Learning Models of World Dynamics Using Bayesian Networks

by

Edward Tolson

S.B., Electrical Engineering and Computer Science
Massachusetts Institute of Technology (2001)

Submitted to the Department of Electrical Engineering and Computer Science in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2002

Copyright 2002 Edward Tolson. All rights reserved.

The author hereby grants to MIT permission to reproduce and distribute publicly paper and electronic copies of this thesis document in whole or in part and to grant others the right to do so.

Author

Department of Electrical Engineering and Computer Science

May 23, 2002

Certified by

Leslie Pack Kadibling
Thesis Supervisor

Accepted by

Arthur C. Smith
Chairman, Department Committee on Graduate Theses
Learning Models of World Dynamics Using Bayesian Networks

by

Edward Tolson

Submitted to the Department of Electrical Engineering and Computer Science

May 23, 2002

In Partial Fulfillment of the Requirements for the Degree of Master of Engineering in Electrical Engineering and Computer Science

Abstract

Dynamic Bayesian networks (DBN's) provide a framework for learning in the context of stochastic time-series. DBN's are factored and allow learning of hidden state; in fact both hidden Markov models and Kalman filters can be thought of as subclasses of DBN's. In this thesis, I explore the use of DBN's to learn models of world dynamics. DBN's are learned both under assumption of the Markov property, and in the pursuit of learning non-Markovian dynamics, and the resulting models are evaluated.

Thesis Supervisor: Leslie Pack Kaelbling
Title: Professor of Computer Science and Engineering, Associate Director of the MIT Artificial Intelligence Laboratory
Acknowledgements

I would like to first thank Professor Leslie Pack Kaelbling for her supervising role in this thesis. Her intelligence, her care for her students, and her passion for Artificial Intelligence made this project a very fun experience. I would also like to thank my teammate, Luke Zettlemoyer, who made himself selflessly available for help in all aspects of this thesis.

I thank my parents who supported and encouraged me. I thank Jeremy and Christi, who put their computers at my disposal during the final weeks. I thank Theresa for being caring, supportive, and understanding. Finally, the greatest thanks are due to God, who gives me every ability and every opportunity I have.
# Contents

1 **Introduction** ........................................................................................................... 7  
   1.1 Motivation ............................................................................................................ 7  
   1.2 Outline .................................................................................................................. 8  

2 **Bayesian Networks** ............................................................................................. 9  
   2.1 Introduction to Bayesian Networks ....................................................................... 9  
   2.2 Learning a Bayesian Network from Data ........................................................... 11  
      2.2.1 Learning Parameters for a Fixed Structure .................................................. 11  
      2.2.2 Choosing Between Structures ..................................................................... 12  
         2.2.2.1 Bayesian Attempts ................................................................................. 13  
         2.2.2.2 Choosing a Single Network Structure ................................................. 14  
         2.2.2.3 Complexity of Searching Among Structures .................................. 17  

3 **Dynamic Bayesian Networks** .............................................................................. 19  
   3.1 Learning a Dynamic Bayesian Network from Data .......................................... 20  
   3.2 Improving DBN Learning in Worlds ..................................................................... 22  

4 **Learning Hidden State** ...................................................................................... 23  
   4.1 Learning DBN Parameters in the Presence of Hidden Variables .................... 24  
      4.1.1 The EM Algorithm for Choosing Model Parameters .................................. 24  
      4.1.2 The Forward-Backward Algorithm for Inference in the E-step ................. 25  
      4.1.3 Complexity of Calculating Expected Statistics ....................................... 28  
   4.2 Choosing Between DBN Structures in the Presence of Hidden Nodes ......... 30  
   4.3 Discovering Hidden State .................................................................................... 31  
      4.3.1 Using a k-Slice DBN to Detect Hidden State ............................................ 32  
      4.3.2 Improving Hidden State Detection in Worlds .......................................... 34  

5 **Worlds** ................................................................................................................. 39
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 The Driving World</td>
<td>39</td>
</tr>
<tr>
<td>5.2 The Left Signaling World</td>
<td>41</td>
</tr>
<tr>
<td>5.3 The Signaling World</td>
<td>41</td>
</tr>
<tr>
<td>5.4 The Slippery Driving World</td>
<td>42</td>
</tr>
<tr>
<td>5.5 The Driving World with Signaling and Slipping</td>
<td>43</td>
</tr>
<tr>
<td>6 Implementation</td>
<td>44</td>
</tr>
<tr>
<td>6.1 Class Overview</td>
<td>44</td>
</tr>
<tr>
<td>6.2 Modularity</td>
<td>46</td>
</tr>
<tr>
<td>7 Results and Evaluation</td>
<td>48</td>
</tr>
<tr>
<td>7.1 Learning Models Without Hidden Nodes</td>
<td>48</td>
</tr>
<tr>
<td>7.2 Discovering and Learning Hidden State</td>
<td>50</td>
</tr>
<tr>
<td>7.2.1 The Signaling Worlds</td>
<td>51</td>
</tr>
<tr>
<td>7.2.2 The Slippery Driving World</td>
<td>58</td>
</tr>
<tr>
<td>7.2.3 The Driving World with Signaling and Slipping</td>
<td>64</td>
</tr>
<tr>
<td>8 Conclusions</td>
<td>67</td>
</tr>
<tr>
<td>A Concise Descriptions of Each World</td>
<td>68</td>
</tr>
<tr>
<td>A.1 The Driving World</td>
<td>68</td>
</tr>
<tr>
<td>A.2 The Left Signaling World</td>
<td>69</td>
</tr>
<tr>
<td>A.3 The Signaling World</td>
<td>69</td>
</tr>
<tr>
<td>A.4 The Slippery Driving World</td>
<td>70</td>
</tr>
<tr>
<td>A.5 The Driving World with Signaling and Slipping</td>
<td>70</td>
</tr>
</tbody>
</table>
# List of Figures

2-1 An example Bayesian network..............................................................10  
2-2 The GREEDY-SEARCH algorithm......................................................17  
3-1 An example dynamic Bayesian network.............................................20  
3-2 Initial DBN structure when learning a model of the dynamics of a world.22  
4-1 Unrolling a DBN................................................................................26  
4-2 The generic STRUCTURAL-SEARCH-EM algorithm..........................32  
4-3 Converting a k-slice DBN to a DBN with hidden nodes......................33  
4-4 The K-SLICE-DBN-HIDDEN-STATE-DISCOVERY algorithm.................34  
4-5 Adding a hidden variable to improve the model of a reward..................36  
4-6 The WORLD-HIDDEN-STATE-DISCOVERY algorithm..........................37  
5-1 The distance-thermometer of the Driving World..................................41  
6-1 The major classes of the implementation and their relationships..........45  
6-2 The architecture of the learning implementation.................................46  
7-1 Learning a DBN without hidden nodes................................................49  
7-2 Comparison of learning strategies in the Left Signaling World...............51  
7-3 Comparison of learning strategies in the Signaling World......................51  
7-4 A model of the Left Signaling World without hidden nodes..................52  
7-5 An imperfect model of Left Signaling World with hidden nodes.............52  
7-6 An optimal model of the Left Signaling World....................................54  
7-7 A model of the Signaling World without hidden nodes........................54  
7-8 An imperfect model of Signaling World with hidden nodes...................55  
7-9 An optimal model of the Signaling World..........................................56  
7-10 A model of the Signaling World with erroneous edges.......................57  
7-11 Comparison of learning strategies in the Slippery Driving World with perfect information accuracy..................................................59  
7-12 Comparison of learning strategies in the Slippery Driving World with eighty percent information accuracy.............................................59  
7-13 Comparison of learning strategies in the Slippery Driving World with sixty percent information accuracy..............................................59  
7-14 A model of the Slippery Driving World without hidden nodes................60  
7-15 An optimal model of the Slippery Driving World...............................61  
7-16 A model of the Slippery Driving World with one erroneous edge...........62  
7-17 Comparison of learning strategies in the Driving World with Signaling and Slipping.................................................................64
Chapter 1

Introduction

1.1 Motivation

Imagine that you are put in charge of a complex set of controls that regulate various operating parameters at a factory. Though the intended function of each control is known, changing various settings has effects that are not well understood. For example, changing the speed at which a certain machine operates may influence the temperature in that part of the factory, which may in turn influence the function of another machine. The manager of the factory would like you to determine the side-effects associated with each control and under what conditions they occur.

Or imagine that you wish to create an artificial intelligence system that trades securities for profit. The system receives information about the current state of the market and has the ability to place orders through different mechanisms and at different prices and order sizes. In order to have the ability to learn profitable trading strategies, you would like the system to learn what factors affect security prices, including both external factors and the system's own influence on the market through the orders it places.

The worlds in both of these examples are composed of a set of factored observations, of which some are specially identified as rewards (e.g., the amount and quality of widget production in the factory, and making a profitable trade in the securities
example). We wish to learn a model of these stochastic, dynamic worlds that can then be used to maximize the rewards received over time. The model should explicitly represent important features of the world, such as causal relationships. In this thesis I investigate the use of dynamic Bayesian networks to learn such models.

1.2 Outline

Chapter 2 discusses Bayesian networks and the techniques used to learn a Bayesian network from data. Chapter 3 discusses dynamic Bayesian networks and techniques for learning dynamic Bayesian networks from data. Chapter 4 presents the need for hidden variables, discusses learning Bayesian networks in the presence of hidden variables, and presents two algorithms for discovering hidden state in the underlying world. Chapter 5 presents the worlds that are used in this project. Chapter 6 discusses the implementation. Chapter 7 presents experimental results. Chapter 8 discusses conclusions.
Chapter 2

Bayesian Networks

A Bayesian network (also known as a belief network or a probabilistic network) is a graphical representation of a joint distribution of random variables. Bayesian networks have become a very popular choice in the field of Artificial Intelligence and have been successfully used in a wide range of applications. Bayesian networks are currently used in many real-world applications, including military applications, systems related to healthcare, and fraud detection systems (Neil et al., 2001; Andreassen et al., 1991; Kahn et al., 1995; Havorka et al., 1993; Taniguchi et al., 1998; Ezawa and Schuermann, 1995).

2.1 Introduction to Bayesian Networks

A Bayesian network is a directed, acyclic graph and an associated set of parameters. The graph contains one node for each random variable of interest. Each edge in the graph represents a direct dependence between two variables. More specifically, given its parents, a variable is conditionally independent from all nodes that are not its descendants (Pearl, 1988). The parameters of a Bayesian network consist of a conditional probability table associated with each variable. A variable’s conditional probability table provides a distribution over the variable's value, given a configuration of the values of its parents.
Figure 2-1: An example Bayesian network modeling the relationships between the Flu, Malaria, and a few related symptoms.

For simplicity, only binary random variables are used in this project. Therefore, to fully represent a variable's distribution, the conditional probability table need only specify the probability that the variable is true. The conditional probability table for a random variable, $X_i$, whose parents in the Bayesian network are $\text{Pa}(X_i)$, must then have $2^{|\text{Pa}(X_i)|}$ entries, each of which specifies the probability that $X_i$ is true given a particular configuration of the variables in $\text{Pa}(X_i)$. Figure 2-1 shows an example Bayesian network modeling the probabilities that an arbitrary patient entering a hospital has the Flu, Malaria, and a few related symptoms. The edges in this graph have been chosen to connect cause to effect, though they do not have to run in this direction.
2.2 Learning a Bayesian Network from Data

In this section we examine the task of learning a Bayesian network to “fit” observed data. This goal can be broken down into two tasks: the relatively easy task of choosing the parameters for a given graph structure, and the more difficult task of choosing the best graph structure.

2.2.1 Learning Parameters for a Fixed Structure

Given a fixed network structure, the task of choosing the best parameters for the conditional probability tables is relatively easy, and a closed form equation can be derived. We would like to choose the set of parameters, $\Theta$, for the network, given a set of observations, $D = \{X[1], X[2], ..., X[T]\}$, and background or prior knowledge, $\xi$. As shown by Heckerman (1996), the parameters $\Theta_{Xi}$ for each node can be analyzed individually. Let the parameter $\Theta_{Xi|c}$ represent the probability that variable $X_i$ is true given that the parents of $X_i$, $Pa(X_i)$, take on configuration $c$. The sufficient statistics for the parameter $\Theta_{Xi|c}$ are $t$ and $f$, where $t$ is the number of times in the training data that $Pa(X_i)$ took configuration $c$ and $X_i$ took the value true, and $f$ is the number of times in the training data that $Pa(X_i)$ took configuration $c$ and $X_i$ took the value false. We wish to choose the parameter $\Theta_{Xi|c}$ to be the probability that variable $X_i$ will take the value true in a new sample taken independently, where $Pa(X_i)$ takes configuration $c$. If we view our background knowledge, $\xi$, as a set of "equivalent" training instances, in which $Pa(X_i)$
took configuration c and $X_i$ took the value *true* $T$ times, and $\text{Pa}(X_i)$ took configuration c and $X_i$ took the value *false* $F$ times, then the probability we desire is the following (Heckerman, 1996):

$$\Theta_{X|c} = \frac{t + T}{t + f + T + F}.$$ 

Calculating each parameter, then, requires only counting the number of instances of certain configurations in the data. The total number of parameters of a Bayesian network with binary variables is $O(N2^p)$, where $N$ represents the number of nodes in the network and $P$ is the maximum number of parents of any one node. In calculating all parameters for a Bayesian network, each data sample, $X[d]$, will contribute exactly one count to the statistics for each node. Deciding which statistic it contributes to requires $O(P)$ operations for each node. Counting the expected statistics from the data therefore requires $O(TNP)$ operations. Since each parameter must be initialized before the expected statistics are counted, the total number of operations required to calculate all the model parameters from a data set requires $O(N(TP + 2^P))$ operations.

### 2.2.2 Choosing Between Structures

In this section we move to the more difficult task of choosing a network structure. We first examine Bayesian approaches and why this type of approach does not seem feasible for modeling world dynamics. We then move to the approach of selecting a single structure, and examine the complexity of doing so.
2.2.2.1 Bayesian Attempts

I first discuss why we would wish to choose a single network structure at all. A Bayesian approach to any situation is to construct a prior distribution, and then use the prior distribution and the observations to create a posterior distribution. We could adopt this approach in the case of Bayesian network structure. Given a prior distribution \( \Pr(G | \xi) \) over graph structures and a prior distribution \( \Pr(\Theta | G, \xi) \) over parameters for a given structure, we can examine the posterior distribution over graph structures. Note, however, that any directed acyclic graph (DAG) is a valid network structure, and the number of DAG's is super-exponential in the number of nodes in the network (Tong and Koller, 2001). This makes examining the likelihood of each possible network structure impractical in networks with more than a few nodes. A solution proposed by Friedman and Koller (2000), is to calculate the likelihood of certain features of the structure, such as whether the variable \( X_i \) is a parent of the variable \( X_j \). For a given feature, \( f \), we have the following posterior distribution:

\[
\Pr(f | D, \xi) = \frac{\Pr(f | \xi) \Pr(D | f, \xi)}{\Pr(D | \xi)}.
\]

Accordingly, to compute this distribution we must sum over the super-exponential number of possible structures, which again is prohibitive in most domains. However, when the choice of network structures is restricted to a particular topological ordering of the nodes, a closed form equation for summing the likelihood of many desirable features exists (Friedman and Koller, 2000). While this is a very useful discovery, for the purposes of modeling world dynamics it does not seem like a viable approach. We desire a concise model of the dynamics of a world, and a distribution over network features is
far from this. On the other hand, a single Bayesian network can provide a very concise and rich model, since not only can desired distributions be calculated from the network, but the network structure often explicitly illustrates causality and conditional independence. Hence the best approach seems to be to choose a single Bayesian network which represents the data well, and in fact this is the usual choice in Bayesian network learning.

2.2.2.2 Choosing a Single Network Structure

We wish to choose a single network structure that "fits" the data well and captures the underlying distribution. A first observation is that if a network "fits" the data well, the likelihood of the data given the network should be relatively high. Therefore, we could adopt the na"ive approach of choosing a network that maximizes the probability of the data given the network. Such an approach, however, results in over-fitting. It is easy to show that any fully connected network structure (containing the maximum number of edges while still being acyclic), with parameters chosen appropriately, will always maximize the likelihood of the data given the network (Friedman and Yakhini, 1996). Rissanen (1987) presents a minimum description length (MDL) criterion for scoring a model which attempts to balance the complexity of the model with the "fit" of the model to the data. Using this score we attempt to minimize the number of bits required to encode the model and the data. We would first encode the model, which requires encoding each parameter in the model. After the model is encoded, the amount of bits required to encode the data will vary with the likelihood of the data given the model.
Though there is no exact closed form for the encoding length, it can be approximated as follows (Rissanen, 1987):

$$\text{MDL}(G, \Theta, D) \equiv \frac{1}{2} (|\Theta| \log |D|) - \log \Pr(D \mid G, \Theta).$$

We wish to choose a network that minimizes the minimum description length, and hence we want a network that maximizes the following score function:

$$\text{Score}(G, \Theta : D) = \log \Pr(D \mid G, \Theta) - \frac{1}{2} (|\Theta| \log |D|).$$

This MDL score is in fact the same as the Bayesian information criterion (BIC) score presented by Schwarz (1978), which was derived as an approximation to the posterior likelihood of a graph structure. Note that the log likelihood term measures the "fit" of the model to the data, while the second term penalizes based upon the complexity of the model. The log likelihood term grows in proportion to the amount of data while the penalizing term grows in proportion to the logarithm of the amount of data. Thus, as the amount of data increases, the scoring function places more emphasis on choosing a model that "fits" the data well.

An important property of this scoring function is its decomposability. The scoring function decomposes into a sum of terms, each of which is associated with a single node. Using the chain rule for Bayesian networks (Getoor et al., 1999), the log probability term decomposes in the following manner:

$$\log \Pr(D \mid G, \Theta) = \sum_{d=1}^{|D|} \log \Pr(X = x[d] \mid G, \Theta)$$

$$= \sum_{d=1}^{|D|} \sum_{i=1}^{N} \log \Pr(X_i = x_i[d] \mid \text{Pa}(x_i)[d], G, \Theta)$$

$$= \sum_{i=1}^{N} \sum_{d=1}^{|D|} \log \Pr(X_i = x_i[d] \mid \text{Pa}(x_i)[d], G, \Theta).$$
If the amount of training data is viewed as a constant, the penalizing term is a sum over the number of parameters, and hence each node contributes to the penalty in proportion to the number of parameters in its conditional probability table. This decomposition property will be used to improve the efficiency of searching among structures.

Now that a scoring function has been established, we are presented with the task of finding a network structure that maximizes the scoring function. As there is no closed form solution and there is a super-exponential number of structures to choose among, there is no practical way of finding the global optimum. Given a current graph structure, we can consider adding or removing an edge to create a "neighboring" structure. Casting the problem in this manner makes it a discrete form of gradient ascent, and standard strategies including greedy search, TABU-search, and simulated annealing are all options. As is always the case with gradient ascent, the contour of the search space has a large impact on the successfulness of the search. Adding the reversal of an edge to the choice of operations leading to neighbors can improve the contour of the space, and hence improve the search results. In this project, both a greedy searching method and a greedy searching method with random restarts were implemented. Because the greedy search without restarts ran much faster and generally produced results almost as good as the greedy search with random restarts, the former was chosen for use in learning. Pseudocode for the GREEDY-SEARCH algorithm is listed in Figure 2-2. Note that the order in which the nodes are examined is randomized during each iteration of the algorithm to prevent the ordering of the nodes in the graph from affecting the search results.
2.2.2.3 Complexity of Searching Among Structures

When examining a neighbor of the current graph structure, the decomposition property of the scoring function is used to make calculating the score of the neighboring structure more efficient. In particular, only the scores of nodes with different parents in the neighboring structure need be recalculated. In the case of adding or removing an edge there is only one such node, while in the case of reversing an edge there are two. Hence, the time required to evaluate the score change when performing one of these operations is
independent of the overall size of the network, given the number of parents of the affected nodes. In order to determine the score of a neighboring structure, a new conditional probability table must be calculated for each affected node, and then the score change at each node can be determined.

Calculating the new conditional probability parameters for a single node requires examining each data sample and updating the count for the appropriate row of the conditional probability distribution. This requires $O(TP)$ operations, where $T$ is the number of data samples and $P$ is the maximum number of parents of any node. Determining the score associated with a particular node requires calculating the log likelihood of the associated random variable in each data sample and penalizing for complexity. Penalizing for complexity is relatively simple, and most penalizing terms can be calculated in $O(P)$ operations. Calculating the log likelihood of the associated random variable requires examining the value of the variable and its parents in each data sample, and hence requires $O(TP)$ operations. We have, then, that setting the parameters and calculating the score for a single node requires $O(TP)$ operations. Since at most two nodes are affected by any of the structure operations, the total number of operations required to evaluate a neighboring model is $O(TP)$. 
Chapter 3

Dynamic Bayesian Networks

A dynamic Bayesian network (DBN) is a Bayesian network used to represent the distribution of successive observations of a stochastic dynamic system. There is a node in the network representing the value of each observation at some arbitrary time $t$, and a node representing the value of each observation at the next time step, time $t+1$ (this second node may be omitted for observations whose behavior we are not interested in modeling). Since we are interested in predicting future observations given the present, all edges will be incident on nodes representing a variable at time $t+1$, and so the nodes representing variables at time $t$ will be roots in the graph. I use $N_i$ to refer to the node that represents the value of variable $X_i$ at time $t$, and $N_i'$ to refer to the node that represents the value at time $t+1$. An example DBN modeling how the weather of an arbitrary city varies from day to day is shown in Figure 3-1.

A DBN model of the behavior of a stochastic system provides a compact representation of the system's dynamics. Several useful attributes of the system's behavior are explicitly represented by the DBN structure. In general, an edge in the network that connects a node $N_i$, which represents the variable $X_i$ at time $t$, to a node $N_j'$, which represents the variable $X_j$ at time $t+1$, can be viewed as representing a direct dependence between variable $X_i$ and variable $X_j$, which is often related to causality. Moreover, when we are interested in the distributions of the variables $X[t+1]$ given
values for the variables at time $t$, $X[t]$, a particular variable $X_i[t+1]$ is dependent only on variables $X_j[t]$ for which $N_i$ is an ancestor of $N_i'$ in the network. Finally, the steady-state distribution of each variable is readily available from the DBN. Each node $N_i$ has no parents, and therefore the associated conditional probability table has only one row. This row contains the steady-state distribution for the variable $X_i$.

### 3.1 Learning a Dynamic Bayesian Network from Data

In this section I discuss how to choose a DBN network structure and parameters to create a model that represents the behavior of a world well under the assumptions that the world is time-invariant and satisfies the Markov property. Consider a dynamic, stochastic
world whose behavior we would like to model. The world has a (perhaps empty) set of actions, \( A \), and a set of binary observations, \( O \). During each time unit, the agent may take one action from the set \( A \), and will then receive a set of observations from the world.

A set of random variables for the DBN can be constructed as \( X = A' \cup O \), where \( A' \) is a set of binary variables, of which, only the one corresponding to the action taken at this time step is \emph{true}. Each variable in \( A' \) will be represented by only one node in the DBN, because while the action taken influences the system, we are not interested in trying to predict which action will be taken in the future.

I assume that in order to learn a model for the behavior of the system, we are given a data set consisting of a sequence of \( T \) observations, \( \{o[1], o[2], \ldots, o[T]\} \), as well as a sequence of \( T-1 \) actions, \( \{a[1], a[2], \ldots, a[T-1]\} \), that the agent took when generating the observation sequence. We can then combine these two sets into a set of \( T-1 \) training samples of the form \( \{o[t], a'[t], o[t+1]\} \), which are drawn from the distribution associated with the DBN. This data set allows us to learn a DBN using the methods described above for generic Bayesian networks under the restriction that the network remain a DBN (i.e., that no edges are created from a node at time \( t+1 \) to a node at time \( t \)). Notice, however, that in learning a Bayesian network from a data set, we assumed that each sample in the data set was independent. This assumption is violated in the case of learning a DBN, and this can present a problem. For example, if some aspects of the world were not "explored" during the sequence of actions from which the training data is generated, the model learned will certainly be weak in modeling those aspects of the world.
3.2 Improving DBN Learning in Worlds

The choice of the initial starting point for the network search algorithm still remains. In choosing this starting point, we can take advantage of our knowledge that we are learning the dynamics of a world. We can first assume that there is correlation between two successive values of any given observation. To incorporate this into the structure starting point, we connect an edge from the node representing each observation at time $t$ to the node representing the same observation at time $t+1$. We can further assume that the actions taken directly influence the observations. In light of this, we can add an edge from the node representing each action at time $t$ to the node representing each observation at time $t+1$. This initialization, which is illustrated in Figure 3-2, is then used as a starting location for the network structural search algorithm.
Chapter 4

Learning Hidden State

Until this point we have assumed that the dynamic systems that we are modeling satisfy the Markov property. That is, we have assumed that given the observations from time t and the action taken at time t, the state of the world at time t+1 is independent of all previous observations and actions. In many real-world domains this is not the case, and this assumption limits the accuracy of the model that is learned.

There are two ways in which common real-world domains exhibit non-Markovian dynamics. The first is the simple case where past observations or actions affect the current state in ways that are not visible in present observations. Therefore, in order to accurately predict the future, the model needs memory of the past. In some worlds, however, simply adding memory is not enough, because an underlying state of the world may be non-deterministic, even given perfect memory of the past. There may be an underlying, unobservable state of the world, of which some observations provide imperfect information. In this case, the model needs to piece together this imperfect information as it arrives in order to best understand the current state of the world. In order to allow a DBN to capture these non-Markovian dynamics, we can add hidden variables.
4.1 Learning DBN Parameters in the Presence of Hidden Variables

This section addresses the task of learning the parameters, $\Theta$, for a given network structure in the case of a DBN with hidden variables. The DBN represents a set of variables, $X = O \cup H$, where $O$ is the set of observable variables and $H$ is the set of hidden variables. I assume that we have a set of training data in the form of a sequence of $T$ observations of the observable variables, $\{o[1], o[2], ..., o[T]\}$. If all the variables were observed, that is, if we also had observations for the variables in $H$, choosing the set of model parameters would follow the closed form solution presented in section 2.2.1. In the presence of hidden variables, however, there is no closed form solution and a gradient ascent algorithm must be used. The standard choice is the Expectation Maximization (EM) algorithm.

4.1.1 The EM Algorithm for Choosing Model Parameters

Beginning with an initial set of parameters, each iteration of the parametric EM algorithm takes two steps: the E-step and the M-step. During the E-step, expected statistics are calculated under the assumption that the current model parameters are correct. The M-step then uses the expected statistics to recalculate the values of the model parameters according to the closed form equation. Each iteration of the EM algorithm is guaranteed to improve $Pr(O \mid G, \Theta)$, the likelihood of the observations given the graph structure and parameters. As with all applications of the EM algorithm, the starting point is very
important, and best results are obtained by taking the best of several runs of the algorithm, each using a different initial parameter setting.

4.1.2 The Forward-Backward Algorithm for Inference in the E-step

During the E-step, expected sufficient statistics for every model parameter are calculated. An expected statistic for a model parameter, $\Theta_{X_{ij}}$, is the expected number of times in the data that $X_i$ took a certain value, $val$, while $Pa(X_i)$ took configuration $c$. We can calculate this expected statistic by summing over the probability that this event occurred at each time as follows:

$$E[\text{Number of times } X_i = val \text{ and } Pa(X_i) = c] = \sum \text{Pr}(X_i[t] = val, Pa(X_i)[t] = c).$$

As some of the variables involved in this probability may be hidden variables, we need to perform inference in order to calculate this probability. Inference in Bayesian networks is a difficult problem in general, and it is seemingly made even more difficult in the case of DBN's because all of the hidden nodes are interdependent. That is, given that the data is composed of a sequence of $T$ steps, we can imagine “unrolling” the DBN into one large Bayesian network with $T(|O|+|H|)$ nodes as illustrated in Figure 4-1. In general, any two hidden variables in any two time slices will not be independent given the observed data. We can, however, take advantage of the structure of this unrolled DBN to improve the efficiency of inference. This unrolled DBN is very similar to a factorial hidden Markov model, and as such we can use the forward-backward algorithm (Ghahramani and Jordan, 1997).
The forward backward algorithm calculates the following alpha and beta terms for each configuration, $c$, of the hidden variables at each time step, $t$ (Rabiner, 1989):

\[
\alpha_t[c] \equiv \Pr(O[1], O[2], ..., O[t], H[t] = c)
\]

\[
\beta_t[c] \equiv \Pr(O[t+1], O[t+2], ..., O[T] | H[t] = c, O[t]).
\]

The alpha and beta terms are calculated recursively using the following two equations (Rabiner, 1989):

\[
\alpha_t[c] = \sum_d \alpha_{t+1}[d] \Pr(H[t] = c, O[t] | O[t-1], H[t-1] = d)
\]

\[
\beta_t[c] = \sum_d \beta_{t+1}[d] \Pr(H[t+1] = d, O[t+1] | H[t] = c, O[t]).
\]
The alpha and beta terms are then combined to get the following distributions over hidden state transitions (Rabiner, 1989):

$$\xi_{t}[c,d] = \frac{\Pr(H[t] = c, H[t+1] = d | O[1], O[2], ..., O[T])}{\Pr(O[1], O[2], ..., O[T])}$$

$$= \frac{\Pr(H[t] = c, H[t+1] = d, O[1], O[2], ..., O[T])}{\Pr(O[1], O[2], ..., O[T])}$$

$$= \sum_c \alpha_{t}[c] \beta_{t+1}[d] \Pr(H[t+1] = d, O[t+1] | H[t] = c, O[t]) .$$

We wish to calculate the probability that $X_i$ and its parents took a specific configuration, $Pr(X_i[t] = val, Pa(X_i)[t] = c)$, at each time $t$. The variables involved in this probability can be divided into the sets $O_1, H_1, O_2,$ and $H_2$, where $O_1$ and $H_1$ are the observable and hidden variables, respectively, whose value at time $t-1$ is involved in this probability, and $O_2$ and $H_2$ are the observable and hidden variables whose value at time $t$ is involved in this probability. Using the values specified by the configuration $c$ and the value $val$ we can construct $o_1, h_1, o_2,$ and $h_2$, the respective configurations of each of the variables specified in this probability. We can then formulate this probability using the transition probabilities as follows:

$$Pr(X_i[t] = val, Pa(X_i)[t] = c)$$

$$= Pr(O_1[t-1] = o_1, H_1[t-1] = h_1, O_2[t] = o_2, H_2[t] = h_2)$$

$$= \sum_o [o_1] [O_2[t] = o_2] \sum_a [H_1(a) = h_1] \sum_b [H_2(b) = h_2] \xi_{t+1}[a,b] ,$$

where $[S[t] = s]$ is 1 if the variables in set $S$ take the values specified by $s$, and 0 otherwise, and $T(c)=t$ means that the hidden variables specified in set $T$ take the values specified by $t$ in configuration $c$. 

27
Summing over the probability $\Pr(X_i[t] = \text{val}, \Pa(X_i)[t] = c)$ at each time step, we arrive at the expected sufficient statistics associated with the model parameter $\Theta_{Xik}$. These expected sufficient statistics are then used in the M-step to update the model parameters according to the closed form equation presented in section 2.2.1.

### 4.1.3 Complexity of Calculating Expected Statistics

The time and space requirements of calculating the expected statistics are unfortunately exponential in the number of hidden variables, and this will place restrictions on the number of hidden variables that can be incorporated into a model. There are $2^H$ configurations of the hidden variables, where $H$ represents the number of hidden variables in the network. The forward-backward algorithm calculates a transition probability term, $\xi_t[c][d]$ for each combination of configurations at each time step. Storing these transition probabilities therefore requires $O(T2^H)$ space.

The alpha and beta terms are calculated recursively using the following two equations:

$$\alpha_t[c] = \sum_d \alpha_{t+1}[d] \Pr(H[t] = c, O[t] | O[t-1], H[t-1] = d)$$

$$\beta_t[c] = \sum_d \beta_{t+1}[d] \Pr(H[t+1] = d, O[t+1] | H[t] = c, O[t]).$$

Note that the probability term in both of these equations is directly computable from the conditional probability tables of the DBN. Calculating each alpha and beta term requires summing over $2^H$ products, each of which requires exactly one lookup in the conditional probability table of each node in the network representing a variable at time $t+1$. 
Looking up an entry in a node’s conditional probability table requires time proportional to the number of parents of that node. Putting these terms together, we find that calculating all the alpha and beta terms requires $O(TNP^{2H})$ operations, where $N$ is the total number (observable and hidden) of nodes in the network and $P$ is the maximum number of parents of any node.

The transition probability terms are calculated according to the following equation:

$$
\xi_t[c,d] = \frac{\alpha_t[c] \beta_{t+1}[d] \Pr(H[t+1] = d, O[t+1] \mid H[t] = c, O[t])}{\sum_c \alpha_t[c] \beta_t[c]}
$$

The denominator of this equation is the likelihood of the observations given the model. This need only be calculated once, and calculating it requires $O(2^H)$ operations. The probability term in the numerator can be directly calculated from the node conditional probability tables of the DBN, and hence requires $O(NP)$ operations. There are $O(T2^H)$ transition probability terms, and calculating all of them therefore requires $O(TNP^{2H})$ operations.

Calculating a single expected statistic requires summing over $O(2^H)$ transition probabilities and examining $O(P)$ observations at each time step, and hence requires $O(T(2^H + P))$ operations. There are $O(N2^p)$ parameters associated with a network, each of which has two associated sufficient statistics. Therefore, using the transition probabilities to calculate all of the expected statistics requires $O(TN2^p(2^H + P))$ operations. This is hence the most computationally expensive phase of calculating the expected statistics. Calculating the expected statistics of a model given a set of observations therefore requires $O(TN2^p(2^H + P))$ operations and $O(T2^H + N2^p)$ space.
4.2 Choosing Between DBN Structures in the Presence of Hidden Nodes

This section discusses how to choose a DBN graph structure in the presence of hidden nodes. As in the case of choosing a graph structure of a Bayesian network without hidden nodes, we would like to maximize the following score function:

$$\text{Score}(G, \Theta : O) = \log \Pr(O \mid G, \Theta) - \text{Penalty}(G).$$

If the network contains hidden variables, calculating the log probability term of this score requires summing over all possible values that the hidden variables may take, and this requires a distribution over the hidden variables. We can use a second (possibly the same) model to generate this distribution over the hidden variables, and hence we arrive at the expected score of a model, \((G, \Theta)\), with respect to a second model \((G', \Theta')\) as presented by Friedman (1997):

$$\text{Score}(G, \Theta : O, G', \Theta') = \mathbb{E}[\log \Pr(O, H \mid G, \Theta) - \text{Penalty}(G)],$$

where the expectation is taken over \(H\) with respect to \(\Pr(H \mid O, G', \Theta')\). We would like to find a model, \((G^*, \Theta^*)\), which maximizes \(\text{Score}(G^*, \Theta^* : O, G^*, \Theta^*)\). Given that we can use the parametric EM algorithm as described in section 4.1 to compute parameters for a DBN with a given structure, a naive approach would be to proceed with a gradient ascent method similar to the method for learning structure in the case of a Bayesian network without hidden nodes. Given an initial network structure, we could examine each neighboring structure, each time performing the parametric EM algorithm to choose appropriate network parameters. This method, however, is not practical because of the
computational complexity of the EM algorithm. Recall that using the forward-backward algorithm to calculate expected sufficient statistics during the E-step requires \(O(TN^2P(2^H + P))\) operations. Running the EM algorithm to choose model parameters requires running both the E-step and the M-step several times, and doing this each time a neighboring structure is examined is extremely inefficient.

As an alternative Friedman (1998) presents two algorithms (MS-EM and AMS-EM) which both use the innovative idea of alternating between parameter and structure steps. A parameter step makes one iteration of the parametric EM algorithm as presented in section 4.1 to improve the parameters associated with the current model. A structure step chooses a neighboring structure with a higher score using expected statistics generated from the current model. Friedman shows that these two algorithms are guaranteed to monotonically increase the score function. Combining these two algorithms, we get the generic STRUCTURAL-SEARCH-EM algorithm shown in Figure 4-2.

### 4.3 Discovering Hidden State

We now have an algorithm for choosing the structure and parameters of a model with hidden nodes. We still need a method for deciding when hidden nodes need to be added to the network. Moreover, as the success of STRUCTURAL-SEARCH-EM algorithm, like any EM algorithm, is highly dependent upon the initial starting point, we need a method for deciding how to initially incorporate new hidden nodes into the network.
4.3.1 Using a k-Slice DBN to Detect Hidden State

As discussed earlier, the need for hidden nodes arises when the system to be modeled violates the Markov property. To detect violation of the Markov property, Boyen et al. (1999) propose learning the edges for a k-slice DBN, and then using non-Markovian edges to detect the need for hidden state. Whereas a standard DBN contains nodes representing each variable at two consecutive time slices, a k-slice DBN represents each variable at k consecutive time slices. Similar to a standard DBN, only edges incident on the last time slice are allowed. After training a k-slice DBN, all non-Markovian edges can be replaced with hidden nodes that "remember" the value of past variables. An edge that connects a node in time slice t-d to a node in slice t+1 requires the addition of d hidden nodes. Each hidden node is connected to the previous hidden node and also to the
Figure 4-3: Converting a k-slice DBN to a DBN with hidden nodes. The non-Markovian edge connecting observation 2 at time t-1 to observation 1 at time t+1 is replaced by hidden variable 1. Similarly, the non-Markovian edge from observation 2 at time t-2 to observation 2 at time t+1 is replaced by hidden nodes 2 and 3.

An example of a conversion from a k-slice DBN to a DBN with hidden nodes is illustrated in Figure 4-3. The conditional probability table of each hidden node is initialized to make it "remember" the previous variable, with noise biased towards maintaining its own state. Once the k-slice DBN has been converted into a standard DBN with hidden variables in this manner, the STRUCTURAL-SEARCH-EM algorithm is used to further improve the model. This algorithm for the discovery of
hidden state, which I refer to as the K-SLICE-DBN-HIDDEN-STATE-DISCOVERY algorithm, is listed in Figure 4-4.

### 4.3.2 Improving Hidden State Detection in Worlds

In the case of learning models of world dynamics, we can attempt to take advantage of our goals to improve the efficiency and success of our search. Specifically, we are most concerned with and would like the highest level of accuracy in the model’s ability to predict the reward observations. I propose the following strategy for incorporating hidden nodes to improve reward observations. We begin by learning a DBN without hidden nodes to model the world. We then repeatedly add hidden variables for the purpose of improving the model’s ability to accurately predict the reward observations.
The **STRUCTURAL-SEARCH-EM** algorithm is called to incorporate each new hidden variable into the network as a cause of a reward that we would like to better model. If the **STRUCTURAL-SEARCH-EM** algorithm fails in incorporating the hidden variable, this is an indication that there is no more hidden state associated with this reward.

The key to the success of this approach lies in how the two nodes representing the new hidden variable are initially incorporated into the network, as the success of the **STRUCTURAL-SEARCH-EM** algorithm will depend heavily upon the initial starting point. Let $h$ and $h'$ be the nodes in the DBN representing the new hidden variable at time $t$ and time $t+1$ respectively. Similarly, let $r$ and $r'$ be the nodes associated with the reward that we are trying to better model. We will want an edge that connects $h$ to $r'$, so that the new hidden variable can have a direct effect on the reward. If the hidden variable is to maintain a hidden state, there will also need to be an edge from node $h$ to node $h'$. These two edges should be considered necessary, and the **STRUCTURAL-SEARCH-EM** algorithm should not be allowed to remove them.

The hidden state to be represented by this newly added hidden variable is likely affected by observations or actions in the world. To capture this, we will want edges in the network incident on the node $h'$. Though the proper choice of these edges will need to be determined by the **STRUCTURAL-SEARCH-EM** algorithm, we can attempt to provide a starting point that facilitates their discovery. It is likely that actions taken by the agent affect the hidden state, and in light of this, we connect each action node to the node $h'$. We can gain further clues about the cause of this hidden state from our current model of the world. We know that the parents of the node $r'$ influence the value of the reward, and in light of this, they may be related to the hidden state as well. As such, we add an edge.
Figure 4-5: Initial configuration when adding a hidden variable to improve the model of a reward.

connecting each parent of the node $r'$ to the node $h'$. An example of this initial incorporation of a new hidden variable is illustrated in Figure 4-5.

To finish the initial incorporation of the new hidden variable into the DBN, the network parameters must be set. The parents of the node $r'$ have changed, and hence this node needs a new conditional probability table. Nodes $h$ and $h'$, which have just been created, also need appropriate conditional probability parameters. As with any EM algorithm, a random initialization is a good choice, though in order to facilitate the new hidden variable being used to represent a hidden state, the parameters of $h'$ can be biased towards maintaining the value of $h$.

When the STRUCTURAL-SEARCH-EM algorithm terminates, we can examine the resulting graph structure to determine if the new hidden variable has been successfully incorporated into the model. If the new hidden variable has not been successfully incorporated into the network in a way that is meaningfully improving the model, the penalizing term in the score function will result in as many edges incident on $h'$ being
WORLD-HIDDEN-STATE-DISCOVERY (Graph G, Parameters Θ, Observations O)

1  rewardsToImprove ← all rewards in the world
2  while rewardsToImprove is not empty
3      reward ← choose reward from rewardsToImprove which minimizes
4           \log \Pr(\text{reward data} | G, Θ, O)
5      (G', Θ') ← copy of (G, Θ)
6      G' ← add hidden nodes h and h' to G'
7      r ← node representing reward at time t in G'
8      r' ← node representing reward at time t+1 in G'
9      G' ← add edge in G' from r to h'
10     for a ← each action
11        G' ← add edge in G' from a to h'
12     for p ← each parent of r'
13        G' ← add edge in G' from p to h'
14     Θ'(r) ← choose randomly
15     Θ'(h) ← choose randomly
16     Θ'(h') ← choose randomly biased towards h
17     ops ← addition or removal of edges incident on h, h', r, or r'
18     (G', Θ') ← STRUCTURAL-SEARCH-EM (G, Θ, O, ops)
19     if node h' has more than one parent in G' or
20        \Pr(O | G', Θ') > (1+ε) \Pr(O | G, Θ)
21        G ← G'
22     Θ ← Θ'
23     numHiddenNodes[reward] ← numHiddenNodes[reward] + 1
24     if numHiddenNodes[reward] >= maxHiddenNodes[reward]
25        remove reward from rewardsToImprove
26   else
27      remove reward from rewardsToImprove
28  return (G, Θ)

Figure 4-6: The WORLD-HIDDEN-STATE-DISCOVERY algorithm. Hidden nodes are added in an attempt to improve the model of reward observations.

removed as possible. We have mandated that the edge from node h to h' not be removed by the STRUCTURAL-SEARCH-EM algorithm, but if h' has any parents besides h when the algorithm terminates, we can conclude that the hidden variable has been successfully incorporated into the model. There are scenarios in which successful incorporation of the hidden variable into the model does not mandate that the node h' have other parents. This scenario can occur in a world that has underlying hidden state of which observations
provide imperfect information (e.g. the Slippery Driving World presented in Section 5.4). In light of this, we also check for a significant increase in the log likelihood of the data and interpret this as a sign that the new hidden variable is improving the model. This strategy for the discovery and incorporation of hidden state, which I refer to as the WORLD-HIDDEN-STATE-DISCOVERY algorithm, is listed algorithmically in Figure 4-6.
Chapter 5

Worlds

This chapter describes the worlds used in experimentation in this project. Appendix A summarizes this information in table form.

5.1 The Driving World

The Driving World is a simple simulation of highway driving. The agent's car moves down a highway at a constant speed and has the ability to change lanes. The agent's "goal" is to avoid colliding with other cars on the road. The idea for this world is taken from McCallum (1996).

The agent's car moves down a three-lane highway at a constant speed of 25 m/s. The agent shares the highway with other vehicles, each of which is either moving at a speed of 20 m/s or 30 m/s. These other vehicles travel straight down the highway at a constant speed and without changing lanes. Movement is simulated in discrete time steps at the rate of one step per second. At each time step the agent may change one lane in either direction or remain in the same lane. If a faster vehicle collides with the agent's car, the vehicle will drive over the agent's car causing damage, and the agent will receive a penalty. Similarly, if the agent's car collides with a slower vehicle, the agent will drive over the other vehicle and will receive a penalty for causing damage. If the agent is in the leftmost lane and attempts to switch lanes further to the left, or is in the rightmost lane
and attempts to switch lanes to the right, the tires of the agent's car will scrape against the curb, and the agent will receive a penalty.

Apart from the three penalties mentioned above, the world provides observations telling whether the agent is in the leftmost or rightmost lane and telling whether there is a vehicle directly to the left or directly to the right of the agent's car. The world also provides six sets of observations providing information about the nearest car to the front and the back in three lanes: the lane the agent's car is in, the lane to the left of the agent's car, and the lane to the right of the agent's car. For each of these six cars, the agent receives a single bit observation indicating whether the car is a faster car or a slower car, and a five bit distance-thermometer. The distance-thermometer measures the distance (parallel to the road) from the agent's car to the vehicle on a logarithmic scale. If there is no vehicle present, all five bits of the thermometer are true. If the vehicle is over forty meters away, the first four bits of the thermometer are true and the last is false. If the vehicle is between twenty and forty meters away, the first three bits are true and the remaining two are false. If the vehicle is between ten and fifteen meters away, the first two bits are true and the remaining three are false. If the vehicle is ten meters away, the first bit is true and the remaining four are false, and finally if the vehicle is next to or on top of the agent's car, all five bits are false. The readings received from the distance-thermometer are illustrated in Figure 5-1.
Distance from the agent's car to the vehicle being measured by the thermometer

| Distance from the agent's car to the vehicle being measured by the thermometer | Values of the distance-thermometer bits |
|---|---|---|---|---|---|
| Bit 1 | Bit 2 | Bit 3 | Bit 4 | Bit 5 |
| No vehicle in sight | *true* | *true* | *true* | *true* | *true* |
| Distance between 45 and 100 meters | *true* | *true* | *true* | *true* | *false* |
| Distance between 20 and 40 meters | *true* | *true* | *true* | *false* | *false* |
| Distance between 10 and 15 meters | *true* | *true* | *false* | *false* | *false* |
| Distance exactly 5 meters | *false* | *false* | *false* | *false* | *false* |
| Distance exactly zero | *false* | *false* | *false* | *false* | *false* |

Figure 5-1: The distance-thermometer of the Driving World and Driving World with Signaling and Slipping.

### 5.2 The Left Signaling World

The Left Signaling World provides an extremely simple example of the need for hidden state to act as memory. The agent is driving a car down a road and has the ability to change lanes to the left or right. The car also has a turning signal that can be turned on and off, however the agent has no feedback as to the current state of the turning signal – it must be remembered. The five actions in the world are "Change lanes to the left", "Change lanes to the right", "Signal left", "Turn signal off", and "Do nothing". The world has one observation, which is a penalty for changing lanes to the left without the turning signal being turned on.

### 5.3 The Signaling World

The Signaling World is similar to the Left Signaling World except that the turning signal must be used for changing lanes in both directions. The turning signal has three
positions: left, right, and off. The world has one observation, which is a penalty for
turning without the turning signal being in the correct position. Because the turning
signal has three states, two binary hidden variables are needed to fully capture the
dynamics of the turning signal.

5.4 The Slippery Driving World

The Slippery Driving World is a very simple world designed to illustrate the need for
hidden state in the model to collect imperfect information regarding an underlying hidden
state of the world. The agent drives a car down a road and has the ability (among other
actions that do nothing) to change lanes to the left. The agent's dilapidated car has
trouble turning and the tires often screech when the agent changes lanes. The tires
screeching bothers the agent and other drivers, and as such a penalty is received when the
tires screech. What is not apparent to the agent is that sections of the road he is driving
over may be slightly wet, and when driving on wet sections of road the agent's tires are
more likely to screech.

Specifically, the world dynamics are composed of two parameters, α and β. The
first parameter, α, is a small fraction representing the probability that the wetness of the
road will change at any given instant. The second parameter, β, specifies the accuracy of
the information the agent receives. If the road is wet when the agent changes lanes, the
tires will screech with probability β. If the road is not wet when the agent changes lanes,
the tires will screech with probability 1-β.
5.5 The Driving World with Signaling and Slipping

The Driving World with Signaling and Slipping combines the Driving World with the Signaling World and the Slippery Driving World. All the actions, observations, and penalties from the Driving World are present in the Driving World with Signaling and Slipping. Moreover, three signaling actions, "Signal Left", "Signal Right", and "Turn off signal" are added, as is a penalty for changing lanes without properly signaling, and a penalty for the tires screeching.
Chapter 6

Implementation

This chapter presents the software implementation of this project. The first section presents the major classes and their relationships, and the second section discusses the modularity of the learning implementation.

6.1 Class Overview

This project was implemented in the Java™ language. The major classes in the implementation and their relationships are illustrated in Figure 6-1. The BayesNet class represents a single Bayesian network, and similarly the DynamicBayesNet class, which extends the BayesNet class, represents a single dynamic Bayesian network. The StatisticGenerator abstract class represents the sufficient statistics or expected sufficient statistics that can be used to score a network or to set network conditional probability parameters. The DataStatisticGenerator and DBNStatisticGenerator classes both extend the StatisticGenerator class and provide the statistics associated with completely observed data or observed data with hidden variable values filled in through inference, respectively. The Scorer abstract class provides an interface for scoring a network. The Scorer class is extended by the LogLikelihoodScorer class which scores according to the log likelihood of the data given the model, and by the MDLScorer class which scores according to the BIC score (and MDL approximation) described in section 2.2.2.2. The
Figure 6-1: The major classes of the implementation and their relationships.

StructureSearcher abstract class is an interface for any model searching algorithm. This abstract class is extended by the GreedyStructureSearcher class, the ParamSelectionEM class, and the ModelSelectionEM class. The GreedyStructureSearcher class uses the GREEDY-SEARCH algorithm (Figure 2-2) to search over the space of network structures when there are no hidden nodes. The ParamSelectionEM class uses the parametric EM algorithm to improve the conditional probability parameters of a model with a fixed
6.2 Modularity

A major goal for the architecture used in the implementation of this project is strong modularity among different elements of the learning system. The architecture of the learning implementation is illustrated in Figure 6-2. The modularity of this...
implementation allows easy mixing and matching of different search algorithms and scoring metrics. Any scoring metric that adheres to the decomposition property described in section 2.2.2.2 can be implemented in a class that extends the Scorer abstract class, and then can immediately be used by any model searching algorithm without further adaptation. Similarly, any algorithm for model selection can be implemented in a class that extends the StructureSearcher abstract class, and then can be immediately used with existing scoring metrics.

Scoring metrics and model selection algorithms do not examine the data directly, but instead query sufficient statistics from a StatisticGenerator class. The statistics returned may be exact statistics as counted from the data or may be the expected statistics generated through inference in the case of hidden variables. This allows networks with hidden variables to be seamlessly plugged into any existing scoring criterion, and allows for easy adaptation of structural searching algorithms for networks without hidden variables to structural searching algorithms for networks with hidden variables.
Chapter 7

Results and Evaluation

This chapter discusses experiments that were run for the purpose of evaluating the use of dynamic Bayesian networks to model world dynamics. The first section examines the use of DBN's without hidden nodes, and the second section examines the use of hidden nodes to discover hidden state.

7.1 Learning Models Without Hidden Nodes

This section discusses the results of learning a DBN without hidden nodes to model the Driving World. The GREEDY-SEARCH algorithm was used to learn a DBN using varying amounts of training data. During each run the training data was generated by first initializing the world to a random state and then randomly selecting actions. After a model was learned, twenty thousand independently drawn steps of test data were used to test the accuracy of the model. The negative log likelihood of the test data, given the model and the actions used to generate the test data, was used to measure the accuracy of the model. This negative log likelihood was divided by the number of time steps and the number of observations in the world to arrive at the average negative log likelihood of predicting each individual observation. As a point of comparison, a naïve model that predicts each observation to be equally likely to be true or false would produce an average negative log likelihood of 0.301.
This experiment was performed under two settings of the background knowledge. The first setting is a uniform prior distribution over all model parameters. This results in "equivalent" prior counts of $T = 1$ and $F = 1$. The second setting of the background knowledge uses the assumption that elements of a world are more likely to stay the same at any given instant than they are to change. To incorporate this knowledge, the "equivalent" prior count for an observation value remaining the same at consecutive time steps was set to 3, and the prior count for the value changing was set to 1. Each of these experiments was performed ten times. The average results are illustrated in Figure 7-1.

Biasing the prior distribution over model parameters to favor observations staying the same perhaps helped when the number of training steps was very few, however as the amount of training data increased, the bias quickly became irrelevant. There are nearly ten million possible configurations of the observations and the action taken during one step in the Driving World. Nevertheless, with only a few thousand steps of training data,
models are learned for which the negative log likelihood of the test data is substantially lower than the likelihood of the test data given a naïve model (0.301), and by this criterion at least, the learned models presumably model the world reasonably well. This demonstrates that the DBN learning is effectively detecting patterns in the data and using them to make valid generalizations.

7.2 Discovering and Learning Hidden State

This section examines the use of hidden nodes to improve a DBN model by comparing DBN's learned with and without hidden nodes. Models learned by the GREEDY-SEARCH algorithm without the use of hidden nodes, models learned by the K-SLICE-DBN-HIDDEN-STATE-DISCOVERY algorithm, and models learned by WORLD-HIDDEN-STATE-DISCOVERY algorithm were evaluated on several worlds. In each case, the following experiment was run: A model is learned using training data generated by taking five thousand steps in the world, with actions selected randomly. This model is then used to predict the likelihood of twenty thousand independently drawn test steps. The average negative log likelihood of each observation in the test steps, given the model and the actions taken to generate the test steps, is used to measure the accuracy of the model. In these experiments a 4-slice DBN was used by the K-SLICE-DBN-HIDDEN-STATE-DISCOVERY algorithm.
Average negative log likelihood of each observation of 20,000 test steps after being trained on 5,000 training steps in the Left Signaling World. Each experiment was performed ten times.

<table>
<thead>
<tr>
<th>Trained using the GREEDY-SEARCH algorithm to learn a DBN without hidden nodes</th>
<th>Trained using the K-SLICE-DBN-HIDDEN-STATE-DISCOVERY algorithm with a 4-Slice DBN</th>
<th>Trained using the WORLD-HIDDEN-STATE-DISCOVERY algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>5.394 E-2</td>
<td>3.870 E-2</td>
</tr>
<tr>
<td>Minimum</td>
<td>5.320 E-2</td>
<td>2.710 E-2</td>
</tr>
<tr>
<td>Maximum</td>
<td>5.571 E-2</td>
<td>6.286 E-2</td>
</tr>
</tbody>
</table>

Figure 7-2: Comparison of learning strategies in the Left Signaling World.

Average negative log likelihood of each observation of 20,000 test steps after being trained on 5,000 training steps in the Signaling World. Each experiment was performed ten times.

<table>
<thead>
<tr>
<th>Trained using the GREEDY-SEARCH algorithm to learn a DBN without hidden nodes</th>
<th>Trained using the K-SLICE-DBN-HIDDEN-STATE-DISCOVERY algorithm with a 4-Slice DBN</th>
<th>Trained using the WORLD-HIDDEN-STATE-DISCOVERY algorithm</th>
</tr>
</thead>
</table>

Figure 7-3: Comparison of learning strategies in the Signaling World.

7.2.1 The Signaling Worlds

The Left Signaling World and the Signaling World provide trivial examples of the need for hidden state to act as memory. The last action taken involving the turning signal must be remembered in order to know whether changing lanes will result in a penalty being received. The parameters of the WORLD-HIDDEN-STATE-DISCOVERY algorithm were set such that the algorithm looks for at most one hidden node in the Left Signaling World and at most two in the Signaling World. The results of experimentation in these worlds
Figure 7-4: A DBN without hidden nodes modeling the Left Signaling World. This is the best structure without hidden nodes for modeling this world.

Figure 7-5: A model of the Left Signaling World. The hidden variable in this model serves to remember whether the signal was turned to the left two steps ago. This is not enough to fully capture the dynamics of the turning signal.
are shown in Figure 7-2 and Figure 7-3. As measured by the likelihood of the test data, both of the algorithms for learning models with hidden nodes decisively outperformed the algorithm learning models without hidden nodes. Furthermore, the WORLD-HIDDEN-STATE-DISCOVERY algorithm significantly outperformed the K-SLICE-DBN-HIDDEN-STATE-DISCOVERY algorithm.

Figure 7-4 shows the DBN structure modeling the Left Signaling World that was chosen by each run of the algorithm not using hidden nodes. This structure is in fact the best structure without hidden nodes for modeling this world. This same structure was also chosen by three of the ten runs of the K-SLICE-DBN-HIDDEN-STATE-DISCOVERY algorithm. Figure 7-5 shows a structure representative of the models chosen by the remaining seven runs of the K-SLICE-DBN-HIDDEN-STATE-DISCOVERY algorithm. The hidden variable in this model serves to remember whether the left turning signal was turned on two time steps ago. Though this is not enough to accurately determine the state of the turning signal, perhaps this provides more information than having no hidden variable at all. Figure 7-6 shows a structure representative of the model chosen during each of the ten runs of the WORLD-HIDDEN-STATE-DISCOVERY algorithm. The hidden variable in this model maintains the state of the turning signal, and hence this structure fully captures the underlying dynamics of the world.

Figure 7-7 shows the DBN structure modeling the Signaling World that was chosen by each run of the algorithm not using hidden nodes. This is again the best structure without hidden nodes for modeling this world. This same structure was also generated during six runs of the K-SLICE-DBN-HIDDEN-STATE-DISCOVERY algorithm. The remaining four runs of the K-SLICE-DBN-HIDDEN-STATE-DISCOVERY algorithm
Figure 7-6: An model of the Left Signaling World. The hidden variable represents the current state of the turning signal (on or off). This structure is optimal in that it fully captures the dynamics of the turning signal.

Figure 7-7: A model of the Signaling World without hidden variables. This is the best structure without hidden variables for modeling this world.
Figure 7-8: A model of the Signaling World. This structure cannot fully capture the dynamics of the turning signal.

Learning models of the Signaling World generated models similar to that illustrated in Figure 7-8. Though this structure has two hidden variables, they do not fully capture the state of the turning signal.

Six runs of the WORLD-HIDDEN-STATE-DISCOVERY algorithm on the Signaling World generated models similar to the one shown in Figure 7-9. The first hidden variable in this model indicates when the turning signal is to the right, and the second hidden variable determines whether it is to the left or off (when it is not to the right). Hence, this
structure can fully capture the dynamics of the turning signal. The models chosen by the remaining four runs of the WORLD-HIDDEN-STATE-DISCOVERY algorithm all had two hidden nodes, though they did not perfectly capture the dynamics of the turning signal. One such model is illustrated in Figure 7-10. This model has erroneous edges connecting the nodes associated with changing lanes to the hidden variables. It is also missing edges connecting the turn signal actions to the hidden variables.

Figure 7-9: A model of the Signaling World. This structure is an optimal structure in that it can fully capture the dynamics of the turning signal.
As exemplified by the structures in Figure 7-6 and Figure 7-9, The WORLD-HIDDEN-STATE-DISCOVERY algorithm often generated excellent models. Though Figure 7-10 illustrates a less successful model, perhaps this type of model would be generated less frequently if more training data were used. Recall that having more training data results in the relative weight of the penalizing term in the scoring function decreasing. Therefore, with more training data the algorithm may add the missing edge in Figure 7-10 from "Switch signal off" to the second hidden variable. Depending upon the
parameters, this would allow the second hidden variable to determine decisively whether or not the turning signal is in one specific position, for instance whether or not the turning signal is to the left. Adding this edge would presumably increase the log likelihood of the data given the model. With this edge added, another edge could be added from "Switch signal right" to the first hidden variable that would allow that hidden variable to determine whether the signal was last switched to the right or switched off. With the addition of this edge, the state of the turning signal at any time could be determined. Because they would no longer contribute anything, the erroneous edges connecting "Change lanes left" and "Change lanes right" to the first hidden variable would presumably be removed. Therefore, an increase in the amount of training data may result in models like that illustrated in Figure 7-10 being generated less frequently.

7.2.2 The Slippery Driving World

The Slippery Driving World presents a rather simple example of the need for hidden state in a model to gather and collect noisy information about an underlying hidden state in the world. There are no direct observations as to whether the section of road that the agent's car is currently driving on is wet, but noisy feedback is received each time the agent changes lanes because the car's tires are more likely to screech if the road is wet. Experimentation was done varying the accuracy of the information received when the agent changes lanes (the parameter \( \beta \)). The parameters of the WORLD-HIDDEN-STATE-DISCOVERY algorithm were set such that the algorithm looks for at most one hidden node.
Average negative log likelihood of each observation of 20,000 test steps after being trained on 5,000 training steps in the Slippery Driving World with $\alpha=0.01$ and $\beta=1.0$. Each experiment was performed ten times.

<table>
<thead>
<tr>
<th></th>
<th>Trained using the K-STATE-DISCOVERY algorithm to learn a DBN without hidden nodes</th>
<th>Trained using the K-STATE-DISCOVERY algorithm to learn a 4-Slice DBN</th>
<th>Trained using the K-STATE-DISCOVERY algorithm to learn a DBN with C=0.01</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>5.524 E-2</td>
<td>2.316 E-2</td>
<td>1.582 E-2</td>
</tr>
<tr>
<td>Minimum</td>
<td>5.321 E-2</td>
<td>1.426 E-2</td>
<td>1.434 E-2</td>
</tr>
</tbody>
</table>

Figure 7-11: Comparison of learning strategies in the Slippery Driving World with the accuracy parameter, $\beta$, set to 1.0.

Average negative log likelihood of each observation of 20,000 test steps after being trained on 5,000 training steps in the Slippery Driving World with $\alpha=0.01$ and $\beta=0.8$. Each experiment was performed ten times.

<table>
<thead>
<tr>
<th></th>
<th>Trained using the K-STATE-DISCOVERY algorithm to learn a DBN without hidden nodes</th>
<th>Trained using the K-STATE-DISCOVERY algorithm to learn a 4-Slice DBN</th>
<th>Trained using the K-STATE-DISCOVERY algorithm to learn a DBN with C=0.01</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>6.057 E-2</td>
<td>6.048 E-2</td>
<td>5.580 E-2</td>
</tr>
<tr>
<td>Minimum</td>
<td>5.918 E-2</td>
<td>5.991 E-2</td>
<td>5.128 E-2</td>
</tr>
</tbody>
</table>

Figure 7-12: Comparison of learning strategies in the Slippery Driving World with the accuracy parameter, $\beta$, set to 0.8.

Average negative log likelihood of each observation of 20,000 test steps after being trained on 5,000 training steps in the Slippery Driving World with $\alpha=0.01$ and $\beta=0.6$. Each experiment was performed ten times.

<table>
<thead>
<tr>
<th></th>
<th>Trained using the K-STATE-DISCOVERY algorithm to learn a DBN without hidden nodes</th>
<th>Trained using the K-STATE-DISCOVERY algorithm to learn a 4-Slice DBN</th>
<th>Trained using the K-STATE-DISCOVERY algorithm to learn a DBN with C=0.01</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum</td>
<td>5.933 E-2</td>
<td>5.919 E-2</td>
<td>5.946 E-2</td>
</tr>
</tbody>
</table>

Figure 7-13: Comparison of learning strategies in the Slippery Driving World with the accuracy parameter, $\beta$, set to 0.6.
Figure 7-14: A model of the Slippery Driving World without hidden nodes. This is the best structure for modeling this world.

Figure 7-11, Figure 7-12, and Figure 7-13 show the results of the three different learning algorithms with the accuracy parameter set to 1.0, 0.8, and 0.6 respectively.

The algorithm learning models without hidden nodes generated the same model during each run regardless of the accuracy parameter. The structure of this model is shown in Figure 7-14. This is indeed the best structure without hidden nodes for modeling this world.

The K-SLICE-DBN-HIDDEN-STATE-DISCOVERY algorithm performed relatively poorly in the Slippery Driving World as compared to the WORLD-HIDDEN-STATE-DISCOVERY algorithm. When the accuracy parameter was set to 1.0, perfect information about the wetness of the road is received each time the agent changes lanes. Because of
Figure 7-15: A model of the Slippery Driving World. This is an optimal structure for modeling this world.

This strong relationship with the past, the k-slice DBN learning phase of the algorithm often found non-Markovian edges. However, the STRUCTURAL-SEARCH-EM algorithm was often unable to incorporate the hidden nodes created in a way that accurately represented the underlying hidden state. When the accuracy was set to 0.8 or 0.6 the k-slice DBN learning phase failed to generate any non-Markovian edges, and hence the structure without hidden nodes illustrated in Figure 7-14 was generated by each run. The K-SLICE-DBN-HIDDEN-STATE-DISCOVERY algorithm was designed with the idea of adding hidden nodes to act as memory of past states. Whereas in the Signaling Worlds
Figure 7-16: A model of the Slippery Driving World. The edge from "Do nothing" to the hidden variable is erroneous.

memory is exactly what is needed, in the Slippery Driving world hidden state in the model is needed not to act as deterministic memory, but rather to gather imperfect information about an underlying state in the world. This may explain why this algorithm did relatively poorly at modeling this world.

The WORLD-HIDDEN-STATE-DISCOVERY algorithm clearly outperformed the other two algorithms when the accuracy parameter was set to 1.0. The algorithm generated the structure illustrated in Figure 7-15 during eight of the ten runs. The hidden node in this structure represents the wetness of the road, and this structure optimally captures the
dynamics of the world. In the remaining two runs the algorithm generated a structure similar to that shown in Figure 7-16. This structure comes close to capturing the dynamics of the world, though it has one erroneous edge.

As the accuracy parameter was turned down to 0.8 and 0.6 the log likelihood of the test data given the models created by the WORLD-HIDDEN-STATE-DISCOVERY algorithm sharply decreased. Despite this indication of failure, the optimal model in Figure 7-15 was generated during many of these runs, and near optimal models generated in the remaining runs. The structure illustrated in Figure 7-15 was generated during three of the runs when the accuracy parameter was set to 0.8 and nine of the runs when the parameter was set to 0.6. The remaining runs in each of these two cases all generated structures similar to that illustrated in Figure 7-16 or a similar structure with two erroneous edges. Though they have erroneous edges, these structures seem to model the underlying dynamics of the world much better than the structure without hidden nodes illustrated in Figure 7-14. Despite the apparent quality of the models generated, the log likelihood of the data given the models is only slightly better than that of the models without hidden nodes. Note, though, that when the accuracy parameter is low, little information about the state of the road is received when a lane change is made, and even if the state of the road is known, there is still high uncertainty about whether the tires will actually screech or not. I propose, then, that the sharp drop in the log likelihood of the data given the model as the accuracy parameter decreases is not an indication that the quality of the models learned by the WORLD-HIDDEN-STATE-DISCOVERY algorithm dropped sharply, but only an indication that the world became much harder to predict.
Average negative log likelihood of each observation of 20,000 test steps after being trained on 5,000 training steps in the Driving World with Signaling and Slipping with $\alpha=0.01$ and $\beta=0.8$. Each experiment was performed five times.

<table>
<thead>
<tr>
<th></th>
<th>Trained using the GREEDY-SEARCH algorithm to learn a DBN without hidden nodes</th>
<th>Trained using the K-SLICE-DBN-HIDDEN-STATE-DISCOVERY algorithm with a 4-Slice DBN</th>
<th>Trained using the WORLD-HIDDEN-STATE-DISCOVERY algorithm</th>
</tr>
</thead>
</table>

Figure 7-17: Comparison of learning strategies in the Driving World with Signaling and Slipping.

### 7.2.3 The Driving World with Signaling and Slipping

The Driving World with Signaling and Slipping presents a large and complex world with several types of hidden state. The world has six actions and forty-five observations meaning that a DBN model of the world needs at least ninety-six nodes. As in the Signaling World, hidden state is needed to remember the current state of the turning signal, and as in the Slippery Driving World, hidden state is needed to gather imperfect information about the wetness of the road. Moreover, the distance-thermometers also contain hidden state, as a model would need to "count" how long a distance-thermometer has had a certain reading in order to know when the reading will change. The results of learning in the Driving World with Signaling and Slipping are shown in Figure 7-17. The accuracy parameter, $\beta$, associated with the penalty received for the tires screeching was set to 0.8 during these experiments. The parameters of the WORLD-HIDDEN-STATE-DISCOVERY algorithm were set such that the algorithm looks for at most two hidden
nodes to improve the model of the penalty for changing lanes without blinking and at most one hidden node to improve the model of the penalty for the tires screeching.

Based upon the likelihood of the test data, the models learned by the K-SLICE-DBN-HIDDEN-STATE-DISCOVERY algorithm performed slightly worse than models learned without hidden nodes. The incorporation of hidden nodes related to the penalty for changing lanes without signaling and the penalty for the tires screeching were exactly analogous to the results in the Signaling World and the Slippery Driving World, respectively. The algorithm generated hidden variables to model the state of the turning signal during only two out of the five runs, and during these two runs they were not incorporated into the model correctly. As was the case in the Slippery Driving World when the accuracy parameter set to 0.8, the algorithm never incorporated hidden variables to model the wetness of the road. Each run of the algorithm generated one other hidden variable not related to either of these penalties, though the role of this variable in the model varied from run to run.

Though the difference in the log likelihood of the training data is seemingly not significant, the models generated by the WORLD-DISCOVER-HIDDEN-STATE algorithm seem significantly better than the models generated by the other two algorithms. The algorithm's incorporation of hidden variables to model the penalty for changing lanes without signaling and the penalty for the tires screeching were similar to the algorithm's results in the Signaling World and the Slippery Driving World, respectively. Each run incorporated two hidden nodes to model the turning signal and a single hidden node to model the wetness of the road. This demonstrates that learning models of large worlds
that require many nodes does not hinder the algorithm's ability to incorporate hidden state
to improve the model of rewards.
Chapter 8

Conclusions

In this thesis I discussed Bayesian networks and how to learn Bayesian networks from data. I then moved on to dynamic Bayesian networks which can be used to model dynamic systems. I discussed how to learn DBN's from data and how to improve this learning in the case of modeling world dynamics. I next discussed the need to capture non-Markovian dynamics and introduced adding hidden nodes to the DBN for this purpose.

I demonstrated that DBN learning can be used to learn effective world models from relatively small data sets that are extremely far from exhaustively enumerating the possible world states. The success of DBN learning with limited data implies that the learning is indeed making valid generalizations.

I demonstrated that incorporating hidden nodes into the DBN can allow for the modeling of non-Markovian properties of the world. Hidden nodes were successfully incorporated into DBN models to act as memory and to model situations where observations provide imperfect information about an underlying state of the world.
Appendix A

Concise Descriptions of Each World

This appendix describes each world used in this project in terms of the actions, rewards, observations, and underlying hidden state.

A.1 The Driving World

<table>
<thead>
<tr>
<th>Actions</th>
<th>Change lanes to the left</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Change lanes to the right</td>
</tr>
<tr>
<td></td>
<td>Do nothing</td>
</tr>
<tr>
<td>Rewards</td>
<td>Penalty for hitting a slower car</td>
</tr>
<tr>
<td></td>
<td>Penalty for being hit by a faster car</td>
</tr>
<tr>
<td></td>
<td>Penalty for hitting the curb</td>
</tr>
<tr>
<td>Observations</td>
<td>Whether the agent's car is in the leftmost lane</td>
</tr>
<tr>
<td>other than the</td>
<td>Whether the agent's car is in the rightmost lane</td>
</tr>
<tr>
<td>rewards</td>
<td>Whether there is a car directly to the left of the agent's car</td>
</tr>
<tr>
<td></td>
<td>Whether there is a car directly to the right of the agent's car</td>
</tr>
<tr>
<td></td>
<td>Whether the closest car in front of the agent's car in the lane to the left of the agent's car is traveling faster or slower than the agent's car</td>
</tr>
<tr>
<td></td>
<td>Five bit distance thermometer measuring the distance to the closest car in front of the agent's car in the lane to the left of the agent's car</td>
</tr>
<tr>
<td></td>
<td>Whether the closest car behind the agent's car in the lane to the left of the agent's car is traveling faster or slower than the agent's car</td>
</tr>
<tr>
<td></td>
<td>Five bit distance thermometer measuring the distance to the closest car behind the agent's car in the same lane as the agent's car</td>
</tr>
<tr>
<td></td>
<td>Whether the closest car in front of the agent's car in the same lane as the agent's car is traveling faster or slower than the agent's car</td>
</tr>
<tr>
<td></td>
<td>Five bit distance thermometer measuring the distance to the closest car in front of the agent's car in the same lane as the agent's car</td>
</tr>
<tr>
<td></td>
<td>Whether the closest car behind the agent's car in the same lane as the agent's car is traveling faster or slower than the agent's car</td>
</tr>
<tr>
<td></td>
<td>Five bit distance thermometer measuring the distance to the closest car behind the agent's car in the same lane as the agent's car</td>
</tr>
</tbody>
</table>
Whether the closest car in front of the agent's car in the lane to the right of the agent's car is traveling faster or slower than the agent's car

Five bit distance thermometer measuring the distance to the closest car in front of the agent's car in the lane to the right of the agent's car

Whether the closest car behind the agent's car in the lane to the right of the agent's car is traveling faster or slower than the agent's car

Underlying hidden state

There is hidden state associated with the distance-thermometers. For instance, if four bits are true and the last is false, we know that the vehicle is between 45 and 100 meters away from the agent's car, but we do not know exactly how far.

### A.2 The Left Signaling World

<table>
<thead>
<tr>
<th>Actions</th>
<th>Change lanes to the left</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Change lanes to the right</td>
</tr>
<tr>
<td></td>
<td>Switch on turning signal</td>
</tr>
<tr>
<td></td>
<td>Switch off turning signal</td>
</tr>
<tr>
<td></td>
<td>Do nothing</td>
</tr>
</tbody>
</table>

Rewards: Penalty for changing lanes to the left when the turning signal is off

Observations other than the rewards: None

Underlying hidden state: The state of the turning signal (on or off)

### A.3 The Signaling World

<table>
<thead>
<tr>
<th>Actions</th>
<th>Change lanes to the left</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Change lanes to the right</td>
</tr>
<tr>
<td></td>
<td>Switch the turning signal to the left</td>
</tr>
<tr>
<td></td>
<td>Switch the turning signal to the right</td>
</tr>
<tr>
<td></td>
<td>Switch the turning signal off</td>
</tr>
<tr>
<td></td>
<td>Do nothing</td>
</tr>
</tbody>
</table>

Rewards: Penalty for changing lanes without the turning signal signaling in the correct direction

Observations other than the rewards: None

Underlying hidden state: The state of the turning signal (left, right, or off)
### A.4 The Slippery Driving World

<table>
<thead>
<tr>
<th>Actions</th>
<th>Change lanes to the left</th>
<th>Do nothing</th>
<th>Do nothing</th>
<th>Do nothing</th>
<th>Do nothing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rewards</td>
<td>Penalty for the tires screeching. When the agent changes lane, the tires screech with probability $\beta$ if the road is wet, and probability $1-\beta$ if the road is not wet.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Observations other than the rewards</td>
<td>None</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Underlying hidden state</td>
<td>The state of the road (wet or dry) is hidden. The wetness of the road changes at any given instant with probability $\alpha$.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### A.5 The Driving World with Signaling and Slipping

<table>
<thead>
<tr>
<th>Actions</th>
<th>Change lanes to the left</th>
<th>Change lanes to the right</th>
<th>Switch the turning signal to the left</th>
<th>Switch the turning signal to the right</th>
<th>Switch the turning signal off</th>
<th>Do nothing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rewards</td>
<td>Penalty for hitting a slower car</td>
<td>Penalty for being hit by a faster car</td>
<td>Penalty for hitting the curb</td>
<td>Penalty for changing lanes without the turning signal signaling in the correct direction</td>
<td>Penalty for the tires screeching. When the agent changes lane, the tires screech with probability $\beta$ if the road is wet, and probability $1-\beta$ if the road is not wet.</td>
<td></td>
</tr>
<tr>
<td>Observations other than the rewards</td>
<td>Whether the agent's car is in the leftmost lane</td>
<td>Whether the agent's car is in the rightmost lane</td>
<td>Whether there is a car directly to the left of the agent's car</td>
<td>Whether there is a car directly to the right of the agent's car</td>
<td>Whether the closest car in front of the agent's car in the lane to the left of the agent's car is traveling faster or slower than the agent's car</td>
<td></td>
</tr>
<tr>
<td>Underlying hidden state</td>
<td>The state of the turning signal (left, right, or off) must be remembered.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The state of the road (wet or dry) is hidden. The wetness of the road changes at any given instant with probability $\alpha$.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>There is hidden state associated with each distance-thermometer. For instance, if four bits are true and the last is false, we know that the vehicle is between 45 and 100 meters away from the agent's car, but we do not know exactly how far.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Five bit distance thermometer measuring the distance to the closest car in front of the agent's car in the lane to the left of the agent's car.
- Whether the closest car behind the agent's car in the lane to the left of the agent's car is traveling faster or slower than the agent's car.
- Five bit distance thermometer measuring the distance to the closest car behind the agent's car in the lane to the left of the agent's car.
- Whether the closest car in front of the agent's car in the same lane as the agent's car is traveling faster or slower than the agent's car.
- Five bit distance thermometer measuring the distance to the closest car in front of the agent's car in the same lane as the agent's car.
- Whether the closest car behind the agent's car in the same lane as the agent's car is traveling faster or slower than the agent's car.
- Five bit distance thermometer measuring the distance to the closest car behind the agent's car in the same lane as the agent's car.
- Whether the closest car in front of the agent's car in the lane to the right of the agent's car is traveling faster or slower than the agent's car.
- Five bit distance thermometer measuring the distance to the closest car in front of the agent's car in the lane to the right of the agent's car.
- Whether the closest car behind the agent's car in the lane to the right of the agent's car is traveling faster or slower than the agent's car.


