Learning Probabilistic Relational Dynamics for Multiple Tasks

by

Ashwin Deshpande

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

Masters of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2007
[June 2007]

Author .....
Department of Electrical Engineering and Computer Science
May 25, 2007

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

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

Learning Probabilistic Relational Dynamics for Multiple Tasks

by

Ashwin Deshpande

Submitted to the Department of Electrical Engineering and Computer Science
on June 2, 2007, in partial fulfillment of the
requirements for the degree of
Masters of Engineering in Computer Science and Engineering

Abstract

While large data sets have enabled machine learning algorithms to act intelligently in complex domains, standard machine learning algorithms perform poorly in situations in which little data exists for the desired target task. Transfer learning attempts to extract trends from the data of similar source tasks to enhance learning in the target task. We apply transfer learning to probabilistic rule learning to learn the dynamics of a target world. We utilize a hierarchical Bayesian framework and specify a generative model which dictates the probabilities of task data, task rulesets and a common global ruleset. Through a greedy coordinated-ascent algorithm, the source tasks contribute towards building the global ruleset which can then be used as a prior to supplement the data from the target ruleset. Simulated experimental results in a variety of blocks-world domains suggest that employing transfer learning can provide significant accuracy gains over traditional single task rule learning algorithms.

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

Acknowledgments

I would like to thank my advisor Leslie Kaelbling and my collaborators Brian Milch and Luke Zettlemoyer for guiding me through the research process.

Bibliographic Notes

The bulk of this thesis is based on the following paper:

"Learning Probabilistic Relational Dynamics for Multiple Tasks" by Ashwin Deshpande, Brian Milch, Luke Zettlemoyer, and Leslie Kaelbling. The paper will appear in the *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence (UAI 2007).*

Contents

List of Figures

Chapter 1

Introduction

In the past twenty years, a trend towards accumulating large amounts of information has pushed artificial intelligence to rely more and more on data driven techniques. Standard machine-learning techniques using large data sets have proved very successful in a variety of domains from natural language processing to planning.

However, traditional machine learning techniques adapt poorly to new distorted tasks even if the model of the original tasks are well known. Transfer learning attempts to boost performance in new target tasks with little task-specific data by incorporating knowledge gained from similar source tasks.

This thesis describes the methods of applying transfer learning to probabilistic rule learning. Probabilistic rule learning is the process of learning a ruleset that describes the probabilistic effects of performing actions in a world. For example, consider a robot trying to learn the dynamics in the domain of driving a car. For a particular task, for example, a particular car, the robotic can learn that if the accelerator is pushed, the car will go faster; if the brake is pushed, the car will most likely decelerate, but due to the possibility of unforeseen circumstances like ice, with some probability, braking may not affect the speed of the car. Under single-task probabilistic rule learning, each of these actions, accelerating and braking, requires a large body of examples to learn the structure and parameters of the associated ruleset from scratch. If the task is changed, for example, a new car is introduced, the dynamics must be completely relearned.

Transfer learning reduces the need for a large training set in the target task by using structural and parametric data gathered in related source tasks as a prior for the target task's ruleset. Going back to the driving example, if the robot has previously seen how several models of cars operate, if a new car with a different weight is introduced, the robot can transfer knowledge about the structure and parameters of its driving model from the previous cars to the model of the new car to quickly approximate its behavior. Similarly, even if the dynamics of the new car are structurally different than previously seen examples, for example, the robot is presented with a manual transmission car when it has only been previously exposed to automatic transmission cars, the robot can adapt common notions of car dynamics, whether manual or automatic, to learning the dynamics of the new car with less data than it would require to learn the dynamics from scratch. Overall, transfer learning can greatly reduce the need for target specific data when general domain specific data will suffice to fill in the information gaps.

Our model uses a hierarchical Bayesian structure to transfer knowledge from the source tasks to the target task. All data is assumed to have been created from a local ruleset which in turn is created from a common global ruleset via a specified generative model. Using this framework, the data in the source tasks can help build a model of the associated local rulesets and ultimately a common global ruleset. The global ruleset is then used as a prior on the target task and combined with support from examples in the target task to create the target task ruleset.

Section 2 describes some related work in both hierarchical Bayesian models and probabilistic rule learning. Next, section 3 states the world representation and the generative model which ties together the global ruleset, local rulesets, and examples. Section 4 describes the optimization strategy used to approximate the global ruleset and local rulesets. Section 5 displays the results of the hierarchical rule learning algorithm. Finally, section 6 describes possible future work in extending the algorithm and concludes the thesis.

Chapter 2

Related Work

Transferring knowledge between different tasks has been addressed in the past by hierarchial Bayesian models [5]. In these methods, local models, describing task-specific examples, are assumed to be sampled from a global model. By learning both the local and global models simultaneously, one can boost commonalities among the local models and increase the overall accuracy of local models with little data. This approach has been successfully used in a variety of statistical domains[1, 3, 6, 9, 11]. However, there is no previous work which uses a hierarchical Bayesian model to the structure of relational rules between tasks.

The notion of probabilistic planning rules was introduced by Kushmerick[4] and successfully used in probabilistic planning using STRIPS rules by Blum and Langford[2]. Zettlemoyer et. al developed an algorithm to learn the probabilistic planning rules for a single task[8, 10]. In that framework, a heuristic function, rather than a generative model, is used to score the complexity of a ruleset. Incremental changes to the ruleset are proposed and chosen if the combined complexity and error is lowered. This method was shown to be successful on a variety of both hand-crafted and simulated domains. This thesis borrows many concepts from these works and extends the framework of the learning algorithms to allow for knowledge transfer between tasks.

Chapter 3

Representation and Generative Model

3.1 Representation

This section describes the representation of states, actions, and rules in the world. Many of the concepts defined below are borrowed from Pasula et. al[8].

An example in a particular task is a triplet (S_{t+1}, A_t, S_t) where S describes a world state consisting of a number of objects and their properties and A describes an action. Thus, an example shows the effects of performing action A_t in the world described by S_t and the resulting state S_{t+1}. The ultimate goal of the problem is to learn a ruleset that effectively describes the probabilistic effects of A in a given task.

We start by defining a language F of a set of functions. Each function f maps an ordered list of f_{args} arguments to a discrete value in the finite set f_{values}. We define a literal l to be a triplet $(f, [args], value)$ where f is a function, $[args]$ is an ordered list of objects, variables, and constants, and $value$ is a element from f_{values}. In addition, we define a formula to be a set of non-duplicate and non-contradictory literals.

We assume that the world is completely observable and all observations can be trusted. Given these assumptions, we define a world state S to be a complete formula that describes the world. Thus, for every function $f \in F$ and every set of objects $[objs]$ of size f_{args}, there exists a single literal of the form $(f, [objs], value)$ where

$value \in f_{values}$. In effect, the difference between any two world states is always in the values of functions rather than in the structure of the literals themselves.

An action A effects a change in the world and thus has the potential to change the values of literals within the world state S. The action itself contains an ordered list of arguments which is filled with different objects from the world[1]. In the case of multiple actions, each action is learned independently of all other actions. Thus, for the scope of this thesis, we examine only one action per domain.

A local rule is defined as a triplet (A, C, O) where A is an action, C describes the context, or precondition, of the rule and O describes the various outcomes associated with the rule. C is defined by a formula. O is a set of outcomes $[o_1, ..., o_k]$ each represented by a formula representing possible changes in the world and a parameter $o_{i,prob}$ which describes the probability of outcome o_i being chosen. In our model, we require that no two outcomes in O overlap, that is, for any world state S_t, the resulting world states $S_{t+1,1}$ and $S_{t+1,2}$ created by applying two local outcomes $o_i, o_j \in O$ must differ.[2] O also contains a special outcome o_{noise} which indicates the probability that the resulting world state is unpredictable or too complicated for the rule to learn specific outcomes.

A ruleset consists of a set of non-overlapping local rules.[3] Thus, in this model, for any world state S and action A, at most one rule from the local ruleset can apply. In addition, every ruleset contains a special rule called the *default rule* which applies when no other local rules apply. The default rule contains only two outcomes: the noise outcome and a *no change* outcome. The inclusion of the default rule greatly simplifies the complexity of rulesets which describe tasks in which actions frequently

[1]The requirement for non-repeating objects in the action arguments is not strictly necessary. However, in order to transfer knowledge between rules in various domains, maintaining a one-to-one mapping of world objects to variables allows rules to be directly compared. If one-to-one mapping was not required, then comparing two rules would an exponential time to search over all semantically equivalent variations of each literal. For example, if $arg1$ and $arg2$ refer to the same object, then a literal $(f, [arg1, arg2], val)$ will also need to be represented as $(f, [arg1, arg1], val)$, $(f, [arg2, arg2], val)$, and $(f, [arg2, arg1], val)$ for comparison.

[2]This is a requirement of the optimization strategy used rather than of the model. This requirement can be removed by introducing variational inference techniques into the optimization strategy.

[3]This requirement, like that of non-overlapping outcomes, is a requirement of our optimization strategy rather than the framework itself. However, due to computational requirements, we enforced a strict non-overlapping policy for local rulesets.

do not change the world state.

Finally, the last items to be described are global rules and the global ruleset. The global ruleset and global rules serve as priors for the various local rulesets and local rules. The global model concepts are very similar to those of local rules and local rulesets, but with a few important differences explained below.

Global rulesets do not require non-overlapping rules. As global rules serve as the prior for local rules, by our generative model, syntactically similar local and global rules are more likely to correspond and ultimately transfer information. While this choice led to greater optimization requirements, in the end, we felt the extra transfer gained by allowing overlapping global rules was more important. Note that identical global rules are still not permitted.

In addition, global rules do not require non-overlapping outcomes. We made the choice to allow outcomes to overlap in global rules as the global rule serves as a prior for local rules. Thus, constraining the global rule's outcomes to not overlap may produce a new global rule that is semantically equal but transfers knowledge poorly to local rules. As before, identical global outcomes are not permitted.

Global rules also have one additional outcome over local rules: the *new outcome.* The new outcome allows local rules to create new outcomes completely different than anything existing in the global rule. The generative model section will describe the use of the new outcome.

Finally, global outcomes do not have outcome probabilities like local outcomes. Instead, a global outcome o_i has an associated Polya hyperparameter $o_{i,param}$. The next sections will describe how this hyperparameter is used in the generative model.

3.2 Generative Model

This section describes the hierarchical model used to transfer information between tasks.

The rule learning framework is set up as a three-tier hierarchy as shown in Fig. 3-1. In this model, there are n source tasks $t_1, ..., t_n$ and a single target task t_{n+1}.

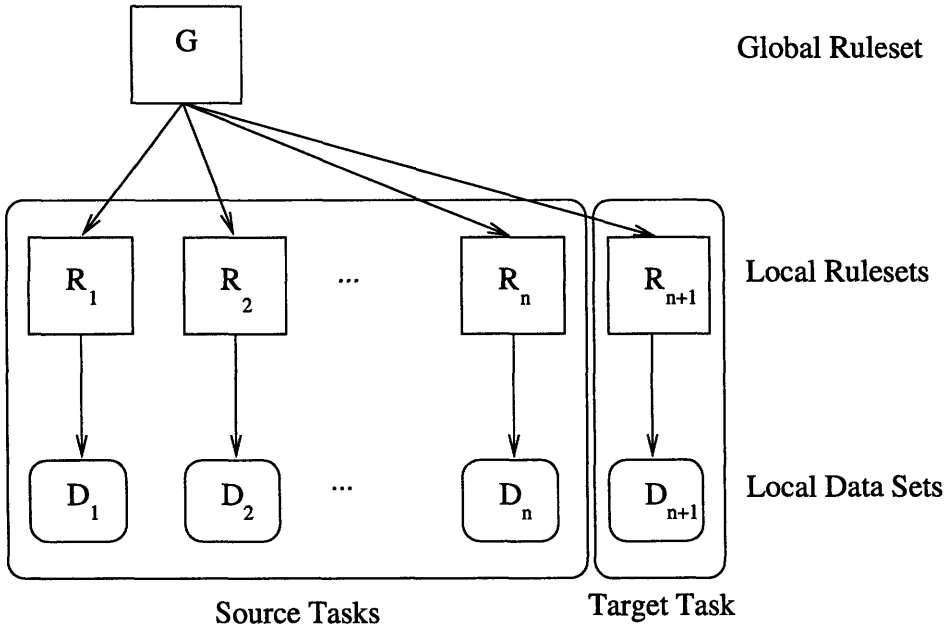

Figure 3-1: A hierarchical Bayesian model with n source tasks and 1 target task.

Each task t_i has its own data set D_i. Each task data set D_i is derived from a local ruleset R_i. Finally, all local rulesets $R_1, ..., R_{n+1}$ are derived from a global ruleset G.

We describe a generative model that can give the probability of deriving a local ruleset R_i from the global ruleset G and the probability of deriving a data set D_i from a local ruleset G. Although we never actually use the generative model to generate local rulesets and data sets, we use the probabilities derived from the model to shape the local and global rulesets.

In the generative model, it is assumed that all examples derived from a local ruleset are independent of all other examples given the local ruleset. Furthermore, we assume that all local rulesets are independent of each other given the global ruleset. Thus, for clarity, we only describe how a particular data set D is derived from a local ruleset R, and how R is derived from the global ruleset G. Finally, we end the section by describing the generative model for the global ruleset G.

3.2.1 Generating Data Sets from Local Rulesets

In this section, we describe how a data set D is generated from a local ruleset R. Each example E in D is generated independently given R. Thus, the probability of creating an data set D from a local ruleset R is $p(D) = \prod_{E \in D} p(E|R)$. Therefore, in essence, this section will describe how E is generated from R and how to compute $p(E|R)$.

The first step involves creating an initial world state S_t and an action A_t. Each initial world state is created independently of all other world states; therefore, the initial world states do not follow a trajectory. S_t is generated by populating a world with various objects. The world state is then specified as a complete formula as described previously. As the ruleset R defines a conditional probability distribution, the initial state of the world is not of critical importance. However, in order to sample over the example space thoroughly, a roughly uniform distribution on initial world states is preferred. Following the creation of S_t the action A_t is specified by picking various world objects to satisfy the action arguments. Thus, the initial world state S_t and action A_t can be specified.

The next step involves applying the ruleset R to the initial world state S_t and action A_t to generate the resulting world state S_{t+1}. Rulesets are used to generate future world states as described below. Fig. 3-2 demonstrates the steps described below for a sample world and ruleset. For each rule $U \in R$, the applicability of the rule must be assessed. If no standard rules apply, then the default rule applies. Assume that a particular rule U is tested for applicability. First, a one-to-one mapping Ω of world objects and constants in S to action arguments in the context of U is calculated. If no such Ω exists, i.e. no mapping of world objects to action arguments will satisfy all literals in the context of U, then the rule U cannot be used to predict the subsequent state. If a mapping does exist, then the outcomes of U describe the predicted future world state. Each outcome o_j is chosen with probability $o_{j,prob}$. Assume outcome o^* is ultimately chosen. The inverse map of Ω is calculated and applied to formula associated with o^* to create a formula of changes $f^*_{changes}$. Finally, the resulting world

$$
\begin{aligned}
S_t: &\ On(Block_A, Block_B) = true, Clear(Block_A) = true, \\
&\ Clear(Block_B) = false, Size(Block_A) = Size2, Size(Block_B) = Size1 \\
A_t: &\ Pickup(Block_A, Block_B) \\
U: &\ Pickup(arg_1, arg_2) \\
&\ On(arg1, arg2) = true \quad \rightarrow \begin{cases} .18: & Size(arg1) = Size1 \\ 82: & Size(arg1) = Size2, On(arg1, arg2) = false \end{cases} \\
\Omega: &\ Block_A \rightarrow arg1, Block_B \rightarrow arg2 \\
o^*: &\ Size(arg1) = Size2, On(arg1, arg2) = false \\
\Omega^{-1}: &\ arg1 \rightarrow Block_A, arg2 \rightarrow Block_B \\
f^*_{changes}: &\ Size(Block_A) = Size2, On(Block_A, Block_B) = false \\
S_{t+1}: &\ On(Block_A, Block_B) = false, Clear(Block_A) = true, \\
&\ Clear(Block_B) = false, Size(Block_A) = Size2, Size(Block_B) = Size1
\end{aligned}
$$

Figure 3-2: This figure describes how a local ruleset can be applied to a world state and action to produce the subsequent world state.

state S_{t+1} is as a copy of $S_t \cup f^*_{changes}$ in which all duplicate or contradictory literals are resolved in favor of $f^*_{changes}$. The triplet (S_t, A_t, S_{t+1}) forms the final example.

To compute the $p(E|R)$, or the probability of generating a example given the local ruleset, one need only retrace the generative process. As all steps of the generative model are deterministic until an outcome o^* is chosen, the probability of an example utilizing utilizing outcome o^* is simply o^*_{prob}.

3.2.2 Generating Local Rulesets from Global Rulesets

In this section, we describe how a local ruleset R is generated from a global ruleset G.

The first step in creating a local ruleset is to choose the number of rules to use. As the default rules are always present in both the local and global ruleset, we do not count their presence in the number of rules in a ruleset. Let m_R be the size of the local ruleset and m_G be the size of the global ruleset. We bias the local ruleset to have a similar number of rules as the global ruleset. Therefore, we define $p(m_R|m_G)$ to be

large when m_R is close to m_G and small when the number of rules differs radically. To achieve a smooth distribution, we use the following probability function:

$$p(m_R|m_G) = \begin{cases} \text{Geom}[\alpha](m_R - m_G) & \text{if } m_R > m_G \\ (1-\alpha)\text{Binom}[m_G, \beta](m_R) & \text{otherwise} \end{cases} \quad (3.1)$$

With this distribution, if $m_R > m_G$, we use a geometric distribution parameterized by a constant α. Thus, the probability mass of $p(m_R > m_G) = \alpha$. The remaining probability mass, $1 - \alpha$, is assigned if $m_R \leq m_G$. This distribution is a binomial distribution parameterized by a different constant β. For any $0 < \alpha, \beta < 1$, this distribution provides a bias for local rulesets to have a similar number of rules as the global ruleset.

Once the number of rules is chosen, each local rule is created almost independently. The only constraint between the rules is that the final local ruleset must not have overlapping rules. This constraint is enforced by discarding any created ruleset with overlapping rules. Although this strategy places a bias towards creating local rules with complex contexts (as rules with long contexts are less likely to overlap), it does provide a near uniform bias. Finally, as the rules in the ruleset can be permuted in any order and still yield the same overall ruleset, a factor of $m_R!$ is added. Thus, letting $p(U_i|G)$ be the probability of creating rule i, the final probability for generating a ruleset of size m_R is:

$$p(R|G) = m_R! * p(m_R|m_G) * \sum_{i=1}^{m_R} p(U_i|G) \quad (3.2)$$

It is helpful to now discuss how a formula is created given a formula in global ruleset as a prior, as this process is used in the creation of both the context and outcomes of a local rule. Let the formula of the local rule be f_L and the formula of the global rule be f_G. Creating f_L requires two phases. In the first phase, all literals from f_G are copied into f_L. For each literal in f_L, with some probability α_{remove}, that literal is removed from f_L. Let m_{remove} be the size of f_L after this phase. The second

phase involves adding literals to f_L. First, the number of literals to add m_{new} is chosen from a geometric distribution parameterized by a constant α_{add}. Given, m_{new}, each literal is chosen independently. For each literal l_{new} to be added, a function is uniformly chosen from the language F. Let the number of arguments accepted by l be l_{arg}. Next, each argument is randomly chosen without repeats from the set of action arguments and constants T_{arg}. If the created formula contains any repeated or contradictory functions, it is resampled. While in theory, we must integrate over the possibility of removing and readding a particular literal, as this probability is negligible, we only observe the probability of the most likely generation pathway from f_G to f_L. This yields the overall probability of creating f_L from f_G as:

$$p(f_L|f_G) = \text{Binom}[m_G, \alpha_{remove}](m_{remove}) * \text{Geom}[m_{new}, \alpha_{add}] * m_{new}! * \prod_{l \in l_{new}} \binom{l_{arg}}{T_{arg}} \frac{1}{|F|} \tag{3.3}$$

We will now describe how a local rule U is derived from the global ruleset G. This process requires four steps: first, a corresponding global rule must be chosen; then, the local rule context must be determined; next, the local outcomes are chosen; finally, the local outcome distribution is fixed.

The first step of creating U is to choose a corresponding global rule E from G. With a constant probability q_{new} no existing global rule is chosen, and the global new rule set to be the parent. With the remaining probability mass, $1 - q_{new}$, a E is chosen uniformly from all rules in G. Note that the global default rule is never allowed to be the corresponding rule for any local rule except the local default rule. Once the corresponding rule is chosen, the context for U can be easily created by the formula modification procedure given above.

The next step is to choose outcomes for U. This procedure is very similar to choosing local rules given the global ruleset. First, the number of outcomes m_{out} is chosen from a probability distribution specified in equation 3.1. Each outcome o_i is mapped to a corresponding global outcome y_i from E. With some constant probability $q_{o,new}$, a local outcome is mapped to the global new outcome. Otherwise,

the remaining probability mass is equally distributed among the global outcomes. As before, only the local noise outcome maps to the global noise outcome. Once the outcome correspondences have been determined, the local outcomes are generated by perturbing the global outcome by the formula modification procedure. All local outcomes mapping to the global new outcome are created from an empty global formula using the same method. Thus, the overall probability of the outcomes O given E is:

$$p(O|E) = m_{out}! * \prod_{o_i \in O} p(y_i|E)p(o_i|y_i) \tag{3.4}$$

The last step is to choose the outcome distribution for the local outcomes O given the global outcome distribution. This involves sampling from a Dirichlet distribution to determine the probability of each local outcome. However, the hyperparameters specified in the global outcome distribution may be unsuitable to sample from, as many local outcomes might correspond to a single global outcome and no local outcomes might correspond to some global outcomes. Thus, another set of pseudo-hyperparameters must be created with a pseudo-hyperparameter for each local outcome. Let W be the original set of global outcome hyperparameters and α be the new set of pseudo-hyperparameters.

The first case to consider is if no local outcome maps to a particular global outcome i. In this case, we define the generative model to not include the hyperparameter for i, effectively shrinking down the dimensionality of the problem. Note that the value of the removed hyperparameter is inconsequential as it is no longer a part of the probability distribution.

The second, and more interesting, case to consider is when one or more local outcomes $o_1, ..., o_a$ map to a single global outcome with hyperparameter W_j. We made the decision to set the sum of the expected probabilities of $o_1, ..., o_a$ to the same expected probability of a single hypothetical outcome o^*.[4] The expected probability

[4]We also entertained the possibility of letting each outcome in $o_1, ..., o_a$ receive the full prior support of W_j. However, this would imply that other outcomes would effectively lose support from their global hyperparameter priors. We decided to use our current method as it provides more probability transitions when splitting across functions (mapping multiple local outcomes to a single

for o^* when all outcome probabilities are sampled from the Dirichlet distribution is:

$$E[o^*] = \frac{W_j}{\sum_{W_i \in W} W_i} \tag{3.5}$$

Thus, to set the expected probability of each outcome $o_1, ..., o_a$ to be $1/a$ of the quantity specified above requires that:

$$\forall o_i \in \{o_1, ..., o_a\}, \alpha_i = \frac{W_j}{a} \tag{3.6}$$

Once all the local hyperparameters are chosen, the local outcome probabilities are sampled from a Dirichlet with hyperparameter vector α. Let B be the vector of sampled local probabilities. Then, the probability of creating an local outcome distribution is simply:

$$p(B|W) = \text{Dir}[\alpha](B) \tag{3.7}$$

Here $\text{Dir}[\alpha](b)$ is simply the probability of generating B from the Dirichlet hyperparameters α. Note that the process of generating α from W is deterministic, and thus has probability 1.

By following the steps listed above, a local ruleset can be derived from a global ruleset. While no local rulesets are ever created using the generative model, the optimization strategy does use the generative model's ability to provide the probability of creating a local ruleset from a global ruleset.

3.2.3 Generating Global Rulesets

The last section of the generative model is the generative model to create a global ruleset G. As the complexity of each local ruleset generated by G is most completely determined by the complexity of G, the generative model for G serves to bias the entire hierarchical model away from complexity and over-fitting.

The generative model for creating a global ruleset G is very similar to that of

global outcome).

creating a local ruleset. First, the number of rules m_{rules} is chosen from a geometric distribution with parameter $\alpha_{g,rule}$. Given m_{rules}, each rule U is independently created. If two rules have *exactly* the same context, then the ruleset is resampled. Overlapping rules are permitted. Thus, the probability for creating a global ruleset G is:

$$p(G) = m_{rules}! * \prod_{U \in G} p(U) \tag{3.8}$$

To select a context for a sample global rule U, three steps are required: first the context of the rule is determined, then the outcomes are determined, and last the outcome distribution is picked.

The generative process is far simpler for creating a global rule than a local rule as the corresponding rule prior is assumed to be empty. For each rule U, the context $U_{context}$ uses the formula modification procedure starting with an empty formula. Next, the number of outcomes m_{outs} is sampled using equation 3.1 using 0 outcomes as the prior. Each outcome o_i in the global outcome set O is created from scratch using the formula modification procedure. If two outcomes are *exactly* the same, the the rule must be resampled (overlapping outcomes are permitted in global outcome sets). In our current generative model, we do not have any prior on outcome hyperparameters. Thus, the outcome hyperparameters are chosen as positive real numbers. Overall, the probability of creating a rule U is:

$$p(U) = p(U_{context}|nil) * m_{outs}! * \prod_{o_i \in O} p(o_i|nil) \tag{3.9}$$

By using the methods above, one can find the probability of an entire hierarchial model given the global ruleset, local rulesets, and data sets. The following section will describe the optimization strategy used to maximize the overall probability of the entire model.

Chapter 4

Optimization Strategy

4.1 Overview

This section describes how the hierarchical framework and generative model are used as a basis to create local rulesets for the various local tasks and a global ruleset which ties them all together.

By the problem formulation, the input to the algorithm is a set of examples from both the sources tasks and the target task. Let tasks $t_{1...N}$ represent the sources tasks and task t_{N+1} represent the target task. Let D_i represent the data from the ith task. The quantity of interest is the probability of the target data D_{N+1} given the source data $D_{1...}N$. However, by the generative model, the probability of a set of data D_i can be calculated given a local ruleset R_i, and the probability of a local ruleset R_i can be calculated given the global ruleset G. Furthermore, by the generative model, all data sets are independent given their corresponding local rulesets, and all local rulesets are independent given the global ruleset. Thus, working in log space:

$$logp(D_{N+1}|D_{1...N}) \propto \int_{G,R_{1...N+1}} logp(G) + \sum_{i=1}^{N+1} logp(D_i|R_i)logp(R_i|G) \qquad (4.1)$$

Since a single target ruleset is the output of the optimization, the most probable local ruleset R_{N+1} can be selected from the above equation leading to the new

optimization equation:

$$R_{N+1} = \arg\max_{R_{N+1}} \int_{G,R_{1...N}} logp(G) + \sum_{i=1}^{N+1} logp(D_i|R_i) + logp(R_i|G) \qquad (4.2)$$

However, the above optimization quantity is extremely complex as each local ruleset R_1 to R_N and the global ruleset G consist of multiple rules, each with a context, outcomes, and an outcome distribution. A great deal of this complexity can be removed if the integration over all local rulesets and global rulesets is replaced with choosing the most likely estimate of each ruleset leading to the new optimization quantity:

$$R_{N+1} = \arg\max_{R_{N+1}} \max_{G,R_{1...N}} logp(G) + \sum_{i=1}^{N+1} logp(D_i|R_i) + logp(R_i|G) \qquad (4.3)$$

However, this quantity is still too complex to optimize directly. To approximately optimize this, we use a coordinated ascent algorithm. Using this method, the global ruleset is held constant while each local ruleset from source tasks is optimized. Thus, for a task i, the optimization using a constant global ruleset $\hat{G}$ is:

$$R_i = \arg\max_{R_i} logp(D_i|R_i) + logp(R_i|\hat{G}) \qquad (4.4)$$

The second step of the coordinated ascent algorithm estimates the global ruleset. To do this, all local rulesets $R_{1...N}$ from source tasks are held constant. The ruleset from the target task is not used in the optimization of the global ruleset because, as specified by the problem formulation itself, the target task has little data and cannot contribute much to the global ruleset. Thus, for constant local source rulesets $\hat{R_{1...N}}$, the optimization for the global ruleset is:

$$G = \arg\max_{G} logp(G) + \sum_{i=1}^{N} logp(\hat{R}_i|G) \qquad (4.5)$$

The global ruleset is calculated by alternately performing the first optimization over the local rulesets from source tasks and the second optimization over the global ruleset. When all rulesets stabilize, the first optimization is used one last time on the

target task to create the target task ruleset.

The global ruleset is initialized to be empty except for the default rule which is initialized with small Polya hyperparameters to allow local rulesets to easily create task specific rules. Each local ruleset is initialized to be empty of all rules except the default rule.

4.2 Learning a Local Ruleset given a Global Ruleset

This section describes how a local ruleset is created from task data and a fixed global ruleset. At the highest level, a greedy search is run which suggests the addition, deletion, or modification of rules in the local ruleset. Given each potential suggestion, the optimal outcomes and outcome distribution is acquired through another greedy search process. High level ruleset changes are proposed and applied greedily until no further modifications to the ruleset lead to a higher ruleset probability.

4.2.1 Ruleset Modification

The highest-level greedy search in optimizing a local ruleset proposes the addition, deletion, and modification of rules in the local ruleset. In the case of additions or modifications to rules, only the rule(s) context needs to be specified as the outcomes and outcome distribution are determined by a secondary greedy search. There are five search operators which propose changes:

- **Add Rule** - This operator proposes that a new rule be added to the local ruleset. This rule context can be derived either by copying a rule context verbatim from the global ruleset or by extracting all functions from a local example which deal with an object that is modified through the rule's action. In order to maintain overlapping rules, all rules which do not have a contradictory function in the context are removed from the ruleset.

- **Remove Rule** - This operator simply removes a rule from the local ruleset. All local examples explained by the removed rule must then be explained by the default rule.

- **Add Function** - This operator adds a function to the context of an existing rule (provided the function is not already present in the context and no existing function in the context contradicts the new function).

- **Remove Function** - This operator removes a function from the context of an existing rule. If the modified rule overlaps with any other rules in the ruleset, these rules are removed.

- **Split Function** - This operator breaks apart a single existing rule into multiple rules. Each new rule has the same context as the old rule with the addition of an additional function that has identical arguments across all new rules but differing values. Note that the set of new rules covers the same example space as the single existing rule.

In each case, if examples become explained or unexplained through the modification of the ruleset, the default rule must be updated. As the default rule does not learn new outcomes, it learns a new outcome distribution through parameter estimation which is described later.

With each search operator, the parent rule of each affected local rule must be recalculated through the generative model. As an approximation to allowing any local rule to be derived from any global rule, we define the parent rule of each local rule to be the global rule which has the most similar context according the to global rule. In addition to this parent rule, we also calculate the probability of the local rule being generated from scratch. In order to find the probability of the outcome set and outcome distribution of each local rule, a second greedy search is needed in order to create a set of outcomes.

4.2.2 Outcome Set Modification

When a rule enters the outcome set modification greedy search, it has a single outcome: the noise outcome. Thus, all examples which are described by the rule are considered noise, leading to a terrible rule score. Thus, the rule has a great incentive to learn new outcomes and reduce the dependence on the noise outcome.

As before, there are multiple search operators that are used when building an outcome set. In this search, a set of candidate outcomes are constructed before the greedy search is started. Candidate outcomes include all outcomes from the corresponding global rule and all outcomes which can explain a local example. The operators include:

- **Remove Outcome** - This operator removes a outcome from the outcome distribution. All local examples explained by the removed outcome are then delegated to the noise outcome.
- **Add Function** - This operator adds a function to an existing outcome (as before, no duplicate or contradictory outcomes are allowed).
- **Merge Outcomes** - This operator merges the context of an existing outcome and a candidate outcome by taking the union of their outcome changes and removing duplicate functions. This operator is only valid when the existing outcome and candidate outcome do not contradict one another.
- **Split Function** - This operator breaks apart a single outcome into multiple outcomes as in the ruleset "Split Function" operator. In addition to the context of the old outcome, each newly proposed outcome contains an additional common function with different values.
- **Add Outcome*** - This operator proposes that a new outcome be added to the outcome set. The proposed outcome is chosen from the candidate outcomes.
- **Remove Function*** - This operator removes a function from an existing outcome.

As before, any new unexplained examples are delegated to the noise outcome. The starred operators above can potentially create overlapping outcomes. Thus, whenever either of those operators are proposed, a secondary greedy search is run to ensure that the outcomes remain non-overlapping.

In the secondary search, all overlapping outcomes are noted in a pairwise manner. Next, for each pair of outcomes, a set of functions which contradict one of the pair is proposed to be appended to the second of the pair, which results in the pair not overlapping. Finally, the function proposal which leads to the greatest number of non-overlapping outcomes is chosen. In certain cases, the secondary search modifies two outcomes to be identical. If such a deadlock is found, one of the offending outcomes is removed. The secondary search is repeated until all outcomes do not overlap.

After the search operators listed above are proposed, the probability of the entire rule and outcome set can be calculated by the generative model. The only missing probability is the probability of the outcome distribution given the global outcome distribution which is described next.

4.2.3 Learning the Outcome Distribution

After a set of local outcomes has been chosen, the corresponding global rule acts as a prior for the outcome distribution. The process of sampling from a rule probability distribution from the Dirichlet parameters of a global rule and sampling examples from the subsequent probability distribution specified in the generative model can be modeled by the Polya distribution[7].

This framework requires that each local outcome be mapped to the most likely global outcome as an approximation to calculating all pathways of deriving the local outcome set from the global outcome set. This mapping can be calculated by finding the closest global outcome to each local outcome specified by the generative ruleset. In some cases, the "new" outcome in the global ruleset may be closer to a local outcome than an existing global outcome. By using this method, many local outcomes may map to the same global outcome, and some global outcomes may not have any mappings.

Given a one-to-one mapping of local outcomes to global outcomes, the outcome probabilities can be calculated via the Polya distribution. Let b_k be the probability of the kth local outcome, K be the total number of local outcomes, n_k be the number of examples supporting outcome k, and α_k be the Polya hyperparameter for outcome k. Then, the Polya distribution is:

$$p(n_1...n_K|\alpha_1...\alpha_K) \propto \int_{b_1...b_K} \frac{\Gamma(\sum_{i=1}^{k} \alpha_i)}{\prod_{i=1}^{K} \Gamma(\alpha_i)} \prod_{k=1}^{K} b_k^{n_k+\alpha_k-1} \tag{4.6}$$

Taking the expected value of the probability parameters b_i yields the expected probabilities:

$$b_i = \frac{n_i + \alpha_i}{\sum_{k=1}^{K} n_k + \alpha_k} \tag{4.7}$$

When there is not a one-to-one mapping of local outcomes to global outcomes, a set of pseudo hyperparameters must be created to create a one-to-one mapping. In order for a global outcome to give an equal prior weight to all derived outcomes, the hyperparameter value for the single global outcome is divided equally among each pseudo hyperparameter of derived local outcomes. This distribution of weight evenly divides the prior among the derived local outcomes and does not affect the estimation of other local outcomes. Using this method, one can derive the the expected values of the outcome probabilities for arbitrary local and global outcomes.

A second factor which must be calculated is the probability of the data given the pseudo hyperparameters. In this case, the probabilities of local outcomes can be factored out of the overall probability yielding:

$$log p(c_1...c_K|\alpha_1...\alpha_K) \propto log\Gamma(\sum_{k=1}^{K} \alpha_k) - log\Gamma(\sum_{k=1}^{K} n_k + \alpha_k) + \sum_{k=1}^{K} log\Gamma(n_k + \alpha_k) - log\Gamma(\alpha_k) \tag{4.8}$$

Using these methods, one can find a point estimate for an outcome probability distribution and the probability of deriving an example distribution from a global rule's Polya hyperparameters. This procedure can be used repeatedly within the

outcome set search and ultimately the ruleset search to calculate the probability of modifying the outcome set or ruleset to eventually arrive at a locally optimum ruleset.

4.3 Learning a Global Ruleset given a Set of Local Rulesets

The counterpart of learning a local ruleset from a set of examples and constant global ruleset is learning a global ruleset from a set of constant local rulesets. The search framework is very similar to that of the local ruleset search. In the outer loop, there is a greedy global ruleset search which adds, removes, and modifies global rules. For each proposed change, all rules relearn outcome sets through another greedy search. Finally, the Polya hyperparameters are calculated within each step taken within the greedy outcome set search.

4.3.1 Global Ruleset Modification

The highest-level greedy search in optimizing the global ruleset proposes the addition, deletion, and modification of global rules as in the local ruleset modification search. As before, in the case of additions or modifications to global rules, only the global rule(s) context needs to be specified as the outcomes and outcome distribution are determined by a secondary greedy search. In the global ruleset search, there are five search operators to modify the global ruleset:

- **Add Global Rule** - This operator proposes that a new global rule be added to the global ruleset. The global rule's context is formed by copying the context of a local rule. As rules are allowed to overlap in the global ruleset, no rules are removed when using this operator.

- **Remove Global Rule** - This operator removes a global rule from the global ruleset. All local rules explained by the removed global rule are then required to find new parent rules.

- **Add Function** - This operator adds a function to the context of an existing global rule (barring duplicate and contradictory functions).

- **Remove Function** - This operator removes a function from the context of an existing global rule.

- **Split Function** - This operator breaks apart a single existing global rule into multiple global rules. As in the case for local rulesets, each new global rule copies the context of the original global rule and appends a common function to each new rule's context. Each new rule assumes a different value of the appended function.

As described in the previous chapter, each local rule is paired with the most probable parent global rule to approximate the generative model which states that any local rules can be derived from any global rule. In order to calculate the mappings between local and global rules, the contexts of all local and global rules are compared and the probability that a local rule is derived from a global rule is partially estimated by the generative model. Each local rule is initially assigned to the closest global rule.

However, the generative model also specifies that any local rule can be created from scratch without a global rule parent. In order to accommodate this possibility, after the outcome distribution for a global rule has been calculated, any local rule mapped to that global rule has the option of declaring itself to be created from scratch. Local rules which have a higher probability of being created from scratch than of being derived from the global rule are disowned from the global rule. Following the disownment of any local rules, the global outcome set is recalculated once again and the process repeated until all owned local rules have a higher probability of being derived from the global rule than being created from scratch.

Whenever the global ruleset is modified, it is possible that local rules will switch parent global rules. As it is impossible to predict what effects a single search operator may have on the remapping of local rules to global rule parents, the outcome set and distribution of each rule must be recalculated at every step.

The only global rule which need not be recalculated at every step is the global default rule. The global default rule is automatically the parent of all local default rules. As the local rulesets are held constant, and no other local rules can be mapped to the global default rule, the outcome distribution of the global default rule need only be calculated at the beginning of the search. Note that by the generative model, like the local default rules, the global default rule cannot learn any new outcomes, so an outcome search is unnecessary.

As in the local ruleset optimization, once the context for a global rule is set, the next step involves creating a outcome set and outcome distribution. The following two sections will describe those components of the global ruleset search.

4.3.2 Global Rule Outcome Set Modification

The global rule outcome set search is very similar to the local rule outcome set search with one important difference: the outcome set for a global rule is allowed to have overlapping outcomes.

As with the local outcome set search, a list of candidate outcomes is constructed before the greedy search is started. Each outcome of each owned local rule is considered as a candidate outcome. Following this initialization, as in the local outcome set optimization search, the main greedy search is driven by several search operators:

- **Remove Outcome** - This operator removes an outcome from the outcome distribution.

- **Add Function** - This operator adds a function to an existing outcome (the duplicates or contradictory outcomes rule applies here as well).

- **Merge Outcomes** - This operator merges the context of an existing outcome and a candidate outcome by taking the union of their outcome changes and removing duplicate functions. This operator is only valid when the existing outcome and candidate outcome do not contradict one another.

- **Split Function** - This operator breaks apart a single outcome into multiple outcomes as in the ruleset "Split Function" operator. In addition to the context of the old outcome, each newly proposed outcome contains an additional common function with different values.

- **Add Outcome*** - This operator proposes that a new outcome be added to the outcome set. The proposed outcome is chosen from the candidate outcomes.

- **Remove Function*** - This operator removes a function from an existing outcome.

After the global rule's outcome set is modified, a new mapping from the outcome sets of associated local rules must be calculated. To do this, the same mapping algorithm described in the previous section on local outcome set learning is used. Each local outcome in each associated local rule is assigned to the global outcome which is closest via the generative model. Note again, that the closest global outcome may be the "new" outcome which specifies that the local outcome is created from scratch.

With mapping from local outcomes to global outcomes, the probability of deriving each associated local rule from the global rule can be calculated except for the probability related to the outcome distributions. The following part will describe how to calculate the global rule outcome distribution and calculate the probability of all associated local rule outcome distribution probabilities.

4.3.3 Learning the Global Rule Outcome Distribution

This section describes how to calculate the global rule outcome distribution given a mapping from associated local rule outcome sets to the global rule set. Once the global rule outcome distribution has been calculated, one can use the generative model to calculate the probability of deriving the associated local ruleset outcome distributions.

Deriving the Probability of the Global Outcome Set Distribution

As discussed earlier in section 4.2.3, it is possible that several local outcomes from within a particular rule will map to a common global outcome. As the immediate objective is to find the global rule outcome distribution, the differences between local outcomes which map to the same global outcome can be disregarded and those outcomes can be replaced by a generic local outcome with an example count equal to the sum of the example counts of the replaced local outcomes.

This procedure results in a matrix of example counts n where n_{ik} refers to the total number of example counts associated with global outcome k from local rule i. Let α refer to a vector of global outcome distribution hyperparameters, K refer to the total number of global outcomes and I refer to the total number of associated local rules. As the outcome distribution of each local rule is independent of any other local rule's outcome distribution given the global outcome set, the log probability of the count matrix is simply the sum of the log probabilities of the count vectors of each local rule. As specified by the generative model, there is a uniform prior on all hyperparameters of the global rule outcome distribution, and so the prior on these hyperparameters does not factor into the overall probability. Thus:

$$logp(n|\alpha) \propto \sum_{i=1}^{I} logp(n_i|\alpha) \tag{4.9}$$

Plugging in the Polya probability distribution into the probability of the example count vector for each task yields:

$$logp(n|\alpha) \propto \sum_{i=1}^{I} log\Gamma(\sum_{k=1}^{K} \alpha_k) - log\Gamma(\sum_{k=1}^{K} n_{ik} + \alpha_k) +$$

$$\sum_{k=1}^{K} log\Gamma(n_{ik} + \alpha_k) - log\Gamma(\alpha_k) \tag{4.10}$$

However, this is still not the true quantity that describes the probability of the

example count matrix. The generative model implies that if a global outcome is not selected, then its hyperparameters should not be factored into the probability. To accommodate this, the first two sums over outcomes must be selective in which hyperparameters are allowed to be added. Note that by the generative model, the hyperparameters new outcome and the noise outcome are not discounted, even if there are 0 example counts for either outcome. This fact can be neatly factored into the math equations by initializing the example counts for the two outcomes with a small number like .01. Discounting hyperparameters with 0 example counts yields:

$$logp(n|\alpha) \propto \sum_{i=1}^{I} log\Gamma(\sum_{k=1,n_{ik}>0}^{K} \alpha_k) - log\Gamma(\sum_{k=1,n_{ik}>0}^{K} n_{ik} + \alpha_k) +$$

$$\sum_{k=1}^{K} log\Gamma(n_{ik} + \alpha_k) - log\Gamma(\alpha_k) \tag{4.11}$$

Finally, through experimentation, we discovered that using the above equation to calculate a new global rule outcome distribution led to very large hyperparameters which often dwarfed pertinent data from local tasks. In fact, if there is only a single associated rule for a global rule, the probability-maximizing hyperparameters are infinity, and thus no amount of local data can outweigh the prior. To combat that trend, we add an additional term to the probability which penalizes the global rule outcome distribution for having very large hyperparameters. We assign a weight W to dictate the strength of the this bias. Thus, the final probability distribution is:

$$logp(n|\alpha) \propto -W * log(\sum_{k=1}^{K} \alpha_k) + \sum_{i=1}^{I} log\Gamma(\sum_{k=1,n_{ik}>0}^{K} \alpha_k) - log\Gamma(\sum_{k=1,n_{ik}>0}^{K} n_{ik} + \alpha_k) +$$

$$\sum_{k=1}^{K} log\Gamma(n_{ik} + \alpha_k) - log\Gamma(\alpha_k) \tag{4.12}$$

Setting the Global Hyperparameters

The next step involves setting the global rule outcome distribution hyperparameters in order to maximize the above probability. As the quantity is complex and highly non-linear, an intensive convex optimization search is needed. We used the the Newton iteration method to converge to the appropriate hyperparameters.

The main criterion for being able to use a convex optimization scheme is the convexity of the underlying distribution. To determine the convexity of our probability function, we tested if the hessian of the probability function was negative semidefinite. Let f be the log probability function and $H(f|\alpha)$ be the hessian of f. Then:

$$H(f|\alpha) = \begin{pmatrix} \frac{\partial^2 f}{\partial \alpha_1^2} & \frac{\partial^2 f}{\partial \alpha_1 \partial \alpha_2} & \cdots & \frac{\partial^2 f}{\partial \alpha_1 \partial \alpha_K} \\ \frac{\partial^2 f}{\partial \alpha_1 \partial \alpha_2} & \frac{\partial^2 f}{\partial \alpha_2^2} & \cdots & \frac{\partial^2 f}{\partial \alpha_2 \partial \alpha_K} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial^2 f}{\partial \alpha_1 \partial \alpha_K} & \frac{\partial^2 f}{\partial \alpha_2 \partial \alpha_K} & \cdots & \frac{\partial^2 f}{\partial \alpha_K^2} \end{pmatrix}$$

$$\frac{\partial^2 f}{\partial \alpha_k^2} \propto \frac{W}{(\sum_{m=1}^{K} \alpha_m)^2} + \sum_{i, n_{ik}>0} \Psi'(\alpha_k) - \Psi'(n_{ik}+\alpha_k) + \Psi'(\sum_{m=1, n_{im}>0}^{K} n_{im}+\alpha_m) - \Psi'(\sum_{m=1, n_{im}>0}^{K} \alpha_m) \tag{4.13}$$

$$\frac{\partial^2 f}{\partial \alpha_k \partial \alpha_j} \propto \frac{W}{(\sum_{m=1}^{K} \alpha_m)^2} + \sum_{i, n_{ik}>0, n_{ij}>0} \Psi'(\sum_{m=1, n_{im}>0}^{K} n_{im} + \alpha_m) - \Psi'(\sum_{m=1, n_{im}>0}^{K} \alpha_m) \tag{4.14}$$

where

$$\Psi(x) = \frac{\partial log \Gamma(x)}{\partial x} \tag{4.15}$$

$$\Psi'(x) = \frac{\partial \Psi(x)}{\partial x} \tag{4.16}$$

A convex probability function should have non-positive determinants over all square sub-matrices of the hessian including the top-left element. Unfortunately, it is immediately clear that even the top left element of the hessian may be positive in some circumstances. To combat non-convexity, we used a standard technique of

subtracting constant from the main diagonal to force the probability distribution to behave convexly[7].

To start the convex optimization, an initial point must be specified. We adapted a common initialization point for the standard Polya distribution to suit our probability distribution. We first calculate P, the proportion of each outcome occurrence for each task:

$$P_ik = \frac{n_{ik}}{\sum_{j=1}^{K} n_{ij}} \tag{4.17}$$

Next, we calculate the average A and squared average S of each outcome using all non-zero proportions. Thus:

$$A_i = \frac{\sum_{k=1,P_{ik}>0}^{K} P_{ik}}{Count_{k=1}^{K}(P_{ik} > 0)} \tag{4.18}$$

$$S_i = \frac{\sum_{k=1,P_{ik}>0}^{K} P_{ik}^2}{Count_{k=1}^{K}(P_{ik} > 0)} \tag{4.19}$$

The intensity s of the initial probability distribution is calculated as follows:

$$s = median_{k=1}^{K}\left(\frac{A_k - S_k}{S_k - A_k^2}\right) \tag{4.20}$$

Finally, the initial guess α is chosen as the product of the intensity and the mean of the outcomes:

$$\alpha_k = s * A_k \tag{4.21}$$

Once an initial point is chosen, the Newton iteration method is repeated until convergence. Thus, for initial guess α_{old}, an update consists of updating the guess to α_{new} by the stated formula. Note that as mentioned earlier, in order to ensure convexity and an invertible hessian, a negative constant is subtracted from the main diagonal if needed:

$$\alpha_{new} = \alpha_{old} - H^{-1}(f|\alpha_{old}) * \nabla(f|\alpha_{old}) \tag{4.22}$$

Using this procedure, one is able to estimate a likely value of the global hyperparameters.

Using the Global Hyperparameters

The third step of global outcome distribution step is estimating the probabilities of the local outcome distributions given the global outcome distribution. Unfortunately, the probability function shown earlier cannot be used to calculate the probability of all local outcome distributions at once as steps were initially taken to merge similarly mapped local outcomes together. Thus, the probabilities of each local rule outcome distribution must be calculated given the global outcome hyperparameters by the method described in section 4.2.3. The probability of the global outcome distribution can then be reported as the sum of all local outcome distributions.

Using the methods outlined here, the global ruleset search can test ruleset changes, build outcome sets for each proposed change, and calculate outcome distribution probabilities for each step of the outcome set step. By greedily searching through many levels, the global ruleset eventually stops searching when no further ruleset operators increase the probability of the global ruleset.

By iterating between optimizing source task rulesets and the global ruleset, the algorithm arrives at a likely global ruleset. This is then used as a prior for the target task. The following section will show the results of the optimization strategy.

Chapter 5

Results

This section will describe the results of the hierarchical rule transfer model.

The following blocks-world domains each have a generative model different than the one specified in chapter 3. Thus, it is informative to note that our generic generative model is widely applicable to a large class of domains. We call the generative models that create test tasks domain generators. In the first four domains, task rulesets are generated by copying all rules from the global ruleset. In domains with structural variation between tasks, specific functions constituting parts of the context or outcome set of rules are chosen when each task ruleset is initialized. Finally, the probability distribution assigned to each local rule is sampled from a Dirichlet with the hyperparameters of the corresponding global rule. In the fifth random domain, the generative model is explained in the section for the random domain.

Once each task ruleset has been created, a set of examples is sampled by creating random worlds and applying rules from the task rulesets. The task of the learner is to reconstruct the target task ruleset from these examples.

Both the learning techniques, using transfer and not using transfer, eventually converge to the optimal ruleset given enough examples. Thus, the measure of success in transfer learning is how well the transfer learners can perform with a small number of examples. To quantify accuracy, we use a measure called *sampled variational distance.*

When using sampled variational distance, we attempt to determine the "distance"

between the actual target ruleset R_{real} and the learned target ruleset $R_{learned}$. To do this, an initial world state S_t and action A_t are created. Then, the actual target task ruleset R_{real} is used to generate the subsequent state S_{t+1}. The probability of arriving at state S_{t+1} is measured using both rulesets. The variational distance VD_{sample} for that sample is then defined as:

$$VD_{sample} = |P(S_t, A_t, S_{t+1}|R_{real}) - P(S_t, A_t, S_{t+1}|R_{learned})|$$

The overall variational distance is simply the average variational distance of many samples. Let there be b total samples. Then, the total variational distance VD_{total} is:

$$VD_{total} = \frac{1}{b}\sum_{i=1}^{b} VD_i$$

The computational requirements of the transfer learning algorithm are demanding in both time and space. The running time of the multi-tiered coordinate ascent algorithm was never formally analyzed; however, it is most likely that the run time is polynomial in the size of the language, the number of source tasks, and the number of examples in each of the provided tasks. Despite the polynomial run time, the polynomial degree behind the run time is quite large leading to lethargic behavior when the example sets, number of source tasks, or domain complexity is too big.

The optimization strategy detailed in chapter 4 was used to gather the results. For all domains, a no-transfer learner was added to determine the benefit of using transfer learning. We were able to use the same generative model for the no-transfer learner by simply setting the global ruleset to be empty. The generative model was used nearly exactly for the model probability function. However, we found it necessary to modify the penalty of adding new literals in the formula modification process. Under the generative model, it is very easy to remove literals from a formula, but extremely difficult to add literals to the formula. To counteract this, we raised the penalty of adding a literal to a formula to the power .5 to reduce the overall impact. We found this necessary to allow meaningful global rules to be created.

Domain	No Transfer	1x5000	2x2500
Gripper Size	3.96	30.42	147.49
Slippery Gripper	13.85	33.09	140.15
Slippery Gripper Size	4.64	30.46	104.20

Domain	No Transfer	1x1000	4x250	10x100
Random	8.10	11.02	314.28	626.07

Figure 5-1: Computation time(seconds) per search iteration with 2000 target task examples.

Each point on the following graphs represents the average of 20 iterations. Variational distance was averaged over 1000 samples. The notation "1x5000" indicates that there was 1 source task with 5000 examples. The average running times for a given number of source tasks and examples are given in figure 5-1. In each case, there was 2000 target task examples. The random domain was run with fewer examples due to the longer computation time.

The next sections will describe the dynamics of the test domains and the obtained results.

5.1 Gripper Size Domain

The gripper size domain is a relatively simple domain with only a single rule per task ruleset. There are 3 functions in the domain: size, location, and weight. In each task, only heavy blocks of a certain task-specific size can be picked up successfully. Since the task-specific size varies between tasks, the rules for different tasks have different structures. The location function is simply a distracter. Figure 5-2 shows the domain generator for the gripper size domain.

As the gripper size domain has only one rule, the global ruleset almost always learns the correct global rule. Figure 5-3 shows a sample learned global ruleset. This rule efficiently serves as a prior for local rulesets as it can be easily specialized with a task-specific size literal to model a specific task. As can be seen in figure 5-4,

$$
\begin{aligned}
Pickup(X): \quad & OnTable(X) = true, \\
& Size(X) = random(Size1, Size2, Size3, Size4, Size5, Size6, Size7) \\
\rightarrow & \begin{cases} 500.0: & OnTable(X) = true \\ 300.0: & OnTable(Y) = false \end{cases}
\end{aligned}
$$

Figure 5-2: Domain generator for the gripper size domain

$$
\begin{aligned}
Pickup(X): \quad & OnTable(X) = true \\
\rightarrow & \begin{cases} .09: & noise \\ 50.21: & OnTable(X) = true \\ 224.87: & OnTable(X) = false \end{cases}
\end{aligned}
$$

Figure 5-3: Sample learned global rule for the gripper size domain

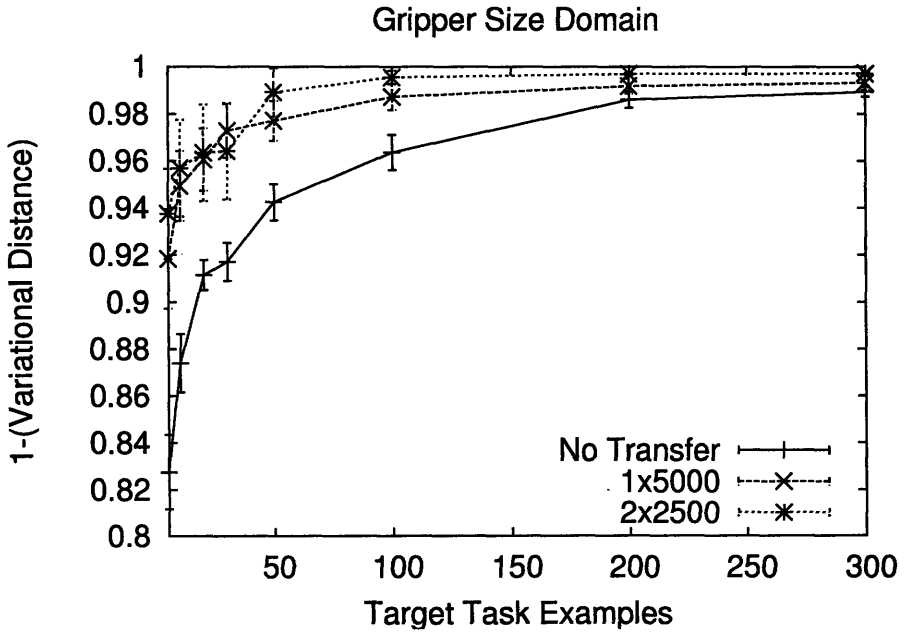

Figure 5-4: Accuracy results for the gripper size domain comparing transfer learners vs a non-transfer learner.

$$Pickup(X,Y): \quad On(X,Y) = true, GripperFree() = true, Wet() = true, BaseType(Y) = block$$
$$\rightarrow \begin{cases} 6.6: & On(X,Y) = true \\ 6.6: & On(X,Y) = false, GripperFree() = true \\ 6.6: & On(X,Y) = false, GripperFree() = false \end{cases}$$

$$Pickup(X,Y): \quad On(X,Y) = true, GripperFree() = true, Wet() = false, BaseType(Y) = block$$
$$\rightarrow \begin{cases} 2.0: & On(X,Y) = true \\ 4.0: & On(X,Y) = false, GripperFree() = true \\ 14.0: & On(X,Y) = false, GripperFree() = false \end{cases}$$

$$Pickup(X,Y): \quad On(X,Y) = true, GripperFree() = true, Wet() = true, BaseType(Y) = table$$
$$\rightarrow \begin{cases} 10.0: & On(X,Y) = true \\ 10.0: & On(X,Y) = false \end{cases}$$

$$Pickup(X,Y): \quad On(X,Y) = true, GripperFree() = true, Wet() = false, BaseType(Y) = table$$
$$\rightarrow \begin{cases} 4.0: & On(X,Y) = true \\ 16.0: & On(X,Y) = false \end{cases}$$

Figure 5-5: Domain generator for the slippery gripper domain

the transfer learners consistently perform significantly better than the non-transfer learner. The global ruleset is quickly able to learn that the *OnTable* predicate is relevant. Furthermore, the global rule learns the correct outcome distribution which biases the target ruleset to learn the correct proportions of the outcomes with relatively little data.

5.2 Slippery Gripper Domain

The slippery gripper domain is a complex domain adapted from a single task variant of the ruleset used in probabilistic planning[4]. Unlike the gripper size domain, the slippery gripper domain contains a significant amount of parametric variation rather than structural variation. There are four rules in the domain. The language consists of 4 functions: location, wetness, status of the gripper, and base type. In each task, a combination of the wetness and base type of the targeted block affects the probability of successfully picking up the block, unsuccessfully dislodging the block, or unsuccessfully not altering the system. The domain generator can thus be modeled by figure 5-5.

$$Pickup(X,Y): \quad On(X,Y) = true, GripperFree() = true, BaseType(Y) = table$$
$$\rightarrow \begin{cases} .09 : noise \\ 36.12 : \ On(X,Y) = true \\ 52.54 : \ On(X,Y) = false \end{cases}$$

$$Pickup(X,Y): \quad On(X,Y) = true, GripperFree() = true, BaseType(Y) = block$$
$$\rightarrow \begin{cases} .07 : noise \\ 8.97 : \ On(X,Y) = true \\ 10.54 : \ On(X,Y) = false, GripperFree() = true \\ 17.37 : \ On(X,Y) = false, GripperFree() = false \end{cases}$$

Figure 5-6: Sample learned global rules for the slippery gripper domain

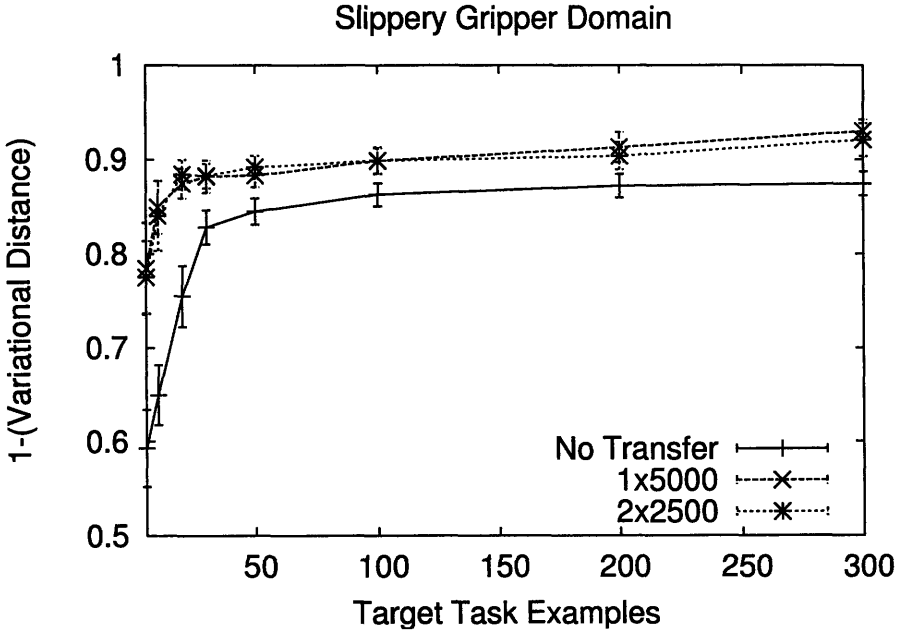

Figure 5-7: Accuracy results for the slippery gripper domain comparing transfer learners vs a non-transfer learner.

The slippery gripper domain is significantly more challenging domain than the gripper size domain due to the presence of four rules. Even with 2000 examples, the no-transfer learner cannot learn the optimal ruleset. However, the transfer learners can learn portions of the correct ruleset and allow for easier structural and parametric modifications for local rulesets to match the true ruleset. Figure 5-6 shows a sample global ruleset learned by the optimization algorithm. Although the global ruleset only contains two rules rather than four, it allows local rulesets to both specialize each rule into multiple rules (splitting along wetness for example) if required or simply use a broad generalized rule in the local ruleset. As can be seen in figure 5-7, the transfer learners are able to utilize the global ruleset effectively to rule a model of the task quicker than the non-transfer learner.

5.3 Slippery Gripper with Size Domain

The third domain, the slippery gripper with size domain, is a cross of the slippery gripper domain and gripper size domain. The domain sports 4 rules and structural variance across tasks. As in the slippery gripper domain, the wetness and base type of the secondary block play roles in modifying the outcome distributions. In addition, as in the gripper size domain, only blocks of a certain task-specific size can activate the rules. Thus, the slippery gripper size domain domains both significant structure and parametric variation. The domain generator is given in figure 5-8.

The slippery gripper with size domain is the most challenging domain due to presence of four rules with both parametric and structural variation between tasks. The learned global ruleset for this domain, figure 5-9, is very similar to that of the slippery gripper domain. In fact, it is sometimes difficult to tell the two apart based on the global ruleset. The transfer learners still manage to perform significantly better than the non-transfer learner in this domain as seen in figure 5-10. In addition to aiding in parametric learning, the global ruleset helps the local rulesets easily specialize the global rules into task-specific local rules by the addition of a single size literal. Overall, the transfer learners perform better than the non-transfer learner in

$$S = random \quad (Size1, Size2, Size3, Size4, Size5, Size6, Size7)$$

$$Pickup(X,Y): \quad On(X,Y) = true,\ GripperFree() = true,\ Wet() = true,\ BaseType(Y) = block \\ Size(X) = S$$

$$\rightarrow \begin{cases} 6.6: & On(X,Y) = true \\ 6.6: & On(X,Y) = false,\ GripperFree() = true \\ 6.6: & On(X,Y) = false,\ GripperFree() = false \end{cases}$$

$$Pickup(X,Y): \quad On(X,Y) = true,\ GripperFree() = true,\ Wet() = false,\ BaseType(Y) = block \\ Size(X) = S$$

$$\rightarrow \begin{cases} 2.0: & On(X,Y) = true \\ 4.0: & On(X,Y) = false,\ GripperFree() = true \\ 14.0: & On(X,Y) = false,\ GripperFree() = false \end{cases}$$

$$Pickup(X,Y): \quad On(X,Y) = true,\ GripperFree() = true,\ Wet() = true,\ BaseType(Y) = table \\ Size(X) = S$$

$$\rightarrow \begin{cases} 10.0: & On(X,Y) = true \\ 10.0: & On(X,Y) = false,\ GripperFree() = true \end{cases}$$

$$Pickup(X,Y): \quad On(X,Y) = true,\ GripperFree() = true,\ Wet() = false,\ BaseType(Y) = table \\ Size(X) = S$$

$$\rightarrow \begin{cases} 4.0: & On(X,Y) = true \\ 16.0: & On(X,Y) = false,\ GripperFree() = true \end{cases}$$

Figure 5-8: Domain generator for the slippery gripper with size domain

$$Pickup(X,Y): \quad On(X,Y) = true,\ GripperFree() = true,\ BaseType(Y) = table,\ Size(X) = Size2$$

$$\rightarrow \begin{cases} .09: & noise \\ 27.93: & On(X,Y) = true \\ 39.32: & On(X,Y) = false \end{cases}$$

$$Pickup(X,Y): \quad On(X,Y) = true,\ GripperFree() = true,\ BaseType(Y) = block$$

$$\rightarrow \begin{cases} .07: & noise \\ 4.82: & On(X,Y) = true \\ 6.62: & On(X,Y) = false,\ GripperFree() = true \\ 10.22: & On(X,Y) = false,\ GripperFree() = false \end{cases}$$

Figure 5-9: Sample learned global rules for the slippery gripper with size domain

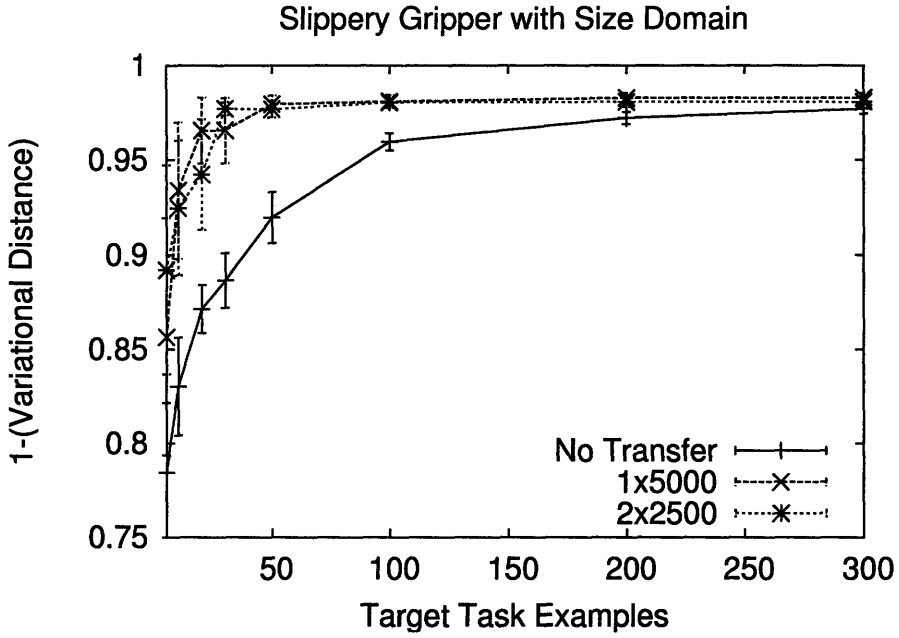

Figure 5-10: Accuracy results for the slippery gripper with size domain comparing transfer learners vs a non-transfer learner.

this domain.

5.4 Random Domain

The random domain presents an interesting challenge for the transfer learning algorithm: will misinformation force the algorithm to learn a bad ruleset and lead to worse performance than the non-transfer case?

The random domain has 4 functions {A,B,C,D} each with two possible boolean values. The random domain generator randomly selects to generate 1-4 rules. Each rule's context is randomly selected to be 1-4 literals long. Each rule is given 1-4 outcomes each with 1-4 literals. Finally, each outcome distribution is randomly selected. Constraints, such as the fact that no two functions in the context can be identical, require many samples before suitable random rulesets can be generated.

Interestingly, the transfer learners perform on par or slightly better compared to the non-transfer learner as can be seen in figure 5-12. This advantage most probably

$$
\begin{aligned}
&Pickup(X): \quad A() = true \\
&\rightarrow \begin{cases}
.13 : noise \\
45.38 : \ A() = true, B() = true \\
84.28 : \ A() = true, B() = false, C() = true \\
64.85 : \ A() = true, B() = false, C() = false, D() = true \\
17.43 : \ A() = false, C() = false, D() = true \\
24.93 : \ A() = false, C() = true, D() = false \\
37.41 : \ A() = false, C() = true, D() = true \\
11.01 : \ A() = false, B() = true, C() = true, D() = false
\end{cases}
\end{aligned}
$$

Figure 5-11: Sample learned global rule for the random domain.

stems from the fact that the global ruleset learns rules in the transfer case but is empty in the non-transfer case. Thus, regardless of the amount of information contained in each of the global rules, the fact that the global ruleset contains one or more global rules places a bias on the target task ruleset to learn rules. The rules learned by the global ruleset are not very helpful as can be seen in figure 5-11; however, overall, it appears that the transfer learners are not severely affected by unrelated source rulesets.

In the three structured tested domains, the transfer learners are able to utilize the target task data more efficiently than the non-transfer learner. Moreover, in the random domain, the transfer learners perform no worse.

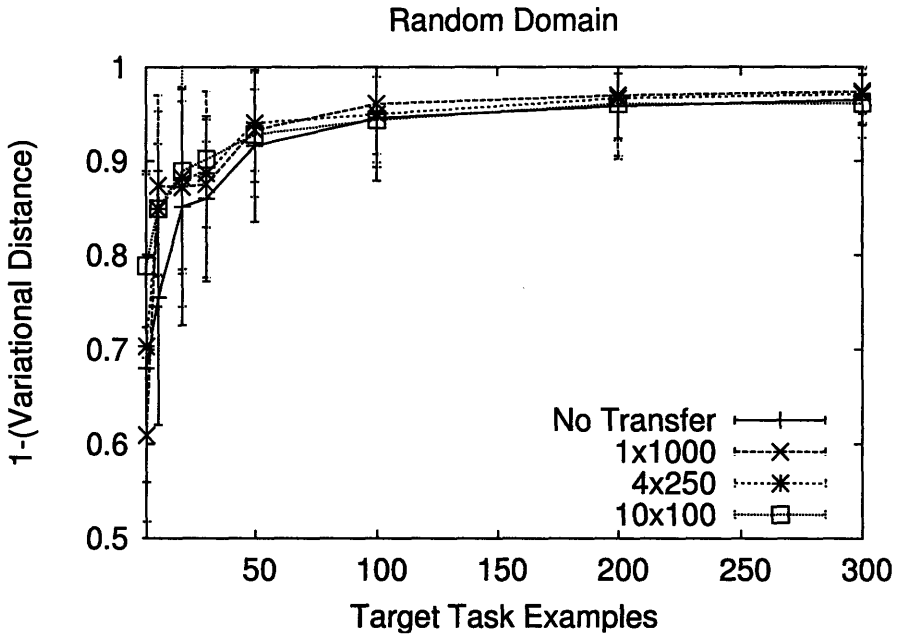

Figure 5-12: Accuracy results for the random domain comparing transfer learners vs a non-transfer learner.

Chapter 6

Future Work and Conclusions

Overall, it can be seen that the overall model proposed is conducive to knowledge transfer in the tested domains. However, there was one major problem with the optimization strategy: it requires a significant amount of data for the source tasks in order to learn an appropriate global ruleset. To counter this, we developed an alternative direct optimization strategy rather than the coordinated-ascent strategy described in the past few chapters.

6.1 Direct Optimization Strategy

In testing the optimization strategy previously presented, we noticed some unsatisfying trends in the optimization of the global ruleset. Namely, in domains like the slippery gripper domain, the global ruleset would often learn only a subset of the ideal global ruleset when the source tasks each contained small data sets. This behavior arose because the local source task rulesets were not able to learn the complete dynamics of the domain themselves. As the probability of the model is strengthened when the global and local rulesets are similar, both types of rulesets would learn over-generalized rules to explain the domain. A local ruleset would be unwilling to specialize its rule as that would decrease the probability of the local ruleset, and the global ruleset would be unwilling to create additional specializing rules without support from the local rulesets. This produced a catch-22 type behavior in the opti-

mization algorithm, even though the generative model may give a higher probability to a more complex model.

To combat this unwanted behavior, we proposed a direct optimization strategy rather than a coordinate-ascent based optimization strategy. Under the new strategy, only the global ruleset is optimized. However, for each proposed change in the global ruleset, all local rulesets are recalculated using the newly proposed global ruleset. This optimization technique requires considerably more time than the previous method due to the frequency of local ruleset searches.

Our preliminary results suggest that the direct optimization strategy is much better than the coordinate-ascent strategy at learning complete global rulesets with small amounts of data in each source task. In fact, using 10 source tasks with 200 examples each, a setup which would usually cause the coordinate-ascent algorithm to learn a single global rule, often produced the complete 4 rule slippery gripper global ruleset. The downside to the method is its run-time. While the run-time was not carefully measured, on the domains tested, it was 20-100 times slower than the coordinate-ascent algorithm. As this method was not extensively explored, it is uncertain if the learning benefits in the tested domains are broadly applicable, but the method appears to hold promise.

6.2 Conclusion

Overall, it appears that transfer learning can be applied to relational rule transfer. The generative model we presented in chapter 3 can apply to a wide range of domains. Furthermore, the generative model presented is not absolute: it can easily be modified if domain specific knowledge is known. The coordinate-ascent optimization strategy described in chapter 4 proved to be robust in the various testing domains. This thesis presented a framework with which to transfer relational rule structure between multiple tasks which can be extended in the future to work in more complex domains, languages, and data sets.

Bibliography

[1] J. Baxter. A Bayesian/information theoretic model of learning to learn via multiple task sampling. *Machine Learning*, 28:7–39, 1997.

[2] A. L. Blum and J. C. Langford. Probabilistic planning in the Graphplan framework. In *Proc. 5th European Conf. on Planning*, 1999.

[3] R. E. Kass and D. Steffey. Approximate Bayesian inference in conditionally independent hierarchical models. *JASA*, 84(407):717–726, 1989.

[4] N. Kushmerick, S. Hanks, and D. S. Weld. An algorithm for probabilistic planning. *Artificial Intelligence*, 76:239–286, 1995.

[5] D. V. Lindley. The estimation of many parameters. In V. P. Godambe and D. A. Sprott, editors, *Foundations of Statistical Inference*. Holt, Rinehart and Winston, Toronto, 1971.

[6] Z. Marx, M. T. Rosenstein, L. P. Kaelbling, and T. G. Dietterich. Transfer learning with an ensemble of background tasks. In *NIPS Workshop on Inductive Transfer*, 2005.

[7] T. P. Minka. Estimating a Dirichlet distribution. Available at http://research.microsoft.com/~minka/papers/dirichlet, 2003.

[8] H. M. Pasula, L. S. Zettlemoyer, and L. P. Kaelbling. Learning probabilistic relational planning rules. In *Proc. 14th ICAPS*, 2004.

[9] K. Yu, V. Tresp, and A. Schwaighofer. Learning Gaussian processes from multiple tasks. In *Proc. 22nd ICML*, 2005.

[10] L. S. Zettlemoyer, H. M. Pasula, and L. P. Kaelbling. Learning planning rules in noisy stochastic worlds. In *Proc. 20th AAAI*, 2005.

[11] J. Zhang, Z. Ghahramani, and Y. Yang. Learning multiple related tasks using latent independent component analysis. In *NIPS 18*. MIT Press, 2006.