Learning Probabilistic Relational Planning Rules

by

Luke S. Zettlemoyer

B.S. Computer Science
B.S. Applied Mathematics
North Carolina State University, 2000

Submitted to the
Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August 2003

Author .....
Department of Electrical Engineering and Computer Science
August 29, 2003

Certified by .....
Leslie Pack Kaelbling
Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by.....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Learning Probabilistic Relational Planning Rules

by

Luke S. Zettlemoyer

Submitted to the Department of Electrical Engineering and Computer Science
on August 29, 2003, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

To learn to behave in complex, uncertain domains, robots must represent and learn compact models of the world dynamics. These models will have a significant bias towards the types of regularities we see in the world around us. Probabilistic planning rules are a strong first step towards an appropriately biased representation; as we will see in this thesis, they can be learned efficiently from little training data.

Thesis Supervisor: Leslie Pack Kaelbling
Title: Professor of Computer Science and Engineering

Acknowledgments

I would like to thank my advisor, Leslie Kaelbling. She is a constant source of inspiration and an endless fountain of ideas. She has also assembled a wonderful group of students and post-docs, all of whom I would like to thank for their endless camaraderie. All of this work for this thesis was done in close collaboration with and would not have been completed without Hanna Pasula. Finally, I would like to thank all of my friends and family for their constant support and encouragement.

This work was supported in part by a NDSEG Graduate Research Fellowship.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Imagine robots that live in the same world that we do; they must be able to predict the consequences of their actions both efficiently and accurately.

Programming a robot for advanced problem solving in a complicated environment is an hard, open problem. Engineering a direct solution has proven difficult. Even the most sophisticated robot programming paradigms (Brooks, 1991), are difficult to scale to human-like robot behaviors.

If robots could learn to act in the world, then much of the programing burden would be removed from the robot engineer. There has been work in reinforcement learning on this problem (Smart & Kaelbling, 2000), but in these cases the robot learns policies for achieving particular goals, without gathering any general knowledge of the world dynamics. As a result, the robots can learn to do particular tasks but have trouble generalizing to new ones.

Instead, if robots could learn how their actions affect the world, then they would be able to behave more robustly in a wide range of situations. This type of learning allows the robot to develop a *world model* that represents the immediate effects of its action in the world. Once this model is learned, the robot could use it to behave robustly in a wide variety of situations.

Imagine, for example, that you are an engineer that must design a robot to deliver mail in wide range of different office environments. You could try to directly program all of the possible floor plans the robot would need to navigate. This would be almost impossible. Or, you could use reinforcement learning to learn to achieve each goal in each

office layout, such as finding a particular office. While this approach could work, it would be very inefficient.

Model learning provides a reasonable alternative. You would first provide the robot with a set of actions, such as following corridors and reading office numbers. The robot could then, in any office environment, execute these action, explore the office, and learn a model of where things are and how to navigate between them.

There are many different ways of representing world models, but one representation, rules, stands out. Rules represent situations in which actions have specified effects. Because each situation can be considered independently, rules can be learned incrementally without having to understand the whole world. Rules also encode assumptions about how the world works and what is possible when actions are performed. These assumptions, as we will see in detail later, correspond to the dynamics of our world and simplify learning and reasoning.

Once rules have been learned, then acting with them is a well studied research problem. Probabilistic planning approaches are directly applicable (Blum & Langford, 1999). And work in this area has shown that compact representations, like rules, are essential for scaling probabilistic planning to large worlds (Boutilier, Dearden, & Goldszmidt, 2002).

But, of course, learning world action models could be just as difficult as programming the robot in the first place. This thesis explores the learning of a simple probabilistic planning rule language in worlds that are much less complex than our office example above. We will see that this language has a strong bias that enables learning of accurate world models from less training data than required by traditional representations of action.

This is an encouraging result, but it is just the first step. As we will see, these rules are actually too restrictive; they make so many assumptions that they are not directly useful in the real world. But, the learning results are still encouraging. They provide hope that planning rules can be extended to model our world without losing their computational advantages.

1.1 Structured Worlds

Developing representations that are structured to efficiently encode the models they will represent has many advantages. These representations provide a *bias* for learning algorithms. Because the representations should be able to efficiently encode the desired models, the learning algorithms can search for compact models: models that will generalize better and can be learned from fewer training examples. And, in general, as the representation becomes more biased towards modeling specific types of worlds, fewer examples will be required to learn an accurate model. However, this increase comes at a cost. These biased representations generally represent worlds that violate their assumptions very poorly.

So, our goal is to make as many assumptions as possible without losing the representational capacity that is required to model the real world. In the planning literature, the following *planning world structure* (PWS) assumptions are common:

- **Frame Assumption:** When an agent takes an action in a world, anything not explicitly changed by that action stays the same.
- **Object Abstraction:** The world is made up of objects, and the effects of actions on these objects generally depend on their attributes rather than their identity.
- **Action Outcomes:** When an action is performed there is a small set of possible changes that can happen to the world. The action can succeed, fail, or have some number of other outcomes.

The first two assumptions have been captured in almost all planning representations, such as STRIPS rules (Fikes & Nilsson, 1971) and more recent variants (Penberthy & Weld, 1992). Recently, the third assumption has been made for modeling probabilistic planning domains with rules (Blum & Langford, 1999), in the situation calculus (Boutilier, Reiter, & Price, 2001), and in the equivalence-class approach of Draper, Hanks, and Weld (1994).

Since these assumptions are so commonly made and are essential for efficient action selection, it is important to understand how they can improve learning. It is commonly acknowledged that planning assumptions are almost certainly too restrictive; however,

demonstrating algorithms that learn them from little data provides hope that less restricted representations can be developed for use in robots that learn the dynamics of the real world.

1.2 Thesis Overview

The rest of this thesis is organized as follows. First, Chapter 2 formally defines probabilistic relational planning rules. These rules are discussed in more detail in Chapter 3 which shows that they encode PWS more efficiently than traditional action representations. Chapter 4 presents a rule learning algorithm that is biased by planning rule structure. The experiments of Chapter 5 provide validation that this learning algorithm is biased appropriately. Finally, Chapter 6 and Chapter 7 present related and future work.

All of the work in this thesis was done in close collaboration with Hanna Pasula. These ideas were first presented in (Pasula, Zettlemoyer, & Kaelbling, 2003) and this thesis details more of our continuing collaborative research.

Chapter 2

Probabilistic Relational Planning Rules

This chapter presents a formal definition of relational planning rules and the world descriptions that they manipulate. World descriptions represent what the agent knows about the world in which it lives and rules represent what the agent believes will happen when it performs actions in these worlds.

World descriptions and planning rules are built from a subset of the syntax of first-order logic. They have predicates, constants, variables, universal quantification, and conjunctive connectives. But, they lack functions, disjunctive connectives, and existential quantification. Literals, ground literals, truth values, and substitutions are defined as usual. Finally, in this thesis, sets of literals and conjunctions of literals are used interchangeably.

2.1 Representing Worlds

An agent's description of the world at time t, also called the state S^t, is represented syntactically as a set of ground literals. Semantically, these literals represent all of the important aspects of this world. The constants map to the objects in the world. The literals encode the truth values of every possible property for all of the objects and all of the relations that are possible between objects.

For example, imagine a simple blocks world. The objects in this world include blocks, a table and a gripper. Blocks can be on other blocks or on the table. A block that has nothing on it is clear. The gripper can hold one block or be empty. The following state

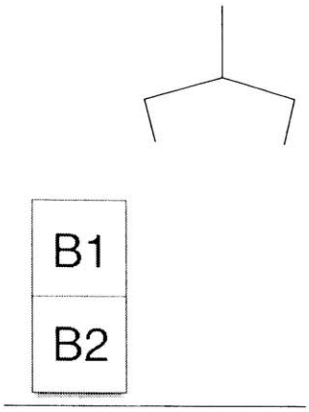

Figure 2-1: A blocks world with two blocks in a single stack and an empty gripper.

$$\begin{aligned} &on(\text{B1}, \text{B2}), on(\text{B2}, \text{TABLE}), inhand(\text{NIL}), clear(\text{B1}), block(\text{B1}), \\ &block(\text{B2}), \neg clear(\text{B2}), \neg on(\text{B2}, \text{B1}), \neg on(\text{B1}, \text{TABLE}), \neg inhand(\text{B1}), \\ &\neg inhand(\text{B2}), \neg block(\text{TABLE}) \end{aligned} \tag{2.1}$$

represents a blocks world where there are two blocks in a single stack on the table. Block B1 is on top of the stack, while B2 is below B1 and on the TABLE. This state is illustrated in Figure 2-1.

Finally, consider when these world descriptions are useful. Because they must contain all of the important details of the world, they can only be used when the world is *fully observable*, when no important aspects of the world are hidden for the agent's perception. All of the worlds in this thesis are fully observable. Investigating the *partially observable* case, where the truth values of some of the literals are not specified, is an important area for future work.

2.2 Representing Rules

A rule set is a model of how a world will change as an agent performs actions in it. For example, the rule set we will explore in this section models how the simple blocks world changes state as it is manipulated by a robot arm. This arm can attempt to pick up blocks and put them on other blocks or the table. However, the arm is faulty so its actions can

succeed, fail to change the world, or fail by knocking the block onto the table.

This section first presents the syntax of rule sets. Then, the semantics of rule sets is described procedurally by showing how the rules are used by an agent. The running example explores the dynamics of blocks world, as just described.

A rule set, $\mathbf{R}$, is simply a set of rules. Each $r \in \mathbf{R}$ is a four-tuple $(C, \mathbf{O}, P_{\mathbf{O}}, A)$. The rule's action $r.A$ is a positive literal. The *context* $r.C$ is a set of literals. The *outcome set* $r.\mathbf{O}$ is a non-empty set of outcomes. Each *outcome* $O \in r.\mathbf{O}$ is a set of literals. Finally, $r.P_{\mathbf{O}}$ is a discrete distribution over the set of outcomes that must assign a non-zero probability to each outcome. Rules may contain variables; however, every variable appearing in $r.C$ or $r.\mathbf{O}$ must also appear in $r.A$. Figure 2-2 is a rule set with four rules for the blocks world domain. Each rule's context is the set of literals to the left of the arrow, with the exception of the last literal which is the action. Each rule's right side shows outcomes paired with their probability from $r.P_{\mathbf{O}}$.

A rule set, $\mathbf{R}$, is a full model of a world's dynamics. With this rule set, the agent can predict the effects of an action, a, when it is performed in a state, $\mathbf{S}^t$. It can also look at a transition from $\mathbf{S}^t$ to $\mathbf{S}^{t+1}$ that occurred when a was executed and determine how likely that particular effect was. Both of these uses are described in this section. They have a common subproblem of determining which rule, from a set, governs the change for a particular initial state, $\mathbf{S}^t$, and action, a. This problem is discussed first.

2.2.1 Rule Selection

Given an action, a, a state, $\mathbf{S}$, and a rule set, $\mathbf{R}$, an agent will often need to find the rule $r \in \mathbf{R}$ that *covers* $\mathbf{S}$ given a. This is a three-step process that ensures that r's action models a, that the r is well formed given a, and that r's context is satisfied by $\mathbf{S}$. As a running example, imagine an agent wants to predict the effects of executing *pickup*(B1, B2) in the world described in Equation 2.1 given the model represented by the rule set in Figure 2-2.

The first step is to build the set of rules $\mathbf{R}' \subseteq \mathbf{R}$ whose actions model a. For every $r \in \mathbf{R}$, the agent attempts to unify $r.A$ with a. Because of the rule's limited syntax, unification will succeed if a and $r.A$ have the same predicate, the same arity, and there is

not an index i such that the ith argument to $r.A$ and the ith argument to a are different constants. If unification is successful, then the agent computes an *action substitution* $\theta_{(a,r)}$ that maps all of the variables in $r.A$ to the corresponding constants in a. If unification fails, r is removed from $\mathbf{R}'$. The *pickup*(B1, B2) action in our running example unifies with the action of the first rule in $\mathbf{R}'$ producing the substitution $\theta_{(a,r)} = \{X/\text{B1}, Y/\text{B2}\}$; fails to unify with the second rule's action, because B2 doesn't equal TABLE; and fails to unify with the remaining rules since they have different predicates.

Now, the agent has to check that every $r \in \mathbf{R}'$ is well formed given a. r is well formed if the conjunctions $\theta_{(a,r)}(r.C)$ and $\theta_{(a,r)}(O)$, for all $O \in r.\mathbf{O}$, do not contain any contradictions. In our running example, the only $r \in \mathbf{R}'$ is well formed given a. However, if the action a had been *pickup*(B1, B1) then the first outcome of the first rule in Figure 2-2 would have a contradiction since it would contain *clear*(B1) and $\neg$*clear*(B1). Any rules with contradictions are removed from $\mathbf{R}'$.

Then, for each remaining $r \in \mathbf{R}'$, the agent checks whether $r.C$ is satisfied by the state $\mathbf{S}$ using the *selector function* $\delta(\mathbf{C_1}, \mathbf{C_2})$ that takes two ground literal sets as input and returns 1 if $\mathbf{C_1} \subseteq \mathbf{C_2}$, and 0 otherwise. More specifically, $\mathbf{S}$ satifies $r.C$ if $\delta(\theta_{(a,r)}(r.C), \mathbf{S}^t) = 1$. Note that the application of the action substition to the context, $\theta_{(a,r)}(r.C)$, will always be ground because we assumed that all variables in $r.C$ are also in $r.A$. Any r whose context isn't satisfied by $\mathbf{S}$ is removed from $\mathbf{R}'$. To finish our running example, $\theta_{(a,r)}(r.C)$ for the only rule r remaining in $\mathbf{R}'$ is $\{$*on*(B1, B2), *clear*(B1), *inhand*(NIL), *block*(B2)$\}$. Since this set is a subset of the state in Equation 2.1, r is the rule that will model the effects of *pickup*(B1, B2).

Notice that under an action a the state $\mathbf{S}$ could be covered by zero, one, or many rules. A rule set is *valid* if every possible state $\mathbf{S}$ is covered by at most one rule. All of the rule sets in this thesis are assumed to be valid. Dealing with multiple rules that make conflicting predictions about future worlds is an important area for future work.

$$
\begin{array}{rcl}
on(X,Y), clear(X), inhand(\text{NIL}), \\
block(Y), pickup(X,Y)
& \rightarrow &
\left\{
\begin{array}{ll}
.7: & inhand(X), \neg clear(X), \neg inhand(\text{NIL}), \\
 & \neg on(X,Y), clear(Y) \\
.2: & on(X,\text{TABLE}), \neg on(X,Y), clear(Y) \\
.1: & \text{no change}
\end{array}
\right. \\
on(X,\text{TABLE}), clear(X), inhand(\text{NIL}), \\
pickup(X,\text{TABLE})
& \rightarrow &
\left\{
\begin{array}{ll}
.66: & inhand(X), \neg clear(X), \neg inhand(\text{NIL}), \\
 & \neg on(X,\text{TABLE}) \\
.34: & \text{no change}
\end{array}
\right. \\
clear(Y), inhand(X), block(Y), \\
puton(X,Y)
& \rightarrow &
\left\{
\begin{array}{ll}
.7: & inhand(\text{NIL}), \neg clear(Y), \neg inhand(X), \\
 & on(X,Y), clear(X) \\
.2: & on(X,\text{TABLE}), clear(X), inhand(\text{NIL}), \\
 & \neg inhand(X) \\
.1: & \text{no change}
\end{array}
\right. \\
inhand(X), \\
puton(X,\text{TABLE})
& \rightarrow &
\left\{
\begin{array}{ll}
.8: & on(X,\text{TABLE}), clear(X), inhand(\text{NIL}), \\
 & \neg inhand(X) \\
.2: & \text{no change}
\end{array}
\right.
\end{array}
$$

Figure 2-2: Four relational rules that model the world dynamics of a simple blocks world.

2.2.2 Rule Execution

Now, an agent can predict the effects of executing action a in state $\mathbf{S}^t$ as follows. First, it finds the rule $r \in \mathbf{R}$ that covers $\mathbf{S}^t$ given a. If there are no such rules, then the agent can assume, given the frame assumption, that executing a would not change the world. If there is an r and a is performed, then the rule's list of outcomes, $r.\mathbf{O}$, and its distribution over them, $r.P_{\mathbf{O}}$, define what will happen. First, an outcome O is selected by sampling from $r.P_{\mathbf{O}}$. This outcome is then used to construct the next state, $\mathbf{S}^{t+1}$, as follows. The agent initializes $\mathbf{S}^{t+1}$ by copying the literals from $\mathbf{S}^t$. Then, the agent grounds O by applying $\theta_{(a,r)}$ to it. For each literal in $\theta_{(a,r)}(O)$, the agent next finds the literal in $\mathbf{S}^{t+1}$ that has the same atomic formula and checks that their truth values match. It they do, nothing happens. Otherwise, the agent flips the truth value of the literal in $\mathbf{S}^{t+1}$. Figure 2-3 shows an example where the first outcome from the first rule in Figure 2-2 predicts that effects of $pickup(\text{B1}, \text{B2})$ to the state of Equation 2.1. The states are represented pictorially and annotated with only the true literals, all others are assumed to be false. As the outcome predicts, $inhand(\text{B1})$ and $clear(\text{B2})$ become true while $on(\text{B1}, \text{B2})$, $clear(\text{B1})$, and $inhand(\text{NIL})$ become false.

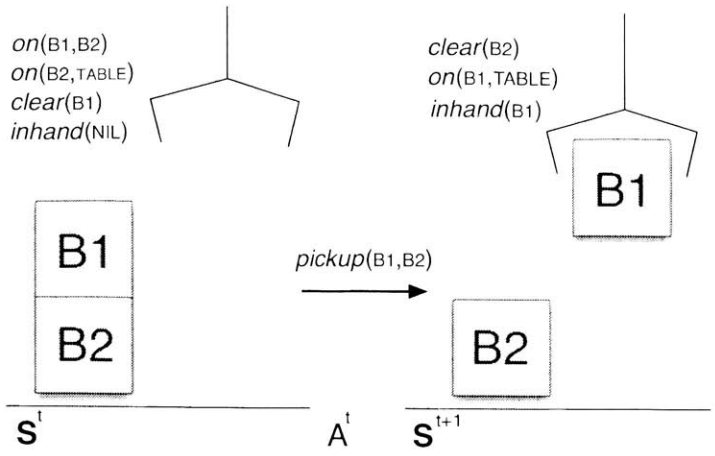

Figure 2-3: Two subsequent states of the blocks world with two blocks. The pictured states are represented by the neighboring lists of true propositions. Everything that is not listed is false. The action *pickup*(B1, B2) was performed successfully

2.2.3 Likelihood Estimation

Given two states $\mathbf{S}^t$ and $\mathbf{S}^{t+1}$, an action a and a rule set $\mathbf{R}$, another problem that an agent often faces, especially when learning rules, is to determine the probability that a caused $\mathbf{S}^t$ to become $\mathbf{S}^{t+1}$, $P(\mathbf{S}^{t+1}|\mathbf{S}^t, a, \mathbf{R})$. Just like before, the agent first finds the unique rule $r \in \mathbf{R}$ that covers $\mathbf{S}^t$ given a. If there is no rule, the agent assumes that there should be no change to the initial state, $\mathbf{S}^t = \mathbf{S}^{t+1}$. If this is true, then $P(\mathbf{S}^{t+1}|\mathbf{S}^t, a, \mathbf{R})$ is one, otherwise the model has failed since an impossible transition has occurred. Assuming a unique $r \in \mathbf{R}$ is found, then the probability of the transition is

$$P(\mathbf{S}^{t+1}|\mathbf{S}^t, a, r) = \sum_{O \in r.\mathbf{O}} \delta(\theta_{(a,r)}(O), \mathbf{S}^{t+1})\delta(\mathbf{S}^{t+1} - \theta_{(a,r)}(O), \mathbf{S}^t) r.P_{\mathbf{O}}(O) \,. \tag{2.2}$$

The two δ functions select the outcomes that describe the transition from $\mathbf{S}^t$ to $\mathbf{S}^{t+1}$: the first one checks whether the literals in the ground outcome are all in the new state, and the second one enforces the frame assumption that the literals in $\mathbf{S}^{t+1}$ but not in the outcome must have the same truth values that they did in $\mathbf{S}^t$. If both of these delta functions return one, then the outcome O *covers* the state transition from $\mathbf{S}^t$ to $\mathbf{S}^{t+1}$ The probability of each $\mathbf{S}^{t+1}$ is the sum of all the outcomes that cover the transition from $\mathbf{S}^t$ to it.

Notice that each O can only cover one transition to any particular $\mathbf{S}^{t+1}$, since $\mathbf{S}^t$ and O

uniquely determine $\mathbf{S}^{t+1}$, as described in Section 2.2.2. This fact guarantees that $P(\mathbf{S}^{t+1}|\mathbf{S}^t, a, r)$ is a well defined distribution.

2.2.4 Overlapping Outcomes

Consider the phenomenon of *overlapping outcomes*. Given an initial state $\mathbf{S}^t$, more than one outcome could cover the transition to an unique next state $\mathbf{S}^{t+1}$. When this happens the outcomes are called overlapping. For example,

$$\mathit{inhand}(X), \mathit{block}(X), \mathit{paint}(X) \rightarrow \begin{cases} .8 : \mathit{painted}(X), \mathit{wet} \\ .2 : \text{no change} \end{cases}$$

is a rule for painting blocks. When this rule is used for an action $\mathit{paint}(\text{B1})$ in an initial state that contains *wet* and $\mathit{painted}(\text{B1})$ the outcomes overlap. Both outcomes describe changes that lead to the same next state, in this case they expect no change to occur. Overlapping outcomes are an integral part of planning rules. They also significantly complicate rule learning, as we will see in Chapter 4.

Chapter 3

Representing Structured Worlds

When representing worlds, it is important to leverage their inherent structure. Every representation makes different assumptions about the model it encodes, and choosing the most appropriate representation is a difficult problem. This chapter explores representations for encoding planning world structure (PWS). It shows that planning rules have a built-in bias that makes them particularly well suited for representing PWS and that enables the creation of special-purpose learning algorithms, as described in later chapters.

3.1 PWS in the Simple Blocks World

Because this chapter focuses on efficiently representing PWS, this section first reviews this structure by showing how it is displayed by our running example, the simple blocks world.

The blocks world of Chapter 2 exhibits the planning world structure assumptions that were introduced in Section 1.1. The execution of the *pickup*(B1, B2) action from Figure 2-3 demonstrates each assumption. First, the *frame assumption* guarantees that all of the propositions that aren't changed by the action don't change state. For example, *on*(B2, B1) is false and remains false. Also, notice that *variable abstraction* ensures that the names of the blocks don't really matter. If B2 had been on B1 the gripper still could have picked it up. Finally, remember that the action has a set of possible *action outcomes*. We only see one outcome in the figure, but the gripper could have dropped the block or failed to move it at all.

3.2 Propositional Representations

First, for simplicity, this section considers representing world structure without the use of variable abstraction. This is an important subproblem because it is the case where well established representations with learning algorithms are available for comparison. In Section 3.3, we will consider the full case.

The most common and well-studied representation for probabilistic relationships among random variables is Bayesian networks (Pearl, 1988). Bayesian networks make conditional independence assumptions that do not leverage all of the PWS. This fact is explored in this section and contrasted with planning rules that leverage the structure more fully.

This representational discrepancy will be particularly important for learning as we will see in Chapter 5, which shows that planning rule learning is able to leverage the representation bias.

3.2.1 Planning Rules

Propositional planning rules (PPRs) have exactly the same syntax and semantics as the relational planning rules of Chapter 2, with a single exception: variables are not allowed to appear in any part of propositional rules. As an example, consider the PPRs in Figure 3-1 that model a blocks world with two blocks. Notice, in particular, that although these rules look very similar to relational rules, the fact that they lack variables makes each literal, for all practical purposes, an atomic proposition.

Although they encode the same informational as their relational counterparts, propositional rules are much less compact. For example, all of the $n(n-1)$ propositional rules for how to pick up one block from another in an blocks world with n blocks could be represented as a single rule with variable abstraction.

PWS assumptions are tightly integrated into PPRs. The procedural semantics, as described in Section 2.2, makes frequent use of the frame assumption to fill in all aspects of the world's dynamics that were not directly represented in the rules. The structure of the rules directly encodes the action outcomes assumption: each rule has a set of possible outcomes.

$$
\begin{array}{rcl}
on(\text{B1},\text{B2}), clear(\text{B1}), inhand(\text{NIL}), \\
pickup(\text{B1},\text{B2})
& \rightarrow &
\begin{cases}
.7: & inhand(\text{B1}), \neg clear(\text{B1}), \neg inhand(\text{NIL}), \\
 & \neg on(\text{B1},\text{B2}), clear(\text{B2}) \\
.2: & on(\text{B1},\text{TABLE}), \neg on(\text{B1},\text{B2}), clear(\text{B2}) \\
.1: & \text{no change}
\end{cases} \\
on(\text{B2},\text{B1}), clear(\text{B2}), inhand(\text{NIL}), \\
pickup(\text{B2},\text{B1})
& \rightarrow &
\begin{cases}
.7: & inhand(\text{B2}), \neg clear(\text{B2}), \neg inhand(\text{NIL}), \\
 & \neg on(\text{B2},\text{B1}), clear(\text{B1}) \\
.2: & on(\text{B2},\text{TABLE}), \neg on(\text{B2},\text{B1}), clear(\text{B1}) \\
.1: & \text{no change}
\end{cases} \\
on(\text{B1},\text{TABLE}), clear(\text{B1}), inhand(\text{NIL}), \\
pickup(\text{B1},\text{TABLE})
& \rightarrow &
\begin{cases}
.66: & inhand(\text{B1}), \neg clear(\text{B1}), \neg inhand(\text{NIL}), \\
 & \neg on(\text{B1},\text{TABLE}) \\
.34: & \text{no change}
\end{cases} \\
on(\text{B2},\text{TABLE}), clear(\text{B2}), inhand(\text{NIL}), \\
pickup(\text{B2},\text{TABLE})
& \rightarrow &
\begin{cases}
.66: & inhand(\text{B2}), \neg clear(\text{B2}), \neg inhand(\text{NIL}), \\
 & \neg on(\text{B2},\text{TABLE}) \\
.34: & \text{no change}
\end{cases} \\
clear(\text{B2}), inhand(\text{B1}), \\
puton(\text{B1},\text{B2})
& \rightarrow &
\begin{cases}
.7: & inhand(\text{NIL}), \neg clear(\text{B2}), \neg inhand(\text{B1}), \\
 & on(\text{B1},\text{B2}), clear(\text{B1}) \\
.2: & on(\text{B1},\text{TABLE}), clear(\text{B1}), inhand(\text{NIL}), \\
 & \neg inhand(\text{B1}) \\
.1: & \text{no change}
\end{cases} \\
clear(\text{B1}), inhand(\text{B2}), \\
puton(\text{B2},\text{B1})
& \rightarrow &
\begin{cases}
.7: & inhand(\text{NIL}), \neg clear(\text{B1}), \neg inhand(\text{B2}), \\
 & on(\text{B2},\text{B1}), clear(\text{B2}) \\
.2: & on(\text{B2},\text{TABLE}), clear(\text{B2}), inhand(\text{NIL}), \\
 & \neg inhand(\text{B2}) \\
.1: & \text{no change}
\end{cases} \\
inhand(\text{B1}), \\
puton(\text{B1},\text{TABLE})
& \rightarrow &
\begin{cases}
.8: & on(\text{B1},\text{TABLE}), clear(\text{B1}), inhand(\text{NIL}), \\
 & \neg inhand(\text{B1}) \\
.2: & \text{no change}
\end{cases} \\
inhand(\text{B2}), \\
puton(\text{B2},\text{TABLE})
& \rightarrow &
\begin{cases}
.8: & on(\text{B2},\text{TABLE}), clear(\text{B2}), inhand(\text{NIL}), \\
 & \neg inhand(\text{B2}) \\
.2: & \text{no change}
\end{cases}
\end{array}
$$

Figure 3-1: Eight propositional rules that define the transition dynamics of a simple two-block blocksworld. Each rule has a context that selects the set of states that will be changed by its action according to its distribution over possible outcomes. Notice that the literals in these rules can also be considered atomic propositions since there is no way to abstract their arguments.

Because planning rules were designed to model PWS, they can model the dynamics of our simple blocks world compactly and accurately.

3.2.2 Dynamic Bayesian Networks

A Bayesian network (BN) (Pearl, 1988) is a directed acyclic graph, $\mathbf{G} = (\mathbf{X}, \mathbf{E})$, where the nodes $\mathbf{X}$ correspond to discrete random variables, and the edges $\mathbf{E}$ encode a set of conditional independence assumptions. Associated with each node X_i is a *conditional probability model* (CPM), $\Pr(X_i|Pa_i)$, where the set Pa_i contains the parents of X_i. The product of all these conditional probabilities defines a unique joint probability distribution over all the variables.

A *dynamic Bayesian network* (DBN) (Dean & Kanazawa, 1988) is a BN that models change over time by representing the state of the world at two subsequent times, t and $t+1$. Thus, each variable X_i is represented twice, as X_i^t and as X_i^{t+1}. DBN graph structure is restricted so that edges must not point backwards through time.

A set of DBNs encodes a world model when there is a unique DBN for every possible action in the world. Figure 3-2 shows the DBN that models the *pickup*(B1, B2) action from Figure 3-1. Notice that this DBN is more of a distributed representation than the corresponding rule. Although it encodes the same information, there is no one part of the network that encodes action outcomes or the frame assumption.

3.2.3 Comparison

Because propositional planning rules were designed to encode PWS it is not surprising that they can encode the simple blocksworld more efficiently than DBNs. However, this representational discrepancy is not as large as it might originally appear. In this chapter, we will see that, with only a few modifications, traditional DBNs can changed to represent the same world structure as propositional planning rules. This process of converting a DBN to propositional rules can be done in four steps. Each step highlights one way in which propositional rules are designed to encode structured world dynamics.

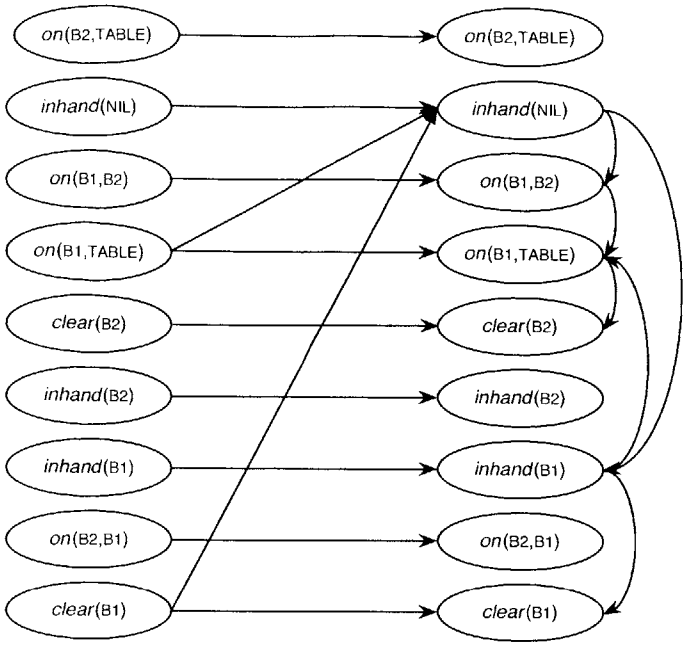

Figure 3-2: A DBN transition model encoding the effects of the $pickup(\text{B1}, \text{B2})$ action.

Step 1: Hidden Outcome Node

The first major difference between DBNs and propositional rules is the way that they encode the effects of actions. DBNs represent dependencies amongst their variables directly while propositional rules have a set of outcomes that mediate how the world changes.

Structure like the rules' outcomes can be added to a DBN by introducing a hidden node and fixing the CPMs of the nodes at time $t + 1$. This hidden *outcome node* has discrete values that correspond to the outcomes of a propositional rule (success, failure, etc.). The CPMs for the nodes at time $t + 1$ have a fixed deterministic structure: they take a value assigned by the outcome node, if there is one, or their values from time t. The result is an outcome DBN like the one shown in Figure 3-3. The CPM of the hidden node now has all of the uncertainty. The remaining links encode the frame assumption and the structure of the outcomes.

Step 2: Context Specific Independence

At first glance, it appears that DBNs require significantly more parameters to represent the world. For example, the outcome node in Figure 3-3 has a number of parameters that is

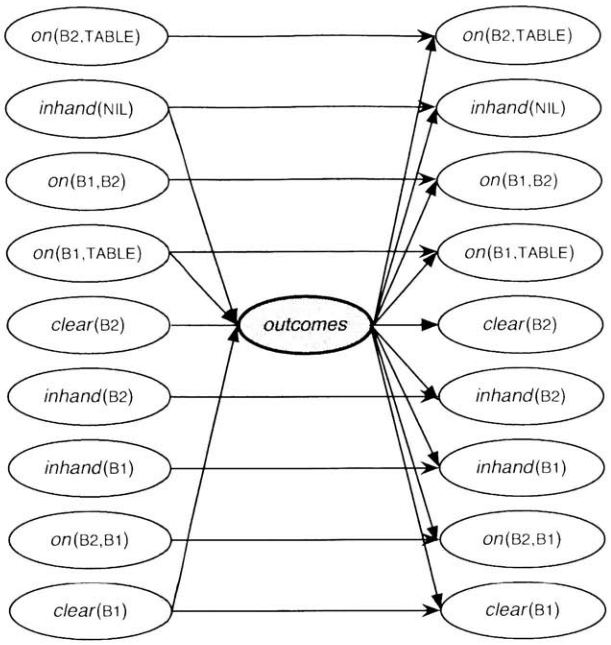

Figure 3-3: An outcome-node DBN encoding the $pickup(\text{B1}, \text{B2})$ action. The outcome node has three values, corresponding to the three distinct outcomes presented in Figure 3-1.

exponential in the number of parents it has, but the corresponding rules would only have one parameter for each outcome.

However, there is no need to explicitly represent all of the exponentially many contexts in the outcome node's CPM. Just like planning rules, BNs have been modified to encode *context specific independence* (CSI) (Boutilier, Friedman, Goldszmidt, & Koller, 1996). CSI allows the CPMs of a BN to be represented more compactly. Several alternative representations have been studied, including trees (Friedman & Goldszmidt, 1998), graphs (Chickering, Heckerman, & Meek, 1997), and rules (Poole, 1997).

The second step in transforming DBNs to propositional planning rules is to allow the DBNs to encode their CPMs with CSI rules. Figure 3-4 shows the set of CSI rules that encode the outcome DBN of Figure 3-3. Starting in the left column, the first rule has all of the probabilistic structure. The next three encode contexts where the action has no effect. The following eight encode how the first two outcomes change the world. All of the other rules encode the dynamics when the action doesn't change the world.

$$
\begin{array}{rcl}
on(\text{B1},\text{B2}), clear(\text{B1}), \\
inhand(\text{NIL}) & \rightarrow & \left\{ \begin{array}{l} .7 : O_1 \\ .2 : O_2 \\ .1 : O_3 \end{array} \right. \\
\neg on(\text{B1},\text{B2}) & \rightarrow & O_3 \\
\neg clear(\text{B1}) & \rightarrow & O_3 \\
\neg inhand(\text{NIL}) & \rightarrow & O_3 \\
O_1 & \rightarrow & inhand(\text{B1}) \\
O_1 & \rightarrow & \neg inhand(\text{NIL}) \\
O_1 & \rightarrow & \neg clear(\text{B1}) \\
O_1 & \rightarrow & \neg on(\text{B1},\text{B2}) \\
O_1 & \rightarrow & clear(\text{B2}) \\
O_2 & \rightarrow & on(\text{B1},\text{TABLE}) \\
O_2 & \rightarrow & \neg on(\text{B1},\text{B2}) \\
O_2 & \rightarrow & clear(\text{B2}) \\
O_3, on(\text{B1},\text{B2}) & \rightarrow & on(\text{B1},\text{B2}) \\
O_3, \neg on(\text{B1},\text{B2}) & \rightarrow & \neg on(\text{B1},\text{B2})
\end{array}
$$

$$
\begin{array}{rcl}
O_3, clear(\text{B2}) & \rightarrow & clear(\text{B2}) \\
O_3, \neg clear(\text{B2}) & \rightarrow & \neg clear(\text{B2}) \\
\neg O_1, inhand(\text{NIL}) & \rightarrow & inhand(\text{NIL}) \\
\neg O_1, \neg inhand(\text{NIL}) & \rightarrow & \neg inhand(\text{NIL}) \\
\neg O_1, clear(\text{B1}) & \rightarrow & clear(\text{B1}) \\
\neg O_1, \neg clear(\text{B1}) & \rightarrow & \neg clear(\text{B1}) \\
\neg O_1, inhand(\text{B1}) & \rightarrow & inhand(\text{B1}) \\
\neg O_1, \neg inhand(\text{B1}) & \rightarrow & \neg inhand(\text{B1}) \\
\neg O_2, on(\text{B1},\text{TABLE}) & \rightarrow & on(\text{B1},\text{TABLE}) \\
\neg O_2, \neg on(\text{B1},\text{TABLE}) & \rightarrow & \neg on(\text{B1},\text{TABLE}) \\
on(\text{B2},\text{TABLE}) & \rightarrow & on(\text{B2},\text{TABLE}) \\
\neg on(\text{B2},\text{TABLE}) & \rightarrow & \neg on(\text{B2},\text{TABLE}) \\
on(\text{B2},\text{B1}) & \rightarrow & on(\text{B2},\text{B1}) \\
\neg on(\text{B2},\text{B1}) & \rightarrow & \neg on(\text{B2},\text{B1}) \\
inhand(\text{B2}) & \rightarrow & inhand(\text{B2}) \\
\neg inhand(\text{B2}) & \rightarrow & \neg inhand(\text{B2})
\end{array}
$$

Figure 3-4: The context-specific, outcome-node rules for $pickup(\text{B1}, \text{B2})$.

Step 3: The Frame Assumption

The DBN rule set after Step 2 has a significant amount of deterministic structure. These deterministic rules fall into two categories. There are the rules that encode what happens for each outcome and the rules that represent the frame assumption. Because the frame assumption is built into the semantics of propositional planning rules, all of the DBN rules of the second type can be removed from the rule set. Then, combining the remaining rules yields:

$$
\begin{array}{rcl}
on(\text{B1},\text{B2}), clear(\text{B1}), inhand(\text{NIL}), \\
pickup(\text{B1},\text{B2}) & \rightarrow & \left\{ \begin{array}{ll} .7 : & inhand(\text{B1}), \neg clear(\text{B1}), \neg inhand(\text{NIL}), \\ & \neg on(\text{B1},\text{B2}), clear(\text{B2}) \\ .2 : & on(\text{B1},\text{TABLE}), \neg on(\text{B1},\text{B2}), clear(\text{B2}) \\ .1 : & \text{no change} \end{array} \right.
\end{array}
$$

which, for the simple blocksworld, is exactly equivalent to the the corresponding proba-

bilistic planning rule.

Step 4: The Arity of the Hidden Node

Although the first three steps were adequate for converting DBNs of the simple blocksworld into rules, in general, there is one more step that must be done when an action has more than one rule associated with it. Consider the more complicated $pickup(\text{B1}, \text{B2})$ below:

$$\begin{array}{rl} on(\text{B1},\text{B2}), clear(\text{B1}), inhand(\text{NIL}) \\ slippery(\text{B1}), pickup(\text{B1},\text{B2}) \end{array} \rightarrow \{ \quad 1: \text{ no change}$$

$$\begin{array}{rl} on(\text{B1},\text{B2}), clear(\text{B1}), inhand(\text{NIL}), \\ \neg slippery(\text{B1}), pickup(\text{B1},\text{B2}) \end{array} \rightarrow \begin{cases} .7: & inhand(\text{B1}), \neg clear(\text{B1}), \neg inhand(\text{NIL}), \\ & \neg on(\text{B1},\text{B2}), clear(\text{B2}) \\ .2: & on(\text{B1},\text{TABLE}), \neg on(\text{B1},\text{B2}), clear(\text{B2}) \\ .1: & \text{no change} \end{cases}$$

Because these two rules describe the same action, their transition dynamics would be encoded in a single DBN. Converting this DBN to rules, using steps 1-3, yields

$$\begin{array}{rl} on(\text{B1},\text{B2}), clear(\text{B1}), inhand(\text{NIL}), \\ slippery(\text{B1}), pickup(\text{B1},\text{B2}) \end{array} \rightarrow \begin{cases} 0: & inhand(\text{B1}), \neg clear(\text{B1}), \neg inhand(\text{NIL}), \\ & \neg on(\text{B1},\text{B2}), clear(\text{B2}) \\ 0: & on(\text{B1},\text{TABLE}), \neg on(\text{B1},\text{B2}), clear(\text{B2}) \\ 1: & \text{no change} \end{cases}$$

$$\begin{array}{rl} on(\text{B1},\text{B2}), clear(\text{B1}), inhand(\text{NIL}), \\ \neg slippery(\text{B1}), pickup(\text{B1},\text{B2}) \end{array} \rightarrow \begin{cases} .7: & inhand(\text{B1}), \neg clear(\text{B1}), \neg inhand(\text{NIL}), \\ & \neg on(\text{B1},\text{B2}), clear(\text{B2}) \\ .2: & on(\text{B1},\text{TABLE}), \neg on(\text{B1},\text{B2}), clear(\text{B2}) \\ .1: & \text{no change} \end{cases}$$

since the hidden outcomes node in the DBN has a fixed number of outcomes for all of the contexts.

This is less than optimal but easy to fix. The final step in our conversion process is to

remove all zero probability outcomes from the rules, thereby allowing each context specific rule to have a different number of outcomes.

The final result is propositional planning rules. This process shows us that propositional planing rules are DBNs with a hidden outcome node, CSI, and fixed deterministic structure to encode the frame assumption.

3.2.4 Summary: Matching Models to Worlds

The process of converting DBNs to propositional planning rules shows the representational differences between the two. Probabilistic planning rules are, in essence, specially structured BNs that are designed to encode PWS efficiently. In Chapter 5, we will see that this structure allows rules to be learned from fewer examples in worlds with PWS, which is especially important since they are meant to eventually be used by robots learning in the real world. But, we will also see that there are many other worlds, such as flipping n independent coins, where the more general DBNs are far superior. In principle, there is no reason why a future learning system couldn't attempt to diagnose which world it is in and choose its representation appropriately. This will be an interesting challenge for future work.

3.3 Relational Representations

In the relational case, abstract schemata generalize over the identity of objects in the world. The generalization allows models to more compactly represent worlds in which there are many objects that behave similarly. In this section, we will see two types of relational schemata: relational planning rules and probabilistic relational models (PRMs).

Relational planning rules (RPRs) are abstractions of propositional planning rules from Section 3.2.1 and PRMs generalize the Bayesian networks of Section 3.2.2. This section explores how RPRs and PRMs encode world dynamics; we will see that they both incorporate variable abstraction but that RPRs leverage PWS more effectively, just as we saw in the propositional case. We will also see that, as they are currently defined, PRMs are not as general as RPRs. Section 3.3.3 describes the representational limitations of PRMs and

ideas for how they might be extended so that a detailed comparison to RPRs would be possible.

3.3.1 Planning Rules

Relational planning rules were the focus of Chapter 2. The are, essentially, propositional planning rules that have variables that abstract over object identity. This allows single rules with actions such as *pickup*(X, Y), where X and Y can be any block in the world, to replace the large set of more specific rules (*pickup*(B1, B2), *pickup*(B2, B1), etc.), that must name the individual objects involved in the action. Relational planning rules encode the frame assumption and the action outcome assumption just like propositional rules, as described in Section 3.2.1. They also incorporate the PWS assumption of object abstraction directly in their use of variables. The final result is a representation that compactly models worlds with PWS, as in the original set of rules for blocks world from Figure 2-2.

3.3.2 Probabilistic Relational Models

The PRM formalism is the only probabilistic relational knowledge representation that has well established learning algorithms (Getoor, 2001). PRMs are, in essence, a way of writing templates over sets of possible BNs (Pfeffer, 2000). PRMs have two parts, the relational schema and the dependency model. This section describes these parts and then shows how to use them to estimate the likelihood of specific worlds.

PRMs encode distributions over worlds that are composed of objects that have simple attributes, discrete random variables like BNs; and reference attributes, random variables that encode relational structure by ranging over a discrete set of values that correspond to objects in the world. A PRM's *relational schema* lists all of the possible object types and the attributes that objects of each type contain. For example, a blocks world could be represented with a relational schema that has three types of objects: blocks, TABLES, and NILS. Blocks would have two simple boolean attributes, *clear* and *inhand*, and one reference attribute, *on*, that can refer to block objects or table objects. The NIL object would have a simple boolean attribute *inhand* to denote when the gripper is empty.

Uncertainty is included in a PRM through its *dependency model*. This model encodes conditional independence assumptions just like the link structure for BNs does. For each attribute $O.A$ in the relational schema, the model specifies a CPM that is conditioned on a set of parents $P_{O.A}$. Attributes in $P_{O.A}$ are restricted to other attributes of O and attributes of objects that can be reached by following a chain of reference attributes that starts in O. For example, in blocksworld, the dependence model for a block B would allow its attributes to depend on $B.clear$, $B.inhand$, $B.on$, and the attributes of objects that can be referenced by a chain of *on* attributes that begin with B ($B.on$, $B.on.on$, etc.).

A *skeleton* is a set of objects that are instances of the types defined in the PRM's relational schema. These objects can have some of their attributes fixed and others can be uncertain. PRMs have a well defined semantics because, given a skeleton, then can produce a *ground* Bayesian network, B, that defines the likelihood of all of the combinations of attribute values. In the case where reference attributes are not uncertain, B is easily constructed. It has a node for each simple attribute and the link structure that is specified directly in the dependency model. When there are uncertain reference attributes, constructing B is more complex and requires adding an extra node for each relational attribute and filling in a complex arc and CPM structure that properly updates uncertain slot chains. Understanding this complex case is not required for reading this thesis.

A Dynamic PRM (DPRM) (Sanghai, Domingos, & Weld, 2003) is built from a PRM much like a DBN was from a BN in Section 3.2.2. The worlds at time t and $t+1$ are represented as sets of objects and a special action object is introduced. Each object at time $t+1$ has a reference attribute *previous* that refers to the same object at time t and a reference attribute *action* that refers to the action object. The action object has reference attributes that specify the parameters of the action. Figure 3-5 shows the relational schema of a DPRM for the blocks world. Each box represents an object type. The attributes are fields in the boxes. Simple attributes have their range denoted in a set while reference attributes show the type of objects they can refer to. When DPRMs are used as a world model, all of the attributes of the actions, the objects at time t, and the previous and action attributes at time $t+1$ are given. The dependency model encodes a distribution over the uncertain attributes at time $t+1$.

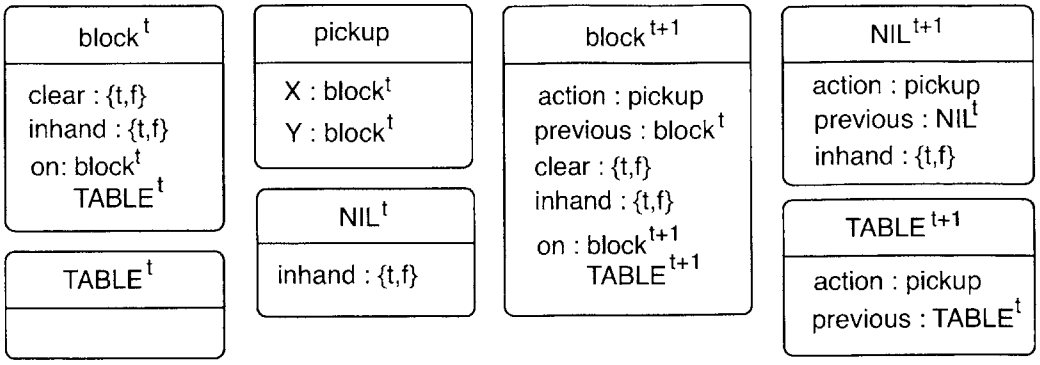

Figure 3-5: The DPRM relational schema for the blocks world.

3.3.3 Comparison

Ideally, the experiments in Chapter 5 would include a detailed comparison of PRM learning and RPR learning. This section highlights two interesting representational issues for PRMs that we discovered while trying to implement model learning with PRMs. The problems can definitely be solved, but, due to time constraints, this work will have to be done in the future.

Constant Object Dependencies

The first limitation of PRMs is that the parents of uncertain attributes in the dependency model must be selected by reference chains. There is no way to encode a dependency on an attribute that can not be reached by a reference chain. For example, the pickup action in blocks world requires that the gripper be empty. In the DPRM of Figure 3-5 this would be represented by the *inhand* attribute of the NIL object at time t having a true value. There are many attributes at time $t+1$, for example all attributes of blocks, that could depend on this property but do not have a reference chain that leads to the NIL object. To encode this dependency efficiently, the PRM would have to allow attributes to depend on the *inhand* attribute of the NIL object without a reference chain.

These *constant object dependencies* could be added to PRMs by a simple extension to the dependency model that allows parent attributes to be chosen by specifying the unique name of the objects they are a part of. However, some sort of strategy would be required

for handling worlds in which the model expects an object that isn't present. For example, these worlds could be declared impossible or multiple CPMs could be included, one for each possible missing constant object case.

Adding constant object dependency would also complicate learning. Since dependency models could include any objects ever observed, PRM learning would have to search through the much larger space of possible parent attributes.

The PIE Problem

The semantics of a PRM is defined by the Bayesian network that its dependency model creates for each possible world. However, there is a set of circumstances where PRMs, as currently defined in the literature, fail to produce a valid ground Bayesian network. This *parent identity equality* (PIE) problem can occur when the dependency model for an attribute allows it to have two different parents which, in some worlds, are the same attribute.

Consider the following simple PRM with no reference uncertainty. Its relational schema has two types of objects, people and books. People have one simple boolean attribute, fame. Books have three attributes. They have a simple boolean attribute, fame, and two relational attributes, author and editor. Both relational attributes refer to people. Part (a) of Figure 3-6 shows the relational schema for this PRM. Assume that the dependency model is some reasonable distribution over the book's fame given the fame of author and the fame of the editor. The prototypical world for this PRM has one book and two people, the book's editor and its author. The well-formed ground Bayesian network for this skeleton is illustrated in Figure 3-6 part (b).

But, consider what happens when the skeleton has only two objects, one book and one person, and the person serves as the book's author and editor. The PIE problem emerges. The standard grounding process creates in ill-formed Bayesian network something like the one in part (c) of Figure 3-6. Even if one of the arcs were omitted, there is no obvious way to fill the CPM for the book's fame. The dependency in this case is in no way related to the case where the author and editor are distinct.

The PIE problem doesn't appear in the PRM literature because objects rarely have two

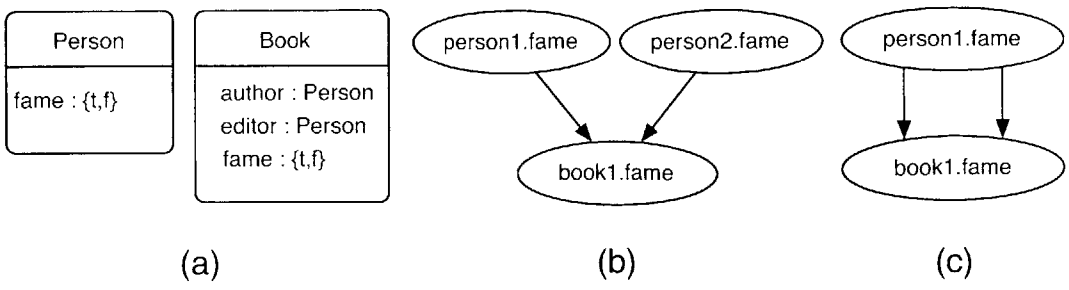

Figure 3-6: (a) The relational structure for the book fame domain. (b) A valid ground Bayesian network. (c) An invalid ground Bayesian network that demonstrates the PIE problem.

reference attributes that refer to the same type of object. However, in planning domains, objects of this type are common. For example, many actions have parameters that are the same type. Consider the following RPR rule for flipping two coupled coins

$$flip(X, Y) \rightarrow \begin{cases} 0.5: & heads(X), heads(Y) \\ 0.5: & \neg heads(X), \neg heads(Y) \end{cases}$$

The DPRM for this action will have an action object that introduces the PIE problem. For fun, the reader might verify this by constructing the ground Bayesian network for the DPRM corresponding to this action in a world with a single coin that fills both arguments to the action. You would see that it contains one node and one arc.

In RPRs, PIE-like problems are localized to the action parameters and avoided by step two of the rule selection semantics of Section 2.2.1 that forbids rules that are ill-formed when a single constant is bound to more than one parameter of the action.

There are several approaches that could be used to extend PRMs so that they are well defined when facing the PIE problem. PRMs could be augmented to encode an assumption that worlds with this reference structure are impossible. They could also encode a set of CPMs for all of the different possible unique sets of parent attributes.

Summary

Because the challenges above slowed our implementation, PRMs are not discussed in the experiments of Chapter 5. When the work is done to facilitate a comparison, the follow-

ing results seem likely for worlds with PWS. Since PRMs have variable abstraction, they should outperform Bayesian networks. Since RPRs leverage the other PWS assumptions, they should outperform PRMs, much like PPRs outperform Bayesian networks. It is hard to predict how PPRs would compare to PRMs, but variable abstraction would probably prove more influential.

However, this is all speculation. For now all that will be shown is that RPRs outperform PPRs, which outperform DBNs; PRMs are left out of the learning comparison entirely.

Chapter 4

Learning

This chapter presents an algorithm for learning probabilistic planning rules, given a training set $\mathbf{D} = D_1 \ldots D_{|\mathbf{D}|}$. Every example $D \in \mathbf{D}$ represents a single transition in the world, and consists of a previous state $D.\mathbf{S}^t$, an action $D.A^t$, and a next state $D.\mathbf{S}^{t+1}$. A rule applies to an example D if it *covers* $D.\mathbf{S}^t$ given $D.A^t$, using the definition of covers from Section 2.2.1. $\mathbf{D}_r$ is the set of examples that a rule r applies to. The algorithm learns a rule set $\mathbf{R}$ that defines the distribution for how the world changes when an action a is executed in it, $P(\mathbf{S}^{t+1}|\mathbf{S}^t, a, \mathbf{R})$.

The sections below are organized as follows. Section 4.1 describes the outer learning loop, *LearnRules*, which is a search through the space of rule sets. This outer loop assumes the existence of a subroutine *InduceOutcomes* which, when given a context and an action, learns the rest of the rule. *InduceOutcomes* is discussed in Section 4.2 and makes use of the subroutine *LearnParameters*, which learns a distribution over a given set of outcomes as presented in Section 4.3.

4.1 Rule Sets

LearnRules performs a greedy search in the space of proper rule sets, where a proper rule set for data set $\mathbf{D}$ is a set of rules $\mathbf{R}$ that has exactly one rule that is applicable to every example $D \in \mathbf{D}$ in which some change occurs and that doesn't have any rules that are applicable to no examples. The examples where no change occurs can be handled by the

frame assumption, so they do not need to have an explicit rule. This section first describes how *LearnRules* scores rule sets. It then describes how *LearnRules* uses *InduceOutcomes* to create a new rule, given an action and a context. Finally, the overall search algorithm is presented in which an initial rule set is created and search operators are used to search for the highest scoring rule set.

During search, *LearnRules* must decide which rule sets are the most desirable. This decision is made by scoring rule sets; those with a higher score are more desirable. *LearnRules* scores rule sets with the following function

$$S(R) = \sum_{D \in \mathbf{D}} \log(P(D.\mathbf{S}^{t+1} | D.\mathbf{S}^t, D.A, R)) - \alpha \sum_{r \in \mathbf{R}} PEN(r)$$

This scoring metric favors rule sets that assign high likelihood to the data and penalizes rule sets that are overly complex. α is a scaling parameter that is set to 0.5 in the experiments of Chapter 5. Cross validation on hold-out data or some other principled technique could also be used to set α. The complexity of a rule is computed with the following penalty term

$$PEN(r) = |r.C| + |r.\mathbf{O}|.$$

The first part of this term penalizes long contexts; the second part penalizes for having too many outcomes. *LearnRules* uses this penalty because it is simple and performed as well as every other penalty term that was tested in informal experiments.

In the learning discussion that follows, we will often want to evaluate how good a single rule is. Notice that the scoring function above can be rewritten as the sum over the scores of individual rules given by

$$S(r) = \sum_{D \in \mathbf{D}_r} \log(P(D.\mathbf{S}^{t+1} | D.\mathbf{S}^t, D.A, r)) - \alpha PEN(r).$$

Sine R is a proper rule set, these formulations are equivalent. The sets of examples, $\mathbf{D}_r$, that the rules in R apply to do not overlap and together include the entire training set $\mathbf{D}$.

Because *LearnRules* creates many new rules during search, it is important to understand how this happens. Whenever a new rule is created, the only thing that *LearnRules* has to

do is specify an action $r.A$ and a context $r.C$. These two parts of the rule select the set of training examples $\mathbf{D}_r$ that the rule could possibly be applicable to. When these examples are passed with the partially specified r to *InduceOutcomes*, this procedure searches for the set of outcomes, $r.\mathbf{O}$, and the distribution, $r.P_{\mathbf{O}}$, that maximizes $S(r)$. Because the rule set score factors into a sum of individual rule scores, *InduceOutcomes* can maximize $S(r)$ directly and not worry about the score of the entire rule set.

Given a dataset $\mathbf{D}_r$, *LearnRules* must first create an initial rule set to start the search. There are many different valid rule sets that could be used. The most general rule set which has a single rule for each action with no context and as many variables as possible, is one option. Another option is the most specific rule set that creates, for every unique $\mathbf{S}^t$, A^t pair in the data, a rule with $r.C = \mathbf{S}^t$ and $r.A = A^t$. However, in the experiments of Chapter 5, *LearnRules* uses a third, intermediate, approach. This approaches creates the most specific set of rules that include only positive literals in their contexts. This can be done by creating the maximally specific rule set, dropping all negative literals from all rule contexts and then removing redundant rules from the set. In informal experimentations in a variety of worlds, starting the search at this intermediate rule set converged to a solution, on average, more quickly than the other two. Since there was no clear difference in the quality of the solution, the intermediate staring point appears to be the best one. Formally exploring how to decide when different starting points are more appropriate is an important area for future work.

At every step of the search, *LearnRules* greedily finds and applies the operator that will increase the total rule set score the most. It uses four search operators which are based on the four basic syntactic operations used for rule search in Inductive Logic Programming (Lavrač & Džeroski, 1994). Each operator selects a rule r, removes it from the rule set, and creates one or more new rules by changing $r.C$ and $r.A$ and calling *InduceOutcomes*. Any new rules created are introduced back into the rule set in a manner than ensures the rule set remains proper. How this is done for each operator is described below.

There are two possible ways to generalize a rule: a literal can be removed from the context, or a constant can be replaced with a variable. The first generalization operator selects a rule from the rule set, creates a new rule by removing a literal from its context and

calling *InduceOutcomes*, and then adds this new rule into the set. The second generalization operator selects a rule and picks one of its constant arguments. Then, this operator invents a new variable and substitutes that variable for every instance of the original constant in the original rule. Finally, it calls *Induce Outcomes* to create a new rule that is added to the set.

When a generalized rule is introduced into a rule set, *LearnRules* must ensure that the rule set remains proper. Generalization may increase the number of examples covered by a rule, thereby making some of the other rules unnecessary. The new rule replaces these other rules, removing them from the set. However, this removal can leave some training examples with no rule, so new, maximally specific rules are created to cover them.

There are also two ways to specialize a rule. A literal can be added to the context or a variable can be replaced with a constant. The first specialization operator selects a rule from the rule set and picks a literal that is not in the rule's context. It then creates two new rules, one with a positive instance of the new literal added to the original context, and one with a negative instance added. These rules are created by, as usual, calling *InduceOutcomes* with the new contexts and the original action. They are then added to the rule set. The second specialization operator selects a rule and a variable that is present in this rule. A set of new rules is created, where each rule corresponds to a possible constant and has every instance of the original variable in the context and the action replaced with that constant. *Induce Outcomes* is called on these new context action pairs to complete the rules. They are then added to the rule set.

Whenever more specific rules are introduced into a rule set *LearnRules* must, again, ensure that it remains proper. This time the concern is that one of the new rules might not be applicable to any of the training examples. Rules that are applicable to no examples are not added to the rule set.

This operators, just like the original ILP operators that motivated them (Lavrač & Džeroski, 1994), are quite general. The specialization operators can be used to create any valid rule set from the most general rule set. Similarly, the generalization operators can build any valid rule set staring from the most specific rule set. Together, they allow the full space of valid rule sets to be searched no matter where the search begins.

LearnRules's search strategy has one large drawback; the set of rules which is learned

is only guaranteed to be proper on the training set and not on testing data. Solving this problem, possibly with approaches based on relational decision trees (Blockeel & Raedt, 1998), is an important area for future work. Its effectiveness is also limited by the efficiency of the *InduceOutcomes* subprocedure since it is called every time a new rule is constructed.

There are also other advanced rule set search operators, such as least general generalization (Plotkin, 1970), which might be modified to create operators that allow *LearnRules* to search the planning rule set space more efficiently.

4.2 Inducing Outcomes

Inducing outcomes is the problem of finding a set of outcomes $\mathbf{O}$ and parameters $P_{\mathbf{O}}$ which maximize the score of a rule r with context $r.C$ and action $r.A$ that applies to a set of examples $\mathbf{D}$. *InduceOutcomes* solves this problem with greedy search and uses the subroutine *EstimateParams*, presented in Section 4.3. Appendix A argues that inducing outcomes is NP-hard which justifies *InduceOutcomes*'s greedy search.

Consider the coins domain, which will be used to explain the learning algorithm. Each coins world contains n coins which can be showing either heads or tails. The action *flip-coupled*, which has no context and no parameters, flips all of the coins to heads half of the time and otherwise flips them all to tails. A set of training data for learning outcomes with two coins might look like part (a) of Figure 4-1 where $h(C)$ stands for $heads(C)$, $t(C)$ stands for $\neg heads(C)$, and $D.\mathbf{S}^t \rightarrow D.\mathbf{S}^{t+1}$ is an example with $D.A^t =$ *flip-coupled*.

InduceOutcomes searches through a restricted subset of possible outcome sets: those that are *valid* on the training examples, where an outcome set is valid if every training example has at least one outcome that covers it and every outcome covers at least one training example (using the definition of covers from Section 2.2.3). *InduceOutcomes* uses two operators, described below, to move through this space. For each set of outcomes it considers, *InduceOutcomes* calls *LearnParameters* to supply the best $P_{\mathbf{O}}$ it can. Search stops when there are no more immediate moves that improve the rule score.

InduceOutcomes creates an initial set of valid outcomes by, for each example, writing down the set of literals that changed values as a result of the action, and then creating an

$$
\begin{array}{lcl}
D_1 & = & t(c1), h(c2) \rightarrow h(c1), h(c2) \\
D_2 & = & h(c1), t(c2) \rightarrow h(c1), h(c2) \\
D_3 & = & h(c1), h(c2) \rightarrow t(c1), t(c2) \\
D_4 & = & h(c1), h(c2) \rightarrow h(c1), h(c2)
\end{array}
$$

(a)

$$
\begin{array}{lcl}
O_1 & = & \{h(c1)\} \\
O_2 & = & \{h(c2)\} \\
O_3 & = & \{t(c1), t(c2)\} \\
O_4 & = & \{\text{no change}\}
\end{array}
$$

(b)

Figure 4-1: (a) Possible training data for learning a set of outcomes. (b) The initial set of outcomes what would be created from the data in (a).

outcome to describe every set of changes observed in this way. In our running example, the initial set of outcomes has the four entries in part (b) of Figure 4-1.

InduceOutcomes then searches through the space of valid outcome sets using two operators.[1] The first is an add operator which picks a pair of outcomes in the set and adds in a new outcome based on their conjunction. For example, it might pick O_1 and O_2 from the running example and combine them, adding a new outcome $O_5 = \{h(c1), h(c2)\}$ to the set. The second is a remove operator that drops an outcome from the set. Outcomes can only be dropped if they were overlapping with other outcomes on every example they cover, otherwise the outcome set would not remain valid. In the outcomes of Figure 4-1 O_4 can be immediately dropped since it only covers D_4, which is also covered by both O_1 and O_2. If we imagine that $O_5 = \{h(c1), h(c2)\}$ has been added with the add operator, then O_1 and O_2 could also be dropped since O_5 covers D_1, D_2, and D_3. This would, in fact, lead to the optimal set of outcomes for the training examples in Figure 4-1.

It is worth recalling that the running example has no context and no action. Handling contexts and actions with constant parameters is easy, since they simply restrict the set of training examples the outcomes have to cover. However, when a rule has variables among its action parameters, *InduceOutcomes* must be able to introduce those variables into the appropriate places in the outcome set. This variable introduction is achieved by applying the inverse of the action substitution to each example's set of changes while computing the

[1] Whenever *InduceOutcomes* proposes a set of outcomes, the outcomes' parameters must be relearned by *LearnParameters*. Often, many of the outcomes will have zero probability. These zero probability outcomes are immediately removed from the outcome set since they don't contribute to the likelihood of the data and they add complexity. This optimization greatly improves the efficiency of the search.

initial set of outcomes. So, for example, if *InduceOutcomes* were learning outcomes for the action *flip(X)* that flips a single coin, our initial outcome set would be $\{O_1 = \{\text{h(X)}\}, O_2 = \{\text{t(X)}\}, O_3 = \{\text{no change}\}\}$ and search would progress as usual from there. *InduceOutcomes* introduces variables aggressively wherever possible, based on the intuition that if any of them should remain a constant, this should become apparent through the other training examples.

Notice that any outcome that *InduceOutcomes* might want to learn can be created with these search operators. Each literal in this outcome has to model a change that was present in some training example, otherwise it could be removed since the frame assumption models everything that does not change. This outcome also has to include every literal that changes in every example it covers, or it would not cover them. Together, these two facts guarantee that any outcome that would be useful for modeling a set of training examples can be created by combining the sets of literals that change state in the examples.

4.3 Learning Parameters

Given a rule r with a context $r.C$ and a set of outcomes $r.\mathbf{O}$, all that remains to be learned is the distribution over the outcomes, $r.P_{\mathbf{O}}$. *LearnParameters* learns the distribution that maximizes the rule score: this will be the distribution that maximizes the log likelihood of the examples $\mathbf{D}_r$ as given by

$$\begin{aligned} & \sum_{D \in \mathbf{D}_r} \log(P(D.\mathbf{S}^{t+1} | D.\mathbf{S}^t, C.A^t, r)) \\ = & \sum_{D \in \mathbf{D}_r} \log \left(\sum_{\{o | D \in \mathbf{D}_o\}} r.P_{\mathbf{O}}(o) \right) \end{aligned} \tag{4.1}$$

where $\mathbf{D}_o$ is the set of examples covered by outcome o, using the definition of covers from Section 2.2.3. When every example is covered by a unique outcome, the problem of minimizing L is relatively simple. Using a Lagrange multiplier to enforce the constraint that $r.P_{\mathbf{O}}$ must sum to 1.0, the partial derivative of L with respect to $r.P_{\mathbf{O}}(o)$ is then $|\mathbf{D}_o|/r.P_{\mathbf{O}}(o) - \lambda$, and $\lambda = |\mathbf{D}|$, so that $r.P_{\mathbf{O}}(o) = |\mathbf{D}_o|/|\mathbf{D}|$. The parameters can be

estimated by calculating the percentage of the examples that each outcome covers.

However, in general, the rule could have overlapping outcomes. In this case, the partials would have sums over os in the denominators and there is no obvious closed-form solution: estimating the maximum likelihood parameters is a nonlinear programming problem. Fortunately, it is an instance of the well-studied problem of minimizing a convex function over a probability simplex. Several gradient descent algorithms with guaranteed convergence can be found in Bertsekas (1999). *LearnParameters* uses the *conditional gradient method*, which works by, at each iteration, moving along the axis with the minimal partial derivative. The step-sizes are chosen using the Armijo rule (with the parameters $s = 1.0$, $\beta = 0.1$, and $\sigma = 0.01$.) The search converges when the improvement in L is very small, less than 10^{-6}. If problems are found where it converges too slowly, there are many other well-known algorithms that could converge more quickly (Bertsekas, 1999).

Chapter 5

Experiments

This chapter describes experiments that demonstrate that the learning algorithm in Chapter 4 is robust and show that the bias described in Chapter 3 improves learning effectiveness. Section 5.1 describes the experimental domains and how they were chosen. Section 5.2 presents results for inducing outcomes in isolation while Section 5.3 investigates learning whole rule sets.

5.1 Domains

The experiments in this chapter involve learning rules for the domains in the following sections. The unique characteristics of each domain showcase particular aspects of the learning procedures.

5.1.1 Coin Flipping

In the coin flipping domain, n coins are flipped using three atomic actions: *flip-coupled*, which, as described in Section 4.2, turns all of the coins to heads half of the time and to tails the rest of the time; *flip-a-coin*, which picks a random coin uniformly and then flips that coin; and *flip-independent*, which flips each of the coins independently of each other. Since the contexts of all these actions are empty, every ruleset contains only a single rule and the whole problem reduces to outcome induction. In Section 5.2 we will see how scaling the

number of coins effects outcome learning for each of these actions.

5.1.2 Slippery Gripper

The slippery gripper domain, inspired by the work of Draper et al. (1994), is a blocks world with a simulated robotic arm, which can be used to move the blocks around on a table, and a nozzle, which can be used to paint the blocks. Painting a block might cause the gripper to become wet, which makes it more likely that it will fail to manipulate the blocks successfully; fortunately, a wet gripper can be dried. Figure 5-1 shows the set of RPRs that model slippery grippers worlds.

Slippery gripper, like most blocks worlds, has a simple parameter that scales the world's complexity, the number of blocks. In Section 5.3, we will explore how the learning algorithms of Chapter 4 compare as the slippery gripper world is scaled in complexity given a fixed number of training examples and how they compare is the number of training examples is scaled in a single complex world.

5.1.3 Trucks and Drivers

Trucks and drivers is a logistics domains, originally from the 2002 AIPS international planning competition (AIPS, 2002), with four types of constants. There are trucks, drivers, locations, and objects. Trucks, drivers and objects can all be at any of the locations. The locations are connected with paths and links. Drivers can board and exit trucks. They can drive trucks between locations that are linked. Drivers can also walk, without a truck, between locations that are connected by paths. Finally, objects can be loaded and unloaded from trucks.

A set of PPR-like rules is shown in Figure 5-2. Most of the actions are simple rules which succeed or fail to change the world. However, the walk action has an interesting twist, represented by its final outcome. When drivers try to walk from one location to another, they succeed most of the time, but some of the time they arrive at a randomly chosen location that has a path to it from their origin location.

The walk action can't be represented efficiently as a set of RPRs. The best encoding

$$
\begin{array}{rcl}
on(X,Y), clear(X), inhand(\text{NIL}), \\
block(X), block(Y), \neg wet, pickup(X,Y)
& \rightarrow &
\begin{cases}
.8: & inhand(X), \neg clear(X), \neg inhand(\text{NIL}), \\
 & \neg on(X,Y), clear(Y) \\
.2: & on(X, \text{TABLE}), \neg on(X,Y), \neg inhand(\text{NIL}) \\
.1: & \text{no change}
\end{cases} \\
on(X,Y), clear(X), inhand(\text{NIL}), \\
block(X), block(Y), wet, pickup(X,Y)
& \rightarrow &
\begin{cases}
.33: & inhand(X), \neg clear(X), \neg inhand(\text{NIL}), \\
 & \neg on(X,Y), clear(Y) \\
.33: & on(X, \text{TABLE}), \neg on(X,Y), \neg inhand(\text{NIL}) \\
.34: & \text{no change}
\end{cases} \\
on(X,Y), clear(X), inhand(\text{NIL}), \\
block(X), table(Y), wet, pickup(X,Y)
& \rightarrow &
\begin{cases}
.5: & inhand(X), \neg clear(X), \neg inhand(\text{NIL}), \\
 & \neg on(X,Y), clear(Y) \\
.5: & \text{no change}
\end{cases} \\
on(X,Y), clear(X), inhand(\text{NIL}), \\
block(X), table(Y), \neg wet, pickup(X,Y)
& \rightarrow &
\begin{cases}
.8: & inhand(X), \neg clear(X), \neg inhand(\text{NIL}), \\
 & \neg on(X,Y), clear(Y) \\
.2: & \text{no change}
\end{cases} \\
clear(Y), inhand(X), block(Y), \\
puton(X,Y)
& \rightarrow &
\begin{cases}
.7: & inhand(\text{NIL}), \neg clear(X), \neg inhand(Y), \\
 & on(X,Y), clear(X) \\
.2: & on(X, \text{TABLE}), clear(X), inhand(\text{NIL}), \\
 & \neg inhand(X) \\
.1: & \text{no change}
\end{cases} \\
inhand(X), \\
puton(X, \text{TABLE})
& \rightarrow &
\begin{cases}
.8: & on(X, \text{TABLE}), clear(X), inhand(\text{NIL}), \\
 & \neg inhand(X) \\
.2: & \text{no change}
\end{cases} \\
block(X), paint(X)
& \rightarrow &
\begin{cases}
.6: & painted(X) \\
.1: & painted(X), wet \\
.3: & \text{no change}
\end{cases} \\
dry
& \rightarrow &
\begin{cases}
.9: & \neg wet \\
.1: & \text{no change}
\end{cases}
\end{array}
$$

Figure 5-1: Eight relational planning rules that model the slippery gripper domain.

has a rule for each origin location and each rule has an outcome for every location that the origin is linked to. This action is difficult to learn but, as we will see in Section 5.3, can be learned with enough training data. Extending the RPR representation to allow actions like walk to be represented as a single rule is an interesting area for future work.

Unlike slippery gripper, trucks and drivers does not have a single parameter that can be used to scale the complexity of the world. Instead, in Section 5.3, we will only see the effects of scaling the number of training examples on the learning algorithms of Chapter 4 in a single, complex trucks and drivers world.

$$
\begin{aligned}
at(T,L), at(O,L), load(O,T,L) &\rightarrow \begin{cases} .9 : \neg at(O,L), in(O,T) \\ .1 : \text{ no change} \end{cases} \\
in(O,T), at(T,L), unload(O,T,L) &\rightarrow \begin{cases} .9 : at(O,L), \neg in(O,T) \\ .1 : \text{ no change} \end{cases} \\
at(T,L), at(D,L), empty(T), \\ board(D,T,L) &\rightarrow \begin{cases} .9 : \neg at(D,L), driving(D,T), \neg empty(T) \\ .1 : \text{ no change} \end{cases} \\
at(T,L), driving(D,T), disembark(D,T,L) &\rightarrow \begin{cases} .9 : \neg driving(D,T), at(D,L), empty(T) \\ .1 : \text{ no change} \end{cases} \\
driving(D,T), at(T,FL), link(FL,TL), \\ drive(T,FL,TL,D) &\rightarrow \begin{cases} .9 : at(T,TL), \neg at(T,FL) \\ .1 : \text{ no change} \end{cases} \\
at(D,FL), path(FL,TL), walk(D,FL,TL) &\rightarrow \begin{cases} .9 : at(D,TL), \neg at(D,FL) \\ .1 : \begin{array}{l} \text{pick } X \text{ s.t. } path(FL,X) \\ at(D,X), \neg at(D,FL) \end{array} \end{cases}
\end{aligned}
$$

Figure 5-2: Six rules that encode the world dynamics for the trucks and drivers domain.

5.2 Inducing Outcomes

Before we investigate learning full rule sets, consider how the *InduceOutcomes* subprocedure performs on some canonical problems in the coin flipping domain.

In order to evaluate *InduceOutcomes* in isolation, a rule was created with an empty context and passed to *InduceOutcomes*. Table 5.1 contrasts the number of outcomes in the initial outcome set with the number eventually learned by *InduceOutcomes*. In these experiments 300 randomly created training examples were provided.

Notice that, given n coins, the optimal number of outcomes for each action is well defined. *flip-coupled* requires 2 outcomes, *flip-a-coin* requires $2n$, and *flip-independent* requires 2^n. In this sense, *flip-independent* is an action that violates our basic structural assumptions about the world, *flip-a-coin* is a difficult problem, and *flip-coupled* behaves like the sort of action we expect to see frequently. The table shows that *InduceOutcomes* can learn the latter two cases, the ones it was designed for, but that actions where a large number of independent changes results in an exponential number of outcomes are beyond its reach. An ideal learning algorithm might notice such behavior and choose to represent it with a factored model.

	Number of Coins				
	2	3	4	5	6
flip-coupled begin	7	15	29.5	50.75	69.75
flip-coupled end	2	2	2	2	2
flip-a-coin begin	5	7	9	11	13
flip-a-coin end	4	6.25	8	9.75	12
flip-independent begin	9	25	47.5	-	-
flip-independent end	5.5	11.25	20	-	-

Table 5.1: The average changes in the number of outcomes found while inducing outcomes in the n-coins world. Results are averaged over four runs of the algorithm. The blank entries did not finish running in reasonable amounts of time.

5.3 Learning Rule Sets

Now that we have seen that *InduceOutcomes* can learn rules that don't require an exponential number of outcomes, this section investigates how *LearnRules* performs.

The experiments are divided between two types of comparisons. Section 5.3.1 shows that propositional rules can be learned more effectively than DBNs while Section 5.3.2 shows that relational rules outperform propositional ones.

These comparisons are performed with four actions. The first two, paint and pickup, are from the slippery gripper domain while the second two, drive and walk, are from the trucks and drivers domain. Each action presents different challenges for learning. Paint is a simple action that has overlapping outcomes. Pickup is a complex action that must be represented by more than one planning rule. Drive is a simple action which has four parameters. And, walk is the most complicated action since there is no single set of planning rules that model it, as described in Section 5.1.3.

All of the experiments use examples, $D \in \mathbf{D}$, generated by randomly constructing an initial state $D.\mathbf{S}^t$ and then executing the action, $D.A^t$, in that state to generate $D.\mathbf{S}^{t+1}$. The distribution of $D.\mathbf{S}^t$'s in the training data is biased to guarantee that in approximately half of the examples $D.A^t$ could possibly change $D.\mathbf{S}^t$. This method of data generation is designed to ensure that the learning algorithms will always have data which is representative of the entire model that they should learn. Thus, the experiments in this chapter ignore the problems an agent would face if it had to generate data by exploring the world.

After training on a set of training examples D, the models are tested on a set of test examples E by calculating the average *variational distance* between the true model P and an estimate $\hat{P}$ given by:

$$VD(P, \hat{P}) = \frac{1}{|\mathbf{E}|} \sum_{E \in \mathbf{E}} |P(E) - \hat{P}(E)|.$$

Variational distance is a natural measure because it favors similar distributions and is well-defined when a zero probability event is observed, as can happen when a rule is learned from sparse data and doesn't have as many outcomes as it should.

5.3.1 Comparison to DBNs

Traditionally, *Dynamic Bayesian Networks* (DBNs) have been used to learn world dynamics. To compare *LearnRules* to DBN learning, variable abstraction is forbidden, thereby forcing the rule sets to remain propositional during learning. The BN learning algorithm of Friedman and Goldszmidt (1998), which uses decision trees to represent its conditional probability distributions, is compared to this restricted *LearnRules* algorithm in Figure 5-3.

Notice the propositional rules consistently outperform DBNs. Although it looks like the DBNs were never able to learn reasonable models, this is not the case. During some of the trials the DBNs performed well, even beating the rules on an occasional training set. [[n out of 10]] But, in many of the trials, DBNs performed horribly. The failures were all instances of the structure search getting stuck in local optima. It is not surprising that the DBNs were more susceptible to local optima since they had to introduce a large number of links to get the proper structure.

5.3.2 The Advantages of Abstraction

The second set of experiments demonstrates that when *LearnRules* is able to use variable abstraction, it outperforms the propositional version. Figure 5-4 shows that the full version consistently outperforms the restricted version as the number of examples is scaled. Also, observe that the performance gap grows with the number of parameters that the action has.

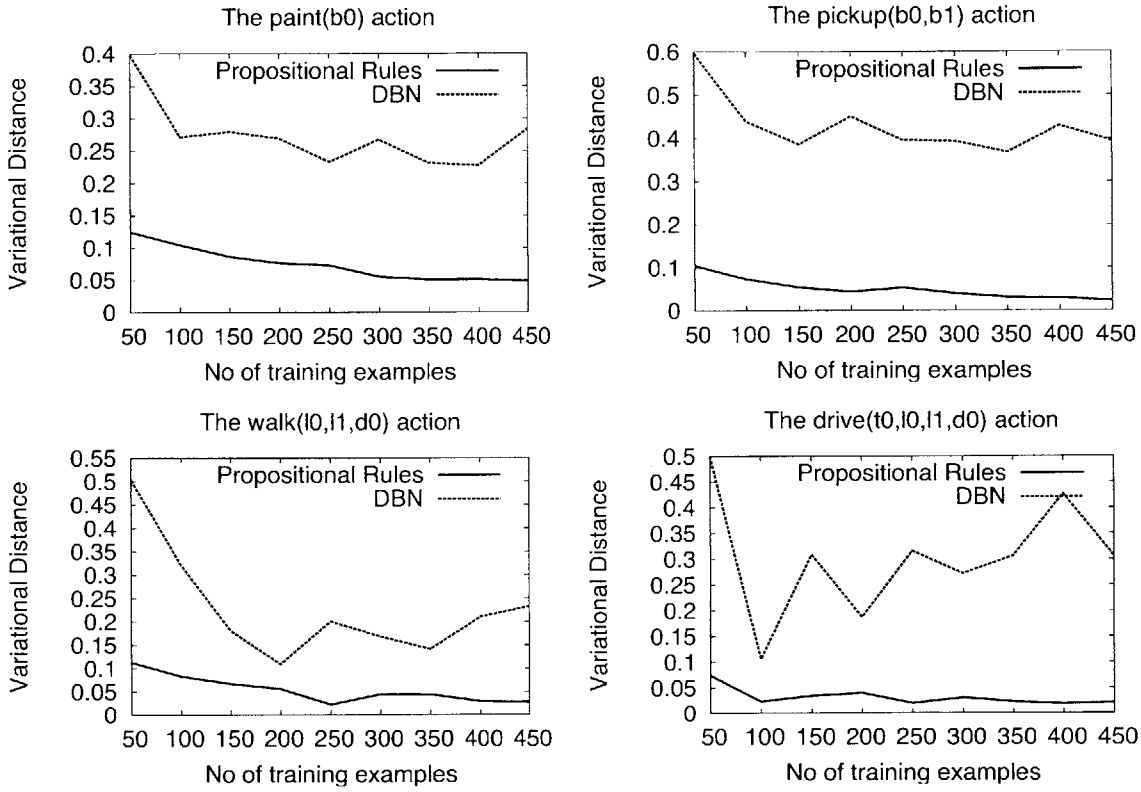

Figure 5-3: Variational distance as a function of the number of training examples for DBNs and propositional rules. The slippery gripper actions were performed in a world with four blocks. The trucks and driver actions were performed in a world with two trucks, two driver, two objects and four locations. The results are averaged over ten trials of the experiment. The test set size was 300 examples.

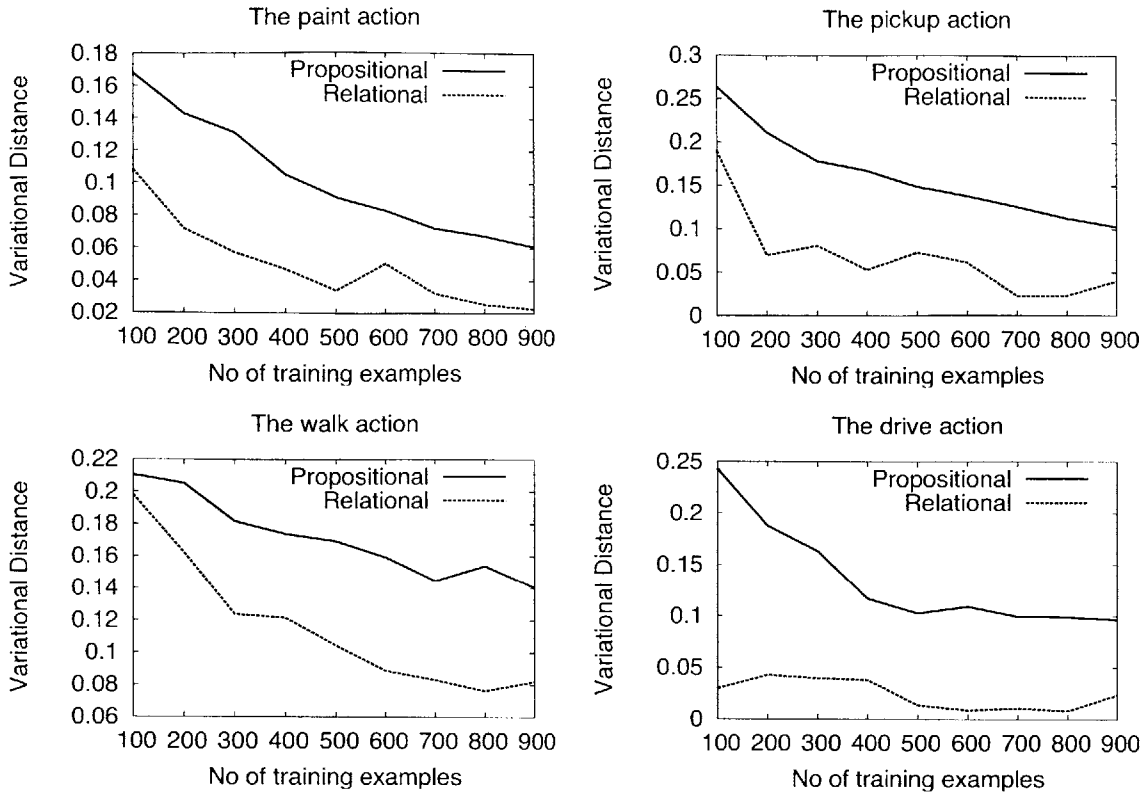

Figure 5-4: Variational distance as a function of the number of training examples for propositional and relational rules. The slippery gripper actions were performed in a world with four blocks. The trucks and driver actions were performed in a world with two trucks, two driver, two objects and four locations. The results are averaged over ten trials of the experiment. The test set size was 400 examples.

This result should not be particularly surprising. The abstracted representation is significantly more compact. Since there are fewer rules, each rule has more training examples and the abstracted representation is significantly more robust in the presence of data sparsity. Another view of the same phenomena is seen in Figure 5-5, where the number of blocks is scaled and the number of training examples is held constant.

5.3.3 Discussion

The experiments of this chapter should not be surprising. Planning rules were designed to efficiently encode the dynamics of the worlds used in the experiments. If they couldn't outperform more general representations and learning algorithms, there would be a serious problem.

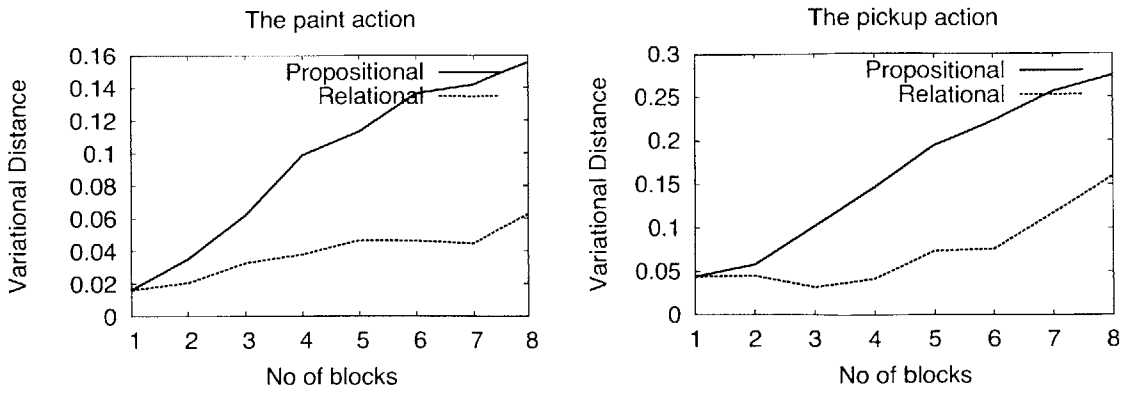

Figure 5-5: An illustration of how the variational distance of the models learned scales with the size of the world, for propositional and relational rules. The experiments were conducted with 500 training examples and 500 test examples. The results are averaged over ten runs of the experiment.

However, these experiments are still an important validation that *LearnRules* is a robust algorithm that does leverage the bias that it was designed for. Because no other algorithms have been designed with this bias, it would be difficult to demonstrate anything else.

Chapter 6

Related Work

There has been essentially no work in learning probabilistic relational planning rules.

The closest work on probabilistic representations and learning is in the Bayesian network and Probabilistic Relational Models community. All of Chapter 3 was devoted to understanding how this work relates to this thesis.

Deterministic rule learning has been explored significantly in the literature. The most relevant example is that of Shen and Simon (1989) and others in the ILP community who have learned deterministic relational rules from data.

There are only two examples of probabilistic rule learning, both of which are significantly different from the work of this thesis. Oates and Cohen (1996) learn propositional probabilistic planning operators. Each rule has a single outcome and multiple rules can apply in parallel, making the model more like a DBN than the planning rules of this thesis. Benson (1996) learns relational action models in continuous worlds that can tolerate noise and execution errors but do not directly model probabilistic action outcomes.

Chapter 7

Conclusions and Future Work

The results of Chapter 5 shows that biasing representations towards the structure of the world they will represent significantly improves learning. The natural next question is how do we bias robots so they can learn in the real world? Extending the work done here to more appropriately model the unique dynamics of our world is an important challenge that can be approached in many ways.

Planning operators exploit an general principle in modeling agent-induced change in world dynamics: each action can only have a few possible outcomes. In the simple examples from this thesis, this assertion was exactly true in the underlying world. In real worlds, this assertion may not be exactly true, but it can be a powerful approximation. If we are able to abstract sets of resulting states into a single generic "outcome," then we can say, for example, that one outcome of trying to put a block on top of a stack is that the whole stack falls over. Although the details of how it falls over can be very different from instance to instance, the import of its having fallen over is essentially the same. We will investigate the concurrent construction of abstract state characterizations that allow us to continue to model actions as having few outcomes.

An additional goal in this work is that of operating in extremely complex domains. In such cases, it is important to have a representation and a learning algorithm that can operate incrementally, in the sense that it can represent, learn, and exploit some regularities about the world without having to capture all of the dynamics at once. This goal originally contributed to the use of rule-based representations. Developing an incremental learning

algorithm that learns models of a world while living in it is am important are for future work.

A crucial further step is the generalization of these methods to the partially observable case. Again, we cannot hope to come up with a general efficient solution for the problem. Instead, algorithms that leverage world structure should be able to obtain good approximate models efficiently.

All of these improvements would lead towards to more general goal of building a representation that will run in a robot and allow it to quickly learn how to survive in our world.

Appendix A

Inducing Outcomes is NP-hard

Consider a set of training examples D, a scalar $\alpha > 0$, and a set of outcomes $\mathbf{O_D}$ that each cover at least one $D \in \mathbf{D}$. Inducing outcomes is the problem of finding the set of outcomes $\mathbf{O} \subseteq \mathbf{O_D}$ and the distribution over these outcomes $P_\mathbf{O}$ that maximize the scoring function $\log L(\mathbf{D}|r) - \alpha|\mathbf{O}|$, subject to the constraint that $|\mathbf{O}| \leq |\mathbf{D}|$.

This formulation of the inducing outcomes problem is different than the one in Chapter 4, but they are essentially the same. In Chapter 4 $\mathbf{O_D}$ was not discussed. It was implicitly assumed that $\mathbf{O_D}$ is the entire set of possible outcomes that cover at least one $D \in \mathbf{D}$. Notice that this is the hardest possible $\mathbf{O_D}$ since all others are subsets of it. The constraint that $|\mathbf{O}| \leq |\mathbf{D}|$ was also not present in Chapter 4. It is required for the following proof and could possibly be relaxed. However, assuming $|\mathbf{O}| \leq |\mathbf{D}|$ is not a large compromise since solutions with $|\mathbf{O}| > |\mathbf{D}|$ are not reasonable for use in planning rules as they were described in this thesis.

The minimal subset problem (Garey & Johnson, 1979) can be reduced to inducing outcomes. This proof first presents a reduction that ignores the likelihood term and directly minimizes the number of outcomes. Then, it shows that the likelihood is bounded and presents an α that will always make the solution with the smallest number of outcomes maximize the overall rule score.

Given a finite set S, and a set C of subsets of S, the minimal subset problem is to find the smallest subset C′ of C such that every element in S belongs to at least one member of C′. Given a minimal subset problem, an induce outcomes problem is created as follows.

The reduction creates a training example D_s for each $s \in \mathbf{S}$. The states in each D_s have predicates p_s for each $s \in \mathbf{S}$. In each D_s, p_s change state from false in $D_s.\mathbf{S}^t$ to true in $D_s.\mathbf{S}^{t+1}$ and other $p \neq p_s$ are set true in both states. Now, for each subset $\mathbf{C}' \in \mathbf{C}$, a possible outcome $O_{\mathbf{C}'} \in \mathbf{O_D}$ is created with positive predicates that correspond to every $s \in \mathbf{C}'$. Notice that the outcomes correspond directly to the subsets and the training examples correspond directly to the elements of $\mathbf{S}$. If a set of outcomes covers the training examples, the corresponding subsets will cover the original set elements. So, finding the smallest set of outcomes is equivalent to finding the smallest subset. Since the conversion can be done in time polynomial in the size of the original minimal subset problem, finding the smallest set of outcomes is NP-hard.

Now, we need to find an α that guarantees that minimizing the number of outcomes will maximize the overall score. First, notice that there is a unique maximum log likelihood for each possible set of outcomes. This term could range from zero to negative infinity. However, we will see that it is, in fact, never less than $|D| \log\left(\frac{1}{|D|}\right)$. This bound ensures that setting $\alpha > -|D| \log\left(\frac{1}{|D|}\right)$ guarantees that whenever the number of outcomes is decreased the overall score will always increase.

First, for any $r.\mathbf{O}$, notice that the likelihood resulting from any setting of the parameters $r.P_{\mathbf{O}}$ is a lower bound on the maximum likelihood. If we set $r.P_{\mathbf{O}}$ to the uniform distribution, then each training example's likelihood will be at least $\log\left(\frac{1}{|O|}\right)$, since there must be at least one $O \in r.\mathbf{O}$ that covers it. Now, because we assumed that $|\mathbf{O}| \leq |\mathbf{D}|$, this also implies that the likelihood of each example is at least $\log\left(\frac{1}{|D|}\right)$, and that the maximum likelihood solution is always greater than or equal to $|D| \log\left(\frac{1}{|D|}\right)$.

In summary, we have just seen that we can set α to guarantee that minimizing the number of outcomes will maximize the score. We also saw that minimizing the number of outcomes is NP-hard. So, we can conclude the inducing outcomes is NP-hard.

Bibliography

AIPS (2002). International planning competition.. http://www.dur.ac.uk/d.p.long/competition.html.

Benson, S. (1996). *Learning Action Models for Reactive Autonomous Agents*. Ph.D. thesis, Stanford.

Bertsekas, D. P. (1999). *Nonlinear Programming*. Athena Scientific.

Blockeel, H., & Raedt, L. D. (1998). Top-down induction of first-order logical decision trees. *Artificial Intelligence*.

Blum, A., & Langford, J. (1999). Probabilistic planning in the graphplan framework. In *ECP*.

Boutilier, C., Friedman, N., Goldszmidt, M., & Koller, D. (1996). Context specific independence in bayesian networks. In *UAI*.

Boutilier, C., Dearden, R., & Goldszmidt, M. (2002). Stochastic dynamic programming with factored representations. *Artificial Intelligence*.

Boutilier, C., Reiter, R., & Price, B. (2001). Symbolic dynamic programming for first-order mdps. In *IJCAI*.

Brooks, R. A. (1991). Intelligence without representation. *AIJ*.

Chickering, D., Heckerman, D., & Meek, C. (1997). A Bayesian approach to learning Bayesian networks with local structure. In *UAI*.

Dean, T., & Kanazawa, K. (1988). Probabilistic temporal reasonin. In *AAAI*.

Draper, D., Hanks, S., & Weld, D. (1994). Probabilistic planning with information gathering and contingent execution. In *AIPS*.

Fikes, R. E., & Nilsson, N. J. (1971). Strips: A new approach to the application of theorem proving to problem solving. *AIJ*.

Friedman, N., & Goldszmidt, M. (1998). Learning Bayesian networks with local structure. In *Learning and Inference in Graphical Models*.

Garey, M. R., & Johnson, D. (1979). *Computers and Intractability: A guide to the theory of NP-completeness*. Freeman.

Getoor, L. (2001). *Learning Statistical Models From Relational Data*. Ph.D. thesis, Stanford.

Lavrač, N., & Džeroski, S. (1994). *Inductive Logic Programming Techniques and Applications*. Ellis Horwood.

Oates, T., & Cohen, P. R. (1996). Searching for planning operators with context-dependent and probabilistic effects. In *AAAI*.

Pasula, H., Zettlemoyer, L., & Kaelbling, L. (2003). Learning probabilistic planning rules. In *Submitted for publication*.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufman.

Penberthy, J. S., & Weld, D. (1992). Ucpop: A sound, complete partial-order planner for adl. In *KR*.

Pfeffer, A. (2000). *Probabilistic Reasoning for Complex Systems*. Ph.D. thesis, Stanford.

Plotkin, G. (1970). A note on inductive generalization. *Machine Intelligence*.

Poole, D. (1997). Probabilistic partial evaluation: Exploiting rule structure in probabilistic inference. In *IJCAI*.

Sanghai, S., Domingos, P., & Weld, D. (2003). Dynamic probabilistic relational models. In *IJCAI*.

Shen, W.-M., & Simon, H. A. (1989). Rule creation and rule learning through environmental exploration. In *IJCAI*.

Smart, W. D., & Kaelbling, L. P. (2000). Practical reinforcement learning in continous spaces. In *ICML*.