}\ldots a^i o^i$ is used to adjust the prediction values. This is the same approach that is used in the myopic gradient descent algorithm. This approach has the disadvantage that the algorithm does not explicitly have the estimate of the current state vector $\hat{y}(Q|h^i)$, although it can be easily computed by successively generating rows for histories between h^{i-d} and h^i using the buffered action-observation pairs.

4.1.4 Extracting the PSR Parameters

If the constrained gradient algorithm is being used online, extracting PSR parameters is unnecessary, since the algorithm maintains up-to-date prediction probabilities for the core tests Q, as well as all other tests in T. However, if a final PSR is necessary, one can be easily generated from the data structure $\hat{y}(T|H)$ used by the constrained gradient algorithm. The necessary parameters are a state vector and the weight vectors m_X.

There are three options for computing the state vector of the PSR: the current state vector, the initial state vector, and the stationary distribution state vector. The current state vector is $\hat{y}(Q|h^i)$, for the most recent history h^i. This state vector represents the state of the system as it was last experienced by the learning algorithm. Extracting a current state vector is possible because the constrained gradient algorithm is an online algorithm; with the exception of the myopic algorithm, previous PSR learning methods are unable to generate a current state vector. The initial state vector is $\hat{y}(Q|\phi)$, the vector that represents the state in the initial distribution of the system. This state vector is used if the generated PSR is for a system that has been reset. However, only the reset-based Monte Carlo algorithm is capable of generating this state vector, because without a reset action, algorithms cannot create an accurate estimate of the initial distribution. Instead, the constrained gradient algorithm uses the stationary distribution of the system, if a stationary distribution exists. The state vector for the stationary distribution can be computed by

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$\frac{1}{|H|}\sum_{h\in H} y(Q|h)$, where H contains all available histories. This is also the approach used by the suffix-history Monte Carlo algorithm, and is the approach used by the constrained gradient algorithm to create a PSR that will be used offline. Of these three types of state vectors, the most important is the current state vector, since it would be used in more practical settings.

Regardless of the type of state vector used, the update parameters of the PSR do not change. The update parameters of the PSR, m_X, are computed using linear regression on the columns of $\widehat{y}(X|H)$. For each $t \in X$, the weight vector m_t is computing by finding the vector that minimizes $|\widehat{y}(Q|H)m_t - \widehat{y}(t|H)|^2$.

4.1.5 The Complete Algorithm

In this section, we summarize the above description of the constrained gradient algorithm, and indicate any implementation details left out of the above discussion for simplicity. Algorithm 1 shows the constrained gradient algorithm.

The computational complexity of the constrained gradient algorithm is dominated by the complexity of the regression step used to compute the parameters m_t for each test t. In this step, m_t is computed by:

$$m_t \leftarrow \left(\widehat{y}(Q|H)^T\widehat{y}(Q|H)\right)^{-1}\widehat{y}(Q|H)^T\widehat{y}(t|H)$$

Note that the first part of this computation is not dependent on the test, t. Thus,

$$A \leftarrow \left(\widehat{y}(Q|H)^T\widehat{y}(Q|H)\right)^{-1}\widehat{y}(Q|H)^T$$

can be computed once at each time step, and each m_t is then computed by

$$m_t \leftarrow A\widehat{y}(t|H)$$

Overall, computing A has complexity $O(|Q|^2|H| + |Q|^3)$ and computing each m_t has complexity $O(|Q||H|)$. This gives a total per-time step complexity of $O(|Q|^2|H| + |Q|^3 + |T||Q||H|)$, or simply $O(|T||Q||H|)$ since $|T| > |Q|$ and generally $|H| \gg |Q|$. The relatively high computational complexity is one of the greatest disadvantages of the constrained gradient algorithm over existing algorithms, which use essentially constant computation per time step. One way to reduce the computation used by the constrained gradient algorithm could be to not re-compute m_t every time step, since it is not likely to change greatly between consecutive time steps. A linear speed-up of n times could be expected if the weights m_t are computed every n time steps, but it could come at the expense of prediction quality.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Algorithm 1 The constrained gradient learning algorithm.

```
Require: 𝒜   // the set of actions
Require: 𝒪   // the set of observations
Require: Q   // the set of core tests
Require: α   // the learning parameter
Require: n   // the number of rows of history kept
Require: a   // the action stream
Require: o   // the observation stream
```

1: initialize T to normalizable set of tests
2: initialize $\widehat{y}(T|h^0)$ to uniform probabilities
3: $i \leftarrow 0$
4: **repeat**
5: $\quad i \leftarrow i+1$
6: $\quad T_1 \leftarrow \{t \,|\, a^i o^i t \in T, t \in T\}$
7: $\quad T_2 \leftarrow T - T_1$
8: $\quad$ `// compute` $\widehat{y}(T_1|h^i)$
9: $\quad$ **for all** $t \in T_1$ **do**
10: $\quad\quad \widehat{y}(t|h^i) \leftarrow \frac{\widehat{y}(a^i o^i t|h^{i-1})}{\widehat{y}(a^i o^i|h^{i-1})}$
11: $\quad$ **end for**
12: $\quad$ `// compute` $\widehat{y}(T_2|h^i)$
13: $\quad$ **for all** $t \in T_2$ **do**
14: $\quad\quad m_t \leftarrow \text{argmin}_m |\widehat{y}(Q|H)m - \widehat{y}(t|H)|^2$
15: $\quad\quad \widehat{y}(t|h^i) \leftarrow \widehat{y}(Q|h^i)m_t$
16: $\quad$ **end for**
17: $\quad$ run normalization step on $\widehat{y}(T|h^i)$
18: $\quad$ `// update observed entries in the row`
19: $\quad$ **for all** $t \in T$ **do**
20: $\quad\quad k \leftarrow$ length of t
21: $\quad\quad$ **if** $t = a^{i+1}o^{i+1} \ldots a^{i+k}o^{i+k}$ **then**
22: $\quad\quad\quad \widehat{y}(t|h^i) \leftarrow \widehat{y}(t|h^i) + \frac{\widehat{y}(\pi(t)|h^i)-\alpha}{1-\alpha}$
23: $\quad\quad\quad$ run normalization step on $\widehat{y}(T|h^i)$
24: $\quad\quad$ **end if**
25: $\quad$ **end for**
26: $\quad$ discard the row $\widehat{y}(T|h^{i-n})$ from $\widehat{y}(T|H)$
27: $\quad$ add the row $\widehat{y}(T|h^i)$ to $\widehat{y}(T|H)$
28: **until** end of streams a, o

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Computing weight vectors m_t requires a matrix inversion step in the linear regression. If the core test matrix $\widehat{y}(Q|H)$ is not full rank, then this inversion cannot be performed. To avoid problems, we use a regularized linear regression by adding a small value λ to the diagonal elements of the matrix that will be inverted. This ensures that the matrix is invertible; it also has the side-effect of biasing the computed solution vectors m_t by penalizing large weights. In this work, we used $\lambda = 10^{-4}$.

Another problem with the algorithm can occur if the entry $\widehat{y}(a^i o^i | h^{i-1})$ is ever zero; *i.e.*, if the algorithm estimates zero probability for an event that actually occurs. This would result in division by zero when the entries for row $\widehat{y}(T|h^i)$ are computed. To avoid this, we place a lower bound of 10^{-5} on the probabilities in the matrix. The lower bound is enforced in the normalization step. The lower bound can affect the quality of learned PSRs, if the PSR actually does have zero probabilities, but with a small enough bound the effect on quality should be negligible. It may be desirable to remove or decrease the lower bound later in learning, when it can be reasonably certain that some events actually have zero probability; however, this was not done in our implementation or investigated in our tests.

When $\widehat{y}(t|h^i)$ is incremented in line 22 of Algorithm 1, the value $1 - \alpha$ is used in the denominator. Thus, we require $\alpha < 1$ to avoid division by zero errors.

In the above algorithm, the row $\widehat{y}(T|h^0)$ is initialized to uniform probabilities. To clarify, this means that for each test $t \in T$, $\widehat{y}(t|h^0)$ is set to $1/|\mathcal{O}|^{len(t)}$. Thus, the probabilities are uniform with respect to their depth. Since no information is known about the system at this point, a uniform distribution seems like a reasonable starting point for the algorithm. If additional information about the system is known, such as the initial distribution or the stationary distribution, the row $\widehat{y}(T|h^0)$ could be initialized appropriately.

One thing to note when adjusting the prediction values of tests to fit the data is to avoid adjusting a prediction value more than once for the same piece of data. For instance, if $y(a^1 o^1 a^2 o^2 | \phi)$ is adjusted, then its modified value will propagate to subsequent rows using the conditional probability property. On the next time step, $y(a^2 o^2 | a^1 o^1)$ should not be adjusted, because it would essentially be double-counting the $a^2 o^2$ data point. Thus, we add the condition that each data point $a^i o^i$ can only be used to adjust a single prediction that ends in $a^i o^i$.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.2 Experiments

In this section, we experimentally investigate the learning capabilities of the constrained gradient algorithm. The purpose of this section is to investigate how the constrained gradient algorithm performs in comparison to other PSR learning algorithms. In Section 4.2.1, we describe the experimental setup and test domains used in these experiments. Section 4.2.2 investigates the effects of parameter selection on the algorithm. Section 4.2.3 presents offline experiments and results, and Section 4.2.4 presents online experiments and results.

4.2.1 Experimental Setup

The experiments in this section, and throughout the rest of this work, use a set of test domains from an online repository [Cassandra, 1999]. This set of domains has become an unofficial standard test set in the PSR literature, as it has been used in experiments in [Singh et al., 2003; James and Singh, 2004; Wolfe et al., 2005]. The domains were all originally designed for experiments with POMDPs, and all are composed of a finite number of nominal states with dynamics that can be modelled by a POMDP. A more detailed explanation of each domain can be found in Appendix B. The number of actions, observations, and linear dimension of each domain is in Table 4.1.

In all of the experiments in this section, the constrained gradient algorithm is provided with a correct set of core tests for the system, because we wish to investigate only the learning capabilities of the constrained gradient algorithm. Each reported result is the mean of 10 trials. In each trial, the constrained gradient algorithm learns a model of the system by processing a data stream of 1,000,000 action-observation pairs. The actions in these data streams were generated using a uniform policy over actions.

In order to determine the performance of the generated models, a PSR was extracted from the algorithm's learned model at various points during learning. Its error was measured using the same method as described by Wolfe et al. [2005]. Prediction error is measured on a test stream of 10,000 action-observation pairs that have been annotated with the true probabilities $y(a^i o_j | h^{i-1})$ for each observation o_j at each time step. At each time step, the difference between the PSR's prediction probabilities, $\widehat{y}(a^i o_j | h^{i-1})$, and the true probabilities, $y(a^i o_j | h^{i-1})$, is recorded. The

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

overall error is:

$$\frac{1}{T}\sum_{i=1}^{T}\frac{1}{|\mathcal{O}|}\sum_{o_j\in\mathcal{O}}\left(\widehat{y}(a^io_j|h^{i-1}) - y(a^io_j|h^{i-1})\right)^2$$

where T is 10,000, the length of the test stream. Note that this is an offline method of measuring error, meaning it does not make any predictions about the stream on which the PSR was learned. While updating the PSR to measure error, normalization is performed after the state vector is updated, in order to prevent small errors from accumulating. The normalization involves restricting the state vector probabilities to the range (0,1]; a small lower bound on prediction error is enforced to prevent division-by-zero errors. Normalization is not performed when computing the observation probabilities $\widehat{y}(a^io_j|h^{i-1})$ that are used to compute the error. Thus, recorded error can possibly be greater than 1.

4.2.2 Parameter Selection Experiments

The constrained gradient algorithm has several parameters that can be tuned, and in this section, we examine how the parameterization of the constrained gradient algorithm affects its performance. The two parameters we appraise are n, the number of rows of history kept (*i.e.*, the size of H), and the decay of α, the learning rate that determines by how much the constrained gradient algorithm follows the gradient. We will use the results of these experiments to determine the parameterization of the algorithm used for the rest of the experiments.

Learning Rate Parameter

In the experiments to select a policy for decay of the learning parameter, α, we tested six different decay policies. In two of them, α remained constant for the duration of learning. We used $\alpha = 0.99999 \approx 1$ as a large constant value, which has the effect of setting all observed probabilities to almost 1 and not-observed probabilities to almost 0 (they are not exactly 0 and 1 because of the lower bound on probabilities). We used $\alpha = 0.1$ as a small constant learning rate. For the other policies, α was initialized to 1 and decayed over time, using two types of decay. Using 'sudden decay', α is halved every k data points. Using 'gradual decay', α is divided by $\sqrt[k]{2}$ at every time step. This has the effect of halving α every k data points, but does so gradually instead of making large changes. We tested $k = 100,000$ and $k = 250,000$ for both styles of decay. The other parameter, $|H|$, was set to 1,000.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

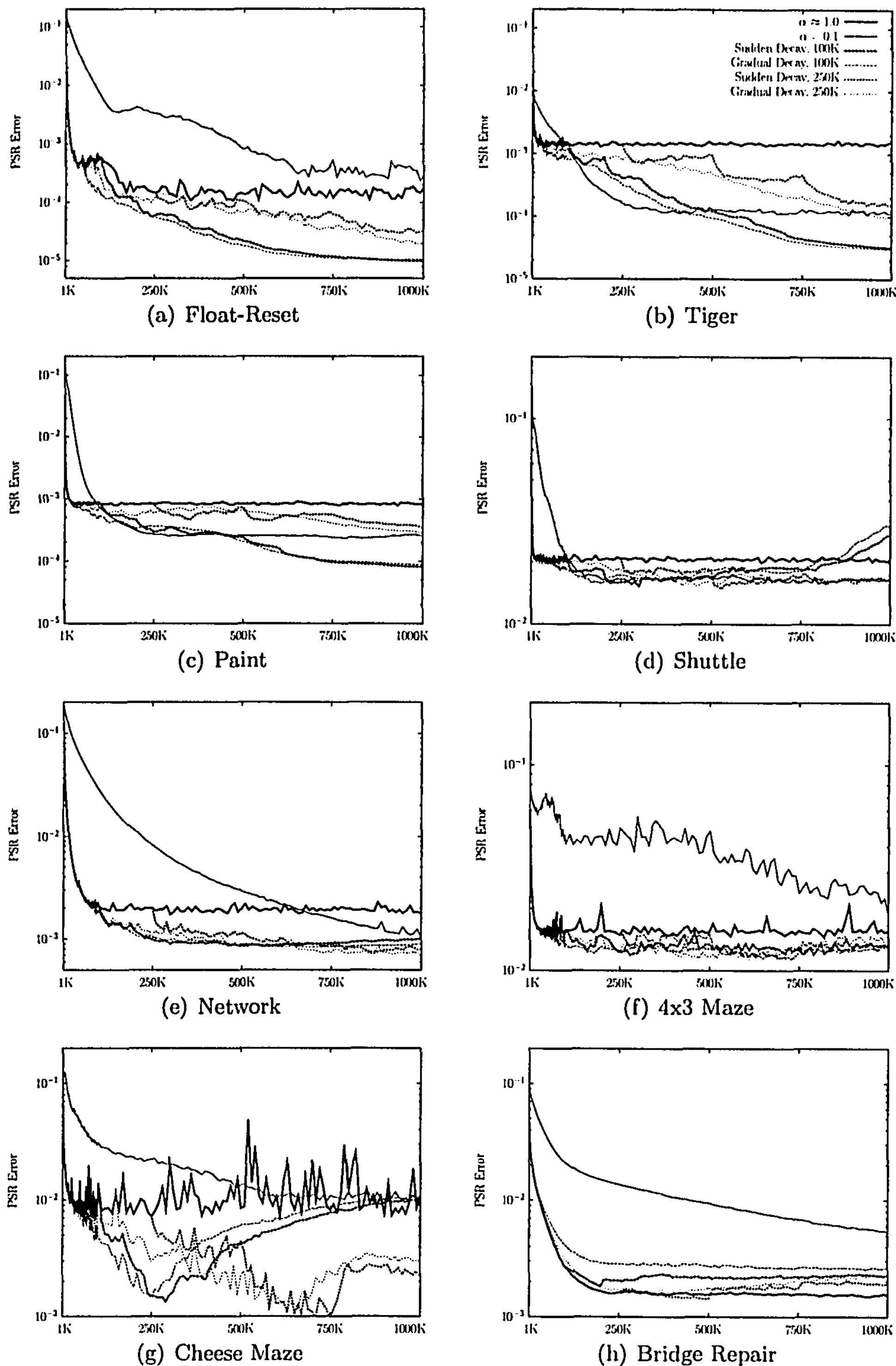


Figure 4.1: The effect of changing the learning parameter, α. The horizontal axis shows the number of data points used, and the vertical axis is the PSR error. Each line represents a different style of α decay.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 4.1 shows the results of the α-decay experiments. Overall, we see that the exact α-decay policy can matter a great deal when learning, and the best decay policy is dependent on the system. Some generalizations can be made, however. Using a constant $\alpha = 1.0$ performs poorly across most domains, and tends to stop learning early, since the prediction adjustments are very coarse grained. Using a large learning rate also seems to create a more unstable representation; as can be seen by the "jaggedness" of the lines in the plots in Figure 4.1. This is a result of the values in $\widehat{y}(T|H)$ changing by a larger amount.

The algorithm performs better using a smaller constant value of $\alpha = 0.1$. Like $\alpha = 1.0$, learning eventually stops, although generally at a more accurate model than with a higher learning rate. This can be best seen in the plots for the Tiger and Paint domains. In all cases, as expected, the early performance of $\alpha = 0.1$ is the worst of all α-decay polices due to slower learning.

Of the decaying policies, the choice of k, the halving interval, matters more than whether gradual or sudden decay was used. For the simpler domains, Float-Reset, Tiger and Paint, using $k = 100,000$ worked best, since it allowed more fine-grained changes in the prediction probabilities. In the Network, 4x3 Maze, and Bridge domains the performance of $k = 100,000$ and $k = 250,000$ were roughly equivalent. Strangely, in the domains Shuttle and Cheese Maze, the models that used a smaller learning rate actually increased in error later in learning. This strange behaviour is investigated further in Section 6.2.2.

Overall, we can see that different domains have different requirements of the learning parameter. Unsurprisingly, simpler domains benefit from a learning rate that decays quickly to allow for more fine-grained learning. More complicated domains require a larger learning rate for a longer period of time. For our remaining experiments with the constrained gradient algorithm, we will use the 'sudden decay' policy with $k = 100,000$, since this policy works fairly well in most domains and does not obscure the increase in errors in the Shuttle and Cheese Maze domains.

Size of History Set

The number of rows of history kept in H is an important parameter. Larger sizes of H mean that the algorithm takes longer to run, since more samples are used in the regression steps, and also that old data is kept longer. Small sizes of H mean there is a higher probability that H may not contain sufficient rows to fully represent

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the system, and also that there are fewer data samples in the regression step which could lead to overfitting. The tested sizes of H were 100, 500, 1,000, 5,000, and 10,000.

The plots in Figure 4.2 show the results of the experiments on the size of H. Once again, we see that the parameterization of the constrained gradient algorithm has a large effect on the performance of the algorithm. One generalization that can be formed from the data is that keeping more histories leads to slower learning. This is best evidenced in the Float-Reset, Tiger and Paint domains. The reason is that, with more histories, each observed data point has a smaller effect on the model as a whole. We also see that the models with 5,000 and 10,000 histories stop learning earlier. There is a relationship between α and the number of histories kept; an α of a given size will have less effect on a model that keeps more rows. Thus, in the models with many histories the size of α becomes negligible faster than in models with fewer histories. A related point is that keeping more histories creates a smoother error line, since the model changes less between data points.

Across all domains, the models that used 100 histories performed poorly. It was also quite evident in the 4x3 Maze and Cheese domains that 100 histories was generally not sufficient to represent the entire state. This shows in the plots by the extreme jaggedness of the error lines. It makes sense that the insufficiency would be most apparent in these domains, since they have the highest linear dimension.

In Tiger, Paint and Shuttle, more evidence is given that error can increase with more data. Notably, this process happens more in models that maintain fewer histories. This indicates, perhaps, a relation between the sufficiency of the set H and the likelihood of falling into a local minima.

Overall, if data is unlimited and learning will continue for a very long time, using more histories and a higher α is probably better, since it will lead to a more stable representation. However, in general we expect that data is limited, or expensive, and both learning and computation speed are important. Therefore, we must choose a trade-off between a sufficient number of histories and a feasible number of histories. For the rest of our experiments, we will use $|H| = 1,000$, since it performs reasonably well across all domains.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

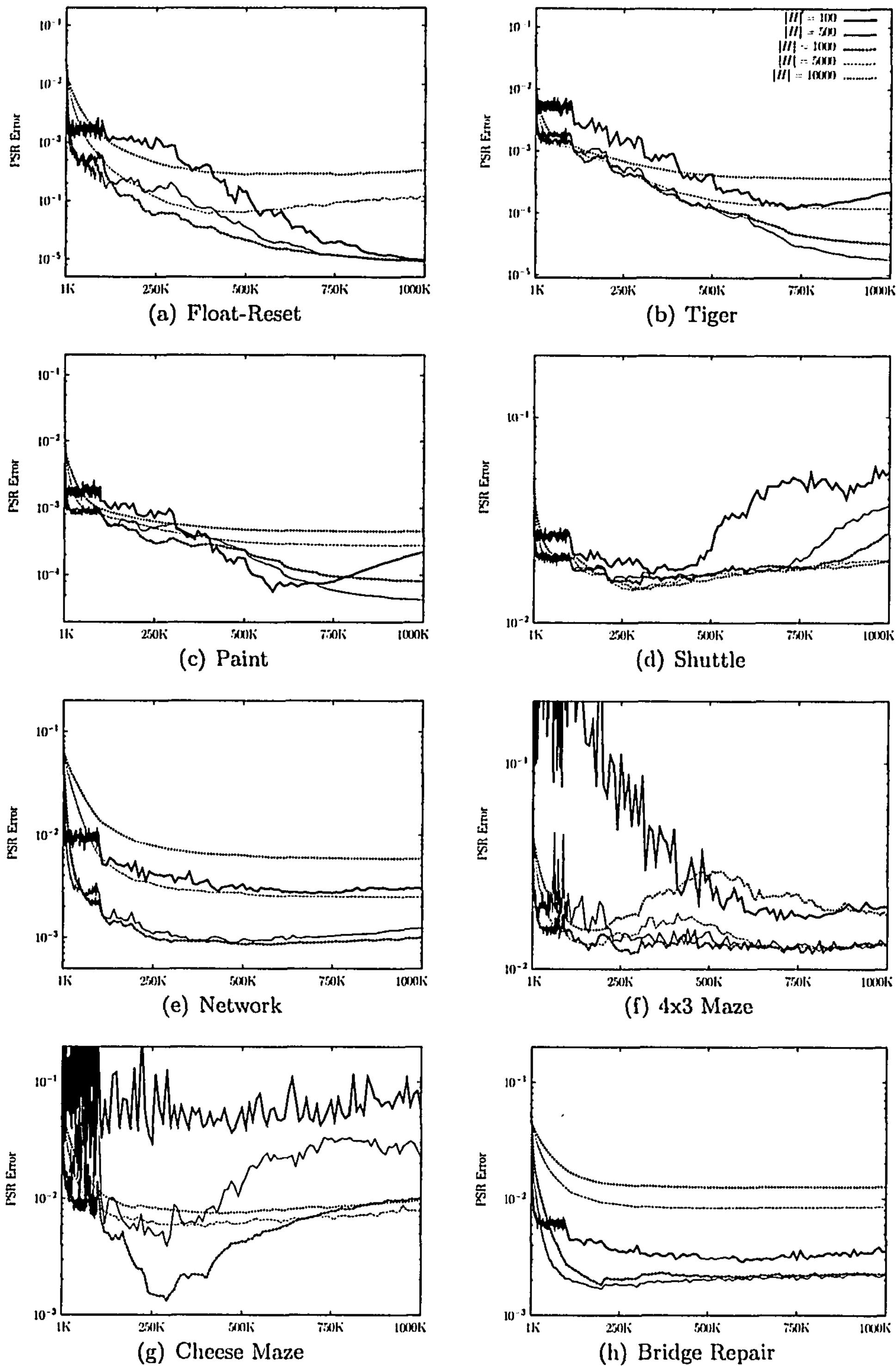


Figure 4.2: The effect of changing the number of histories in H. The horizontal axis shows the number of data points used, and the vertical axis is the PSR error. Each line represents a different number of histories in H.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.2.3 Offline Experiments

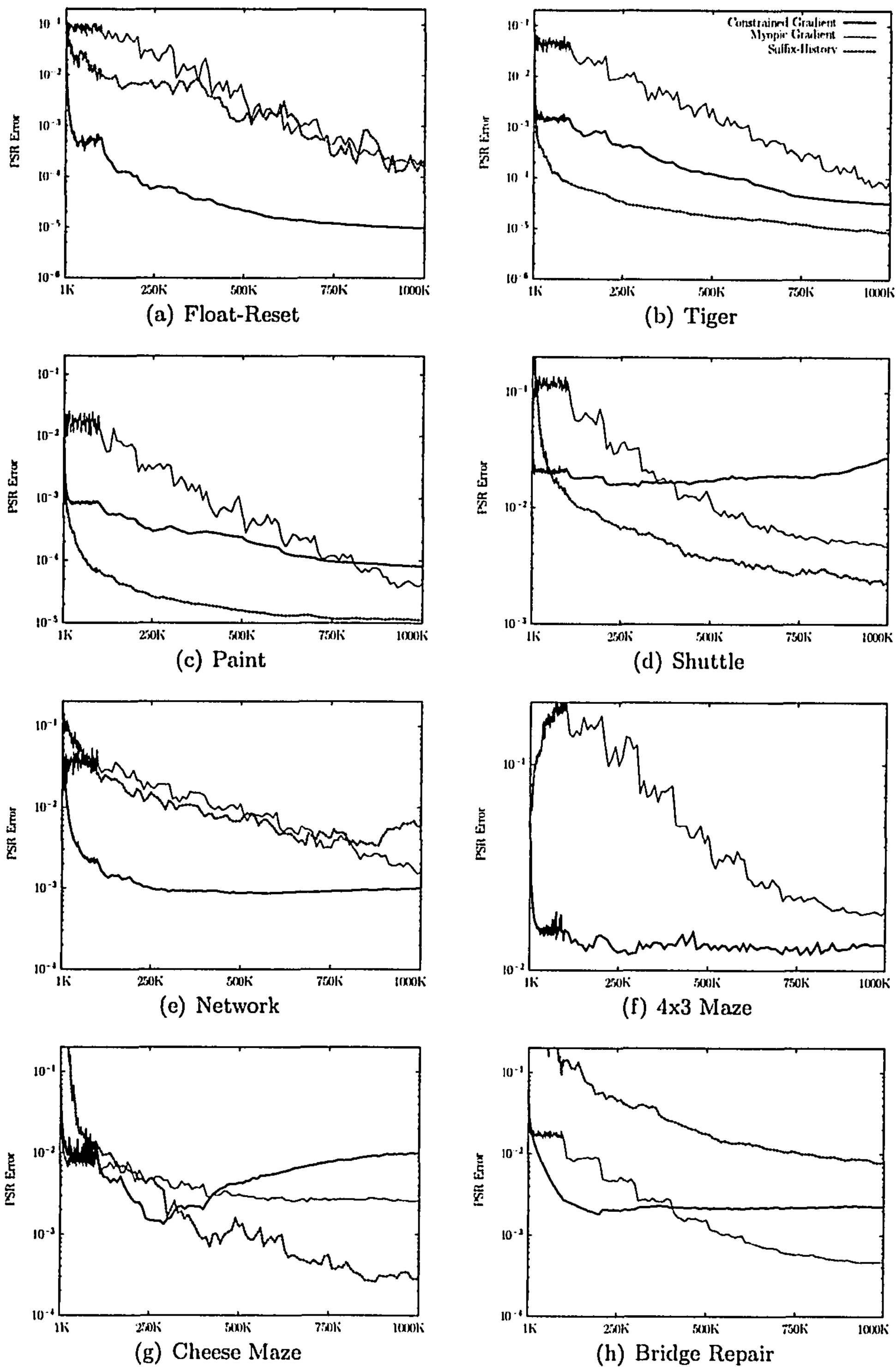
In this section, we examine the performance of the constrained gradient algorithm using the offline measure of PSR error. Its performance is compared to the suffix-history algorithm, with the modification that the suffix-history algorithm is provided with a correct set of core tests. Performance is also compared to the myopic gradient descent algorithm. Figure 4.3 shows the results for these experiments.

For the myopic gradient algorithm, the learning rate α is initialized to 0.5 and halves every 100,000 data points, as was suggested in [Singh et al., 2003]. This is why the performance of the myopic algorithm tends to have sharp transitions. Compared to the myopic gradient algorithm, the constrained gradient algorithm learns an initial model very quickly. This gives evidence for our claim that the constrained gradient algorithm is able to use the constraints on the system dynamics matrix to learn more effectively. However, in many of the domains, like Bridge Repair, Paint and Shuttle, the performance of the constrained gradient algorithm tends to plateau, while the myopic gradient algorithm continues to learn a better model.

One possible explanation is that the constrained gradient algorithm may be more susceptible to local minima than the myopic algorithm. If the algorithm reached a local minima, it would have the same behaviour as evidenced in some of the plots in Figure 4.3. The reason that the constrained gradient algorithm may be more susceptible to minima is that it contains a lot of self-propagating information. After each data point, only a small number of the predictions in the large matrix $\widehat{y}(T|H)$ are changed. Because the entire matrix is used in computing each new prediction, the small changes made from following the gradient on the data may not be enough. Chapter 6 further investigates these issues with the performance of the constrained gradient algorithm.

The results for the suffix-history domain are also shown in Figure 4.3. The algorithm was implemented from the published details [Wolfe et al., 2005]. The performance of the suffix-history algorithm varied greatly among the domains. In some domains, like Tiger, Paint and Cheese Maze, it creates a very good model of the system and performs better than both the myopic algorithm and the constrained gradient algorithm. In other domains its performance is approximately competitive with the myopic gradient algorithm. In the 4x3 Maze domain, the model created by suffix-history is too poor to show on the plot. These results conflict with the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) Float-Reset (b) Tiger (c) Paint (d) Shuttle (e) Network (f) 4x3 Maze (g) Cheese Maze (h) Bridge Repair

Figure 4.3: PSR error in offline tests. The horizontal axis shows the number of data points used, and the vertical axis is the PSR error.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

published results for the algorithm, which show good performance in all of these domains. Informal experiments show that the suffix-history algorithm can be quite sensitive to small inaccuracies in the matrix $\widehat{y}(T|H)$. It is also very sensitive to the choice of suffixes; care was taken to choose a good set for each domain, based on likelihood of occurring and linear relatedness. We believe that the discrepancies between our results for suffix-history and the published results are due to these sensitivities.

4.2.4 Online Experiments

These experiments are intended to determine the performance of the constrained gradient algorithm when used for online predictions. This performance is compared to the performance of the myopic gradient descent algorithm, which is currently the only other PSR learning algorithm capable of making online predictions.

Unlike the previous experiments, the PSR errors shown in the results for these experiments do not use an independent test stream to measure error. Online error at each time step i is measured by $\frac{1}{|\mathcal{O}|}\sum_{o_j \in \mathcal{O}}(\widehat{y}(a^{i+1}o_j|h^i) - y(a^{i+1}o_j|h^i))^2$, which computes the squared prediction error for the next time step, averaged over all observations. Computing this requires that the original data stream is annotated with the true prediction probabilities. In the results presented here, the online error is averaged over the previous 1,000 data points, in order to smooth the values.

Figure 4.4 shows the results for testing online performance. The most striking feature of the online performance is that it matches the offline performance extremely closely. This means that the average prediction error over 10,000 steps (the offline error) is essentially the same as the average error for predicting only a single step ahead (the online error). This indicates that PSRs are not very prone to *drifting*. Drifting is when small errors in the PSR state vector accumulate over time, and eventually create increasingly inaccurate predictions.

4.2.5 Summary of Learning Results

Overall, the learning results for the constrained gradient algorithm are encouraging. They show that the algorithm is capable of quickly building an initial model, faster than the myopic gradient algorithm in all domains except Cheese Maze, and faster than suffix-history in several cases, although the accuracy of the suffix-history results is in question. In the long run, though, the performance of the constrained gradient

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

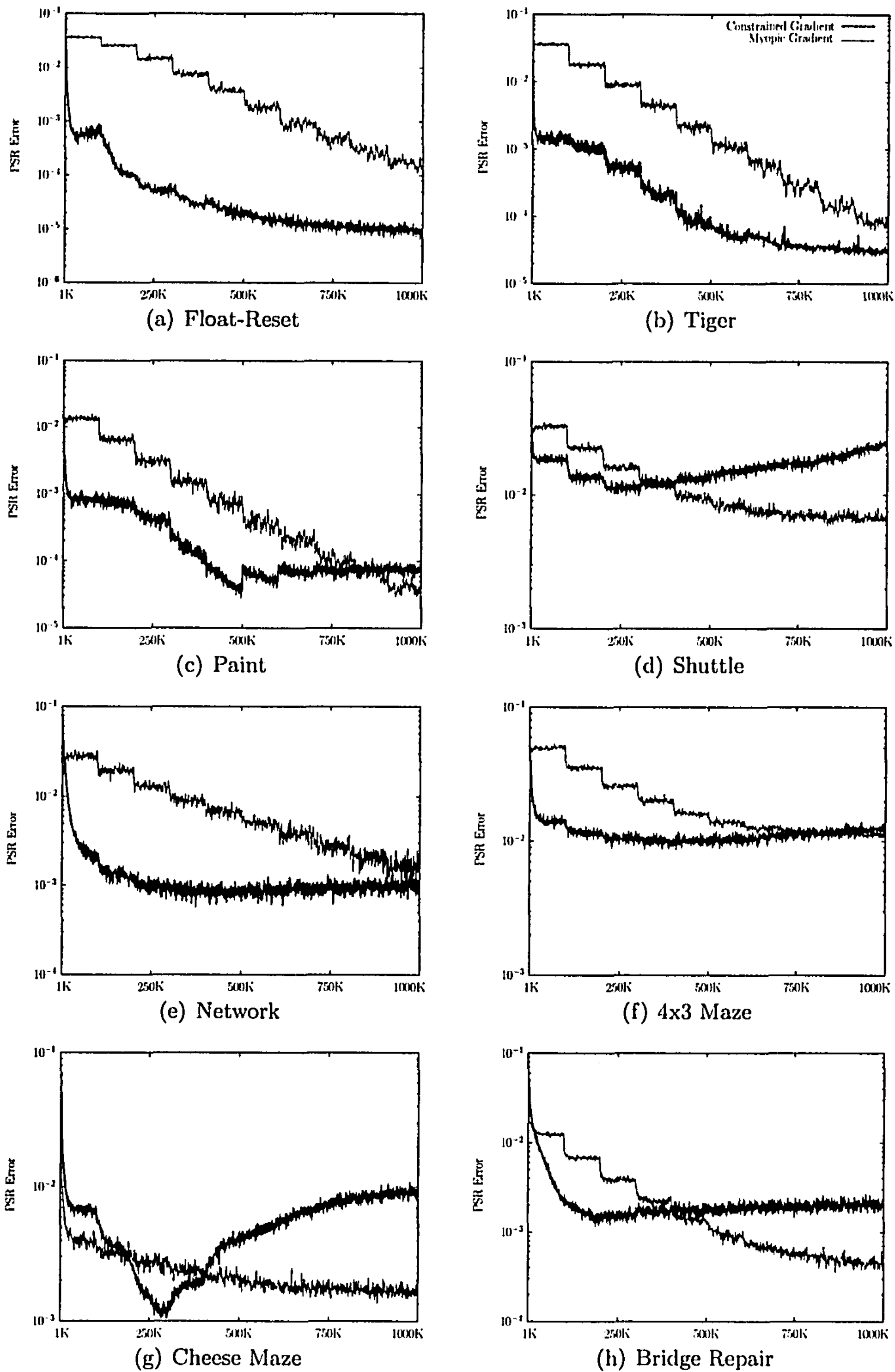


Figure 4.4: PSR error in online tests. The horizontal axis shows the number of data points used, and the vertical axis is the PSR error.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

algorithm tends to plateau, indicating that it may be prone to local minima in the search space.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 5

Discovery of Core Tests

In the previous chapter, we described the learning portion of the constrained gradient algorithm, and assumed that the set of core tests Q was known. In general, though, Q is not known prior to learning. In this chapter, we describe how the constrained gradient algorithm selects core tests. In Section 5.1, we describe how core tests can be identified in the constrained gradient algorithm and we discuss some issues related to the discovery problem. In Section 5.2, we show experimental results for this discovery algorithm on our test domains.

5.1 Core Test Discovery

Our discussion of core test discovery is divided into two sections. In Section 5.1.1, we discuss in general how core tests can be selected from the matrix $\widehat{y}(T|H)$. In Section 5.1.2, we explain in more detail how core test discovery is used in the constrained gradient algorithm.

5.1.1 Selecting Core Tests

In this section we explain how to choose a set of core tests from the set of tests T. Because this selected set is an approximate set of core tests, we use the notation $\widehat{Q}$ instead of Q, which denotes a true set of core tests. The constrained gradient algorithm uses a simple parameterized threshold algorithm to find core tests. This approach is more rudimentary than other PSR discovery and learning algorithms, namely, the rank-estimation method used by the reset-based and suffix-history methods. However, as will be seen in Section 5.2, the constrained gradient algorithm's discovery method is frequently capable of selecting a correct set of core tests with very little data.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The data structure available to the constrained gradient algorithm is $\widehat{y}(T|H)$, the current approximated submatrix of the system dynamics matrix. To begin the discussion, let us make three major assumptions:

1. $\widehat{y}(T|H) = y(T|H)$; the values in $\widehat{y}(T|H)$ are perfectly correct.
2. $Q \subset T$; T contains a complete set of core tests, but the identity of these core tests is unknown.
3. H is sufficient to represent the linear independence of each test; if the columns $y(t_1|H)$ and $y(t_2|H)$ are linearly dependent, no other set H will show them to be linearly independent.

Under these ideal conditions, discovering core tests is simple. Exactly $|Q|$ columns of $\widehat{y}(T|H)$ are linearly independent, and selecting Q is a matter of selecting linearly independent columns until all remaining columns are linearly dependent on the selected set. However, conditions are rarely this ideal. In the next three sections, we will discuss how each of the above conditions affects discovery when using an approximated $\widehat{y}(T|H)$.

Estimating Linear Relatedness

Since the values in $\widehat{y}(T|H)$ are estimated from data, they are only approximations of their true values. If we say that each value $\widehat{y}(t|h) = y(t|h) + n$, with some noise n, then the matrix $\widehat{y}(T|H) = y(T|H) + N$, with a noise matrix N. Assuming the noise is relatively unstructured, all of the columns of N will be linearly independent with very high likelihood, and therefore all of the columns of $\widehat{y}(T|H)$ will also be linearly independent.

The columns of $\widehat{y}(T|H)$ still exhibit different levels of linear relatedness, though, and this can be used as an indication of which tests are actually linearly independent, and therefore which tests should be chosen as core tests. Suppose we have a partial set of core tests in $\widehat{Q}$, and we have two potential new core tests, t_1 and t_2. If test t_1 appears more linearly unrelated to $\widehat{Q}$ than t_2, and there is no reason to believe that $\widehat{y}(t_1|H)$ has more noise than $\widehat{y}(t_2|H)$, then it is more likely that t_1 is a core test than t_2. More formally, given an incomplete set of core tests $\widehat{Q}$, we expect that the test that is most likely to be another core test is the test that is least linearly related to $\widehat{y}(\widehat{Q}|H)$:

$$\text{argmin}_{t \in T}\, \widehat{y}(\widehat{Q}|H) \oplus \widehat{y}(t|H)$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

where $\oplus$ is some measure of the linear relatedness of a matrix and a column. We use the condition number of a matrix to indicate relatedness, as was done in other discovery algorithms [James and Singh, 2004; Wolfe et al., 2005]. Other measures are also possible, such as the angle between the vector and the subspace formed by the matrix. The condition number $\mathrm{cond}(M)$ of a matrix M is the ratio of the largest singular value to the smallest singular value, where the singular values can be obtained using singular value decomposition [Khuri, 2003]. A matrix with a low condition number contains columns that are mostly linearly unrelated. If a matrix has a high condition number it has at least one column that is nearly linearly dependent on the rest of the matrix, and if the condition number is undefined (or infinite), the matrix contains at least one column that is completely linearly dependent on the other columns in the matrix. Using the condition number as a measure of linear relatedness, the above selection procedure becomes:

$$\mathrm{argmin}_{t \in T}\ \mathrm{cond}\Big(\widehat{y}(\widehat{Q} \cup \{t\}|H)\Big)$$

If $\widehat{Q}$ is initialized to $\{\varepsilon\}$, repeatedly applying the above procedure forms a greedy procedure for selecting tests that are likely to be core tests. This procedure is the basic mechanism for core test selection in all current discovery methods for PSRs. However, the procedure lacks a stopping condition, since the number of core tests is not known.

The stopping condition used by the constrained gradient algorithm is simple. A condition threshold, c, is supplied as a parameter to the algorithm. The discovery algorithm stops choosing core tests when it cannot add a core test to $\widehat{Q}$ without raising the condition of $\widehat{y}(\widehat{Q}|H)$ above c. Larger or smaller choices of c yield larger or smaller sets of tests for $\widehat{Q}$.

A threshold is a very direct approach to choosing a stopping condition for the constrained gradient algorithm, and other methods of deciding the size of $\widehat{Q}$ are possible. The reset-based and suffix-history algorithms use a more sophisticated approach to determine how many core tests to select. They use $\widehat{y}(T|H)$ to compute an approximation of the rank of $y(T|H)$ that takes into account the fact that the entries in the matrix are samples. They model the noise in each of the matrix probabilities, using the number of data samples that contributed to the probability. With a model of the noise in the matrix and a confidence parameter, they compute a singular value cutoff; the rank of the matrix is estimated to be the number of singular

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

values larger than the cutoff. This approach is more mathematically rigorous than a simple parameterization; however, this approach cannot be used easily in the constrained gradient algorithm since the number of data points used to create the prediction probabilities is not known.

Sufficiency of T

The above discussion explained how core tests can be selected from $\widehat{y}(T|H)$, despite it being only an approximation of the true matrix $y(T|H)$. The other two properties required to select core tests concern the sufficiency of T and H. We will now discuss how T can be constructed to contain a complete set of core tests. Littman et al. [2002] showed that if the one-step extensions of a proposed set of core tests $\widehat{Q}$ are all linearly dependent on $\widehat{Q}$, then all tests are linearly dependent on $\widehat{Q}$. However, Littman et al.'s proof depends on the existence of a POMDP representation of the system. Since a PSR is more general than a POMDP, this proof does not apply to PSRs in general. Below, we give a novel proof that is similar to Littman et al.'s proof, but does not require a POMDP representation.

Theorem 1. *If $\widehat{Q}$ contains ε, and all of the one-step extensions of $\widehat{Q}$ are linearly dependent on $\widehat{Q}$, then all possible tests are linearly dependent on $\widehat{Q}$.*

Proof. We use proof by induction. For the base case, ε is trivially linearly dependent on $\widehat{Q}$, since $\varepsilon \in \widehat{Q}$. For the inductive step, we show that if t is linearly dependent on $\widehat{Q}$, then aot is also linearly dependent on $\widehat{Q}$, for any action-observation pair ao.

$$\begin{aligned}
y(aot|H) &= \operatorname{diag}(y(ao|H))\, y(t|Hao) && (5.1)\\
&= \operatorname{diag}(y(ao|H))\, y(\widehat{Q}|Hao)m_t && (5.2)\\
&= \operatorname{diag}(y(ao|H))\operatorname{diag}(y(ao|H))^{-1}\, y(ao\widehat{Q}|H)m_t && (5.3)\\
&= \operatorname{diag}(y(ao|H))\operatorname{diag}(y(ao|H))^{-1}\, y(\widehat{Q}|H)M_{ao}m_t && (5.4)\\
&= y(\widehat{Q}|H)M_{ao}m_t && (5.5)\\
&= y(\widehat{Q}|H)m_{aot}, \quad \text{where } m_{aot} = M_{ao}m_t && (5.6)
\end{aligned}$$

In the above, $\operatorname{diag}(x)$ is the diagonal matrix that has vector x on the diagonal, $Hao = \{hao \,|\, \forall h \in H, ao \in \mathcal{A}\times\mathcal{O}\}$, and $ao\widehat{Q} = \{aoq \,|\, \forall ao \in \mathcal{A}\times\mathcal{O}, q \in \widehat{Q}\}$. Steps 5.1 and 5.3 use the conditional probability rule. Steps 5.2 and 5.4, respectively, use the fact that t and $ao\widehat{Q}$ are linearly dependent on $\widehat{Q}$. The remaining steps are algebraic reductions. The above proof assumes that $y(ao|h) > 0$, for all $h \in H$, because

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

otherwise diag($y(ao|H)$) is not invertible. However, in cases where $y(ao|h) = 0$, and therefore $y(aot|h) = 0$, the derived linear dependence still holds.

$$\begin{aligned} y(aot|h) &= y(\widehat{Q}|h)m_{aot} && (5.7) \\ &= y(\widehat{Q}|h)M_{ao}m_t && (5.8) \\ &= 0_{(1\times|\widehat{Q}|)}m_t && (5.9) \\ &= 0 && (5.10) \end{aligned}$$

where $0_{(1\times|\widehat{Q}|)}$ is the zero vector of size $|\widehat{Q}|$. Step 5.9 uses the fact that the probabilities in $y(ao\widehat{Q}|h)$ must all be zero when $y(ao|h)$ is zero, and that the weight matrix M_{ao} must linearly produce those zeros.

Thus, we see that if t is linearly dependent on $\widehat{Q}$, all one-step extensions of t are also dependent on $\widehat{Q}$, and by induction all tests are dependent on $\widehat{Q}$. □

Note that $m_t = m_{a_1o_1\cdots a_{n-1}o_{n-1}a_no_n} = M_{a_1o_1}\cdots M_{a_{n-1}o_{n-1}}m_{a_no_n}$ is the same result that was found by Littman et al. [2002] using POMDPs as the basis of the calculation.

This above theorem is useful, because it shows that it is sufficient to search for additional core tests among the one-step extensions of $\widehat{Q}$. If all one-step extensions of $\widehat{Q}$ are linearly dependent on $\widehat{Q}$, then a complete set of core tests has been found. Otherwise, one of the extensions must be linearly independent, and therefore another core test has been found. This addresses the above requirement that $Q \subset T$; in practise, as long as T contains the one-step extensions of $\widehat{Q}$, T is sufficient to find at least one more core test.

Sufficiency of H

The remaining requirement for finding core tests is that H is sufficient. This means that H contains enough histories to reveal the linear independence between any two tests which might be linearly independent. This assumption is more difficult to address than the previous two assumptions. The issues related to H and discovery are the same as the issues related to H and learning, which were addressed in Section 4.1.2. In general, if H is not sufficient, then some tests will be considered linearly dependent on $\widehat{Q}$, even if they are actually linearly independent. Because there are often multiple possible choices for core tests, mislabelling some tests as linearly dependent is not a problem. However, if H is insufficient enough such that

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

all potential core tests are mislabelled as linearly dependent, then a complete set of core tests cannot be discovered.

5.1.2 Discovery in the Constrained Gradient Algorithm

In the previous section, we described how core tests can be identified in the set T. In this section, we describe more specifically the procedure used by the constrained gradient algorithm to discover core tests. Algorithm 2 shows this procedure. The algorithm shows two variations on the discovery procedure: a cumulative version that adds new tests to the existing set $\widehat{Q}$, and a non-cumulative version that builds $\widehat{Q}$ from scratch each time. The former is called the cumulative version because changes made to the set $\widehat{Q}$ are cumulative; once a test is added to $\widehat{Q}$, it is never removed. The non-cumulative version differs in line 2, in which $\widehat{Q}$ is reset to $\{\varepsilon\}$ before any core tests are found.

In line 5, the set of potential core tests S is initialized to all of the one-step extensions of $\widehat{Q}$ that are also present in T. Because T is updated *after* all of the core tests have been selected, initializing S in this way prevents any tests from being chosen as core tests that have not been present in T since at least the previous run of the core test selection procedure. More simply, if $t \notin T$ at the beginning of the selection procedure, then t cannot be selected as a core test. These tests are excluded because $\widehat{y}(t|H)$ is not available for these tests.

Algorithm 2 The discovery procedure in the constrained gradient algorithm.

Require: $y(T|H)$
Require: c // The condition threshold.
1: **if** not using cumulative discovery **then**
2: $\quad \widehat{Q} \leftarrow \{\varepsilon\}$
3: **end if**
4: **loop**
5: $\quad S \leftarrow \{aoq | a \in \mathcal{A}, o \in \mathcal{O}, q \in \widehat{Q}\} \cap T$
6: $\quad t \leftarrow \text{argmin}_{t \in S}\, cond(y(\widehat{Q} \cup \{t\}|H))$
7: $\quad$ **if** $cond(y(\widehat{Q} \cup \{t\}|H)) \leq c$ **then**
8: $\quad\quad \widehat{Q} \leftarrow \widehat{Q} \cup \{t\}$
9: $\quad$ **else**
10: $\quad\quad$ break from loop
11: $\quad$ **end if**
12: **end loop**
13: $T' \leftarrow$ normalizable set containing $\widehat{Q}$
14: $T_{new} \leftarrow T' - T$
15: initialize $y(T_{new}|H)$
16: $T \leftarrow T'$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The most computationally expensive part of Algorithm 2 is finding the test that is least linearly related to the current set of core tests. For each potential test, finding the condition number of $\hat{y}(\hat{Q} \cup \{t\}|H)$ requires computing the singular value decomposition of the matrix, which takes $O(|\hat{Q}|^2|H|)$ time. Since there are about $|\mathcal{A}||\mathcal{O}||\hat{Q}|$ potential tests, overall the time complexity of running the core test discovery procedure is about $O(|\mathcal{A}||\mathcal{O}||\hat{Q}|^3|H|)$, or about $O(|\hat{Q}|^3|H|)$, if the number of actions and observations is relatively small. While this is expensive, we generally assume that $|\hat{Q}|$ is small. Also, the core test selection procedure is not run at every time step, and thus the cost can be amortized over the number of time steps between successive runs of the procedure. A clever implementation of singular value decomposition could also reduce the time required to run the procedure. Note that each matrix for which the SVD is computed varies by only a single column. If an incremental SVD algorithm is used, the time complexity to compute each decomposition would be $O(|\hat{Q}|^3 + |\hat{Q}||H|)$ [Brand, 2003]. Since we expect $|H| \gg |\hat{Q}|$, this optimization would speed the algorithm by a factor of $|\hat{Q}|$.

As previously mentioned, the discovery procedure is not run at every time step. In batch learning methods, the natural break point to run a core test detection procedure is after each complete pass over the data. This is what is done in the reset-based and suffix-history algorithms. However, in online learning methods there is no such natural break point, since only one pass over the data is made. We have chosen to simply run the core test detection algorithm after every n data points, where n can be configured. A reasonable choice for n is $|H|$, because this allows H to be completely refreshed and a whole new matrix $\hat{y}(T|H)$ to be generated between each time the core test discovery algorithm is run.

After the new set of core tests is selected, a set of new tests is added to T. This set includes all of the one-step extensions of the new core tests, as well as any parent and sibling tests that must be in T to perform the normalization step. When these new tests are added, their columns in the matrix $\hat{y}(T|H)$ are initialized, as in line 15. It is possible to compute reasonable estimates of $\hat{y}(T_{new}|H)$, because many of those tests will be of the form aot, where columns for ao and t are already in the matrix. Thus a prediction of $\hat{y}(aot|h)$ can be computed from $\hat{y}(t|hao)\hat{y}(ao|h)$ in cases where the h is followed by ao. This gets complicated, however, and in practise simply initializing the columns to zeros tends to work well.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.2 Experimental Results

In this section, we show empirical results from running the constrained gradient algorithm's core test selection procedure. We investigate the choice of the condition threshold, and the differences between cumulative and non-cumulative discovery.

In the experiments shown in this section, a history window of size 1,000 was used for H. The learning rate, α, was initialized to 1.0 and halved every 100,000 data points. All values reported are the mean of 10 trials, using the same data sets of length 1,000,000 that were used in the previous chapter. Core test detection was performed every 1,000 data points.

When reporting on the results of discovery experiments, two values are relevant. The first value is $|\widehat{Q}|$, the size of the set of core tests that was selected by the algorithm. The second value is $|\widehat{Q}_{true}|$, which we define to be the number of tests in $\widehat{Q}$ that are linearly independent in the true system. In all cases, $1 \leq |\widehat{Q}_{true}| \leq |\widehat{Q}|$, because $\{\varepsilon\} \subseteq \widehat{Q}_{true} \subseteq \widehat{Q}$ and ε is a core test in all systems. The size of $\widehat{Q}_{true}$ is calculated by computing the number of linearly independent vectors in $y(\widehat{Q}|H^*)$, where H^* is approximated by a very large set of histories.

5.2.1 Condition Threshold Tests

The first experiments we describe were designed to investigate how changing the condition threshold parameter affects the discovery performance of the constrained gradient algorithm. In these trials, the condition threshold was set to values between 1 and 20. Cumulative discovery was used. The sizes of $\widehat{Q}$ and $\widehat{Q}_{true}$ were recorded after 10,000 data points, which in all cases was sufficient for discovery to choose a set $\widehat{Q}$. The results are shown in Figure 5.1.

The results from the condition threshold tests are not surprising. As the condition threshold is increased, the size of $\widehat{Q}$ increases. The size of $\widehat{Q}_{true}$ also increases, until it reaches the true number of core tests for the system, as indicated by the horizontal lines in the plots. In five of the domains (Tiger, Paint, Shuttle, Network and Bridge Repair) the discovery algorithm successfully chooses a complete set of core tests for the system, and in the remaining domains the discovery algorithm very nearly chooses a complete set. In all domains, these set of core tests are chosen after very little data is observed.

In the Float-Reset domain, a seemingly anomalous result occurred. The number

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

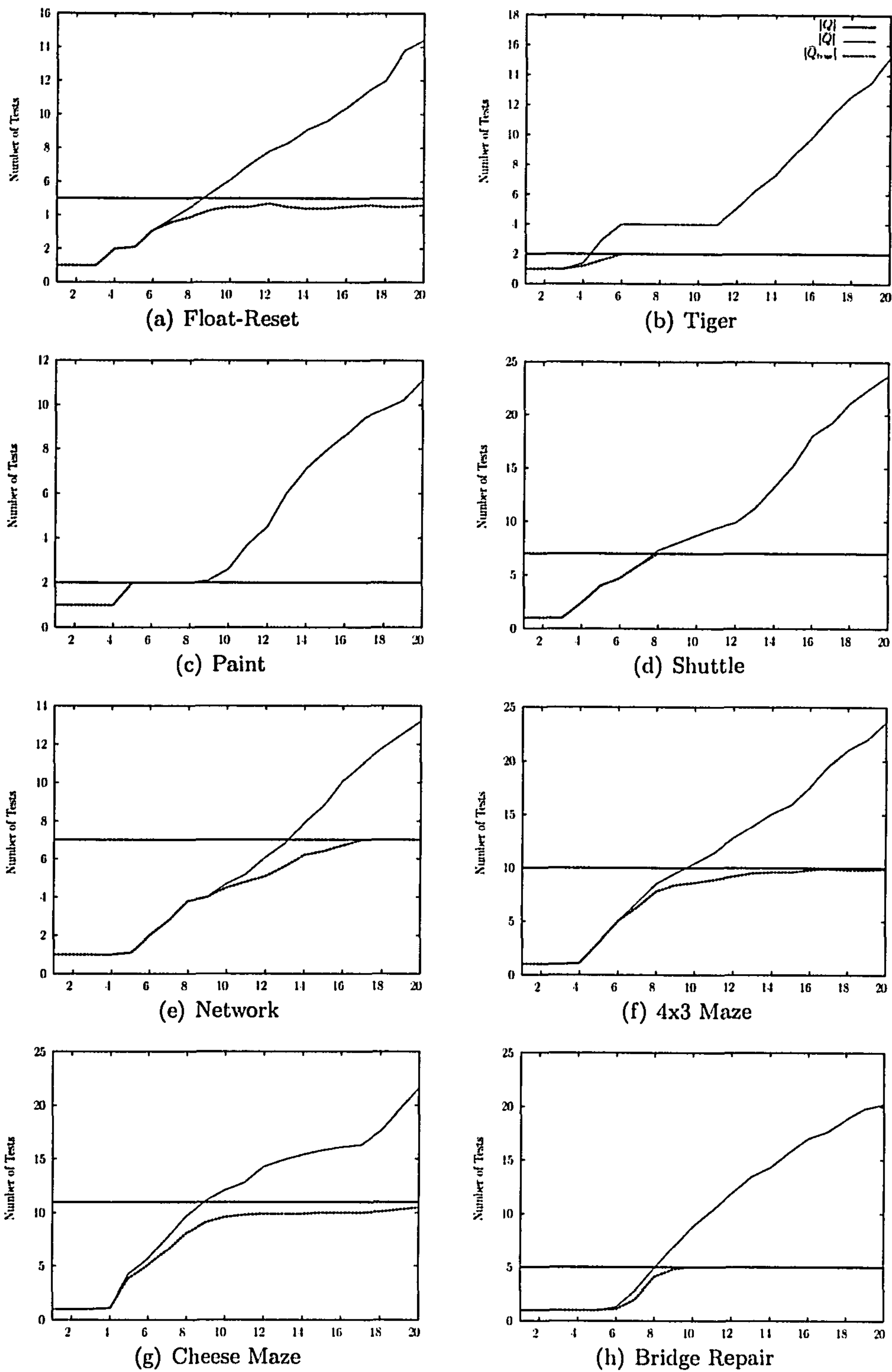


Figure 5.1: The number of core tests selected by the discovery algorithm, for different values of the condition threshold, c.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Domain		Constrained Gradient			Suffix-History	
Name	$\lvert Q\rvert$	$\lvert\hat{Q}\rvert$	$\lvert\hat{Q}_{true}\rvert$	# Data	$\lvert\hat{Q}\rvert = \lvert\hat{Q}_{true}\rvert$	# Data
Float-Reset	5	6.1	4.5	4000	-	-
Tiger	2	4.0	2.0	1000	2	4000
Paint	2	2.6	2.0	4000	2	4000
Shuttle	7	8.7	7.0	2000	7	1024000
Network	7	4.7	4.5	2000	3	2048000
4x3 Maze	10	10.4	8.6	2000	9	1024000
Cheese Maze	11	12.1	9.6	1000	9	32000
Bridge Repair	5	8.8	5.0	2000	5	1024000

Table 5.1: The average number of core tests found by the constrained gradient algorithm, when the condition threshold was 10. The data for the suffix-history algorithm is shown here for comparison.

of correct core tests chosen by the algorithm actually decreases slightly with higher condition thresholds. This can occur mainly in domains that have core tests of length two or more; in these domains, multiple iterations of the discovery procedure are required to select all of the core tests. With a large condition threshold, earlier iterations of the discovery procedure can select incorrect core tests that raise the condition number of $\hat{y}(\hat{Q}|H)$. This essentially blocks tests from being selecting in subsequent iterations of the discovery procedure.

Overall, the discovery procedure used by the constrained gradient algorithm is quite successful at choosing core tests with very little data. However, there is no one condition threshold that works best across all domains. For domains with a small number of core tests, a smaller condition threshold works well; larger thresholds are needed for domains with many core tests. For the domains shown, a threshold of 10 works reasonably well across all domains; this is the threshold used in the remaining experiments with discovery.

Table 5.1 shows the effectiveness of the constrained gradient discovery algorithm with a condition threshold of 10, as compared to the discovery results for the suffix-history algorithm reported in the literature [Wolfe et al., 2005]. The suffix-history algorithm is currently the only other algorithm capable of discovering core tests in systems without a reset action. For the constrained gradient algorithm, the number of data points listed is the maximum number of data points after which no further changes to $\hat{Q}$ occurred. For the suffix-history algorithm, the number of data points shown is the minimum number of data points required to find that number of core tests. Also, the suffix-history algorithm was able to make multiple passes over its

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

data, while the constrained gradient algorithm uses a single pass. Overall, Table 5.1 shows that, in general, the constrained gradient algorithm is able to find as many core tests as the suffix-history algorithm, but with much less data.

Comparing the discovery results for the constrained gradient algorithm and the suffix-history algorithm is somewhat unfair, because the two algorithms take different approaches to the discovery problem. The suffix-history algorithm uses a very conservative approach to discovery; it does not add a test to $\widehat{Q}$ until it is quite certain that the test is really linearly independent to the current set $\widehat{Q}$. For this reason, when using the suffix-history algorithm $\widehat{Q} = \widehat{Q}_{true}$ for all of the above domains, although this is not necessarily true in general. It also means that the suffix-history algorithm requires a lot of data before it adds a core test to $\widehat{Q}$, which explains why the numbers of data points for the suffix-history algorithm are so high. The constrained gradient algorithm, on the other hand, takes a more liberal approach. It adds tests that look likely to be core tests, but with much less data. As a result, $\widehat{Q}$ usually contains tests that are not truly linearly independent.

In theory, including extra tests in $\widehat{Q}$ is not particularly harmful. As long as $\widehat{Q}$ contains a full set of core tests, it has sufficient data to represent the system, and extra tests are simply redundant data. However, including extra tests can have three negative side effects. The first is that they require extra storage and computation, since parts of the learning algorithm are linear, quadratic and even cubic in $|\widehat{Q}|$. The second negative effect is that it could allow overfitting in the regression step by having too many inputs. Even though the extra inputs should theoretically be linearly dependent on the true core tests, small errors in the inputs can be used to compute weights that overfit the data. Finally, including extra tests can block true core tests from being chosen in the discovery step. We have already seen this happen in the Float-Reset results, above. This is potentially the most disastrous effect of extra core tests, since if the algorithm cannot find a complete set of core tests, it will never be able to fully represent the system. In general, it is desirable to avoid selecting incorrect core tests.

5.2.2 Non-Cumulative Selection of Core Tests

The previous section showed results for cumulative selection of core tests. It also gave evidence of a dilemma that cumulative discovery causes: a condition threshold must be high enough to allow selection of all of the true core tests, but not so high

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

that incorrect core tests are chosen early and block true core tests from being chosen. This difficulty is mainly caused by the cumulative nature of the discovery; once a test has been chosen as a core test by the algorithm, there is no way to remove it. Furthermore, discovery happens very early; most of $\widehat{Q}$ is chosen during the first iteration of the selection procedure, which means that discovery does not benefit from any of the data points that occur after the first 1,000 data points.

In this section, we propose a solution to this discovery dilemma. Instead of adding tests to $\widehat{Q}$ each time the discovery procedure is invoked, the algorithm re-selects $\widehat{Q}$ from scratch. This allows incorrectly selected tests to be discarded, even if they appeared linearly independent early during learning.

Figure 5.2 shows the discovery results when a condition threshold of 10 is used, and core tests are selected from scratch every 1,000 data points. The plots show the sizes of $\widehat{Q}$ and $\widehat{Q}_{true}$ as the number of data points increases. Several facts are apparent from viewing these results. One is that, after sufficient data is seen, in all cases the discovery algorithm stops selecting incorrect core tests, and $\widehat{Q} = \widehat{Q}_{true}$. This is an expected result, because after more data is seen by the algorithm, its estimates of the prediction probabilities become more accurate, and the discovery procedure is better able to estimate if two tests are linearly independent.

The other clear result is that the size of $\widehat{Q}$ decreases as more data is seen, even after the algorithm stops selecting incorrect core tests. Eventually, the size of $\widehat{Q}$ settles on a value and remains relatively constant. This is because a constant condition threshold was used, and except for the most simple domains, this threshold simply was not large enough to allow all of the tests in Q to be chosen. Furthermore, the largest jumps in the size of $\widehat{Q}$ correspond to the time steps when the learning rate, α, is reduced. The explanation for this is that, as the estimates in $\widehat{y}(\widehat{Q}|H)$ become more accurate, the condition number of $\widehat{y}(\widehat{Q}|H)$ increases. When the condition number becomes higher than the condition threshold, the algorithm is no longer able to select all of the tests in $\widehat{Q}$, and the size of $\widehat{Q}$ is reduced in subsequent runs of the discovery procedure.

To further investigate this phenomenon, we recorded the condition number of $\widehat{y}(Q|H)$ during learning. We also computed the minimum condition of $\widehat{y}(Q \cup \{t\}|H)$ out of all tests $t \in T$. The purpose of this was to give an indication of how large the condition threshold would have to be to select all of the true core tests, without selecting any incorrect core tests. Figure 5.3 shows the results from these experi-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

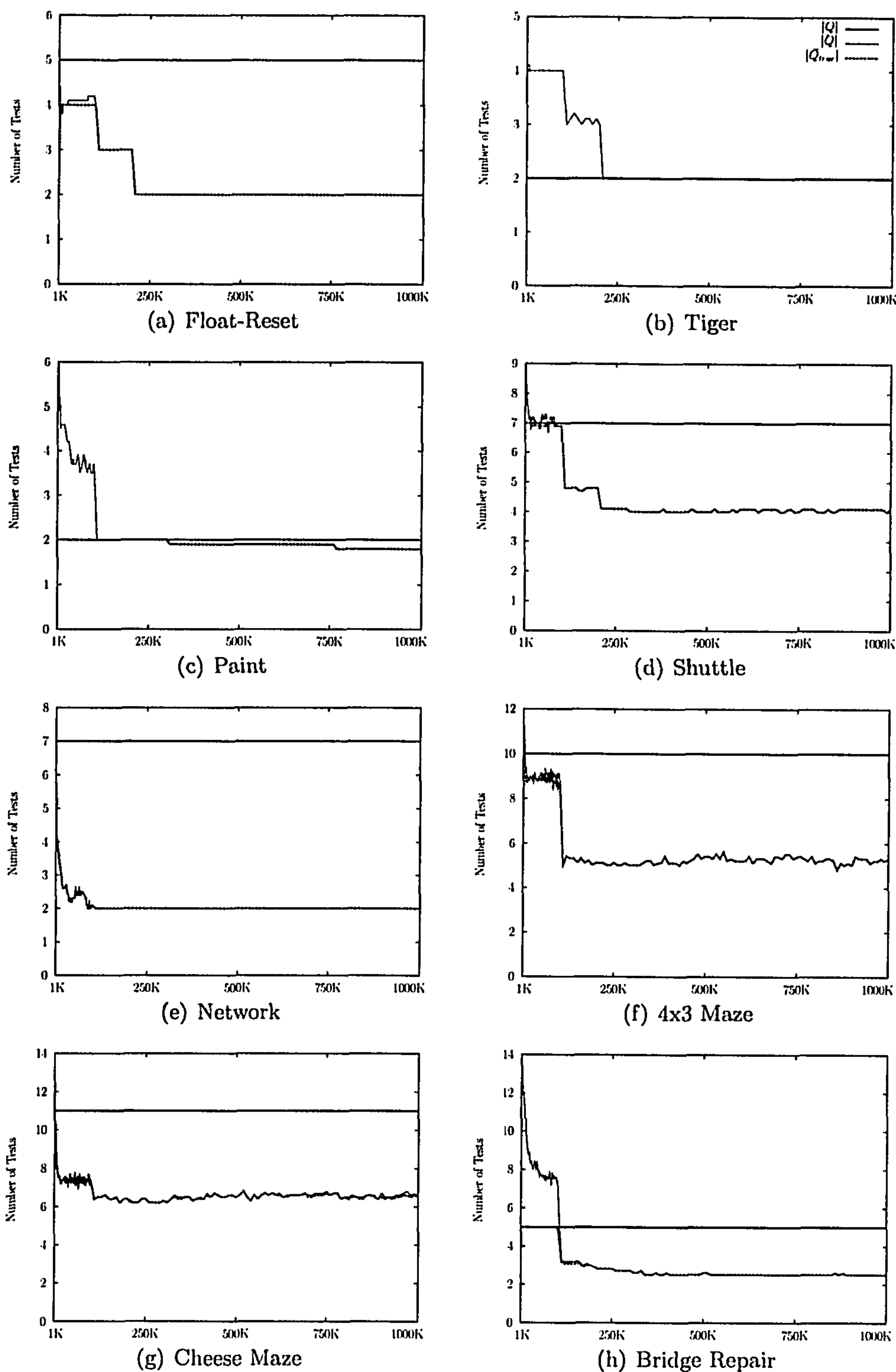


Figure 5.2: Number of core tests selected using non-cumulative discovery. The horizontal axis is the number of data points. Core tests were reselected every 1,000 data points.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ments. These results confirm the behaviour experienced in the discovery results; the condition number of $\widehat{y}(Q|H)$ does increase as more data is seen and the prediction probabilities become more accurate. The condition numbers tend to jump at transition points in α, which correspond to the decreases in core tests selected as seen in Figure 5.2.

The gap between the condition of $\widehat{y}(Q|H)$ and the condition of $\widehat{y}(Q \cup \{t\}|H)$ increases as the columns become more accurate, although the difference is more pronounced in some cases than in others. This is expected; theoretically, the condition of $y(Q \cup \{t\}|H)$ is infinite, because t should be linearly dependent on Q. So we expect that as $\widehat{y}(T|H)$ becomes more accurate overall, the condition of $\widehat{y}(Q \cup \{t\}|H)$ should increase quite quickly. This 'widening of the gap' would be a good thing to take advantage of in future refinements to the constrained gradient discovery algorithm. It indicates that a condition threshold can be chosen that falls in between these two crucial condition numbers, and that choosing such a threshold becomes easier as learning continues.

5.2.3 Summary of Discovery Results

Overall, the discovery results for the constrained gradient algorithm show a definite ability to quickly select sets of tests that contain complete sets of core tests. Its ability to do this is heavily dependent on the choice of the condition threshold parameter; if the parameter is too large, too many non-core tests will be selected, but if the parameter is too small, insufficient true core tests will be selected. Also, the non-cumulative method of discovery appears to be better at selecting true core tests without selecting additional tests. However, the non-cumulative method requires an increasing threshold, to account for the increase in the condition of $\widehat{y}(\widehat{Q}|H)$ over time.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

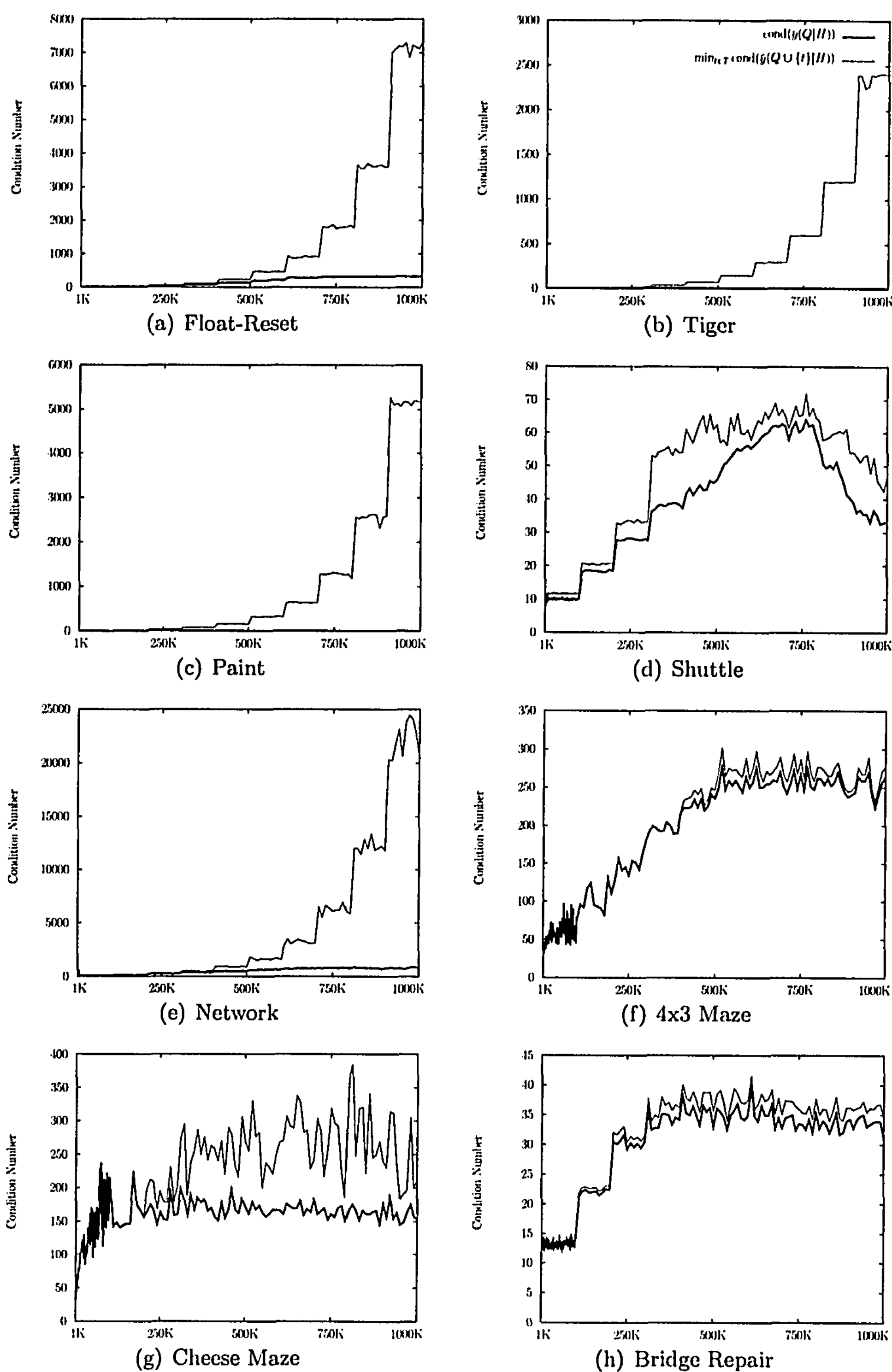


Figure 5.3: The condition of the matrix $\widehat{y}(Q|H)$ as the number of data points increases. Also shown is the condition of $\min_{t \in T} cond(\widehat{y}(Q \cup \{t\}|H))$.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 6

Additional Investigation

This chapter contains additional experiments that further explore the constrained gradient algorithm. In Section 6.1, we see how the constrained gradient algorithm might be expected to perform in practical settings. In Section 6.2, we perform experiments inspired by some of the interesting results from previous sections.

6.1 Discovery and Learning

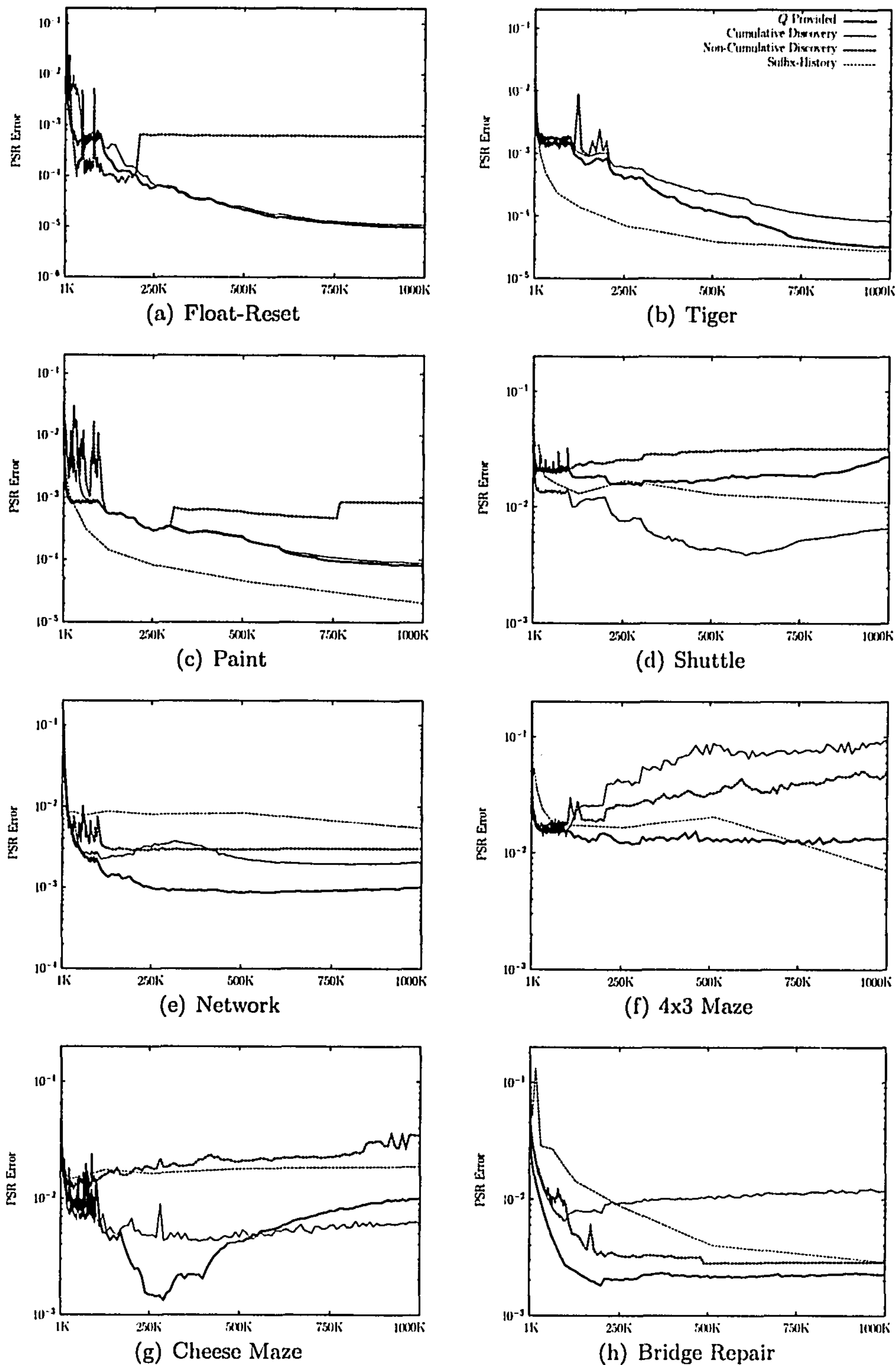
Chapter 4 presented learning results for the constrained gradient algorithm when it was supplied with a correct set of core tests. In this section, we show the performance results for the constrained gradient algorithm using discovered sets of core tests. These experiments are representative of how the constrained gradient algorithm would perform in a practical setting, when a true set of core tests is unknown.

The sets $\widehat{Q}$ discovered in these experiments are sometimes incomplete, and often contain extra tests. A condition threshold of 10 was used for the discovery procedure, and both cumulative and non-cumulative discovery were tested. See the results from Section 5.2.1 and Section 5.2.2 for more information on the sets of tests discovered.

The results for the constrained gradient algorithm using discovery and learning are in Figure 6.1. The plots show average offline prediction error for cumulative discovery, non-cumulative discovery, and suffix-history using discovery. For comparison, the constrained gradient results when provided with Q are also given (repeated from Figure 4.3). Overall, the results in Figure 6.1 are mixed, but are generally explainable when considering the underlying sets of core tests that were found by the algorithm.

As expected, the constrained gradient algorithm did not perform as well when using a discovered $\widehat{Q}$ as when it is provided with Q. Three exceptions are the simpler

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) Float-Reset

(b) Tiger

(c) Paint

(d) Shuttle

(e) Network

(f) 4x3 Maze

(g) Cheese Maze

(h) Bridge Repair

Figure 6.1: The PSR error of the constrained gradient algorithm when learning using discovered sets of core tests.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

domains Tiger, Paint and Float-Reset, in which a discovered set performed as well as, but not better than, the given set. The major exception is the Shuttle domain, in which the performance was much better using $\widehat{Q}$ found with cumulative discovery than when given Q. In all cases, $\widehat{Q}$ contains a complete set of core tests in all trials. This performance difference indicates that some sets of core tests can be better than others; further experiments with different sets of core tests are done in Section 6.2.3.

In most cases, constrained gradient performed better using cumulative discovery than when using non-cumulative discovery. This is because non-cumulative discovery, with a threshold of 10, tends to settle on sets $\widehat{Q}$ that are much smaller than the true set Q, and therefore unable to create a good representation of the system. The exceptions to this, as seen in Figure 6.1, are the Tiger, 4x3 Maze, and Bridge Repair domains. In Tiger this is because non-cumulative discovery settled on an exactly correct set of core tests, while cumulative discovery used extra tests that interfered with learning. In 4x3 Maze and Bridge Repair, we can surmise only that the smaller set of core tests discovered using non-cumulative discovery provided a more stable set with which to create an approximate model than the large set found by cumulative discovery. This result is especially interesting in the Bridge Repair domain, because non-cumulative discovery finds about two core tests on average, while cumulative discovery finds a complete set of core tests, but also includes some extra tests. This indicates that finding extra core tests can actually have a large negative effect on performance with the constrained gradient algorithm; although extra core tests were generally not found to be a problem for the myopic gradient algorithm [Singh et al., 2003].

The results for suffix-history shown in Figure 6.1 are obtained directly from the published results for the algorithm [Wolfe et al., 2005]. Because of this, the suffix-history results were not trained or tested on any of the same data sets, and direct comparisons are therefore somewhat inappropriate. Also, because the suffix-history algorithm was tested as a batch algorithm, it made multiple passes over the data to build its model. Thus, the suffix-history algorithm was able to perform its discovery procedure after seeing *all* of the data, and then make another pass over the data to learn a model. However, because suffix-history is the only other algorithm capable of both discovery and learning of PSRs without a reset, we show the results for comparison. In Figure 6.1 we see that suffix-history outperforms the constrained gradient algorithm in the easier domains, Tiger and Paint, and also in the 4x3 Maze domain

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

because both models discovered with the constrained gradient algorithm performed poorly. In the Network domain, both constrained gradient models outperformed suffix-history, even though the non-cumulative discovery model used only two core tests. In the Shuttle, Cheese and Bridge Repair domains, at least one of the models discovered using the constrained gradient algorithm outperforms suffix-history.

Overall, the constrained gradient algorithm manages competitive performance in most of the domains. With improvements to the condition threshold used by the non-cumulative discovery method, we expect that the constrained gradient algorithm's performance will approach its performance when Q is given.

6.2 Investigative Experiments

The results presented in previous chapters focused on the performance of the constrained gradient algorithm, in both discovery and learning. In this section, we describe the results of experiments designed investigate the inner workings of the constrained gradient algorithm.

6.2.1 Examining Sources of Error

For any gradient descent algorithm, there are at least two sources of error when trying to find the global minimum. One source is approximations made in computing the gradient, and another is the initialization of the parameters of the algorithm. The constrained gradient algorithm experiences error from both of these sources. At each data point, the unknown true gradient is replaced by the myopic gradient of that data point, with the assumption that over many data points, this will approximate the true gradient. Also, the initial parameterization of the algorithm uses uniform probabilities, which may not necessarily be a good place in the search space to start. In this section, we will investigate both of these sources of error for the constrained gradient algorithm, by performing experiments that eliminate these errors. These experiments were inspired by similar experiments performed with the myopic gradient algorithm [Singh et al., 2003].

$$\widehat{y}(t|h) \leftarrow (1-\alpha)\widehat{y}(t|h) + \alpha\widehat{y}(\pi(t)|h) \tag{6.1}$$

$$\widehat{y}(t|h) \leftarrow (1-\alpha)\widehat{y}(t|h) + \alpha y(t|h) \tag{6.2}$$

$$\widehat{y}(t|h) \leftarrow (1-\alpha)\widehat{y}(t|h) + \alpha\widehat{y}(\pi(t)|h)y(t|h,\pi(t)) \tag{6.3}$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Rules 6.1, 6.2 and 6.3 are three different learning rules for the constrained gradient algorithm. Rule 6.1 shows the learning rule of the constrained gradient algorithm; it is the same as learning rule 6.3, with the implicit assumption that $y(t|h, \pi(t)) = 1$ (*i.e.*, the myopic gradient). The theoretically correct learning rule is given in rule 6.2; it is the same as rule 6.3 if $\widehat{y}(\pi(t)|h) = y(\pi(t)|h)$. If this equality is not true, however, the theoretically correct rule cannot be used, since the normalization procedure would modify the value of $\widehat{y}(t|h)$. In this section, we perform experiments using learning rule 6.3; we provide the probability $y(t|h, \pi(t))$ to the algorithm each time it uses the learning rule to modify test t. For these experiments, we used $\alpha \approx 1$, since the true gradient should be followed as much as possible.

Experiments were also performed to see how stable the correct solution is for each domain. To do this, the set H was initialized to 1000 reachable histories, which were sufficient to represent the dynamics of each system. The entries in matrix $\widehat{y}(T|H)$ were initialized to their true probabilities. For learning in this system, $\alpha = 0.001$ was used as a small learning rate, and learning proceeded as normal, using the myopic gradient. The purpose of this experiment is to see how far the learned model drifts from the correct initial model. We expect some drift, because learning with the myopic gradient and $\alpha > 0$ introduces some error into the system.

If both of the above modifications are made to the constrained gradient algorithm, then in all cases the algorithm begins with a perfect model and stays at a perfect model, excepting small errors caused by regularized regression and bounded probabilities. When used individually, the results are more interesting.

Figure 6.2 shows the results of the experiments described above, as well as the performance of regular learned models for comparison (results repeated from Figure 4.3). In the Float-Reset, Tiger and Paint domains, the results were exactly as expected. The models that were initialized to correct values slowly drifted to higher, but still very small, levels of error. The models that were given the true gradient decreased in error very quickly, and eventually plateaued at negligible error levels. Although the models learned using the unaugmented constrained gradient algorithm do not reach the same performance as the augmented models with the data given, we expect that with sufficient data and an appropriately decaying α, they will eventually converge to the same values.

Other domains had more surprising results. The Shuttle domain is an appropriate example. All three experiments converged to models with very similar, and

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

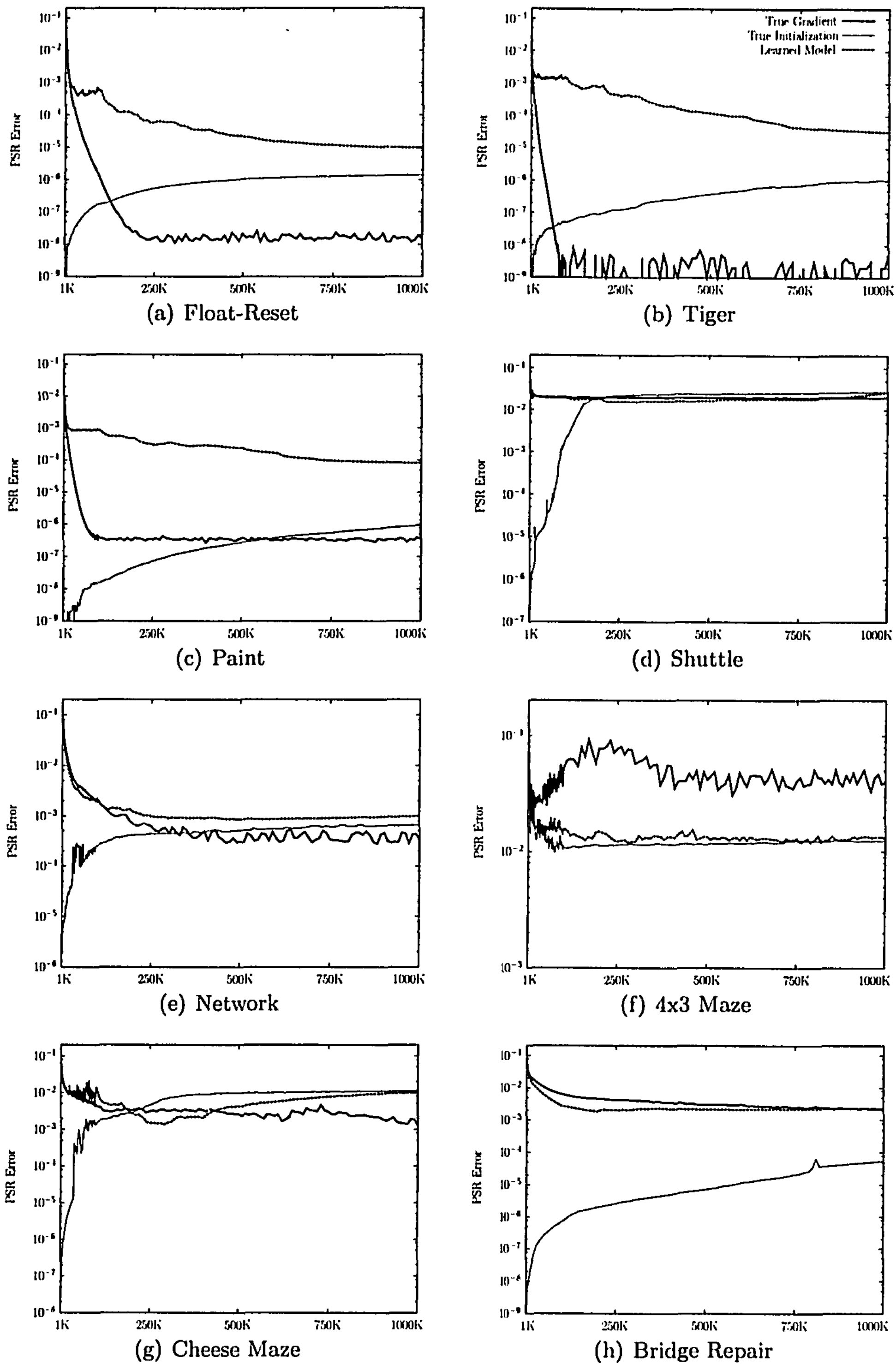


Figure 6.2: Performance of the constrained gradient algorithm when initialized when initialized with true probabilities and when using the true gradient.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

relatively high, levels of error. Using the true gradient does not decrease the error of the learned model much below that achieved when using the myopic gradient, and the properly initialized model drifted in error until it had the same performance as the other models. One possible explanation of this behaviour is that the global minimum might be very shallow, but there exists a hard-to-escape local minimum that traps the developing model.

Another surprising result is that, in the 4x3 Maze, the model using the true gradient learns a worse model than that using the myopic gradient. We do not have an explanation at this time why this would occur.

6.2.2 Momentum in Learning

When the learning results for the constrained gradient algorithm were first presented, in Figure 4.1, it was noted that in some of the domains performance degraded over time, despite continuing learning. This was most apparent in the Shuttle and Cheese Maze domains. In this section, we explore this result.

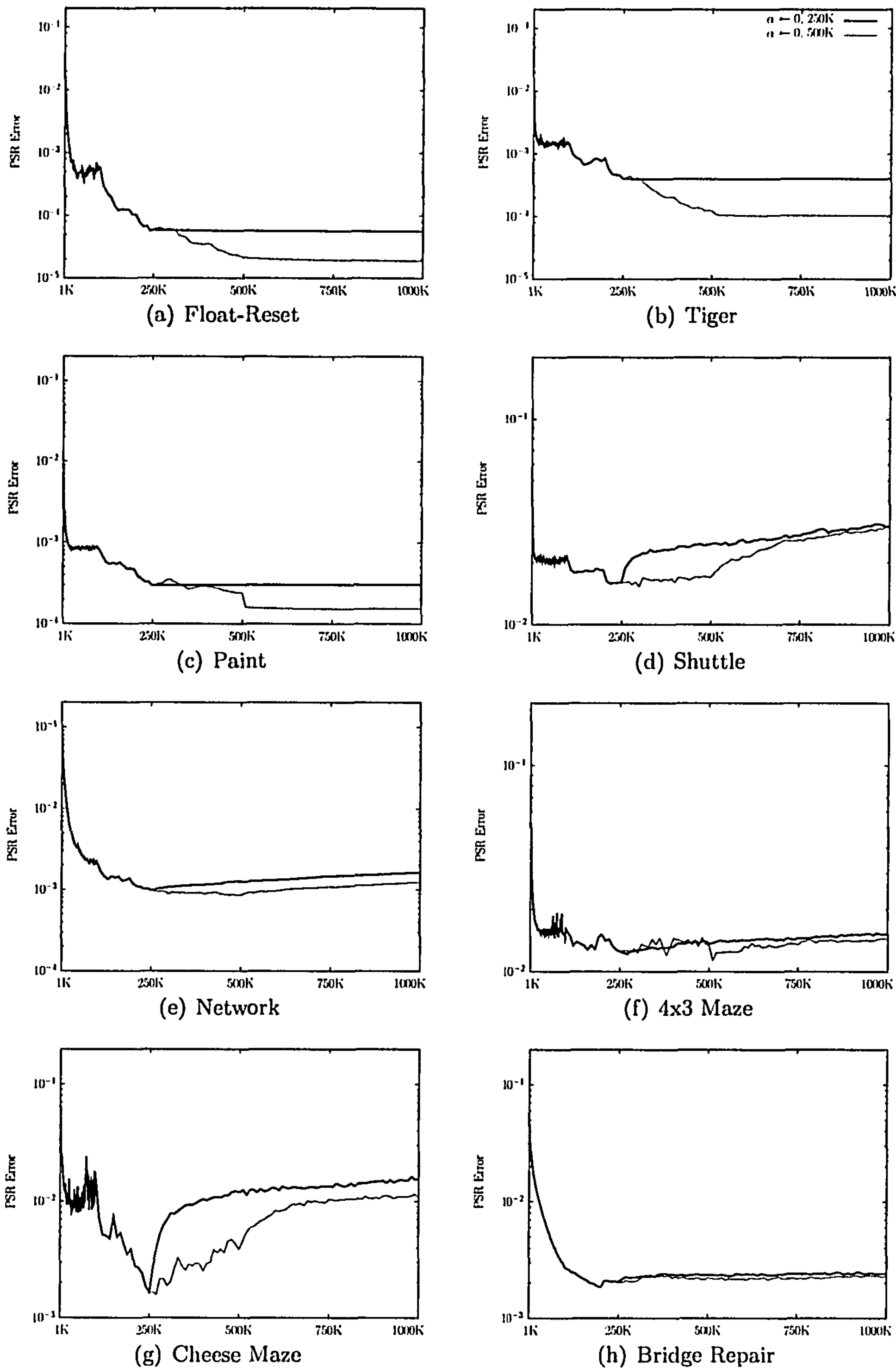
One fact that is apparent in Figure 4.1 is that this degrade in performance is delayed when α is decayed at a slower rate. This suggests that it is not the process of following the gradient that causes the increase in error. Recall that generating a new row involves several steps. At first, the row is estimated based on the existing matrix $\widehat{y}(T|H)$, and then it is adjusted based on the myopic gradient of the data. As α decreases, the effect of the second phase becomes minimal.

Our hypothesis is that the matrix $\widehat{y}(T|H)$ has a *momentum* towards stable points in learning space, in which the matrix satisfies the properties of a system dynamics matrix. This momentum is caused by enforcing the constraints on the system dynamics matrix, using regression and normalization. If the direction of the momentum of the matrix is not the same as the gradient of the data, then once the learning rate α becomes small enough, the momentum of the matrix becomes the primary force of change in the algorithm.

To explore our hypothesis, we designed experiments to see how the performance of the matrix changes when the learning rate is set to zero. Two experiments were run; in one, α was set to zero after 250,000 data points, and in the other, α was set to zero after 500,000 data points.

Figure 6.3 shows the results of the experiments into matrix momentum. Once again, Float-Reset, Tiger and Paint are excellent examples of desired results. In

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) Float-Reset (b) Tiger

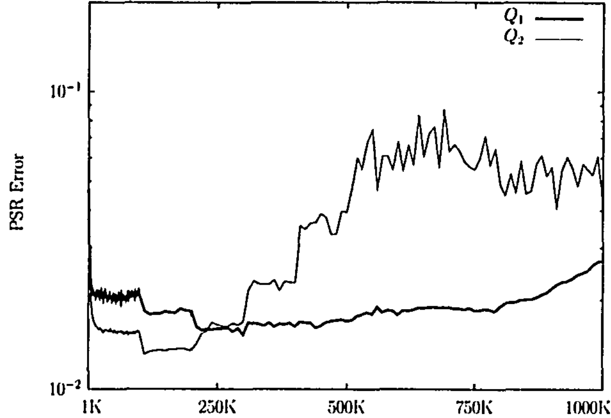
(c) Paint (d) Shuttle

(e) Network (f) 4x3 Maze

(g) Cheese Maze (h) Bridge Repair

Figure 6.3: PSR error for the constrained gradient algorithm, when α is set to zero after 250,000 time steps and 500,000 time steps. This shows the momentum of the $\widehat{y}(T|H)$ matrix.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



$$
\begin{aligned}
Q_1 &= \{\varepsilon, (\text{turn}, \text{see-LRV}), (\text{turn}, \text{nothing}), (\text{forward}, \text{see-MRV}), (\text{forward}, \text{nothing}), \\
&\quad (\text{backup}, \text{nothing}), (\text{backup}, \text{docked-LRV})\} \\
Q_2 &= \{\varepsilon, (\text{turn}, \text{see-LRV}), (\text{turn}, \text{see-MRV}), (\text{forward}, \text{see-LRV}), (\text{forward}, \text{see-MRV}), \\
&\quad (\text{backup}, \text{see-LRV}), (\text{backup}, \text{see-LRV})\}
\end{aligned}
$$

Figure 6.4: Performance results in the Shuttle domain using two different complete sets of core tests. The sets of core tests Q_1 and Q_2 are shown; see Appendix B for more information about the Shuttle domain.

these domains, once α is set to zero, learning stops and the generated model remains stable; essentially, the matrix $\widehat{y}(T|H)$ has no momentum. In fact, in the Paint domain, there is even a small decrease in error immediately after learning stopped; the matrix must have had momentum in the direction of the gradient. In the other domains, however, once $\alpha = 0$, the model has a tendency to drift and produce higher errors. This is especially noticeable in the Shuttle and Cheese Maze domains, which were also the most noticeable cases of error increasing in Figure 4.1. These results tend to support our hypothesis that the momentum of the matrix $\widehat{y}(T|H)$ can counteract the learning process of the constrained gradient algorithm.

6.2.3 Sets of Core Tests

It is known that the set of true core tests Q is not unique; any set of $|Q|$ linearly independent tests are capable of representing a dynamical system. However, this does not mean in practise that all sets Q are equivalent. In this section, we give an example of how different sets Q can alter the performance of the constrained gradient algorithm.

Figure 6.4 shows two average performance lines for the Shuttle domain. The

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

model that produced each line used a different, but sufficient, set of core tests. The set of core tests Q_1 produced better results than the set Q_2, despite the fact that they are theoretically equivalent. The set Q_2 appears to have better performance at first, but is then more susceptible to momentum of the prediction matrix. The constrained gradient algorithm is sensitive to the choice of Q, because this set is directly used to compute many of the predictions in each new row. Thus, different Q create different directions of momentum for the matrix $\hat{y}(T|H)$.

The core test selection used by the constrained gradient algorithm is biased towards selecting tests corresponding to columns that maintain a low condition number. This is the same bias used by other discovery algorithms, as well [James and Singh, 2004; Wolfe et al., 2005]. Other core test selection biases are possible; tests which are shorter could be preferred over longer tests, or tests which are more frequently encountered could be preferred over rarely executed tests. Some of these biases are also used in other discovery algorithms [Wolfe et al., 2005].

6.2.4 Summary of Investigative Tests

The main result of the investigative tests is that the constrained gradient algorithm has two forces behind its model creation: gradient descent and constraint satisfaction, or momentum. It was originally thought that these two forces would complement each other, but this appears to not always be the case. When the learning rate becomes too small to counteract the momentum of the matrix, models created by the constrained gradient algorithm can experience decay of prediction accuracy. The momentum appears to be dependent on the set of core tests used by the algorithm.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 7

Conclusion

In this section, we summarize the contributions presented in this thesis, and describe several avenues for future work with the constrained gradient algorithm.

7.1 Contributions

Our main contribution in this work is the presentation of the constrained gradient algorithm for discovery and learning of predictive state representations. This algorithm is the first online algorithm capable of discovery, and also the first learning algorithm that does not use strictly Monte Carlo learning updates. Experimentally, we have shown that the constrained gradient algorithm is capable of creating excellent models in some domains, and we have also shown domains in which the algorithm has difficulty learning correct models. Experiments were performed that suggest that the momentum of the learned submatrix can sometimes be counterproductive in learning a predictive state representation.

This work also has several other contributions. We have given a clear explanation of the exact constraints on a system dynamics matrix. We have written a proof for the discovery procedure used by most current discovery methods that does not require a POMDP representation of the system. We have performed the first online experiments with predictive state representations. Finally, we have included, as Appendix B, a clear description of the test domains that are frequently used in PSR research.

The goal at the outset of this work was to create an online algorithm for learning PSRs that does not require a reset action in the system, and that is able to extrapolate information based on the known structure of a system dynamics matrix. The constrained gradient algorithm satisfies these goals: it maintains a current state

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

vector, it does not require a reset action, and it uses the normalization procedure to propagate changes in probabilities. The algorithm is particularly successful at finding core tests with very little data, and is competitive with current algorithms at learning PSR models.

7.2 Future Work

Although the constrained gradient algorithm satisfies our initial goals in creating an algorithm for discovery and learning of PSRs, there is still work to be done. Our experimental investigation raised some interesting phenomena which should be addressed. Below, we list some possible avenues of future work with the constrained gradient algorithm.

7.2.1 Discovery Threshold

Two variations of the discovery procedure, cumulative and non-cumulative, have been described and tested in this work. Both versions have strengths and weaknesses. For practical use, though, the non-cumulative version would seem to be the best option, because it can base its decisions on more data than the cumulative version. In our experiments, however, the performance of non-cumulative discovery tended to suffer because a constant condition threshold was used.

To take advantage of non-cumulative discovery, the condition threshold should gradually increase over the course of learning, matching the gradual increase in the condition of $\widehat{y}(\widehat{Q}|H)$. Designing an appropriate schedule of increase for the threshold would be an excellent area for further investigation. Preferably, such a schedule would not require setting an explicit parameter, but instead be based on properties of the system, such as number of core tests selected or the condition number of the previously selected tests.

7.2.2 Selection of History Set

Our experiments with the constrained gradient algorithm used a finite window approach to selecting histories for H. This method has several advantages, but it also has two compounding disadvantages. The first is that H may be insufficient to represent the system, if it does not contain appropriate histories. To reduce the likelihood of insufficient representation, the size of H can be very large. However,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

using a large H causes computational penalties, which is the second main disadvantage; the constrained gradient algorithm is currently the most computationally expensive algorithm for learning PSRs.

A potentially better method of selecting H might be a hybrid approach. Histories selected for H could be a relatively small set of histories whose rows are most linearly unrelated. This addresses both the computation and insufficient representation issues. To ensure that erroneous (but linearly unrelated) rows are not kept permanently, a bound could be placed on the age of any history in the set H. Overall, this method could ensure an up-to-date matrix H, without suffering the insufficient representation and large computational costs of a plain finite-window approach.

7.2.3 Reformulation as Optimization Problem

Currently, each new row $\hat{y}(T|h)$ is computed in a multi-step process. The reason for this is because two goals are being balanced: satisfying the constraints of a system dynamics matrix, and matching the observed data. Furthermore, each prediction value is computed separately in the regression step, and the internal consistency constraints are enforced after all predictions have been computed. Perhaps a better solution to this problem is to compute the entire row at once, as a single optimization problem, so that all values can be calculated with knowledge of their relationship to other values in the row. [1] This approach allows the constraints to be handled more naturally. However, this approach would likely also be more computationally expensive than the current approach.

7.2.4 Enforcing Constraints

In Chapter 6, it was discovered that when α becomes very small, the momentum of the matrix $\hat{y}(T|H)$ can lead to poor representations. This occurs because the use of the constraints generates the entire row, and those values can only be changed a small amount in the direction of the gradient; the impact of the matrix is therefore greater than the impact of the data. In order for the constrained gradient algorithm to be usable for practical problems, this difficulty must be overcome. Unfortunately, it is unclear how to accomplish such a thing. Since the basis of the constrained gradient algorithm is that better performance can be achieved by enforcing system

[1] All of the constraints are linear, and the objective is quadratic. Thus, the optimization could be formulated as a quadratic program.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

dynamics matrix constraints, reducing the impact of the constraints seems counter to the spirit of the algorithm. Finding a better middle ground between constraint satisfaction and gradient descent is still an open problem.

7.2.5 Theoretical Convergence

Our current knowledge of the constrained gradient algorithm's behaviour is based purely on empirical testing. At the moment, we do not have any statements about the theoretical convergence of the algorithm to a local minimum or to a stable system dynamics matrix. Formulation of such theoretical knowledge would be helpful for understanding the properties of the algorithm.

7.3 Summary

Overall, we have found that the constrained gradient algorithm is capable of quickly discovering sets of core tests and building an initial model of system dynamics. However, the long-term performance of the algorithm can be hindered by local minima. There is still much work to be done in the field of using the structure of data to build better predictive representations. In this area, the constrained gradient algorithm provides a starting point and benchmark for future work.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Bibliography

K. J. Aström. Optimal control of Markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications*, pages 10:174–205, 1965.

Matthew Brand. Fast online SVD revisions for lightweight recommender systems. In *SIAM International Conference on Data Mining*, 2003.

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

Anthony Cassandra, Leslie Kaelbling, and Michael Littman. Acting optimally in partially observable stochastic domains. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI)*, 1994.

Lonnie Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI)*, 1992.

Hugh Ellis, Mingxiang Jiang, and Ross B. Corotis. Inspection, maintenance, and repair with partial observability. *Infrastructure Systems*, 1(2):92–99, 1995.

Ronald A. Howard. *Dynamic Probabilistic Systems: Volume II: Semi-Markov and Decision Processes.* John Wiley & Sons, Inc., 1971.

Herbert Jaeger. Discrete-time, discrete-valued observable operator models: a tutorial. Technical report, German National Research Center for Information Technology, 1998.

Michael R. James and Satinder Singh. Learning and discovery of predictive state representations in dynamical systems with reset. In *Proceedings of the 21st International Conference on Machine Learning (ICML)*, pages 719–726, 2004.

Michael R. James and Satinder Singh. Planning in models that combine memory with predictive representations of state. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI)*, 2005.

Michael R. James, Britton Wolfe, and Satinder Singh. Combining memory and landmarks with predictive representations. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI)*, 2005.

André I. Khuri. *Advanced Calculus with Applications in Statistics.* John Wiley & Sons, Inc., 2nd edition, 2003.

N. Kushmerick, S. Hanks, and D. Weld. An algorithm for probabilistic planning. *Artificial Intelligence*, 76(1–2):239–286, 1995.

Michael Littman. Network domain, January 1996.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Michael Littman, Richard S. Sutton, and Satinder Singh. Predictive representations of state. In *Advances in Neural Information Processing Systems 14 (NIPS)*, pages 1555-1561. MIT Press, 2002.

R. Andrew McCallum. Overcoming incomplete perception with utile distinction memory. In *Proceedings of the Tenth International Conference on Machine Learning (ICML)*, 1993.

Peter McCracken and Michael Bowling. Online learning of predictive state representations. In *Advances in Neural Information Processing Systems 18 (NIPS)*. MIT Press, 2006. To appear.

Ronald Parr and Stuart Russell. Approximating optimal policies for partially observable stochastic domains. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI)*, 1995.

Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons, New York, NY, 1994.

Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

Ronald L. Rivest and Robert E. Schapire. Diversity-based inference of finite automata. *Journal of the Association for Computing Machinery*, 41(3):555–589, 1994.

Matthew Rosencrantz, Geoff Gordon, and Sebastian Thrun. Learning low dimensional predictive representations. In *Proceedings of the 21st International Conference on Machine Learning (ICML)*, 2004.

Matthew R. Rudary and Satinder Singh. A nonlinear predictive state representation. In *Advances in Neural Information Processing Systems 16 (NIPS)*. MIT Press, 2004.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 2nd edition, 2003.

Satinder Singh, Michael Littman, Nicholas Jong, David Pardoe, and Peter Stone. Learning predictive state representations. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML)*, pages 712–719, 2003.

Satinder Singh, Michael R. James, and Matthew R. Rudary. Predictive state representations: A new theory for modeling dynamical systems. In *Uncertainty in Artificial Intelligence: Proceedings of the Twentieth Conference (UAI)*, pages 512–519, 2004.

Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

Richard S. Sutton and Brian Tanner. Temporal-difference networks. In *Advances in Neural Information Processing Systems 17 (NIPS)*, pages 1377–1384. MIT Press, 2005.

Richard S. Sutton, Eddie J. Rafols, and Anna Koop. Temporal abstraction in TD networks. Technical report, University of Alberta, 2005.

Brian Tanner and Richard S. Sutton. Temporal-difference networks eligibility traces: TD(lambda) networks. In *Proceedings of the 22nd International Conference on Machine Learning (ICML)*, 2005a.

Brian Tanner and Richard S. Sutton. Temporal-difference networks with history. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI)*, 2005b.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Eric Wiewiora. Learning predictive representations from a history. In *Proceedings of the 22nd International Conference on Machine Learning (ICML)*, 2005.

Britton Wolfe, Michael R. James, and Satinder Singh. Learning predictive state representations in dynamical systems without reset. In *Proceedings of the 22nd International Conference on Machine Learning (ICML)*, 2005.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Appendix A

Defining Test Predictions

Throughout this work, we have used $y(t|h)$ as the value of a test prediction. As noted previously, this deviates from the value of a test prediction used in previous PSR research [Littman et al., 2002; Singh et al., 2004]. In this appendix, we describe the previous approach to defining test predictions, the problems with this approach, and how using $y(t|h)$ as the definition of a test prediction solves these problems.

A.1 Test Predictions

The intention of a state representation is to model the dynamics of a system. In general, it is undesirable for the state representation to model the agent interacting with the system. Furthermore, the policy used by an agent when interacting with a system should not affect the state representation of the system it generates, as long as the policy is sufficiently diverse to uncover the entire complexity of the system. As we show below, however, in previous PSR research the definition of a test prediction meant that PSRs were dependent on the policy used by the agent.

In previous work, a test prediction for a test t was defined as $\Pr(\overline{o}_t|h, \overline{a}_t)$, abbreviated as $p(t|h)$ [Littman et al., 2002]. In words, this is the probability of perceiving all of the observations in $\overline{o}_t$, given that the agent takes all of the actions in $\overline{a}_t$ immediately following history h. Expanding $p(t|h)$, we get:

$$
\begin{aligned}
p(t|h) &\equiv \Pr(\overline{o}_t|h, \overline{a}_t) \\
&= \frac{\Pr(\overline{a}_t\overline{o}_t|h)}{\Pr(\overline{a}_t|h)} \\
&= \prod_{i=1}^{n} \frac{\Pr(a_i|h, a_1o_1 \ldots a_{i-1}o_{i-1})\Pr(o_i|h, a_1o_1 \ldots a_{i-1}o_{i-1}a_i)}{\Pr(a_i|h, a_1 \ldots a_{i-1})} \\
&= \prod_{i=1}^{n} \Pr(o_i|h, a_1o_1 \ldots a_{i-1}o_{i-1}a_i) \prod_{i=1}^{n} \frac{\Pr(a_i|h, a_1o_1 \ldots a_{i-1}o_{i-1})}{\Pr(a_i|h, a_1 \ldots a_{i-1})}
\end{aligned}
$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$$= y(t|h) \prod_{i=1}^{n} \frac{\Pr(a_i|h, a_1 o_1 \ldots a_{i-1} o_{i-1})}{\Pr(a_i|h, a_1 \ldots a_{i-1})}$$

The final step comes from the definition of $y(t|h)$, given in Chapter 2. Thus, $y(t|h)$ and $p(t|h)$ differ due to the existence of $\frac{\Pr(a_i|h, a_1 o_1 \ldots a_{i-1} o_{i-1})}{\Pr(a_i|h, a_1 \ldots a_{i-1})}$ terms in $p(t|h)$. These terms represent probabilities of choosing actions, and show that $p(t|h)$ is dependent on the policy used to generate actions; the value of $p(t|h)$ can be different for different policies. However, because the terms which compose $y(t|h)$ are probabilities of observations only, the value of $y(t|h)$ is not dependent on the policy used to generate the actions.[1] In the special case of policies in which all actions are generated independently of observations, $y(t|h) = p(t|h)$. Also, for any policy and any length one test ao, $y(ao|h) = p(ao|h)$.

A.2 Problems with $p(t|h)$

Predictive state representations and system dynamics matrices are closely related topics. We will use the system dynamics matrix to explain how the definition of a test prediction can affect the state representation. Overall, the system dynamics matrix $p(T^*|H^*)$ has several disadvantages when compared to the system dynamics matrix $y(T^*|H^*)$.

First, the system dynamics matrix $p(T^*|H^*)$ is dependent on the policy used to choose actions. Thus, for a given system, there is an entire family of different system dynamics matrices that specify the system. However, there is only one $y(T^*|H^*)$ matrix for any system. Furthermore, a $p(T^*|H^*)$ corresponding to a particular policy is not directly applicable to a different policy; this means that the prediction probabilities in a model learned using one policy cannot directly be used to ask questions about a different policy. It may be possible, if both polices are known, to convert a prediction $p(t|h)$ for one policy to another policy.

A second disadvantage to using $p(T^*|H^*)$ as a system dynamics matrix is that it can have higher linear dimension than the matrix $y(T^*|H^*)$. Wiewiora [2005] shows that the rank of $p(T^*|H^*)$ is the product of the complexity of the policy and the complexity of the underlying system. The rank of $y(T^*|H^*)$ is the linear complexity of the system only. A system dynamics matrix with higher linear dimension will cause the corresponding PSR to have higher dimension; this means more parameters

[1]The only case when $y(t|h)$ is dependent on policy is when the policy has zero probability of choosing some action after experiencing some history. Thus, we require $\Pr(a|h) > 0$ for all $a \in \mathcal{A}$ and for all histories h.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

must be estimated during learning, and the resulting PSR will have larger space and computation requirements.

Finally, and most importantly, some of the constraints listed in Chapter 3 do not apply to the system dynamics matrix $p(T^*|H^*)$. In particular, the internal consistency constraint and the conditional probability constraint do not apply. Below, we explain why.

The internal consistency constraint, using $p(t|h)$, would be:

$$p(t|h) = \sum_{o \in \mathcal{O}} p(tao|h)$$

However, the above is not necessarily true. Expanding the summation in the constraint, we get:

$$\begin{aligned}
& \sum_{o \in \mathcal{O}} p(tao|h) \\
= & \sum_{o \in \mathcal{O}} \Pr(\bar{o}_t o|h, \bar{a}_t a) \\
= & \sum_{o \in \mathcal{O}} \Pr(\bar{o}_t|h, \bar{a}_t a) \Pr(o|h, ta) \\
= & \Pr(\bar{o}_t|h, \bar{a}_t a) \sum_{o \in \mathcal{O}} \Pr(o|h, ta) \\
= & \Pr(\bar{o}_t|h, \bar{a}_t a) \\
= & \Pr(\bar{o}_t|h, \bar{a}_t) \frac{\Pr(a|h, \bar{a}_t \bar{o}_t)}{\Pr(a|h, \bar{a}_t)} \\
= & p(t|h) \frac{\Pr(a|h, \bar{a}_t \bar{o}_t)}{\Pr(a|h, \bar{a}_t)}
\end{aligned}$$

In the above, we see that the summation portion of the internal consistency constraint is not necessarily equal to $p(t|h)$. The internal consistency constraint is only true for $p(T^*|H^*)$ when the probability of choosing the action a is independent of the observations already seen; essentially, the constraint only holds in the cases where $p(t|h) = y(t|h)$.

The conditional probability constraint, using $p(t|h)$, states:

$$p(t|hao) = \frac{p(aot|h)}{p(ao|h)} \quad \forall a \in \mathcal{A}, o \in \mathcal{O}$$

The rule appears to be a simple application of the conditional probability rule. However, when not using shorthand form, the rule becomes:

$$\Pr(\bar{o}_t|hao, \bar{a}_t) = \frac{\Pr(o\bar{o}_t|h, a\bar{a}_t)}{\Pr(o|h, a)} \quad \forall a \in \mathcal{A}, o \in \mathcal{O}$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Two of the terms in the equation are conditioned on $\bar{a}_t$, but the denominator is not. Thus, this attempt to use conditional probability is incorrect. Corrected, the conditional probability rule should be:

$$\Pr(\bar{o}_t|hao, \bar{a}_t) = \frac{\Pr(o\bar{o}_t|h, a\bar{a}_t)}{\Pr(o|h, a\bar{a}_t)} \quad \forall a \in \mathcal{A}, o \in \mathcal{O}$$

This version of the rule is less useful, however, because the value $\Pr(o|h, a\bar{a}_t)$ is not generally available as an entry in the system dynamics matrix.

The conditional probability property is very important, because the PSR state update rule is based on this property. Therefore, using $p(T^*|H^*)$ restricts the use of PSRs to policies where:

$$\begin{aligned}
\Pr(o|h, a) &= \Pr(o|h, a\bar{a}) \\
\frac{\Pr(ao|h)}{\Pr(a|h)} &= \frac{\Pr(ao\bar{a}|h)}{\Pr(a\bar{a}|h)} \\
\frac{\Pr(ao|h)}{\Pr(a|h)} &= \frac{\Pr(ao|h)}{\Pr(a|h)} \frac{\Pr(\bar{a}|hao)}{\Pr(\bar{a}|ha)} \\
1 &= \frac{\Pr(\bar{a}|hao)}{\Pr(\bar{a}|ha)} \\
\Pr(\bar{a}|hao) &= \Pr(\bar{a}|ha)
\end{aligned}$$

More simply, using $p(T^*|H^*)$ restricts the use of PSRs to policies that choose actions independently of observations.

A.3 Redefining Test Predictions

As shown above, using $p(t|h)$ restricts the use of PSRs to the special case of *blind* policies that do not depend on the observations. It is our belief that the intent behind system dynamics matrices and PSRs is more aptly represented by using $y(t|h)$, rather than $p(t|h)$. This redefinition of test predictions means that PSRs are independent of policy. In this section, we show that the internal consistency and conditional probability constraints are correct when test predictions are defined by $y(t|h)$.

The following theorem shows that the internal consistency constraint is correct.

Theorem 2.

$$y(t|h) = \sum_{o \in \mathcal{O}} y(tao|h) \quad \forall a \in \mathcal{A}$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Proof.

$$\begin{aligned}
y(t|h) &= y(t|h) \times 1 \\
&= y(t|h) \times \sum_{o \in \mathcal{O}} \Pr(o|h, ta) \quad \forall a \in \mathcal{A} \\
&= \sum_{o \in \mathcal{O}} y(t|h) \times \Pr(o|h, ta) \quad \forall a \in \mathcal{A} \\
&= \sum_{o \in \mathcal{O}} y(tao|h) \quad \forall a \in \mathcal{A}
\end{aligned}$$

□

The following theorem shows that the conditional probability constraint is correct.

Theorem 3.

$$y(t|hao) = \frac{y(aot|h)}{y(ao|h)} \quad \forall a \in \mathcal{A}, o \in \mathcal{O}$$

Proof.

$$\begin{aligned}
y(t|hao) &= 1 \times y(t|hao) \quad \forall a \in \mathcal{A}, o \in \mathcal{O} \\
&= \frac{y(ao|h)}{y(ao|h)} \times y(t|hao) \quad \forall a \in \mathcal{A}, o \in \mathcal{O} \\
&= \frac{y(aot|h)}{y(ao|h)} \quad \forall a \in \mathcal{A}, o \in \mathcal{O}
\end{aligned}$$

□

Thus, we see that using $y(t|h)$ is appropriate for creating policy-independent PSRs.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Appendix B

Test Domains

The purpose of this appendix is to explain the dynamics of the eight domains used in the experiments in this work. For each domain, we list the action and observation sets, and describe the dynamics of the system. We also show the sets of core tests, $\mathcal{Q}$, that were used in the learning experiments in Chapter 4. Note that, with the exception of Float-Reset, the original purpose of these domains was for POMDP research; because of this, some domains have features which only make sense in the context of reward. Except for Float-Reset, all domains were obtained from an online POMDP repository [Cassandra, 1999]. Also, all domains in this appendix are representable by POMDPs, and therefore have a small number of specific nominal states; when describing the domains, *state* refers to these nominal states.

B.1 Float-Reset

$$\begin{aligned}
\mathcal{A} &= \{\text{float}, \text{reset}\} \\
\mathcal{O} &= \{0, 1\} \\
\mathcal{Q} &= \{\varepsilon, (\text{reset}, 0), (\text{float}, 0, \text{reset}, 0), (\text{float}, 0, \text{float}, 0, \text{reset}, 0), \\
&\quad (\text{float}, 0, \text{float}, 0, \text{float}, 0, \text{reset}, 0)\}
\end{aligned}$$

The Float-Reset domain [Littman et al., 2002] is pictured in Figure B.1. There are five nominal states, including a special reset state, which is shaded in the diagram. The 'reset' action always moves the system to the reset state. The 'float' action moves the system to the left or right state with uniform probability, except in the end states where it either moves or stays in the same state. The only way to produce a 1 observation is to perform the reset action while the system is in the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

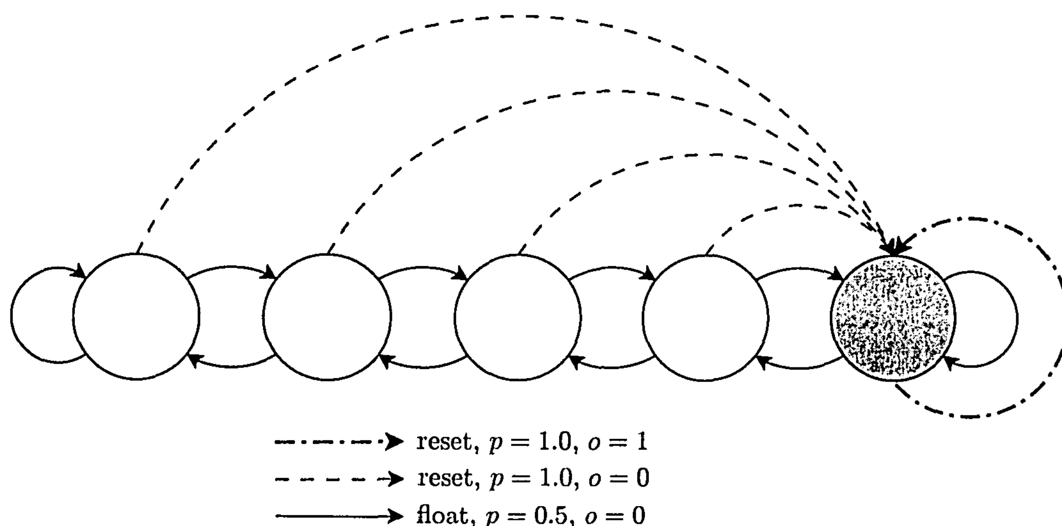


Figure B.1: The Float-Reset domain. (Repeated from Figure 2.1)

reset state. Reseting from any other state produces a 0, as does floating in any state.

B.2 Tiger

$$\begin{aligned}\mathcal{A} &= \{\text{listen}, \text{open-left}, \text{open-right}\}\\ \mathcal{O} &= \{\text{tiger-left}, \text{tiger-right}\}\\ \mathcal{Q} &= \{\varepsilon, (\text{listen}, \text{tiger-right})\}\end{aligned}$$

In the Tiger domain [Cassandra, Kaelbling, and Littman, 1994], an agent begins in a room with two doors, and a tiger is behind one of the doors with uniform probability. If the agent listens, it can hear the tiger, and correctly observes whether the tiger is to the left or the right 85% of the time (and misidentifies 15% of the time). Listening does not change which door the tiger is behind. Taking either door opening action causes a random observation and brings the agent to a new room, identical to the previous room, where the location of the tiger is once again uniformly randomly chosen.

Note that for purposes of representing the system, the actions 'open-left' and 'open-right' are identical. They are included for the control version of the Tiger domain, in which reward is applicable.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

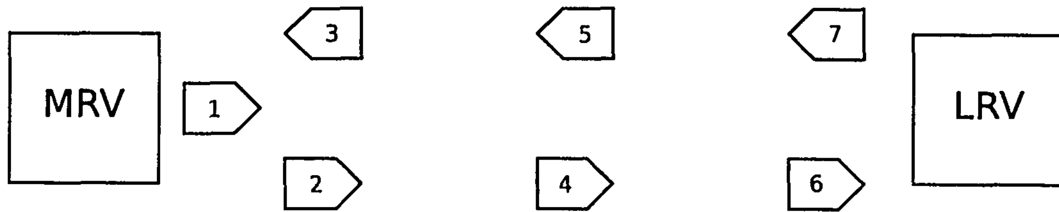


Figure B.2: The Shuttle domain.

B.3 Paint

$$\mathcal{A} = \{\text{inspect}, \text{paint}, \text{ship}, \text{reject}\}$$
$$\mathcal{O} = \{\text{blemished}, \text{not-blemished}\}$$
$$\mathcal{Q} = \{\varepsilon, (\text{inspect}, \text{blemished})\}$$

The Paint domain [Kushmerick, Hanks, and Weld, 1995] describes a widget painting operation. Widgets are either blemished or not-blemished with uniform probability, and blemished widgets can be transformed to non-blemished widgets by painting them. Painting is successful 90% of the time, and the widget remains blemished 10% of the time. Inspecting a widget reveals with 75% accuracy whether it is blemished or not, without changing the state. Shipping or rejecting the widget both present the agent with a new widget. The 'paint', 'ship' and 'reject' actions all produce the 'not-blemished' observation.

In the representation version of the Paint problem, the 'ship' and 'reject' actions are identical. They exist to differentiate states in the control problem.

B.4 Shuttle

$$\mathcal{A} = \{\text{forward}, \text{backup}, \text{turn}\}$$
$$\mathcal{O} = \{\text{nothing}, \text{see-MRV}, \text{see-LRV}, \text{docked-MRV}, \text{docked-LRV}\}$$
$$\mathcal{Q} = \{\varepsilon, (\text{turn}, \text{see-LRV}), (\text{turn}, \text{nothing}), (\text{forward}, \text{see-MRV}), (\text{forward}, \text{nothing}), (\text{backup}, \text{nothing}), (\text{backup}, \text{docked-LRV})\}$$

The Shuttle domain [Chrisman, 1992] describes a shuttle that ferries goods between two identical space stations, that are distinguished by recognizing which was the last one visited by the the shuttle. They are called MRV (most recently visited) and LRV (least recently visited). Figure B.2 shows the seven different positions

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

of the space shuttle. There is no position for being docked in LRV, because once the shuttle docks in the least recently visited station, it becomes the most recently visited.

Turning and going forward always have the expected effect, with the special cases that going forward while directly in front of a station (positions 3 and 6) does not change the position, and turning while docked moves the shuttle to be facing the station. Backing up is a noisy action. When directly in front of a station, facing the station, backing up launches into space 30% of the time, turns the ship around 30% of the time, and has no effect 40% of the time. When the ship is in front of a station with the rear towards the station, backup up docks 70% of the time and has no effect 30% of the time. In space, backing up moves closer to the station 80% of the time, turns the shuttle around 10% of the time, and has no effect 10% of the time. Backing up has no effect when docked.

When the shuttle is directly in front of a station, and facing it (positions 3 and 6), it can see the station. If a shuttle is directly in front of a station and facing away from it (positions 2 and 7), it sees nothing. The shuttle can also detect when it is docked (position 1). When the shuttle is in space between the two stations, its sensors are noisy; 70% of the time it sees the station it is facing, and 30% of the time it sees nothing.

B.5 Network

$$
\begin{aligned}
\mathcal{A} &= \{\text{unrestrict}, \text{steady}, \text{restrict}, \text{reboot}\} \\
\mathcal{O} &= \{\text{up}, \text{down}\} \\
Q &= \{\varepsilon, (\text{unrestrict}, \text{up}), (\text{steady}, \text{up}), (\text{restrict}, \text{up}), (\text{unrestrict}, \text{up}, \text{unrestrict}, \text{up}), \\
&\quad (\text{unrestrict}, \text{down}, \text{unrestrict}, \text{up}), (\text{unrestrict}, \text{up}, \text{steady}, \text{up})\}
\end{aligned}
$$

The Network domain [Littman, 1996] simulates a network with six levels of stability, plus a crashed state. The agent controls the stability of the network by letting network flow be unrestricted, steady, or restricted, and can also reboot the network. Because the dynamics of the domain are fairly complicated, we present the effects of the four actions in Table B.1.

The general summary of the actions in the Network domain is that unrestricted flow tends to increase instability, steady flow tends to maintain the current stability

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

(a) unrestrict

	1	2	3	4	5	6	C
1	0.5	0.3	0.1	0.1	0.0	0.0	0.0
2	0.2	0.3	0.3	0.1	0.1	0.0	0.0
3	0.1	0.1	0.3	0.3	0.1	0.1	0.0
4	0.0	0.1	0.1	0.3	0.3	0.1	0.1
5	0.0	0.0	0.1	0.1	0.3	0.3	0.2
6	0.0	0.0	0.0	0.1	0.1	0.3	0.5
C	0.0	0.0	0.0	0.0	0.0	0.0	1.0

(b) steady

	1	2	3	4	5	6	C
1	0.7	0.2	0.1	0.0	0.0	0.0	0.0
2	0.3	0.4	0.3	0.1	0.0	0.0	0.0
3	0.1	0.2	0.4	0.2	0.1	0.0	0.0
4	0.0	0.1	0.2	0.4	0.2	0.1	0.0
5	0.0	0.0	0.1	0.2	0.4	0.2	0.1
6	0.0	0.0	0.0	0.1	0.2	0.4	0.3
C	0.0	0.0	0.0	0.0	0.0	0.0	1.0

(c) restrict

	1	2	3	4	5	6	C
1	0.8	0.1	0.1	0.0	0.0	0.0	0.0
2	0.5	0.3	0.1	0.1	0.0	0.0	0.0
3	0.2	0.3	0.3	0.1	0.1	0.0	0.0
4	0.1	0.1	0.3	0.3	0.1	0.1	0.0
5	0.1	0.0	0.1	0.3	0.3	0.1	0.1
6	0.0	0.1	0.0	0.1	0.3	0.3	0.2
C	0.0	0.0	0.0	0.0	0.0	0.0	1.0

(d) reboot

	1	2	3	4	5	6	C
1	1.0	0.0	0.0	0.0	0.0	0.0	0.0
2	1.0	0.0	0.0	0.0	0.0	0.0	0.0
3	1.0	0.0	0.0	0.0	0.0	0.0	0.0
4	1.0	0.0	0.0	0.0	0.0	0.0	0.0
5	1.0	0.0	0.0	0.0	0.0	0.0	0.0
6	1.0	0.0	0.0	0.0	0.0	0.0	0.0
C	1.0	0.0	0.0	0.0	0.0	0.0	0.0

Table B.1: The state transitions for the Network domain. Numbers 1 to 6 are stability levels, and C is the crashed state. Each row is a probability distribution over states for the corresponding starting state.

level, and restricted flow tends to make the network more stable. Rebooting the system always makes it completely stable. Nothing changes the crashed state except for a reboot.

The observations in the Network domain are noisy. The network is seen as up or down, depending on how stable it is. At stability levels 1, 2 and 3, the network is always seen as up. At stability level 4, there is a 90% chance of observing the network up, at level 5, 70%, and at level 6, 50%. A crashed network is always down.

B.6 4x3 Maze

$$\mathcal{A} = \{\text{north}, \text{south}, \text{east}, \text{west}\}$$

$$\mathcal{O} = \{\text{neither}, \text{left}, \text{right}, \text{both}, \text{good}, \text{bad}\}$$

$$Q = \{\varepsilon, (\text{north}, \text{left}), (\text{north}, \text{right}), (\text{north}, \text{neither}), (\text{north}, \text{both}), (\text{north}, \text{good}), (\text{south}, \text{left}), (\text{south}, \text{right}), (\text{south}, \text{neither}), (\text{north}, \text{left}, \text{north}, \text{left})\}$$

The 4x3 Maze [Parr and Russell, 1995] is a grid world pictured in Figure B.3. The agent can move in the four compass directions, and can perceive whether there

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure B.3: The 4x3 Maze domain.

Figure B.4: The Cheese domain.

is a wall to its left, to its right, both, or neither. There are two special grid locations, in which the agent perceives special observations; it perceives the position with a '+' to be 'good', and the position with a '-' to be 'bad'. Moving is noisy; 80% of the time, the agent moves in its intended direction, and 10% of the time it moves in one of the directions perpendicular to its intended direction. Moving into a wall does not change the position of the agent. When the agent is in either of the special locations, its next action will randomly transport the agent to any of the non-special locations.

B.7 Cheese Maze

$$\mathcal{A} = \{\text{north}, \text{south}, \text{east}, \text{west}\}$$

$$\mathcal{O} = \{1, 2, 3, 4, 5, 6, 7\}$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

(a) no-repair

	1	2	3	4	5
1	0.80	0.13	0.02	0.00	0.05
2	0.00	0.70	0.17	0.05	0.08
3	0.00	0.00	0.75	0.15	0.10
4	0.00	0.00	0.00	0.60	0.40
5	0.00	0.00	0.00	0.00	1.00

(b) clean-paint

	1	2	3	4	5
1	0.80	0.13	0.02	0.00	0.05
2	0.00	0.80	0.10	0.02	0.08
3	0.00	0.00	0.80	0.10	0.10
4	0.00	0.00	0.00	0.60	0.40
5	0.00	0.00	0.00	0.00	1.00

(c) strengthen-paint

	1	2	3	4	5
1	0.80	0.13	0.02	0.00	0.05
2	0.19	0.65	0.08	0.02	0.06
3	0.10	0.20	0.56	0.08	0.06
4	0.00	0.10	0.25	0.55	0.10
5	0.00	0.00	0.00	0.00	1.00

(d) structural-repair

	1	2	3	4	5
1	0.80	0.13	0.02	0.00	0.05
2	0.80	0.13	0.02	0.00	0.05
3	0.80	0.13	0.02	0.00	0.05
4	0.80	0.13	0.02	0.00	0.05
5	0.80	0.13	0.02	0.00	0.05

Table B.2: The effect of the repair actions on the structural stability of the bridges. Each row represents the starting stability of the bridge, and each column represents the stability after the repair action is taken.

$$\begin{aligned} Q &= \{\varepsilon, (\text{north}, 1), (\text{north}, 2), (\text{north}, 3), (\text{north}, 4), (\text{north}, 5), (\text{south}, 5), \\ &\quad (\text{south}, 6), (\text{east}, 2), (\text{east}, 3), (\text{north}, 5, \text{north}, 1)\} \end{aligned}$$

The Cheese Maze [McCallum, 1993] is a grid world pictured in Figure B.4. The agent can move in the four compass directions, and actions are never noisy. The agent perceives the four walls around it; the state-observation mappings are shown in the diagram. The shaded location has a piece of cheese, and has a special observation. Taking any action in the cheese location causes the agent to be randomly transported to any of the other locations.

B.8 Bridge Repair

$$\begin{aligned} \mathcal{A} &= \{\text{no-repair}, \text{clean-paint}, \text{paint-strengthen}, \text{structural-repair}\} \times \\ &\quad \{\text{no-inspect}, \text{visual-inspect}, \text{ut-inspect}\} \\ \mathcal{O} &= \{\text{less-than-5}, \text{between-5-and-15}, \text{between-15-and-25}, \text{greater-than-25}, \text{failed}\} \\ Q &= \{\varepsilon, (\text{no-repair-and-ut-inspect}, \text{less-than-5}), \\ &\quad (\text{clean-paint-and-ut-inspect}, \text{between-5-and-15}), \\ &\quad (\text{clean-paint-and-ut-inspect}, \text{between-15-and-25}), \\ &\quad (\text{paint-strengthen-and-ut-inspect}, \text{failed})\} \end{aligned}$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

(a) no-inspect

	1	2	3	4	5
1	1.00	0.00	0.00	0.00	0.00
2	1.00	0.00	0.00	0.00	0.00
3	1.00	0.00	0.00	0.00	0.00
4	1.00	0.00	0.00	0.00	0.00
5	1.00	0.00	0.00	0.00	0.00

(b) visual-inspect

	1	2	3	4	5
1	0.80	0.20	0.00	0.00	0.00
2	0.20	0.60	0.20	0.00	0.00
3	0.05	0.70	0.25	0.00	0.00
4	0.00	0.30	0.70	0.00	0.00
5	0.00	0.00	1.00	0.00	0.00

(c) ut-inspect

	1	2	3	4	5
1	0.90	0.10	0.00	0.00	0.00
2	0.05	0.90	0.05	0.00	0.00
3	0.00	0.05	0.90	0.05	0.00
4	0.00	0.00	0.05	0.90	0.05
5	0.00	0.00	0.00	0.00	1.00

Table B.3: The probability distributions over observations generated by each inspection action taken in each state, for the Bridge domain.

The Bridge Repair domain [Ellis, Jiang, and Corotis, 1995] simulates a bridge that has five different degrees of structural strength. It is formulated somewhat differently than other systems. At each time step, the agent takes both a repair action and an inspect action, so the action set is the cross product of these sets. The change in state is dependent only on the repair action taken, and the observation is dependent only on the inspect action. We summarize the effect of these actions on the state and observation in Tables B.2 and B.3. To summarize the repair actions, no-repair tends to maintain the state of the bridge, but has a trend toward worse stability. The action clean-paint is similar to no-repair, but maintains the state slightly more. The action strengthen-paint has a trend toward making the bridge stronger, and structural-repair resets the state of the bridge. Of the inspection actions, no-inspect provides no information, visual inspect tends to report that the bridge is stronger than it is, and ut-inspect provides fairly accurate information about the strength of the bridge.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.