AN ABSTRACT OF THE DISSERTATION OF

Sriraam Natarajan for the degree of Doctor of Philosophy in Computer Science presented on November 20, 2007.

Title: Effective Decision-Theoretic Assistance Through Relational Hierarchical Models

Abstract approved: ______________________________

Prasad Tadepalli

Building intelligent computer assistants has been a long-cherished goal of AI. Many intelligent assistant systems were built and fine-tuned to specific application domains. In this work, we develop a general model of assistance that combines three powerful ideas: decision theory, hierarchical task models and probabilistic relational languages. We use the principles of decision theory to model the general problem of intelligent assistance. We use a combination of hierarchical task models and probabilistic relational languages to specify prior knowledge of the computer assistant. The assistant exploits its prior knowledge to infer the user's goals and takes actions to assist the user. We evaluate the decision theoretic assistance model in three different domains including a real-world domain to demonstrate its generality. We show through experiments that both the hierarchical structure of the goals and the parameter sharing facilitated by relational models significantly improve the learning speed of the agent. Finally, we present the results of deploying our relational hierarchical model in a real-world activity recognition task.

Effective Decision-Theoretic Assistance Through Relational Hierarchical Models

by

Sriraam Natarajan

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented November 20, 2007
Commencement June 2008

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Dr. Prasad Tadepalli, my advisor at Oregon State University. He has been incredibly supportive and patient with me during the past 6 years. His encouragement, constructive criticism, guidance and motivation are the most important factors in my successful completion. He has tolerated my failures and frustrations and provided me with energy and confidence when needed. He has been a fantastic role model for every student with his humility and approachability. I cannot imagine how my progress would have been if not for him. Ever willing to lend an ear and a helping hand, Dr.Tadepalli has been with me through the tough times. I am forever indebted to him.

I also thank my committee members, Dr. Thomas Dietterich, Dr. Alan Fern and Dr. Weng-Keen Wong for the innumerable discussions about different aspects of the thesis. I have had the pleasure of co-authoring several papers with them and have benifitted from their help in those papers. Dr.Dietterich has always been my inspiration with his ability to handle many tasks. Even though he is immensly informed, his thirst for knowledge is unquenchable. I feel fortunate to have had the experience of working with him. Dr.Fern has guided me while working on several papers and has been of tremendous support. His meteoric rise in the past few years makes him a role model for all new researchers like myself. Dr.Wong is mainly responsible for cultivating my interests in graphical models and I am thankful to him for exposing me to this area. He has been easily approachable and has always been looking to share ideas.

I thank Dr.Rod Harter for taking his time off to serve in my committee. I also thank Dr. Hung Bui of SRI International for providing me with an opportunity to work with him. It was a fantastic experience coming at the fag end of my PhD. I also would like to

extend my gratitude to the CS department staff for their help all these years. I am grateful to my lab mates Neville, Aaron, Ronny and Charlie for the numerous discussions (both on research and otherwise) that we have had over the past 5+ years.
Friends! How will life be without them? I am blessed to have the best of friends in my life. To start with, Arvind, Sai, Sharat and Vinod were the best room-mates anyone could have ever had. I do not have words to describe the amazing time we had in the first two years. Every single day was worth remembering during that period. The assignments, the comp exam, the exams, dixon, sai's fabulous lunches, chit chats in Dearborn corridor, the cricket matches,... the list is endless. I cannot forget the numerous trips to Portland or the trips to the bay area to the most recent internship time which was amazing due to Sharat and Sai. Thank you guys for some of the best years of my life and for being with me for so many years.
I would also like to thank the cricket gang (too many of them to mention by name) for the innumerable fantastic games we played. I cannot forget the 2003 world cup experience. I am also thankful to Madhu(s), Kavitha, Anand, Saket, Raja, Ranga, Jana, Neeraja, casa grande gang, nash gang and La gaviota gang for being with me during the different years of my grad school life. I take this opportunity to express my gratitude to Professor Sundar, Ramanujam uncle and aunty. I would also like to thank Kshitij for the assistance in running several experiments, helping me out whenever needed and the several discussions over the past few years.
Karthik, we met in the most unique of circumstances. Since then you have been my friend, philosopher and guide. Your support over the past few years has been incredible. In you, I have found a friend for life. You have been a wonderful motivator and stood by me during my failures. I will always cherish the wonderful discussions that we have had over the years. More recently, Shwetha has been a pillor of support for me over the past 2 years. She has been with me during the toughest of times and served as a source of encouragement during those times. Thank you Shwetha for patiently bearing with me,

listening to my complaints, calming me down whenever I was worked up, providing the energy that was needed and being an amazing friend. I extend my sincere gratitude to Karthik and Shwetha.

My graduation would not have been possible if it wasnt for thatha's blessings. My special thanks to paatis for their wishes. Everyone at Adambakkam have been amazing over the past 6 years. I am deeply indebted to all the mamas, mamis, chitti, chittappa for their blessings and wishes. Raji has been an incredible mentor. Thank you Raji for being there for me providing with superb advises and guidance at every single step. Last but definetly not the least, my degree is dedicated to Amma, Appa and Ramya. You guys have been fantastic. The innumerable number of prayers and wishes and the faith you had in me over the few years at grad school have been the main reasons for my successful completion. The number of prayers would have made pillayaar bored with you guys! I am thankful for the best family that any student can get.

When I look back, I realize that my degree was a collective effort of so many people. I sincerely thank almighty for providing me with so many people who have been with me over the past few years and hopefully for the rest of my life.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF TABLES

DEDICATION

To Amma, Appa and Ramya

Chapter 1 – Introduction

Building intelligent assistants that assist their users in their day-to-day chores has been a long cherished dream of AI. There has been a growing interest in developing intelligent assistant systems over the past decade. These systems aim at helping the user with a specific task by reducing the user's efforts in accomplishing that task. Also, they reduce the need for the user to remember all the information that is relevant to performing the task. An ideal assistant should have a knowledge of the set of tasks of the user, comprehend the current task that the user is working on and help the user in achieving the task. The goal of this research is not to replace a human user, but to aid him so as to reduce his effort. It may be useful to think of this intelligent assistant as a personal secretary. The secretary is not always told what to do. Similarly, the assistant will not always be told what the user is working on. The assistant should infer the user's tasks by observing the actions of the user. Once it has a reasonable confidence about the current task of the user, it could start helping the user. A perfect assistant should also be able to remind the user when he forgets to do a task (for example, when he forgets to attach a document to an email), or when there is an easier method by which the task could be completed (for example, using the shortcuts on the desktop instead of going through the file system). In this work, we take several steps towards designing an intelligent assistant. Though this assistant cannot possibly achieve all the desired qualities, it is a general model of assistance that can be applicable to several problems. Our assistant can handle uncertainty with respect to the user's goals, can generalize across several users of the domain, can handle changes in the domain and can be effective in the real-world.

1.1 Assistantship Model

There has been substantial research in designing intelligent assistants ranging from helping people with dementia[10] to travel planning[2] and rescheduling meetings[68]. However, most of this work consists of well engineered solutions to the particular application domains. There is not a general model of assistance that can be extended across several domains. The goal of this thesis is to formulate a model of assistance that:

- applies to several different domains - For example, the model should be applied to a desktop assistant or an assistant that helps elderly people with their daily chores.
- handles uncertainty - For example, the user's actions might not give complete information about his or her current task. The assistant should still be able to offer some help.
- handles varying user action/cost models - For example, when there is a short deadline, the user would want to complete the task quickly, while on a normal situation, the user might want to explore more ideas. The assistant should be helpful in either case. In other words, the model should not change as the user's preferences change.
- generalizes across different users in the domain - For example, the assistant might be trained by observing the development team and when actually deployed and used by a real user, it should provide effective assistance.
- can handle hierarchical goal structure of the user - For example, an user might decompose his higher level goal of planning a conference trip to booking hotels, flights, registering for the conference etc. The assistant must be able to represent and reason about this type of a rich goal structure.
- performs efficiently in real-time. In particular, inference must be performed in real time for the assistant to be useful in real-world domains.

To this effect, we propose a model of assistance that is based on decision-theoretic principles and applicable across several domains. In order for the model to be generalized across different users and objects in the domain, we draw on our experiences with relational probabilistic models and task hierarchies to represent the prior knowledge about the user. This specification of the prior knowledge also enables the assistant to learn faster about the user preferences. For this richer model to perform efficiently, we have to avoid grounding the relational prior knowledge in the system with all the objects in the domain while tracking the user's progress. We explore the use of Logical Hierarchical HMMs for this purpose.

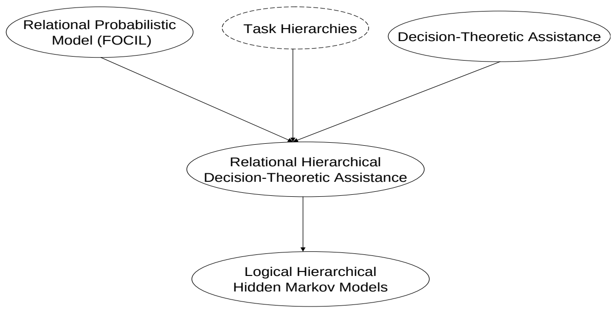

Relational Probabilistic Model (FOCIL)
Task Hierarchies
Decision-Theoretic Assistance
Relational Hierarchical Decision-Theoretic Assistance
Logical Hierarchical Hidden Markov Models

Figure 1.1: Evolution of the thesis. The solid ovals indicate our contributions.

The evolution of the thesis is pictured in Figure 1.1. Initially, we formulate the decision-theoretic assistantship problem. We then incorporate ideas from relational and hierarchical models to represent prior knowledge. The decision-theoretic assistant performs assistance in two steps: first, it estimates the goal of the user and second, it determines the best assistive action. In the relational hierarchical model, the goal estimation is performed using a hand-coded DBN. This led to research in the area of goal estimation (or more generally activity recognition) and we have developed a Logical Hierarchical Hidden Markov Model that extends standard HMMs to the relational domains with hierarchies.

1.2 Overview of the Thesis

Our proposed framework can be understood as trying to achieve the best of three worlds of using hierarchies for plan recognition, using decision-theory for assistance and using relational models for succinct representation. Incorporating relational hierarchies into the decision-theoretic framework would enable us to realistically model the user and provide better assistance. Our major contribution is to demonstrate the usefulness of the combination of hierarchies, relational models and decision-theory in the formulation of intelligent assistants. There has been several attempts to combine hierarchies and decision-theoretic models. For example see [19, 3, 60, 58]. Recently, relational reinforcement learning has also been explored [66]. Our goal is to combine the power of all the three different areas in designing a useful intelligent assistant.

It should be mentioned that our framework is not specific to the problem of assistance and the individual components can be applied to other problems as well. For example, our relational learning algorithms can be used for modeling several aspects of the domain. Our relational hierarchies can be used to specify the prior knowledge irrespective of the model of assistance. Our Logical Hierarchical HMMs can be used for activity recognition in other domains. In this chapter, we motivate the different models that have been presented in this thesis.

1.2.1 Decision-Theoretic Model of Assistance

Our goal is to develop an assistant that would aid a goal-oriented user and one that can be generalized across several domains. The assistant would observe the user performing some actions, infer the user's goals[1] from its observations and determine the action that can best assist the user. The assistant can modify its actions depending upon the user's

[1]In this work, we use the words task and goal interchangeably.

response to its assistance. In real applications, this requires that the assistant be able to handle uncertainty about the environment and agent, to reason about the action costs, to handle unforeseen situations, to take actions that would aid the user, and to adapt to the agent over time.

We formulate the assistantship problem as a Partially Observable Markov Decision Process (POMDP) which naturally handles these features, providing a formal basis for designing intelligent assistants.. We design myopic heuristics for solving the POMDP. We evaluate this model in three different domains: two game-like environments and a real-world folder predictor domain to establish the domain independent nature of the problem and the solution. The results indicate that the model can be used across different environments and that the myopic heuristics achieve comparable performance with more sophisticated but costly solution techniques.

1.2.2 Relational Models

The decision-theoretic model is a first-step towards developing a useful assistant and has several simplifying assumptions. One of the limitations of the model is that the representation is inherently propositional. This model does not capture the relational aspects of the world and ignores the relationships that exists between the objects in the world. If the model can explicitly capture these relationships, it can be very powerful. There has been extensive research in the combination of probabilistic and relational models that aims to achieve generalization among the objects of the domain[29].

One of the main advantages of these relational models is that they share the parameters i.e, different instances of objects of the same kind share the distributions. For example, a model that is learned for an academic researcher can be used for another researcher thus reducing the number of examples needed to learn the model. The other major advantage is that they provide a way to accelerate learning by exploiting the relationships between objects. They also make it possible to simplify the problem of inference as they avoid considering all the objects in the domain. These models, when given a query and the database describing the domain, would construct a smaller network that is needed for the answering that query.

We have developed a language for representing prior relational knowledge about the domain. The syntax consists of a set of rules that describe influence relationships between objects in the domain. Since the model is relational, when instantiated with the values of the domain, there could be multiple instantiations of the same object. For example, the courses a student took or the documents that researcher worked on. The effects due to the different instantiations are combined using either *aggregators* or *combining rules.* We focus on the use of combining rules and develop learning algorithms based on Gradient Descent and EM for learning the parameters of the relational models in the presence of combining rules. We evaluate the algorithms on two domains: a folder predictor domain and a synthetic domain to test the accuracy of the learned models.

1.2.3 Incorporating Relational Hierarchies

Given our experience with the more powerful models, our next goal is to design a richer representation for the decision theoretic assistant framework. The assumption that the user has a fixed set of flat goals is not particularly appropriate in real world. Consider an example of a desktop assistant along the lines of CALO [13] that serves an academic researcher. The researcher could have some high level goals like writing a proposal, preparing for a meeting, writing a paper and preparing class notes etc. In turn each of these goals would involve subgoals like editing a document, looking up references, sending emails etc. The non-hierarchical model would reduce the task to a large number of individual actions. This is not only unintuitive but also wasteful and is impractical for larger problems. Instead, we explore the use of hierarchies which naturally capture the user's goal-subgoal structure and decomposes a higher level goal into a small number of lower level goals.

But hierarchies alone do not suffice in many real-world applications. As we pointed out earlier, the underlying model is relational in most real-world domains and this necessitates the hierarchies to be relational as well. For instance, the task could be to write a paper to a particular conference. If the hierarchies were propositional, we need a separate subtask for writing papers to different conferences. In reality, writing conference papers follow a similar pattern that includes running experiments, editing a particular document, sharing it with co-authors and turning in the paper. The similarity between different papers can be exploited if the task is relational, i.e., the task is parameterized with the parameter being the properties of the paper that is being written. This would enable the model learned from one conference to be used for helping the user in another conference. As explained earlier, this concept of parameter tying makes it possible to share the parameters and hence makes the learning process (and sometimes the inference as well) easier.

The relations between the objects can be used to constrain the number of objects that need to be considered for a particular task. For instance, only a few related documents would be referenced while working on a particular document. To exploit these ideas, the hierarchy must be relational which in turn would make the inference problem simpler as the constraints would restrict the set of objects that need to be considered. For details see chapter 4. In this work, we propose to use a relational hierarchy to model the goal-subgoal structure. Our hierarchies are a particular form of the goal-subgoal hierarchies where goals at higher levels have parameters and make calls to the lower level goals by passing appropriate parameters. We use Dynamic Bayesian Networks [53](DBNs) to estimate the distribution of the goal-subgoal combination of the user and use myopic heuristics to select the best assistive action. We present the results of the assistant model in two toy domains: a grid world domain and a cooking domain.

1.2.4 Logical Hierarchical HMM

The DBN used in the Relational Hierarchical model cannot scale with the number of objects in the domain. This is due to the fact, when we ground the objects of a domain, it can lead to an exponential number of ground values based on the combinations of the values of the objects and relations. This would mean that the inference can become intractable. An alternative is to use sampling mechanisms for the DBNs, but the number of ground states can grow drastically with the number of objects in the domain. For example, the combinations of emails and the files that can be attached to those emails can be very large for a typical user. It is impractical to enumerate all the possible combinations in order to perform goal estimation.

The problem of huge state spaces led us to develop a relational model that can capture this DBN succinctly. In particular, in this work, we have developed a Logical Hierarchical Hidden Markov Model(LoHiHMM) which can be used to track the user's activity. LoHiHMMs can be understood as extending the HMMs to the relational (logical) and hierarchical settings. The basic idea is that given a generalized model of the user's activity in a particular domain, the problem of estimating the user's goal corresponds to the problem of performing inference given the observations(user actions). We propose the use of LoHiHMMs to perform inference on the user's goal-subgoal combinations. We outline the syntax and define the semantics of the LoHiHMM in this work. We also present an inference algorithm based on particle filtering that avoids complete enumeration of the state space. The activity recognition results are demonstrated using a real-world desktop assistant and some synthetic domains.

1.2.5 Outline

The rest of the thesis is organized as follows: in the next chapter, we present the decision-theoretic model of assistance and the experimental results. We provide a detailed overview of the algorithms that we have developed for learning in relational models in the third chapter and present empirical evidence for the performance of these algorithms. In the fourth chapter, we present the relational hierarchical model for assistance and evaluate it in two domains. In the penultimate chapter, we present our logical hierarchical hidden markov model and use it for activity recognition. We conclude the thesis in the final chapter by outlining some possible directions for future work.

Chapter 2 – A Decision-Theoretic Model of Assistance

2.1 Overview

There is a growing interest in intelligent assistants for a variety of applications from organizing tasks for knowledge workers to helping people with dementia. However, a general framework that captures the notion of useful assistance is lacking. In this chapter, we present and evaluate a decision-theoretic model of assistance. The objective is to observe a goal-directed agent and to select assistive actions in order to minimize the overall cost. We describe how to model this problem as an assistant POMDP where the hidden state corresponds to the agent's unobserved goals. This formulation naturally handles uncertainty, varying action costs, and customization to specific agents via learning. Since directly solving the assistant POMDP is often not a practical alternative, we consider an approximate approach to action selection based on the computation of myopic heuristics and bounded search. In addition, we introduce a novel approach to quickly learn the agent policy, based on a rationality assumption, which is critical for the assistant to be useful early in its lifetime. We evaluate our approach in two game-like computer environments where human subjects perform tasks and a real-world domain of providing assistance during folder navigation in a computer desktop environment. The results show that in all three domains the framework results in an assistant that substantially reduces user effort with only modest computation.

2.2 Introduction

The development of intelligent computer assistants has tremendous impact potential in many domains. A variety of AI techniques have been used for this purpose in domains such as assistive technologies for the disabled [10] and desktop work management [54]. However, most of this work has been fine-tuned to the particular application domains. In this chapter, we describe and evaluate a more comprehensive framework for designing intelligent assistants.

We consider a model where the assistant observes a goal-oriented agent and must select assistive actions in order to best help the agent achieve its goals. To perform well the assistant must be able to accurately and quickly infer the goals of the agent and reason about the utility of various assistive actions toward achieving the goals. In real applications, this requires that the assistant be able to handle uncertainty about the environment and agent, to reason about varying action costs, to handle unforeseen situations, and to adapt to the agent over time. Here we consider a decision-theoretic model, based on partially observable Markov decision processes (POMDPs), which naturally handles these features, providing a formal basis for designing intelligent assistants.

The first contribution of this work is to formulate the problem of selecting assistive actions as an assistant POMDP, which jointly models the application environment along with the agent's policy and hidden goals. A key feature of this approach is that it explicitly reasons about the environment and agent, which provides the potential flexibility for assisting in ways unforeseen by the developer as new situations are encountered. Thus, the developer need not design a hand-coded assistive policy for each of a set of preconceived application scenarios. Instead, when using our framework, the burden on the developer is to provide a model of the application domain and agent, or alternatively a mechanism for learning one or both of these models from experience. Our framework then utilizes those models in an attempt to compute, in any situation, whether assistance could be beneficial and if so what assistive action to select.

In principle, given an assistant POMDP, one could apply a POMDP solver in order to arrive at an optimal assistant policy. Unfortunately, the relatively poor scalability of POMDP solvers will often force us to utilize approximate/heuristic solutions. This is particularly true when the assistant is continually learning updated models of the agent and/or environment, which results in a sequence of more accurate assistant POMDPs, each of which needs to be solved. A second contribution of our work is to describe a fast online action selection mechanism that heuristically approximates the optimal policy of the assistant POMDP. The approach is based on a combination of explicit goal estimation, myopic heuristics, and bounded search. We argue that the approach is well-suited to the assistant POMDP in many application domains even when restricted to small search bounds. We propose two myopic heuristics, one based on solving a set of derived assistant MDPs, and another based on the simulation technique of policy rollout.

In order for the above approach to be useful, the assistant POMDP must incorporate a reasonably accurate model of the agent being assisted. A third contribution of our work is to describe a novel model-based bootstrapping mechanism for quickly learning the agent policy, which is important for the usability of an assistant early in its lifetime. The main idea is to assume that that agent is "close to rational" in the decision-theoretic sense, which motivates defining a prior on agent policies that places higher probability on policies that are closer to optimal. This prior in combination with Bayesian updates allows for the agent model to be learned quickly when the rationality assumption is approximately satisfied, which we argue is often the case in many application.

A fourth contribution of our work is to evaluate our framework in three domains. First we consider two game-like computer environments using 12 human subjects. The results in these domains show that the assistants resulting from our framework substantially reduce the amount of work performed by the human subjects. We also consider a more realistic domain, the folder navigator [5] of the Task Tracer project. In the folder-navigator domain, the user navigates the directory structure searching for a particular location to open or save a file, which is unknown to the assistant. The assistant has the ability to take actions that provide "short cuts" by showing the user a set of three potential destination folders. [5] formulated the problem in a supervised learning framework and applied a cost-sensitive algorithm to predict the 3 most relevant folders at the beginning of the navigation process. We model this problem in our framework, which allows for the assistant to recommend folders at any point during navigation, not just at the beginning. The results show that our generic assistant framework compares favorably to the hand-coded solution of [5].

The remainder of this chapter is organized as follows. In the next section, we introduce our formal problem setup, followed by a definition of the assistant POMDP. Next, we present our approximate solution technique based on goal estimation and online action selection. Finally we give an empirical evaluation of the approach in three domains and conclude with a discussion of related and future work.

2.3 Problem Setup

Throughout the thesis we will refer to the entity that we are attempting to assist as the *agent* (and sometimes as the *user*) and the assisting entity as the *assistant*. We assume that the environment of the agent and assistant is a Markov decision process (MDP) described by the tuple $\langle W, A, A', T, C, I \rangle$, where W is a finite set of world states, A is a finite set of agent actions, A' is a finite set of assistant actions, and $T(w, a, w')$ is a transition distribution that represents the probability of transitioning to state w' given that action $a \in A \cup A'$ is taken in state w. We will sometimes use $T(w, a)$ to denote a random variable distributed as $T(w, a, \cdot)$. We assume that the assistant action set always contains the action **noop** which leaves the state unchanged. The component C is an action-cost function that maps $W \times (A \cup A')$ to real-numbers, and I is an initial state distribution over W. We will sometimes treat I as a random variable whose value represents the initial state.

We consider an episodic problem setting where at the beginning of each episode the agent begins in some state drawn from I and selects a goal from a finite set of possible goals G according to some unknown distribution. The goal set, for example, might contain all possible dishes that the agent might be interested in cooking, or all the possible destination folders that the user may possibly navigate to. When an assistant is not available, the episode proceeds with the agent executing actions from A until it arrives at a goal state upon which the episode ends. When an assistant is present it is able to observe the changing state and the agent's actions, but is unable to observe the agent's goal. At any point along the agent's state trajectory the assistant is allowed to execute a sequence of one or more actions from A' ending in **noop**, after which the agent may again perform an action. The episode ends when either an agent or assistant action leads to a goal state. The cost of an episode is equal to the sum of the costs of the actions executed by the agent and assistant during the episode. Note that the available actions for the agent and assistant need not be the same and may have varying costs. Also note that the cost of the assistant's actions should be viewed as the cost of those actions from the perspective of the agent. The objective of the assistant is to minimize the expected total cost of an episode.

More formally, we will model the agent as an unknown stochastic policy $\pi(a|w, g)$ that gives the probability of selecting action $a \in A$ given that the agent has goal g and is in state w. The assistant is a history-dependent stochastic policy $\pi'(a|w, t)$ that gives the probability of selecting action $a \in A'$ given world state w and the state-action trajectory t observed starting from the beginning of the trajectory. It is critical that the assistant policy depend on t, since the prior states and actions serve as a source of evidence about the agent's goal, which is important for selecting good assistive actions. Given an initial state w, an agent policy π, and assistant policy π' we let $C(w, g, \pi, \pi')$, denote the expected cost of episodes that begin at state w with goal g and evolve according to the following process: 1) execute assistant actions according to π' until **noop** is selected, 2) execute an agent action according to π, 3) if g is achieved then terminate, else go to step 1.

Given the above interaction model, we define the assistant design problem as follows. We are given descriptions of an environment MDP, an agent goal distribution G_0, and an agent policy π. The goal is to select an assistant policy π' that minimizes the expected cost $E[C(I, G_0, \pi, \pi')]$, which is simply the expected cost of interaction episodes for initial states and goals drawn according to I and G_0 respectively. Note that the agent policy π and goal distribution G_0 will not necessarily be known to the assistant early in its lifetime. In such cases, the assistant will need to learn these by observing the agent. One of the contributions of this work, described later, is to introduce an approach for bootstrapping the learning of the agent's policy so as to provide useful early assistance.

Before proceeding it is worth reviewing some of the assumptions of the above formulation and the potential implications. For simplicity we have assumed that the environment is modeled as an MDP, which implies full observability of the world state. This choice is not fundamental to our framework and one can imagine relatively straightforward extensions of our techniques that model the environment as a partially observable MDP (POMDP) where the world states are not fully observable. We have also assumed that the agent is modeled as a memoryless/reactive policy that gives a distribution over actions conditioned on only the current world state and goal. This assumption is also not fundamental to our framework and one can also extend it to include more complex models of the user, for example, that include hierarchical goal structures. Such an extension has recently been explored [56].

We have also assumed for simplicity an interaction model between the assistant and agent that involves interleaved, sequential actions rather than parallel actions. This, for example, precludes the assistant from taking actions in parallel with the agent. While parallel assistive actions are useful in many cases, there are many domains where sequential actions are the norm. We are especially motivated by domains such as “intelligent desktop assistants” that help store and retrieve files more quickly, helps sort email, etc., and “smart homes” that open doors, switch on appliances and so on. Many opportunities for assistance in these domains is of the sequential variety. Also note that in many cases, tasks that appear to require parallel activity can often be formulated as a set of threads where each thread is sequential and hence can be formulated as a separate assistant. Extending our framework to handle general parallel assistance is an interesting future direction.

Finally, note that our sequential interaction model assumes that the assistant is allowed to take an arbitrary number of actions until selecting NOOP, upon which the agent is allowed to select a single action. This at first may seem like an unreasonable formulation since it leaves open the possibility that the assistant may “take control” indefinitely long, forcing the agent to pause until the assistant has finished. Note, however, that a properly designed assistant would never take control for a long period unless the overall cost to the agent (including annoyance cost) was justified. Thus, in our framework, with an appropriately designed cost function for assistive and agent actions, the assistant can be discouraged from engaging in a long task without giving control to the user, so the problem of having to wait for the assistant does not arise. Also note that assistant actions will often be on a much smaller time scale than agent actions, allowing the assistant to perform a potentially complex sequence of actions without delaying the user (e.g. carrying out certain file management tasks). In addition, a developer could restrict the number of actions selected by the assistant by defining an appropriate MDP, e.g. one that forces the assistant to select NOOP after each non-NOOP action.

2.4 The Assistant POMDP

POMDPs provide a decision-theoretic framework for decision making in partially observable stochastic environments. A POMDP is defined by a tuple $\langle S, A, T, C, I, O, \mu \rangle$, where the first five components describe an MDP with a finite set of states S, finite set of actions A, transition distribution $T(s, a, s')$, action cost function C, and initial state distribution I. The component O is a finite set of observations and $\mu(o|s, a, s')$ is a distribution over observations $o \in O$ generated upon taking action a in state s and transitioning to state s'. The primary distinction between a POMDP and an MDP is that a controller acting in a POMDP is not able to directly observe the states of the underlying MDP as it evolves. Rather, a controller is only able to observe the observations that are stochastically generated by the underlying state-action sequence. Thus, control of POMDPs typically involves dealing with both the problem of resolving uncertainty about the underlying MDP state based on observations and selecting actions to help better identify the state and/or make progress toward the goal.

A policy for a POMDP defines a distribution over actions given the sequence of preceding observations. It is important that the policy depends on the history of observations rather than only the current observation since the entire history can potentially provide evidence about the current state. It is often useful to view a POMDP as an MDP over an infinite set of belief states, where a belief state is simply a distribution over S. In this case, a POMDP policy can be viewed as a mapping from belief states to actions. Note, however, that the belief-state MDP has an infinite state space (the uncountable space of distributions over S) and hence cannot be directly solved by standard techniques such as dynamic programming which are typically defined for finite state spaces.

We will use a POMDP to jointly model the agent and environment from the perspective of the assistant. Our goal is to define this "assistant POMDP" such that its optimal policy is an optimal solution to the assistant design problem defined in the previous section. The assistant POMDP will allow us to address the two main challenges in selecting assistive actions. The first challenge is to infer the agent's hidden goals, which is critical to provide good assistance. The assistant POMDP will capture goal uncertainty by including the agent's goal as a hidden component of the POMDP state. In effect, the belief states of the assistant POMDP will correspond exactly to a distribution over possible agent's goals. The second challenge is that even if we know the agent's goals, we must jointly reason about the possibly stochastic environment, agent policy, and action costs in order to select the best course of action. Our POMDP will capture this information in the transition function and cost model, providing a decision-theoretic basis for such reasoning.

More formally, given an environment MDP $\langle W, A, A', T, C, I \rangle$, a goal distribution G_0 over a finite set of possible goals G, and an agent policy π we now define the corresponding components of the *assistant POMDP* $\langle S', A', T', C', I', O', \mu' \rangle$:

- The **state space** S' is $W \times G$ so that each state is a pair (w, g) of a world state and agent goal. The world state component will be observable to the assistant, while the goal component will be unobservable and must be inferred. Note that according to this definition, a belief state (i.e. a distribution over S') corresponds to a distribution over the possible agent's goals G.

- The **action set** A' is just the set of assistant actions specified by the environment MDP. This choice reflects the fact that the assistant POMDP is used exclusively for selecting actions for the assistant.

- The **transition function** T' assigns zero probability to any transition that changes the goal component of the state. This reflects the fact that the assistant cannot change the agent's goal by taking actions. Thus the agent's goal remains fixed throughout any episode of the assistant POMDP. Otherwise, for any action a except for **noop**, the state transition from (w, g) to (w', g) has probability $T'((w, g), a, (w', g)) = T(w, a, w')$. This reflects the fact that when the agent takes a non-noop action the world will transition according the transition function specified by the environment MDP. Otherwise, for the **noop** action, T' simulates the effect of executing an agent action selected according to π and then transitioning according to the environment MDP dynamics. That is, $T'((w,g), \textbf{noop}, (w',g)) = T(w, \pi(w,g)) = w'$.

- The **cost model** C' reflects the costs of agent and assistant actions in the MDP. For all actions a except for **noop** we have that $C'((w,g),a) = C(w,a)$. Otherwise we have that $C'((w,g), \textbf{noop}) = E[C(w,a)]$, where a is a random variable distributed according to $\pi(\cdot|w,g)$. That is, the cost of a **noop** action is the expected cost of the ensuing agent action.

- The **initial state distribution** I' assigns the state (w,g) probability $I(w)G_0(g)$, which models the process of independently selecting an initial state and goal for the agent at the beginning of each episode.

- The **observation set** O' is equal to $W \times (A \cup A')$, that is all pairs of world states and agent/assistant actions.

- The **observation distribution** μ' is deterministic and reflects the fact that the assistant can only directly observe the world state and actions. For the **noop** action in state (w,g) leading to state (w',g), the observation is (w',a) where $a \in A$ is the action executed by the agent immediately after the **noop**. Note that the observations resulting from **noop** actions are informative for inferring the agent's goal since they show the choice of action selected by the the agent in a particular state, which depends on the goal. For all other assistant actions the observation is equal to the W component of the state and the assistant action, i.e. $\mu'((w,g),a,(w',g)) = (w',a)$. Note that the observations resulting from non-noop actions are not informative with respect to inferring the agent's goals.

Note that according to the above definition, a policy π' for the assistant POMDP is a mapping from state-action sequences (i.e. the observations are pairs of states and agent actions) to assistant actions. We must now define an objective function that will be used to evaluate the value of a particular policy. For this purpose, we will assume an episodic setting, where each assistant POMDP episode begins by drawing an initial POMDP state (w, g) from I'. Actions are then selected according to the assistant policy π' and transitions occur according to T' until a state (w', g) is reached where w' satisfies the goal g. Note that whenever the assistant policy selects **noop**, the transition dictated by T' simulates a single action selected by the agent's policy, which corresponds to the sequential interaction model introduced in the previous section. Our objective function for the assistant POMDP is the expected cost of a trajectory under π'. Note that this objective function corresponds exactly to our objective function for the assistant design problem $E[C(I, G_0, \pi, \pi')]$ from the previous section. Thus, solving for the optimal assistant POMDP policy yields an optimal assistant.

There are two main obstacles to solving the assistant POMDP and in turn the assistant design problem. First, in many scenarios, initially the assistant POMDP will not be directly at our disposal since we will lack accurate information about the agent policy π and/or the goal distribution G_O. This is often due to the fact that the assistant will be deployed for a variety of initially unknown agents. Rather, we will often only be given a definition of the environment MDP and the possible set of goals. As described in the next section, our approach to this difficulty is to utilize an approximate assistant POMDP by estimating π and G_0. Furthermore, we will also describe a bootstrapping mechanism for learning these approximations quickly. The second obstacle to solving the assistant POMDP is the generally high computational complexity of finding policies for POMDPs. To deal with this issue Section 2.6 considers various approximate techniques for efficiently solving the assistant POMDP.

2.5 Learning the Assistant POMDP

In this section, we will assume that we are provided with the environment MDP and the set of possible agent goals and that the primary role for learning is to acquire the agent's policy and goal distribution. This assumption is natural in situations where the assistant is being applied many times in the same environment, but for different agents. For example, in a computer desktop environment, the environment MDP corresponds to a description of the various desktop functionalities, which remains fixed across users. If one is not provided with a description of the MDP then it is typically straightforward to learn this model with the primary cost being a longer "warming up" period for the assistant.

Relaxing the assumption that we are provided with the set of possible goals is more problematic in our current framework. As we will see in Section 2.6, our solution methods will all depend on knowing this set of goals and it is not clear how to learn these from observations, since the goals, unlike states and actions, are not directly observable to the assistant. Extending our framework so that the assistant can automatically infer the set of possible user goals, or allow the user to define their own goals, is an interesting future direction. We note, however, it is often possible for a designer to enumerate a set of user goals before deployment that while perhaps not complete, allow for useful assistance to be provided.

2.5.1 Maximum Likelihood Estimates

It is straightforward to estimate the goal distribution G_0 and agent policy π by simply observing the agent's actions, possibly while being assisted, and to compute empirical estimates of the relevant quantities. This can be done by storing the goal achieved at the end of each episode along with the set of world state-action pairs observed for the agent during the episode. The estimate of G_0 can then be based on observed frequency of each goal (usually with Laplace correction). Likewise, the estimate of $\pi(a|w,g)$ is simply the frequency for which action a was taken by the agent when in state w and having goal g. While in the limit these maximum likelihood estimates will converge to the correct values, yielding the true assistant POMDP, in practice convergence can be slow. This slow convergence can lead to poor performance in the early stages of the assistant's lifetime. To alleviate this problem we propose an approach for bootstrapping the learning of the agent policy π.

2.5.2 Model-Based Bootstrapping

We will leverage our environment MDP model in order to bootstrap the learning of the agent policy. In particular, we assume that the agent is reasonably close to being optimal. That is, for a particular goal and world state, an agent is more likely to select actions that are closer to optimal. We term this assumption as the *rationality assumption*. This is not unrealistic in many application domains that might benefit from intelligent assistants. In particular, there are many tasks, that are conceptually simple for humans, yet they are quite tedious and require substantial effort to complete. For example, navigating through the directory structure of a computer desktop.

Given the rationality assumption, we initialize the estimate of the agent's policy to a prior that is biased toward more optimal agent actions. To do this we will consider the environment MDP with the assistant actions removed and solve for the Q-function $Q(a,w,g)$ using MDP planning techniques. The Q-function gives the expected cost of executing agent action a in world state w and then acting optimally to achieve goal g using only agent actions. In a world without an assistant, a rational agent would always select actions that maximize the Q-function for any state and goal. Furthermore, a close-to-rational agent would prefer actions that achieve higher Q-values to highly suboptimal actions. We first define the Boltzmann distribution, which will be used to define our prior,

$$\hat{\pi}(a|w,g) = \frac{1}{Z(w,g)} \exp(K \cdot Q(a,w,g))$$

where $Z(w,g)$ is a normalizing constant, and K is a temperature constant. Using larger values of K skews the distribution more heavily toward optimal actions. Given this definition, our prior distribution over $\pi(\cdot|w,g)$ is taken to be a Dirichlet with parameters $(\alpha_1, \ldots, \alpha_{|A|})$, where $\alpha_i = \alpha_0 \cdot \hat{\pi}(a_i|w,g)$. Here α_0 is a parameter that controls the strength of the prior. Intuitively α_0 can be thought of as the number of pseudo-actions represented by the prior, with each α_i representing the number of those pseudo-actions that involved agent action a_i. Since the Dirichlet is conjugate to the multinomial distribution, which is the form of $\pi(\cdot|w,g)$, it is easy to update the posterior over $\pi(\cdot|w,g)$ after each observation. One can then take the mode or mean of this posterior to be the point estimate of the agent policy used to define the assistant POMDP.

In our experiments, we found that this prior provides a good initial proxy for the actual agent policy, allowing for the assistant to be immediately useful. Further updating of the posterior tunes the assistant better to the peculiarities of a given agent. For example, in many cases there are multiple optimal actions and the posterior will come to reflect any systematic bias for equally good actions that an agent has. Computationally the main obstacle to this approach is computing the Q-function, which needs to be done only once for a given application domain since the environment MDP is constant. Using dynamic programming this can accomplished in polynomial time in the number of states and goals. When this is not practical, a number of alternatives exist including the use of factored MDP algorithms [11], approximate solution methods [11, 31], or developing domain specific solutions.

Finally, in this work, we utilize an uninformative prior over the goal distribution. An interesting future direction would be to bootstrap the goal distribution estimate based on observations from a population of agents.

2.6 Solving the Assistant POMDP

We now consider the problem of solving the assistant POMDP. Unfortunately, general purpose POMDP solvers are generally quite inefficient and not practical to use in many cases. This is particularly true in our framework where the assistant POMDP is continually being refined by learning more accurate estimates of the agent goal distribution and agent policy, which requires re-solving the assistant POMDP when the model is updated. For this reason, we adopt a Bayesian goal estimation followed by a heuristic action selection approach that we argue is natural for many assistant POMDPs and we show that it works well in practice. Below, we first give an overview of our solution algorithm and then describe each of the components in more detail.

2.6.1 Overview

Denote the assistant POMDP by $M = \langle S', A', T', C', I', O', \mu' \rangle$ and let $O_t = o_1, ..., o_t$ be an observation sequence observed by assistant from the beginning of the current trajectory until time t. Note that each observation is a tuple of a world state and the previously selected action (by either the assistant or agent). Given O_t and M our goal is to compute an assistant action whose value is close to optimal.

To motivate the approach, it is useful to consider some special characteristics of the assistant POMDP. Most importantly, the belief state corresponds to a distribution over the agent's goal. Since the agent is assumed to be goal directed, the observed agent actions provide substantial evidence about what the goal might and might not be. In fact, even if the assistant does nothing, the agent's goals will often be rapidly revealed by analyzing the relevance of the agent's initial actions to the possible goals. This suggests that the state/goal estimation problem for the assistant POMDP may be solved quite effectively by just observing how the agent's actions relate to the various possible goals, rather than requiring the assistant to select actions explicitly for the purpose of information gathering about the agent's goals. In other words, we can expect that for many assistant POMDPs, purely (or nearly) myopic action selection strategies, which avoid reasoning about information gathering, will be effective. Reasoning about information gathering is one of the key complexities involved in solving POMDPs compared to MDPs. Here we leverage the intuitive properties of the assistant POMDP to gain tractability by limiting or completely avoiding such reasoning.

We note that in some cases, the assistant will have pure information-gathering actions at its disposal, e.g. asking the agent a question. While we do not consider such actions in our experiments, we believe that such actions can be handled naturally in this framework by incorporating only a small amount of look-ahead search.

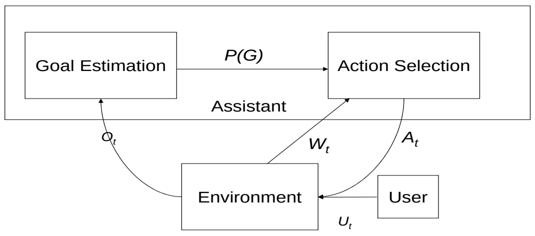


Figure 2.1: Depiction of the assistant architecture. The user/agent has a hidden goal and selects actions U_t that cause the environment to change world state W_t, typically moving closer to the goal. The assistant (upper rectangle) is able to observe the world state along with the observations generated by the environment, which in our setting contain the user/agent actions along with the world state. The assistant is divided into two components. First, the goal estimation component computes a posterior over agent goals $P(G)$ given the observations. Second, the action selection component uses the goal distribution to compute the best assistive action A_t via a combination of bounded search and myopic heuristic computation. The best action might be **noop** in cases where none of the other assistive actions has higher utility for the user.

With the above motivation, our assistant architecture, depicted in Figure 2.1, alternates between *goal estimation* and *action selection* as follows:

1. After observing the agent's next action, we update the goal distribution based on the assistant POMDP model.

2. Based on the updated distribution we evaluate the effectiveness of assistant actions (including **noop** by building a sparse-sampling look-ahead tree of bounded depth (perhaps just depth one), where leaves are evaluated via a myopic heuristic. If the best action is **noop** then control is given to the agent and we go to step 1, otherwise we repeat step 2.

The key element of the architecture is the computation of the myopic heuristics. On top of this heuristic, we can optionally obtain non-myopic behavior via search by building a look-ahead sparse-sampling tree. Our experiments show that such search can improve performance by a small margin at a significant computational cost. We note that the idea of utilizing myopic heuristics to select actions in POMDPs is not new, see for example [14, 26], and similar methods have been used previously with success in applications such as computer bridge [30]. The main contribution here is to note that this approach seems particularly well suited to the assistant POMDP and also in later sections to suggest some efficiently computable heuristics that are specifically designed for the assistant POMDP framework. Below we describe the goal estimation and action selection operations in more detail.

2.6.2 Goal Estimation

Given an assistant POMDP with agent policy π and initial goal distribution G_0, our objective is to maintain the posterior goal distribution $P(g|O_t)$, which gives the probability of the agent having goal g conditioned on observation sequence O_t. Note that since we have assumed that the assistant cannot affect the agent's goal, only observations related to the agent's actions are relevant to the posterior. Given the agent policy π, it is straightforward to incrementally update the posterior $P(g|O_t)$ upon each of the agent's actions by referring to the Bayesian network of Figure 2.2. The node g refers to the current goal of the user and is the only hidden variable that is distributed according to the current goal posterior, A_t denotes the agent's action at time t, and W_t refers to the world state at time-step t. The agent action A_t is distributed according to the agent's policy π.

At the beginning of each episode we initialize the goal distribution $P(g|O_0)$ to G_0. On timestep t of the episode, if o_t does not involve an agent action, then we leave the distribution unchanged. Otherwise, when the agent selects action a in state w, we update the posterior according to $P(g|O_t) = (1/Z) \cdot P(g|O_{t-1}) \cdot \pi(a|w, g)$, where Z is a normalizing constant. That is, the distribution is adjusted to place more weight on goals that are more likely to cause the agent to execute action a in w. The accuracy of goal estimation relies on

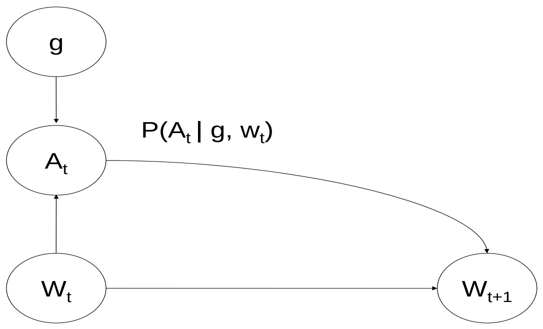


Figure 2.2: The Bayes net used to compute the posterior distribution over the agent's goals. The node A_t represents the agent action that is conditioned on the agent's goal G and current world state W_t. In the simplest setting G is the only hidden variable with both the agent actions and states being observable. In other cases, the agent action may also be hidden, with only the state transition being directly observable.

how well the policy π learned by the assistant reflects the true agent policy. As described above, we use a model-based bootstrapping approach for estimating π and update this estimate at the end of each episode. Provided that the agent is close to optimal, as in our experimental domains, this approach can lead to rapid goal estimation, even early in the lifetime of the assistant.

We have assumed for simplicity that the actions of the agent are directly observable. In some domains, it is more natural to assume that only the state of the world is observable, rather than the actual action identities. In these cases, after observing the agent transitioning from w to w' we can use the MDP transition function T to marginalize over possible agent actions yielding the update,

$$P(g|O_t) = (1/Z) \cdot P(g|O_{t-1}) \cdot \sum_{a \in A} \pi(a|w,g)T(w,a,w').$$

2.6.3 Action Selection

Given the assistant POMDP M and a distribution over goals $P(g|O_t)$, we now address the problem of selecting an assistive action. Our mechanisms utilize a combination of bounded look-ahead search and myopic heuristic computations. By increasing the amount of look-ahead search the actions returned will be closer to optimal at the cost of more computation. Fortunately, for many assistant POMDPs, useful assistant actions can be computed with relatively little or no search. We first describe several myopic heuristics that can be used either for greedy action selection or in combination with search. Next, we describe how to utilize sparse sampling to obtain non-myopic action selection.

2.6.3.1 Myopic Heuristics

To explain the action selection procedure, we introduce the idea of an *assistant MDP* relative to a goal g and M, which we will denote by $M(g)$. The MDP $M(g)$ is identical to M except that we change the initial goal distribution such that $P(G_0 = g) = 1$. That is, the goal is always fixed to g in each episode. Since the only hidden component of M's state space was the goal, fixing the goal in $M(g)$ makes the state fully observable, yielding an MDP. Each episode in $M(g)$ evolves by drawing an initial world state and then selecting assistant actions until a **noop**, upon which the agent executes an action drawn from its policy for achieving goal g. An optimal policy for $M(g)$ gives the optimal assistive action assuming that the agent is acting to achieve goal g. We will denote the Q-function of $M(g)$ by $Q_g(w, a)$, which is the expected cost of executing action a and then following the optimal policy. We assume that the MDP $M(g)$ can be solved easily. We call this assumption as the *tractability assumption*

Our first myopic heuristic is simply the expected Q-value of an action over assistant MDPs. This heuristic has also been called the Q_{MDP} method in [14]. The heuristic value for assistant action a in state w given observations O_t is

$$H(w, a, O_t) = \sum_g Q_g(w, a) \cdot P(g|O_t).$$

Intuitively $H(w, a, O_t)$ measures the utility of taking an action under the assumption that all goal ambiguity is resolved in one step. Thus, this heuristic will not value the information-gathering utility of an action. Rather, the heuristic will favor assistant actions that make progress toward goals with high posterior probability, while avoiding moving away from goals with high probability. When the goal posterior is highly ambiguous this will often lead the assistant to prefer **noop**, which at least does not hurt progress toward the goal. Note that this heuristic, as well as the others below, can be used to evaluate the utility of a state w, rather than a state-action pair, by maximizing over all actions $\max_a H(w', a, O_{t'})$.

The primary computational complexity of computing H is to solve the assistant MDPs for each goal in order to obtain the Q-functions. Technically, since the transition functions of the assistant MDPs depend on the approximate agent policy π, we must re-solve each

MDP after updating the π estimate at the end of each episode. However, using incremental dynamic programming methods such as prioritized sweeping [50] can alleviate much of the computational cost. In particular, before deploying the assistant we can solve each MDP offline based on the default agent policy given by the Boltzmann bootstrapping distribution described earlier. After deployment, prioritized sweeping can be used to incrementally update the Q-values based on the learned refinements we make to π.

When it is not practical to exactly solve the assistant MDPs, we may resort to various approximations. We consider two approximations in our experiments. One is to replace the user's policy to be used in computing the assistant MDP with a fixed default user policy, eliminating the need to compute the assistant MDP at every step. We denote this approximation by H_d. Another approximation uses the simulation technique of *policy rollout* [7] to approximate $Q_g(w, a)$ in the expression for H. This is done by first simulating the effect of taking action a in state w and then using π to estimate the expected cost for the agent to achieve g from the resulting state. That is, we approximate $Q_g(w, a)$ by assuming that the assistant will only select a single initial action followed by only agent actions. More formally, let $\bar{C}_n(\pi, w, g)$ be a function that simulates n trajectories of π achieving the goal from state w and then averaging the trajectory costs. The heuristic H_r is identical to $H(w, a, O_t)$ except that we replace $Q_g(w, a)$ with the expectation $\sum_{w' \in W} T(w, a, w') \cdot \bar{C}(\pi, w', g)$. We can also combine both of these heuristics, using a fixed default user policy and policy rollouts, which we denote by $H_{d,r}$.

2.6.3.2 Sparse Sampling

All of the above heuristics can be used to greedily select assistant actions, resulting in purely myopic action-selection strategies. In cases where it is beneficial to include some amount of non-myopic reasoning, one can combine these heuristics with shallow search in the belief space of the assistant MDP. For this purpose we utilize depth d bounded sparse sampling trees [40] to compute an approximation to the Q-function for a given belief state (w_t, O_t), denoted by $Q^d(w_t, a, O_t)$. Given a particular belief state, the assistant will then select the action that maximizes Q^d. Note that for convenience we represent the belief state as a pair of the current state w_t and observation history O_t. This is a lossless representation of the belief state since the posterior goal distribution can be computed exactly from O_t and the goal is the only hidden portion of the POMDP state.

As the base case $Q^0(w_t, a, O_t)$ will be equal to one of our myopic heuristics described above. Increasing the depth d will result in looking ahead d state transitions and then evaluating one of our heuristics. By looking ahead it is possible to track the potential changes to the belief state after taking certain actions and then determine whether those changes in belief would be beneficial with respect to providing better assistance. Sparse sampling, does such look-ahead by approximately computing:

$$Q^d(w, a, O) = E[C'((w, g), a) + V^{d-1}(w', O')] \tag{2.1}$$

$$V^d(w, O) = \min_a Q^d(w, a, O) \tag{2.2}$$

where g is a random variable distributed according to the goal posterior $P(g|O)$ and (w', O') is a random variable that represents the belief state after taking action a in belief state (w, O). In particular, w' is that world state arrived at and O' is simply the observation sequence O extended with the observation obtained during the state transition. The first term in the above expectation represents the immediate cost of the assistant action a. According to the definition of the assistant POMDP, this is simply the cost of the assistant action in the underlying MDP for non-noop actions. For the noop action, the cost is equal to the expected cost of the agent action.

Sparse sampling approximates the above equations by averaging a set of b samples of successor belief states to approximate the expectation. The sparse-sampling pseudo-code is presented in Table 2.1.Given an input belief state (w, O), assistant action a, heuristic

H, depth bound d, and sampling width b the algorithm returns (an approximation of) $Q^d(w, a, O)$. First, if the depth bound is equal to zero the heuristic value is returned. Otherwise b samples of observations resulting from taking action a in belief state (w, O) are generated. In the case that the action is not **noop**, the observation is created by simulating the effect of the assistant action in the environment MDP and then forming an observation from the pair of resulting state and action a (recall that observations are pairs of states and previous actions). In the case that the assistant action is **noop**, the observation is generated by sampling an agent goal and an action and then simulating the environment MDP for the agent action. The observation is simply the pair of resulting state and agent action. Each observation $o_i = (w_i, a_i)$ corresponds to a new belief state $(w_i, [O; o_i])$ where $[O; o_i]$ is simply the concatenation of o_i to O. The code then recursively computes a value for each of these belief states by minimizing Q^d over all actions and then averages the results.

As b and d become large, sparse sampling will produce an arbitrarily close approximation to the true Q-function of the belief state MDP. The computational complexity of sparse sampling is linear in b and exponential in d. Thus the depth must be kept small for real-time operation.

2.7 Experimental Results

In this section, we present the results of conducting user studies and simulations in three domains: two game-like environments and a folder predictor domain for an intelligent desktop assistant. In the user studies in the two game-like domains, for each episode, the user's and the assistant's actions were recorded. The ratio of the cost of achieving the goal with the assistant's help to the optimal cost without the assistant was calculated and averaged over the multiple trials for each user. We present similar results for the simulations as well. The third domain is a folder predictor domain, where we simulated

Table 2.1: Pseudo-code for Sparse Sampling in the Assistant POMDP

```
• Given: heuristic function $H$, belief state $(w, O)$, action $a$, depth bound $d$, sampling width $b$
• Return: an approximation $Q^d(w, a, O)$ of the value of $a$ in belief state $(w, O)$
1. If $d = 0$ then return $H(w, a, O)$
2. Sample a set of $b$ observations $\{o_1, \ldots, o_b\}$ resulting from taking action $a$ in belief state $(w, O)$ as follows:
    (a) If $a \neq$ noop then $a_i = a$, otherwise,
    (b) If $a =$ noop then
        • Sample a goal $g$ from $P(g|O)$
        • Sample an agent action $a_i$ from the agent policy $\pi(\cdot|w, g)$
    (c) $o_i = (w_i, a_i)$, where $w_i$ is sample from the environment MDP transition function $T(w, a_i, \cdot)$
3. For each $o_i = (w_i, a_i)$ compute $V_i = \min_{a'} Q^{d-1}(w_i, a', [O; o_i])$
4. Return $Q^d(w, a, O) = C(w, a_i) + \frac{1}{b} \sum_i V_i$
```

the user and used one of our heuristics to generate the top 3 recommended folders for the user. We present the number of clicks required on an average for the user to reach his desired folder.

2.7.1 Doorman Domain

In the doorman domain, there is an agent and a set of possible goals such as collect *wood*, *food* and *gold*. Some of the grid cells are blocked. Each cell has four doors and the agent has to open the door to move to the next cell (see Figure 2.3). The door closes after one time-step so that at any time only one door is open. The goal of the assistant is to help the user reach his goal faster by opening the correct doors.

A state is a tuple $\langle s, d \rangle$, where s stands for the the agent's cell and d is the door that is open. The actions of the agent are to open door and to move in each of the 4 directions or to pickup whatever is in the cell, for a total of 9 actions. The assistant can open the doors or perform a **noop** (5 actions). Since the assistant is not allowed to push the agent through the door, the agent's and the assistant's actions strictly alternate in this domain. There is a cost of -1 if the user has to open the door and no cost to the assistant's action. The trial ends when the agent picks up the desired object.

In this experiment, we evaluated the two heuristics: one where we fixed the user policy to the default policy in the assistant POMDP creation (H_d) and the second where we use the policy rollout to calculate the Q-values (H_r). In each trial, the system chooses a goal and one of the two heuristics at random. The user is shown the goal and he tries to achieve it, always starting from the center square. After every user's action, the assistant opens a door or does nothing. The user may pass through the door or open a different door. After the user achieves the goal, the trial ends, and a new one begins. The assistant then uses the user's trajectory to update the agent's policy.

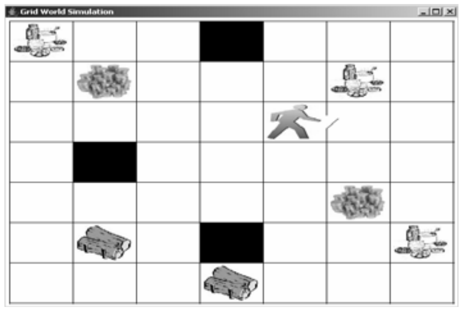


Figure 2.3: Doorman Domain. The agent's goal is to fetch a resource. The grid cells are separated by doors that must be opened before passing through.

The results of the user studies for the doorman domain are presented in Figure 2.2. The first two rows give cumulative results for the user study when actions are selected greedily according to H_r and H_d respectively. The table presents the total optimal costs (number of actions) for all trials across all users without the assistant N, and the costs with the assistant U, and the average of percentage cost savings (*1-(U/N)*) over all trials and over all the users[1].

As can be seen, both the methods reduce the cost to less than 50%. An omniscient assistant who knows the user's goal reduces the cost to 22%(i.e., the assistant performs 78% of the actions). This is not 0 because the first door is always opened by the user. In our experiments, if we do not count the user's first action, the cost reduces to 35%(the assistant performs 65% of the actions). It can be observed that H_r appears to have a slight edge over H_d. One possible reason for this is that while using H_d, we do not re-solve the MDP after updating the user policy, while H_r is always using the updated user policy. Thus, rollout is reasoning with a more accurate model of the user.

Heuristic	Total Actions N	User Actions U	Fractional Savings $1-(U/N)$	Time per action (in secs
H_r	750	339	0.55± 0.055	0.0562
H_d	882	435	0.51 ± 0.05	0.0021
H_r	1550	751	0.543 ± 0.17	0.031
d = 2, b = 1	1337	570	0.588 ± 0.17	0.097
d = 2, b = 2	1304	521	0.597 ± 0.17	0.35
d = 3, b = 1	1167	467	0.6 ± 0.15	0.384
d = 3, b = 2	1113	422	0.623 ± 0.15	2.61

Table 2.2: Results of experiments in the Doorman Domain. The first two rows of the table present the results of the user studies while the rest of the table presents the results of the simulation.

Another interesting observation is that there are individual differences among the users. Some users always prefer a fixed path to the goal regardless of the assistant's actions. Some users are more flexible. From the survey, we conducted at the end of the experiment, we learned that one of the features that the users liked was that the system was tolerant to their choice of suboptimal paths. The data reveals that the system was able to reduce the costs by approximately 50% even when the users chose suboptimal

[1]This gives a pessimistic estimate of the usefulness of the assistant assuming an optimal user and is a measure of utility normalized by the optimal utility without the aid of the assistant.

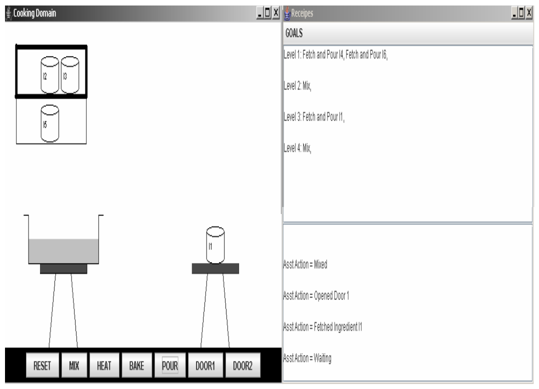
trajectories.

We also conducted experiments using sparse sampling with non-trivial depths. We considered depths of $d = 1$ and $d = 2$ while using sampling widths of $b = 1$ or $b = 2$. The leaves of the sparse sampling tree are evaluated using H_r which simply applies rollout to the user policy. Hence sparse sampling of $d = 0$ and $b = 1$, would correspond to the heuristic H_r. For these experiments, we did not conduct user studies, due to the high cost of such studies, but simulated the human users by choosing actions according to policies learned from their observed actions. The results are presented in the last 5 rows of Table 2.2. We see that sparse sampling increased the average run time by an order of magnitude, but is able to produce a reduction in average cost for the user. This result is not surprising in hindsight, for in the simulated experiments, sparse sampling is able to sample from the exact user policy (i.e. it is sampling from the learned policy, which is also being used for simulations). These results suggest that a small amount of non-myopic reasoning can have a positive benefit with a substantial computation cost. Note, however, that the bulk of the benefit realized by the assistant can be obtained without such reasoning, showing that the myopic heuristics are well-suited to this domain.

2.7.2 Kitchen Domain

In the kitchen domain, the goals of the agent are to cook various dishes. There are 2 shelves with 3 ingredients each. Each dish has a recipe, represented as a partially ordered plan. The ingredients can be fetched in any order, but should be mixed before they are heated. The shelves have doors that must be opened before fetching ingredients and only one door can be open at a time.

There are 8 different recipes. The state consists of the location of each of the ingredient (bowl/shelf/table), the mixing state and temperature state of the ingredient (if it is in the bowl) and the door that is open. The state also includes the action history to preserve



Level 1: Fetch and Pour I4, Fetch and Pour I6,

Level 2: Mix,

Level 3: Fetch and Pour I1,

Level 4: Mix,

Asst Action = Mixed

Asst Action = Opened Door 1

Asst Action = Fetched Ingredient I1

Asst Action = Waiting

Figure 2.4: The kitchen domain. The user is to prepare the dishes described in the recipes on the right. The assistant's actions are shown in the bottom frame.

the ordering of the plans for the recipes. The user's actions are: open the doors, fetch the ingredients, pour them into the bowl, mix, heat and bake the contents of the bowl, or replace an ingredient back to the shelf. The assistant can perform all user actions except for pouring the ingredients or replacing an ingredient back to the shelf. The cost of all non-pour actions is -1. Experiments were conducted on 12 human subjects. Unlike in the doorman domain, here it is not necessary for the assistant to wait at every alternative time step. The assistant continues to act until the **noop** becomes the best action according to the heuristic.

This domain has a large state space and hence it is not possible to update the user policy after every trajectory. The two heuristics that we compare both use the default user policy. The second heuristic in addition uses policy rollout to compare the actions. In other words, we compare H_d and $H_{d,r}$. The results of the user studies are shown in top part of the Table 2.3. The total cost of user's optimal policy, the total cost when the assistant is present, and the average ratio of the two are presented. The number of user actions was summed over 15 users and the cumulative results are presented. It can be observed that $H_{d,r}$ performs better than H_d. It was observed from the experiments that the $H_{d,r}$ technique was more aggressive in choosing non-noop actions than the H_d, which would wait until the goal distribution is highly skewed toward a particular goal.

Heuristic	Total Actions N	User Actions U	Fractional Savings $1-(U/N)$	Time per action (secs)
$H_{d,r}$	3188	1175	0.6361 ±0.15	0.013
H_d	3175	1458	0.5371± 0.10	0.013
$H_{d,r}$	6498	2332	0.6379 ± 0.14	0.013
d = 2, b = 1	6532	2427	0.6277 ± 0.14	0.054
d = 2, b = 2	6477	2293	0.646 ± 0.14	0.190
d = 3, b = 1	6536	2458	0.6263 ±0.15	0.170
d = 3, b = 2	6585	2408	0.645 ± 0.14	0.995

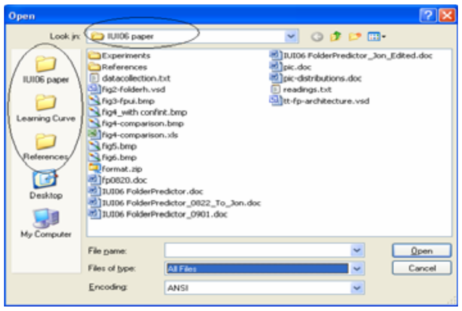
Table 2.3: Results of experiments in the Kitchen Domain. The first two rows of the table present the results of the user studies while the last 5 rows present the results of the simulation.

We compared the use of sparse sampling and our heuristic on simulated user trajectories for this domain as well (see the last 5 rows of Table 2.3). It can be observed that the total number of user actions is much higher for the simulations than the user studies due to the fact that user studies are costly. The simulations had to consider a much larger state-space than the one used in the user studies. Because of this, the policies learned on simulations are significantly cheaper than in the user studies, although they took more time to execute. There is no significant difference between the solution quality of rollouts and sparse sampling on simulations, showing that our myopic heuristics are performing as well as sparse sampling with much less computation. Sparse sampling with higher depths requires an order of magnitude more computation time when compared to the rollout.

2.7.3 Folder Predictor

In this section, we present the evaluation of our framework on a real-world domain. As a part of the Task Tracer project [22], researchers developed a file location system called *folder predictor* [5]. The idea behind the folder predictor is that by learning about the user's file access patterns, the assistant can help the user with his file accesses by predicting the folder in which the file has to be accessed or saved.

In this setting, the goal of the folder predictor is to minimize the number of clicks of the user. The predictor would choose the top three folders that would minimize the cost and then append them to the UI (shown in ovals in Figure 2.5). Also, the user is taken to the first recommended folder. So if the user's target folder is the first recommended folder, the user would reach the folder in zero clicks and reach the second or the third recommended folder in one click. The user can either choose one of the recommendations or navigate through the windows folder hierarchy if the recommendations are not relevant.



Look in: IUI06 paper
File name:
Files of type: All Files
Encoding: ANSI

Figure 2.5: Folder predictor [5].

Bao et al. considered the problem as a supervised learning problem and implemented a cost-sensitive algorithm for the predictions with the cost being the number of clicks of the user [5]. But, their algorithm does not take into account the response of the user to their predictions. For instance, if the user chooses to ignore the recommended folders and navigates the folder hierarchy, they do not make any re-predictions. This is due to the fact that their model is a one-time prediction and does not consider the user responses. Also, their algorithm considers a restricted set of previously accessed folders and their ancestors as possible destinations. This precludes handling the possibility of user accessing a new folder.

Our decision-theoretic model naturally handles the case of re-predictions by changing the recommendations in response to the user actions. As a first step, we used the data collected from their user interface and used our model to make predictions. We use the user's response to our predictions to make further predictions. Also, to handle the possibility of a new folder, we consider all the folders in the folder hierarchies for each prediction. We used a mixture density to obtain the probability distribution over the folders.

$$P(f) = \mu_0 P_0(f) + (1 - \mu_0) P_l(f)$$

Here P_0 is the probability according to Bao et.al's algorithm, P_l is the uniform probability distribution over the set of folders and μ_0 is ratio of the number of times a previously accessed folder has been accessed to the total number of folder accesses.

The idea behind using the above density function is that during early stages of a task, the user will be accessing new folders while in later stages the user will access the folders of a particular task hierarchy. Hence as the number of folder accesses increase the value of μ_0 increases and would converge to 1 eventually and hence the resulting distribution would converge to P_0. The data set consists of a collection of requests to open a file (Open) and save a file (saveAs), ordered by time. Each request contains information such as, the type of request (open or saveAs), the current task, the destination folder etc. The data set consists of a total of 810 open/saveAs requests. The folder hierarchy consists of 226 folders.

The state space consists of 4 parts: the current folder that the user is accessing and the three recommendations two of which are unordered. This would correspond to a state space of size $226 \times 225 \times \binom{224}{2}$. The action of the user is either to choose a recommended folder or select a different folder. The action of the assistant corresponds to choosing the top 3 folders and the action space is of size $225 \times \binom{224}{2}$. The cost in our case was the number of user clicks to the correct folder. In this domain, the assistant and the user's actions strictly alternate as the assistant revises its predictions after every user action. The prior distribution was initialized using the costs computed by the model developed in [5].

We applied the decision theoretic model to the data set. For each request, our assistant would make the prediction using the $H_{d,r}$ heuristic (which uses the default user policy and the rollout method) and then the user is simulated. The user would accept the recommendation if it shortens his path to the goal else would act according to his optimal policy. The user here is considered close to optimal, which is not unrealistic in the real world. To compare our results, we also used the model developed by Bao et al. in the data set and present the results in Table 2.4.

	One-time Prediction	With Repredictions
Restricted folder set	1.3724	1.34
All Folders	1.319	1.2344

Table 2.4: Results of the experiments in the folder predictor domain. The entry in the top left hand cell is the performance of the current Task Tracer, while the one in the bottom right hand cell is the performance of the decision-theoretic assistant

The table shows the average cost of folder navigation for 4 different cases: Bao et.al's original algorithm, their algorithm modified to include mixture distributions and our model with and without mixture distributions. It can be seen that our model with the use of mixture distributions has the least user cost for navigation and hence is the most effective. This improvement can be attributed to the two modifications mentioned earlier; first, the use of re-predictions in our model which is natural to the decision-theoretic framework while their model makes a one-time prediction and hence cannot make use of the user's response to the recommendations. Secondly, the fact that we consider all folders in the hierarchy for prediction and thus considering the possibility of the user accessing a new folder. It can be observed that either of the modifications yields a lower cost than the original algorithm, but combining the two changes is significantly more effective.

2.8 Discussion and Related Work

Our work is inspired by the growing interest and success in building useful software assistants [54]. Some of this effort is focused on building desktop assistants that help with tasks such as email filtering, on-line diagnostics, and travel planning. Each of these tasks typically requires designing a software system around specialized technologies and algorithms. For example, email filtering is typically posed as a supervised learning problem [16], travel planning combines information gathering with search and constraint propagation [2], and printer diagnostics is formulated as Bayesian network inference [65]. Unfortunately the plethora of systems and approaches lacks an overarching conceptual framework, which makes it difficult to build on each others' work. We argue that a decision-theoretic approach provides such a common framework and allows the design of systems that respond to novel situations in a flexible manner reducing the need for pre-programmed behaviors. We formulate a general version of the assistantship problem that involves inferring the user's goals and taking actions to minimize the expected costs.

Earlier work on learning apprentice systems focused on learning from the users by observation [48, 49]. This work is also closely related to learning from demonstration or programming by demonstration [4, 17, 46]. The emphasis in these systems is to provide an interface where the computer system can unobtrusively observe the human user doing a task and learn to do it by itself. The human acts both as a user and as a teacher. The performance of the system is measured by how quickly the system learns to imitate the user, i.e., in the supervised learning setting. Note that imitation and assistance are two different things in general. While we expect our secretaries to learn about us, they are not typically expected to replace us. In our setting, the assistant's goal is to reduce the expected cost of user's problem solving. If the user and the assistant are capable of exactly the same set of actions, and if the assistant's actions cost nothing compared to the user's, then it makes sense for the assistant to try to completely replace the human. Even in this case, the assistantship framework is different from learning from demonstration in that it still requires the assistant to *infer* the user's goal from his actions before trying to achieve it. Moreover, the assistant might learn to solve the goal by itself by reasoning about its action set rather than by being shown examples of how to do it by the user. In general, however, the action set of the user and the assistant may be different, and supervised learning is not appropriate. For example, this is the case in our Folder predictor. The system needs to decide which set of folders to present to the user, and the user needs to decide which of those to choose. It is awkward if not impossible to formulate this problem as supervised learning or programming by demonstration.

Taking the decision-theoretic view helps us approach the assistantship problem in a principled manner taking into account the uncertainty in the user's goals and the costs of taking different actions. The assistant chooses an action whose expected cost is the lowest. The framework naturally prevents the assistant from taking actions (other than noop) when there is no assistive action which is expected to reduce the overall cost for the user. Rather than learning from the user how to behave, in our framework the assistant learns the user's policy. This is again similar to a secretary who learns the habits of his boss, not so much to imitate her, but to help in the most effective way. In this work we assumed that the user MDP is small enough that it can be solved exactly given the user's goals. This assumption may not always be valid, and it makes sense in those cases to learn from the user how to behave. It is most natural to treat this as a case where the user's actions provide exploratory guidance to the system [15, 23]. This gives an opportunity for the system to imitate the user when it knows nothing better and improve upon the user's policy when it can.

There have been other personal assistant systems that are based on POMDP models. However, these systems are formulated as domain-specific POMDPs and solved offline. For instance, the COACH system helped people suffering from Dementia by giving them appropriate prompts as needed in their daily activities [10]. They use a plan graph to keep track of the user's progress and then estimate the user's responsiveness to determine the best prompting strategy. A distinct difference from our approach is that there is only a single fixed goal of washing hands, and the only hidden variable is the user responsiveness. Rather, in our formulation the goal is a random variable that is hidden to the assistant. We note, that a combination of these two frameworks would be useful, where the assistant infers both the agent goals and other relevant hidden properties of the user, such as responsiveness.

In *Electric Elves*, the assistant takes on many of the mundane responsibilities of the human agent including rescheduling meetings should it appear that the user is likely to miss it. Again a domain-specific POMDP is formulated and solved offline using a variety of techniques. In one such approach, since the system monitors users in short regular intervals, radical changes in the belief states are usually not possible and are pruned from the search space [68]. Neither exact nor approximate POMDP solvers are feasible in our online setting, where the POMDP is changing as we learn about the user, and must be repeatedly solved. They are either too costly to run [10], or too complex to implement as a baseline, e.g., Electric Elves [68]. Our experiments demonstrate simple methods such as one-step look-ahead followed by rollouts would work well in many domains where the POMDPs are solved online. In a distinct but related work [21], the authors introduce the setting of interactive POMDPs, where each agent models the other agent's beliefs. Clearly, this is more general and more complex than ordinary POMDPs. Our model is simpler and assumes that the agent is oblivious to the presence and beliefs of the assistant. While the simplified model suffices in many domains, relaxing this assumption without sacrificing tractability would be interesting.

Reinforcement Learning has been explored before in specific interactive settings such as in dialogue management in spoken dialogue systems [64, 69]. For example, the goal of the NJFun system in [64] is to learn a policy for optimally interacting with the user who is trying to query a database in spoken natural language. In particular the system decides when to take initiative in the dialogue, and whether to confirm its understanding of the speaker's utterance. The user's speech is processed through an automatic speech recognition (ASR) system which produces a noisy belief state which is compressed into a set of derived features. A dialogue policy is then learned over this set of derived features. The approach taken is to first learn a user's model through an exploratory phase, and then use offline value-iteration on the model to learn an optimal policy. This work can be viewed as a specific instance of our assistantship framework to dialogue management where the state/goal estimation is done by a separate ASR system and the action selection is done by offline reinforcement learning.

Our work is also related to on-line plan recognition and can be naturally extended to include hierarchies as in the hierarchical versions of HMMs [12] and PCFGs [61]. Blaylock and Allen describe a statistical approach to goal recognition that uses maximum likelihood estimates of goal schemas and parameters [9]. These approaches do not have the notion of cost or reward. By incorporating plan recognition in the decision-theoretic context, we obtain a natural notion of optimal assistance, namely maximizing the expected utility.

There has been substantial research in the area of user modeling. Horvitz et.al took a Bayesian approach to model whether a user needs assistance based on user actions and attributes and used it to provide assistance to user in a spreadsheet application [35]. Hui and Boutilier used a similar idea for assistance with text editing [36]. They use DBNs with handcoded parameters to infer the type of the user and compute the expected utility of assisting the user. It would be interesting to explore these kind of user models in our system to take into account the user's intentions and attitudes while computing the optimal policy for the assistant.

2.9 Summary and Future Work

We introduced a decision-theoretic framework for assistant systems and described the assistant POMDP as an appropriate model for selecting assistive actions. We also described an approximate solution approach based on iteratively estimating the agent's goal and selecting actions using myopic heuristics. Our evaluation using human subjects in two game-like domains show that the approach can significantly help the user. We also demonstrated in a real world folder predictor that the decision-theoretic framework was more effective than the state of the art techniques for folder prediction.

One future direction is to consider more complex domains where the assistant is able to do a series of activities in parallel with the agent. Another possible direction is to assume hierarchical goal structure for the user and do goal estimation in that context. Recently, the assistantship model was extended to hierarchical and relational settings [56] and presented in chapter 4 by including parameterized task hierarchies and conditional relational influences as prior knowledge of the assistant. This prior knowledge would relax the twin assumptions of rationality and tractability that correspond to the user being rational in his action choices and that the MDP can be solved tractably. This knowledge was compiled into an underlying Dynamic Bayesian network, and Bayesian network inference algorithms were used to infer a distribution of user's goals given a sequence of her atomic actions. The parameters for the user's policy were estimated by observing the users' actions.

Our framework can be naturally extended to the case where the environment is partially observable to the agent and/or to the assistant. This requires recognizing actions taken to gather information, e.g., opening the fridge to decide what to make based on what is available. Incorporating more sophisticated user modeling that includes users forgetting their goals, not paying attention to an important detail, and/or changing their intentions would be extremely important for building practical systems. The assistive technology can also be very useful if the assistant can quickly learn new tasks from expert users and transfer the knowledge to novice users during training.

Chapter 3 – Learning First-Order Probabilistic Models with Combining Rules

3.1 Overview

In this chapter, we develop an expressive language, that has a relational structure and a probabilistic interpretation. In the later chapter, this language is used to express the bias or prior knowledge of the decision-theoretic assistant. The language consists of statements that describe probabilistic influences between attributes of objects. Each quantified statement describes probabilistic influences that are independent of other influences. To keep these models succinct, the influences due to different rules and their ground instances on any attribute are combined using fixed "combining rules" such as Noisy-OR. This chapter presents our earlier work on developing algorithms for learning in the presence of such combining rules. Specifically, algorithms based on gradient descent and expectation maximization for different combining rules are derived, implemented, and evaluated on synthetic data and on a real-world task. The results demonstrate that the algorithms are able to learn the parameters of both the individual parent-target distributions and those of the combining rules. For the purposes of this chapter, we temporarily move away from the problem of assistance and focus on modeling in relational domains.

3.2 Introduction

New challenging application problems that involve rich relational data and probabilistic influences have led to the development of relational probabilistic models [29]. The advantage of these models is that they can succinctly represent probabilistic dependencies between the attributes of different related objects, leading to more efficient learning. Models are succinct because parameters are shared between different instantiations of the same "rule" applied to different objects. In many cases, inference is accomplished by "unrolling" the rules or grounding them with all possible parameter bindings yielding a propositional Bayesian network. The nodes of the Bayesian network represent the values of different attributes of objects or relationships between objects.

In many cases, a single parameterized rule can result in multiple instantiated sets of parents that influence a single ground target variable. More over, the number of these parent variables might change from one instance to the other. Consider, for example, the problem of modeling the spread of a disease such as west nile virus. One might posit that the spread depends on the mosquito population in a given location. The size of the population of mosquitos in turn depends on the temperature and the rainfall of each day since the last freeze. In one location, there might have been 19 days since the last freeze, whereas in another location, there might have been only 3 days (see Figure 3.1(a)).

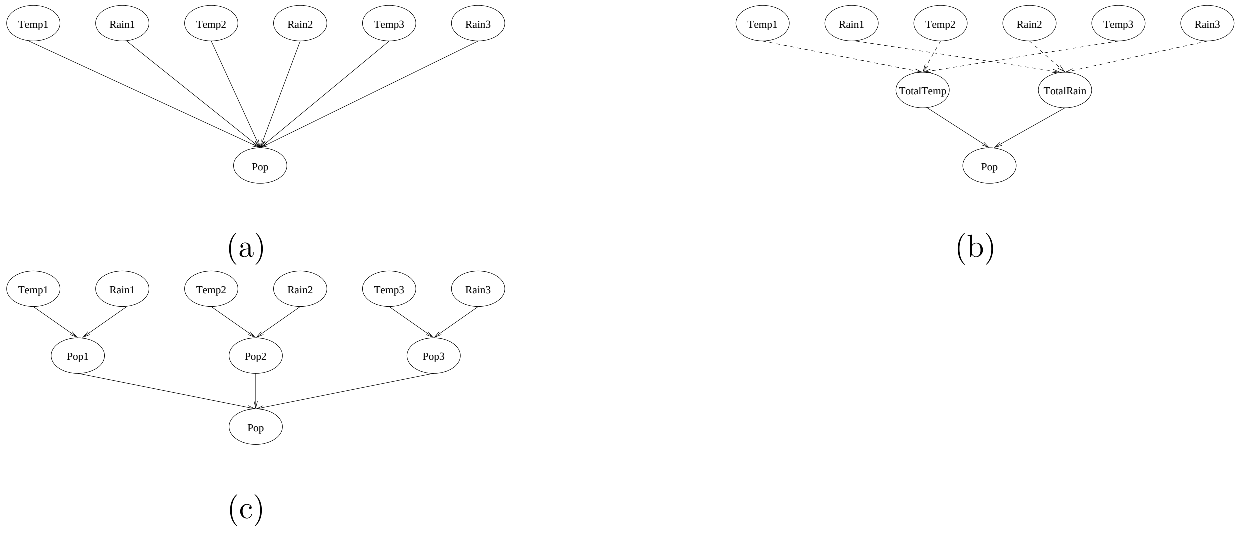


Figure 3.1: Three Bayesian networks describing the influence of daily temperature and rainfall on the population of mosquitos. (a) a network with no aggregation or combination rules leads to a very complex conditional probability distribution, (b) a network with separate aggregation for temperature and rainfall, (c) a network with separate prediction of the mosquito population each day followed by a combining rule to predict the overall population.

There are two main approaches to deal with this "multiple-parent" problem: aggregators and combining rules. An aggregator is a function that takes the *values* of the parent variables and combines them to produce a single aggregate value which then becomes the parent of the target variable. In the mosquito problem, we might define the total temperature and the total rainfall as aggregate variables. These are well-defined for any number of parents, and they can be computed deterministically (shown as dashed lines in Figure 3.1(b)). The population node then has only two parents: TotalTemp and TotalRain.

The second approach to the multiple-parent problem is to have a distribution $P(Pop \mid Temp, Rain)$ for population given a single temperature-rain pair and then combine the *distributions* from all such related pairs via a combining rule such as Noisy-OR, Noisy-AND, or Mean (see Figure 3.1(c)). The advantage of this method is that it can capture interactions between the *Temp* and *Rain* variables that are lost when temperature and rain are aggregated separately.

A special case of this second approach involves "decomposable combining rules," where the combining rule can be seen as summarizing the effects of several "micro-causes." In the above case, we can think of the different days producing different populations of mosquitos, whose combined effect on the total population is modeled by a combining rule. Thus, each parent or set of related parents produces a different child variable, whose values are combined using a deterministic or stochastic function. In the above example, the mosquito population of each day may be made a random function of a single temperature-rain pair and the populations from each day may be combined into a single value through a deterministic (e.g., sum, average) or a stochastic (e.g., random choice) aggregating function (Figure 3.1(c)). Thus the final distribution is a parameterized function of the distributions of the day-wise population variables.

There is, however, an additional complication. There can be multiple independent causal influences on the same target variable which are captured by different rules. For example, spraying of pesticides and the direction of the wind at that time might also influence the mosquito population at a given place and time. The net effect of this rule and that of the previous rule will have to be combined to get a final probability of the mosquito population. This can be achieved by having another combining rule which combines the different distributions of the target variable given the parent variables in each rule.

How can we learn in the presence of aggregators and combining rules? Most aggregators are deterministic and have no adjustable parameters, so they pose no additional problems for learning. However, some aggregators may have internal parameters. Suppose, for example, that we aggregated the temperatures as "degree days above 50 degrees": $DD(50) = \sum_i max(0, Temp_i - 50)$. The threshold of 50 might be learned from training data.

Learning with combining rules is more difficult, because the individual predicted target variables (e.g., Pop1, Pop2, ...) are unobserved, so the probabilistic model becomes a latent variable model. However, the latent variables are constrained to share the same conditional probability distribution, so the total number of parameters remains small. In previous work, Koller and Pfeffer developed an expectation maximization (EM) algorithm for learning in the presence of combining rules and missing data in relational context [44]. Kersting and DeRaedt implemented a gradient descent algorithm for the same [43].

In this work, we generalize and extend the above work to multi-level combining rules. The first level of combining rules combines the distributions of the target variable due to different instantiations of the same parameterized rule. The second level of combining rules operates on the results of the first level and combines the multiple distributions due to different rules.

We present algorithms for learning the parameters of the distributions and the combining rules based on both gradient descent and EM. The algorithms are tested on three tasks: a folder prediction task for an intelligent desktop assistant and two synthetic tasks designed to evaluate the ability of the algorithms to recover the true conditional probabilities.

The rest of the chapter is organized as follows. Section 2 introduces the necessary background on probabilistic relational languages and the need for combining rules. Section 3 presents the two gradient descent and EM algorithms that we have designed for learning the parameters in the presence of combining rules. Section 4 explains the experimental results on a real-world dataset and on the synthetic datasets. Section 5 concludes the chapter and points out a few directions for future research.

3.3 Probabilistic Relational Languages

In this section, we give a brief introduction to our first order conditional influence language (FOCIL), which will serve as a concrete syntax for specifying and learning combining rules. However, we note that our learning techniques are not tied to this particular language and are applicable to other probabilistic modeling languages that share the same underlying abstract syntax or model, e.g., Bayesian Logic Programs (BLPs). We give examples of such languages and translation of their syntax to ours in the next section.

In the spirit of Probabilistic Relational Models (PRMs) [27], we model domains in terms of objects of various types. Each type of object has associated attributes. The attributes have values of different primitive types, e.g., integers, enumerated type, etc. There are also predicates that describe properties of objects or relationships between objects of certain types. In this work, we assume that the domain of objects and relations among the objects are known and that we are interested in modeling the probabilistic influences between the attributes of the objects. The methods are extensible to probabilistic influences between relations in a straightforward way.

3.3.1 Conditional Influence Statements

In this section, we summarize the syntax of our language, FOCIL. The core of FOCIL consists of first-order conditional influence (FOCI) statements, which are used to specify probabilistic influences between the attributes of objects in a given domain. Each FOCI statement has the form:

If $\langle$*condition*$\rangle$ *then* $\langle$*qualitative influence*$\rangle$

where *condition* is a set of literals, each literal being a predicate symbol applied to the appropriate number of variables. A $\langle$*qualitative influence*$\rangle$ is of the form $X_1, \ldots, X_k$ *Qinf* Y, where the X_i and Y are of the form $V.a$, where V is a variable in *condition* and a is an object attribute. This statement simply expresses a directional dependence of the *resultant* Y on the *influents* X_i. Associated with each FOCI statement is a *conditional probability function* that specifies a probability distribution of the resultant conditioned on the influents, e.g. $P(Y|X_1, \ldots, X_k)$ for the above statement. We will use P_i to denote the probability function of the i'th FOCI statement. As an example, consider the statement,

`If {Person(p)} then p.diettype Qinf p.fitness`,

which indicates that a person's type of diet influences their fitness level. The conditional probability distribution $P(p.\text{fitness} \mid p.\text{diettype})$ associated with this statement (partially) captures the quantitative relationships between the attributes. As another example, consider the statement

`If {Takes(Offering,Student,Course)} then Student.iq, Course.diff Qinf Offering.grade`,

which indicates that a student's IQ and a course's difficulty influence the grade of the student in the course. Note that since `Takes` is a many-to-many relation, we have introduced an argument `Offering` to represent the instance of the student taking a course. It can be interpreted as representing a student-course pair.

Given a fixed domain of objects and a database of facts about those objects, FOCI statements define Bayesian network fragments over the object attributes. In particular, for the above statement, the unrolled Bayesian network includes a variable for the grade of each student-course object, the IQ of each student, and the difficulty of each course. The parents of each grade variable are the IQ and difficulty attributes corresponding to the appropriate student and course. Each grade variable has an identical conditional probability table $P(Offering.grade|Student.iq, Course.diff)$—that is, the table associated with the above rule.

In addition, our language supports qualitative constraints such as monotonicity (e.g., a person's consumption monotonically increases with income) and synergies (e.g., the lifestyle of a family synergistically depends on the family members' incomes). Although in this work we do not learn with these constraints, we have well-defined semantics of the constraints in FOCIL and learning algorithms for propositional models with monotonicity constraints [1].

3.3.2 Combining Rules

The following example illustrates the multiple-parent problem described in the introduction. Consider an intelligent desktop assistant that must predict the folder of a document to be saved. Assume that there are several tasks that a user can work on, such as proposals, courses, budgets, etc. The following FOCI statement says that a task and the role the document plays in that task influence its folder.

```
If {role(Doc,Role,Task)} then Task.id,Role.id Qinf Doc.folder.
```

Typically a document plays several roles in several tasks. For example, it may be the main document of one task but only a reference in some other task. Thus there are multiple task-role pairs $(Task_1, Role_1), \ldots, (Task_m, Role_m)$, each yielding a distinct folder distribution $P(Doc.folder \mid Task_i.id, Role_i.id)$. We need to combine these distributions into a single distribution for the folder variable. We could apply some kind of aggregator (e.g., the most frequently-occurring task-role pair) as in PRMs [27]. However, it is easy to imagine cases in which a document is accessed with low frequency across many different tasks, but these individual accesses, when summed together, predict that the document is stored in a convenient top-level folder rather than in the folder of the most frequent single task-role pair. This kind of summing of evidence can be implemented by a combining rule.

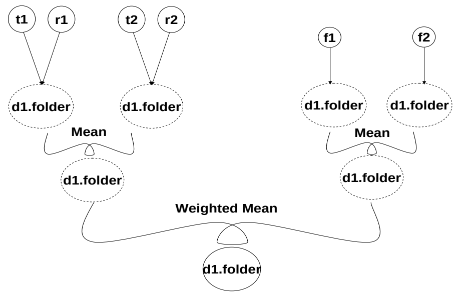


Figure 3.2: Use of Combining rules to combine the influences of task and role on the one hand and the source folder on the other on the folder of the current document.

In the above example, a combining rule is applied to combine the distributions due to different influent instances of a single FOCI statement. In addition, combining rules can be employed to combine distributions arising from multiple FOCI statements with the same resultant. The following example captures such a case (see Figure 3.2 for the unrolled network):

```
WeightedMean{
  If {role(Doc,Role,Task)} then  Task.id, Role.id Qinf (Mean) Doc.folder.
  If {source(Src,Doc)} then Src.folder Qinf (Mean) Doc.folder.
  }
```

Figure 3.3: Example of specifying combining rules in FOCIL.

The expression in Figure 3.3 includes two FOCI statements. One statement is the task-role influence statement discussed above. The other says that the folder of the source document of *Doc* influences *Doc*'s folder. The source of a document is a document that was edited to create the current document. There can be multiple sources for a document. The distributions corresponding to different instances of the influents in the same statement are combined via the "Mean" combining rule. The two resulting distributions are then combined with the "Weighted-Mean" combining rule. The precise meanings of these rules are described in the section 3.3.4.

Formally, the language consists of a set of *influence statements*. Each influent statement has a *resultant*, a set of *influents*, a *logical condition* and a *combining rule* to combine the influences due to the different influents. The set of influent statements that have the same resultant are grouped together and another combining rule is used to combine the distributions arising due to these statements.

The influents and the resultant are attributes of objects in the domain. These can be understood similar to the variables in a PRM. The objects refer to the relations of the PRM and the attributes in our model are similar to the attributes of the relations in the PRM. The logical conditions consist of predicates and relations over the objects in the domain. The predicates serve to bind the values of the objects while the relations are used to tie the values of the different objects in the influence statements. For instance, in the example presented in Figure 3.3,the relation *role* serves to obtain the bindings of the relationship variable *Role*.

3.3.3 Relationship to Other Languages

We describe our learning algorithms using the FOCIL notation. However, our learning algorithms and results are not specific to its concrete syntax. They can be applied to any language that shares its abstract syntax and uses combining rules to combine the results of multiple instantiated rules.

In this section, we represent several first order probabilistic models in the FOCIL syntax, hence establishing the equivalence between these languages and showing that our learning algorithms extend to several first-order models.

Kersting and De Raedt introduced Bayesian Logic Programs [42]. BLPs combine Bayesian Networks with definite clause logic. BLPs view the atoms as random variables. Bayesian Logic Programs consist of two components: a qualitative component that is the logical component which captures the structure of the domain (similar to that of the Bayesian Network structure) and a quantitative component. An example of a BLP clause is as follows:

```
bt(X) | father(F,X) , bt(F), mother(M,X) , bt (M)
```

There is a CPT corresponding to this clause. In this case, the predicates $mother(M, X)$ and $father(F, X)$ would have boolean values. One could then specify the ground facts like $father(John, Tom)$ etc. The function $bt(F)$ represents the blood type of F. The above statement says that a person's blood type is a function of his father's and mother's blood types. The FOCI statement corresponding to the above BLP clause is:

```
If { mother(M,P) , father(F,P)} then M.bt, F.bt Qinf P.bt
```

BLPs also use combining rules for combining the distributions due to multiple instantiations of the parent predicates. The main difference between BLPs and FOCI statements is that in the latter, the conditions are separated out from the clause. BLPs do not make a clear syntactic distinction between the conditions and the influents in the statement itself, although they are distinguished elsewhere in the model.

Another representation that is closely related to both FOCIL and BLPs is Logical Bayesian Networks. They consist of conditional dependency clauses of the form $X|Y_1, ..., Y_k \longleftarrow Z_1, ..., Z_m$. This can be interpreted as the $Y_1, ..., Y_k$ influence X when

$\langle Z_1, ...Z_k \rangle$ are true, where $Y_1, ..., Y_k$ and X are random variables and $\langle Z_1, ..., Z_m \rangle$ are logical literals. The above example of the bloodtype can be represented in LBNs as,

`bt(X) | bt(M), bt(F)` $\longleftarrow$ `Mother(M,X),Father(F,X)`

More recently Getoor and Grant proposed the formalism of Probabilistic Relational Language (PRL) [28]. The main motivation behind this work is to represent the original work on probabilistic relational models (PRMs) [27] in logical notation. While PRMs used aggregators to combine the influences of multiple parents, both aggregators and combining rules can be used in the PRL framework. The entities and the relationships that are represented as predicates form the logical structure of the domain. The probabilistic structure is composed of non-key attributes that form the random variables in the domain. The general structure of the influence statement is: $\mathit{DependsOn}(X(\alpha), Y_1(\alpha), ...Y_n(\alpha)) \longleftarrow Z(\alpha)$ and can be interpreted as $\langle Y_1(\alpha)....Y_n(\alpha)$ influence $X(\alpha)$ when $Z(\alpha)$ is true. Consider, our bloodtype example. In PRL, we can represent it as follows:

`DependsOn(bt(X),bt(M), bt(F))` $\longleftarrow$ `Mother(M,X), Father(F,X)`

The main difference between the PRLs and LBNs lies in the fact that the PRLs allow for aggregate functions explicitly. Also, in [28] the authors show how to represent several kinds of uncertainties like Structure uncertainty, reference uncertainty, existence uncertainty etc in PRL.

Heckerman et.al introduced directed acyclic probabilistic entity-relationship (DAPER) models [34]. DAPER models are to entity-relationship models what PRMs are to relational schema. The arcs in the DAPER models are between the attributes of the entities and relationships. The main difference with the PRMs is that the DAPER models attach arbitrary first-order conditions to the Bayes net arcs. These conditions serve to restrict the set of possible instantiations of the variables. For instance, to represent the bloodtype example, there would be arcs to the bloodtype of a person from the person's father and mother's bloodtypes with the condition on the arcs being $\exists M, Mother(M, X)$ and $\exists F, Father(F, X)$. Essentially, these conditions along with the arc represent the fact that there exists persons M and F who are the mother and father of the person X and whose bloodtype influences X's bloodtype.

One problem with the DAPER models, however, is that they do not allow sharing of variables between the incoming arcs at the same node, and thus prohibit interaction between the influents. For instance, it is impossible to express the following rule in DAPER because the condition $role(Doc, Role, Task)$ involves both the Bayes net parents $Task$ and Doc.

```
If {role(Doc,Role,Task)} then Task.id,Role.id Qinf Doc.folder.
```

This problem can be solved by attaching the conditions jointly to all parents of a node. DAPER models allows us to specify how the distributions are combined if there are more than one possible instantiation of the free variables that satisfy the conditions.

Although, the different models differ from each other in syntactic details, they all share the same underlying semantics, and express equivalent pieces of knowledge. All of them also suffer from the multiple-parent problem, which can be addressed through combining rules. Thus, the algorithms discussed in this chapter are relevant and applicable to all these formalisms and a few others such as Relational Bayesian Networks (RBNs) [38].

However, there are some statistical relational models for which our algorithms do not apply. For example, PRISM [62] uses distributional semantics which is quite different from the possible worlds semantics used by these other models. Both PRISM and Stochastic Logic Programs (SLPs), place distributions on possible proofs rather than on possible worlds [51]. Markov Logic Networks [20] and related Conditional Random Fields [45] are based on undirected graphical models and significantly differ from models based on directed graphs. Markov Logic Networks, for example, are more flexible in allowing knowledge to be expressed as weighted first-order formulas, and have a correspondingly harder inference problem.

3.3.4 Unrolling FOCI Statements

In this section, we formally define the semantics of the FOCI statements by showing the procedure to unroll them into a Bayesian network. Consider a generic influence statement S_i:

if $\langle \textit{condition} \rangle$ then $X_1^i, \ldots, X_k^i$ *Qinf* Y.

We assume without loss of generality that each influence statement S_i ('rule i' for short) has k influents, X_1^i through X_k^i (which we jointly denote as $\mathbf{X}^i$), that influence the target variable. When this rule is instantiated or "unrolled" on a specific database, it generates multiple, say m_i, sets of influent instances, which we denote as $\mathbf{X}_1^i \ldots \mathbf{X}_{m_i}^i$. This is shown in Figure 3.4. In the figure, the instantiations of a particular statement are combined with the *mean* combining rule. The distributions resulting from the different FOCI statements are combined via the Weighted-Mean combining rule.

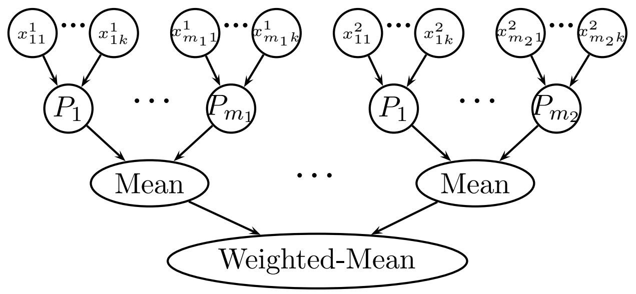


Figure 3.4: Unrolling of FOCI statements

The role of the combining rule is to express the probability $P_i(Y|\mathbf{X}_1^i \ldots \mathbf{X}_{m_i}^i)$ as a function of the probabilities $P_i(Y|\mathbf{X}_j^i)$, one for each j, where P_i is the CPT associated with S_i. Since these instance tuples are unordered and can be arbitrary in number, our combining rule should be symmetric, i.e., its value should not depend on the order of the arguments. For example, with the mean combining rule, we obtain:

$$P(y|\mathbf{X}_1^i \dots \mathbf{X}_{m_i}^i) = \frac{1}{m_i} \sum_{j=1}^{m_i} P_i(y|\mathbf{X}_j^i) \tag{3.1}$$

If there are r such rules, we need to estimate the conditional probability $P(Y|X_{1,1}^1 ... X_{m_r,k}^r)$. Since each rule is distinctly labeled and its instances can be identified, the combining rule need not be symmetric, e.g., Weighted-Mean. If w_i represents the weight of the i^{th} combining rule, the "Weighted-Mean" is defined as:

$$P(Y|X_{1,1}^1 ... X_{m_r,k}^r) = \frac{\sum_{i=1}^r w_i P(Y|\mathbf{X}_1^i \dots \mathbf{X}_{m_i}^i)}{\sum_{i=1}^r w_i} \tag{3.2}$$

We write $x_{j,1}^i, \dots, x_{j,k}^i \equiv \mathbf{x}_j^i$ to denote the values of $\mathbf{X}_j^i$ and y to denote the value of Y. We write $\theta_{y|\mathbf{x}^i}$ to denote $P_i(y|\mathbf{x}^i)$. Note that in this case we omit the subscript j because the parameters θ are shared across the different instantiations of the same rule.

Note that the graph presented in Figure 3.4 is not a Bayesian Network, but is a tree representation of the expression for the conditional probability of the target variable given the inputs. The nodes labeled "Mean" compute the mean of the parent distributions represented by $P_1^1, \dots, P_1^{m_1}$, etc., and the node labeled "Weighted-Mean" computes the weighted mean of its parent distributions.

Consider the equivalent Bayesian Network presented in Figure 3.5. This network is very similar to Figure 3.4. However, the nodes here represent random variables whose values are from the domain of the target variable. The values of the random variables $Y_1^1 .. Y_1^{m_1}$, etc. are sampled from the distributions $P_1^1, \dots, P_1^{m_1}$, etc. The value of the variables Y_1 and Y_2 may be inherited from one of the parents which is chosen uniformly randomly. This ensures that the distribution of Y_i given the root variables is the mean of the distributions of $Y_i^1 .. Y_i^{m_i}$, as specified in Figure 3.4. Similarly the value of the variable

Y may be inherited from Y_1 or Y_2 with probabilities w_1 and w_2 respectively to ensure that the final conditional distribution of Y is the weighted mean of the distributions of Y_1 and Y_2. Similarly, the Noisy-OR combining rule can be implemented by setting the value of Y to be a deterministic function of Y_1 and Y_2, in particular to Y_1 OR Y_2. Here the "noise" enters through the distributions of Y_1 and Y_2 given their parents.

The key idea behind these "decomposable" combining rules is that they can be implemented using deterministic and stochastic functions on the corresponding values of random variables. Given such a value-based implementation of the combining rules, it is possible to use the standard Bayesian learning and inference methods on them to learn their parameters. Our implementation of the EM algorithm, described in Section 3.4.6, can be understood as doing exactly that in the case of the Weighted-Mean combining rule.

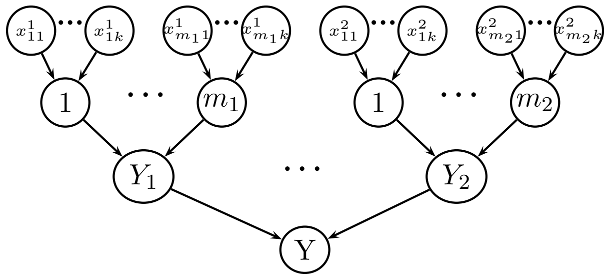


Figure 3.5: Value based Bayesian Network for the FOCI statements. The mean combining rule is replaced by a node that chooses a value using an uniform distribution. The Weighted-Mean rule is replaced by a node that chooses one of its parent values randomly using a biased distribution.

3.4 Learning Model Parameters

In this section, we present algorithms for learning the parameters of the combining rules and the conditional probability tables (CPTs). As mentioned earlier, we are using 2-levels of combining rules, the first to combine the multiple instances of the same rule and the second to combine the distributions due to the different rules. In this work, we use the Mean combining rule at the first level (i.e., to combine the influence due to multiple instances of the same rule) while using Weighted-Mean and Noisy-OR at the second level (i.e., to combine the distributions due to the different rules). We present the algorithms for learning in presence of both the Weighted-Mean and Noisy-OR combining rules.

3.4.1 Gradient Descent Learning

Two of our learning algorithms are based on gradient descent. The idea of gradient descent training is to gradually change the parameters of the probability distributions in the direction of improved performance. We consider two measures of performance. The mean squared error measures the square of the difference between the true probability of a label of the example and the probability of that label predicted by the current model. Naturally, we would like to minimize the squared error. In trying to predict probabilities, it is more common to measure the performance by the loglikelihood, which is the logarithm of the likelihood of the examples under the current model. We would like to maximize the loglikelihood to improve the performance of the learner.

The generic gradient descent pseudocode for learning the parameters of the FOCI statements in the presence of combining rules is given in Figure 3.1. When using the Noisy-OR combining rule, the weight gradient and the update steps are ignored. When the weighted mean combining rule is used, we use different step-size parameters (α and β) for the parameters and the weights respectively. It is important to have much smaller learning rates for the combining rule weights compared to those of the CPT parameters, i.e., $\beta << \alpha$. This is because each iteration of each example only changes a few of the large number of CPT parameters, whereas it changes many of the small number of weights. Hence, the rule weights are updated much more frequently than the conditional probabilities and hence should be updated by smaller amount in each iteration.

Table 3.1: Gradient Descent Algorithm for parameter learning

1. Initialize the parameters θ and weights w_i randomly
2. ParameterGradient Step: for each value of y and for all parent configurations of each rule $\mathbf{x}^i$ and for each instantiation of each rule, compute the gradient $\frac{-\partial E}{\partial \theta_{y|\mathbf{x}^i}}$ where E is the error function that is either the Noisy-OR or the Mean squared Error function
3. WeightGradient Step: Compute the gradient of the weights $\frac{-\partial E}{\partial w_i}$ for each of the rule
4. ParameterUpdate Step: Update each parameter $\theta_{y|\mathbf{x}^i}$ by $\theta_{y|\mathbf{x}^i} = \theta_{y|\mathbf{x}^i} - \alpha \frac{\partial E}{\partial \theta_{y|\mathbf{x}^i}}$ for each value of y
5. GradientUpdate Step: Update each weight w_i by $w_i = w_i - \beta \frac{\partial E}{\partial w_i}$. Normalize the weights so that the sum is preserved
6. Continue steps 2 through 5 until convergence.

Table 3.2: Gradient descent applied to learning parameters of FOCI statements

3.4.2 Gradient Derivation for the Mean-squared Error for Weighted-Mean

In this section, we derive the gradient equation for the mean-squared error function for the prediction of the target variable, when multiple FOCI-statements are present and are combined by the Weighted-Mean combining rule.

Let the l^{th} training example e_l be denoted by $(\langle x^1_{l,1,1}, \ldots, x^{r_l}_{l,m_{l,r_l},k}\rangle, y_l)$, where $x^i_{l,j,p}$ is the p^{th} input value of the j^{th} instance of the i^{th} rule on the l^{th} example. The predicted probability of class y on e_l is given by

$$P(y|e_l) = \frac{1}{\sum_i w_i} \sum_i^{r_l} \frac{w_i}{m_{l,i}} \sum_j^{m_{l,i}} P_i(y|\mathbf{x}^i_{l,j}). \tag{3.3}$$

In the above equation, r_l is the number of rules the example satisfies, i is an index of the applicable rule, and $m_{l,i}$ is the number of instances of rule i on the l^{th} example. The squared error is given by

$$E = \frac{1}{2}\sum_{l=1}^{n}\sum_{y}(I(y_l, y) - P(y|e_l))^2. \tag{3.4}$$

Here y is a class label, and y_l is the true label of l^{th} example. $I(y_l, y)$ is an indicator variable that is 1 if $y_l = y$ and 0 otherwise. Taking the derivative of negative squared error with respect to $P(y|\mathbf{x}^i) = \theta_{y|\mathbf{x}^i}$, we get

$$\begin{aligned}
\frac{-\partial E}{\partial \theta_{y|\mathbf{x}^i}} &= \sum_{l=1}^{n}\sum_{y}\left[(I(y_l, y) - P(y|e_l))\left(\frac{-\partial}{\partial \theta_{y|\mathbf{x}^i}}P(y|e_l)\right)\right] \\
&= \sum_{l=1}^{n}\sum_{y}\left[(I(y_l, y) - P(y|e_l))\left[\frac{1}{\sum_{i'} w_{i'}}\frac{w_i}{m_{l,i}}\#(\mathbf{x}^i|e_l)\right]\right].
\end{aligned} \tag{3.5}$$

The value of $P(y|e_l)$ is given in equation 3.3. Here $\#(\mathbf{x}^i|e_l)$ represents the number of occurrences of the tuple $\mathbf{x}^i$ in the x-instances of the i^{th} rule of example e_l. Gradient descent increments each parameter $\theta_{y|\mathbf{x}^i}$ by - $\alpha\frac{\partial E}{\partial \theta_{y|\mathbf{x}^i}}$ in each iteration. After each iteration, the parameters are normalized so that the distributions are well-defined.

The gradients with respect to rule weights are computed as follows:

$$\begin{aligned}
\frac{-\partial E}{\partial w_i} &= \sum_{l=1}^{n}\sum_{y}\left[(I(y_l, y) - P(y|e_l))\left(\frac{-\partial}{\partial w_i}P(y|e_l)\right)\right] \\
&= \sum_{l=1}^{n}\sum_{y}\left[(I(y_l, y) - P(y|e_l))\frac{-\partial}{\partial w_i}\left(\frac{1}{\sum_i w_i}\sum_{i}^{r_l}\frac{w_i}{m_{l,i}}\sum_{j}^{m_{l,i}}P_i(y|\mathbf{x}^i_{l,j})\right)\right] \\
&= \sum_{l=1}^{n}\sum_{y}\left[(I(y_l, y) - P(y|e_l))\left(\left(\frac{-\partial}{\partial w_i}\frac{1}{\sum_i w_i}\right)\sum_{i}^{r_l}\frac{w_i}{m_{l,i}}\sum_{j}^{m_{l,i}}P_i(y|\mathbf{x}^i_{l,j})\right.\right. \\
&+ \left.\left.\frac{1}{\sum_i w_i}\left(\frac{-\partial}{\partial w_i}\sum_{i}^{r_l}\frac{w_i}{m_{l,i}}\sum_{j}^{m_{l,i}}P_i(y|\mathbf{x}^i_{l,j})\right)\right)\right] \\
&= \sum_{l=1}^{n}\left[\delta(e_l, i) - \frac{1}{r_l}\sum_{r}\delta(e_l, r)\right],
\end{aligned} \tag{3.6}$$

where $\delta(e_l, r)$ is given by

$$\sum_y (I(y_l, y) - P(y|e_l)) \frac{1}{\sum_{i'} w_{i'}} \frac{1}{m_{l,r}} \sum_j P_r(y|\mathbf{x}^r_{l,j}) \tag{3.7}$$

We include the $\frac{1}{r_l}$ term in the equation to properly normalize the weights. When we adjust the weights of the rules using examples, we preserve the sum of the weights of the matching rules in each example by normalizing the weights, so that the overall sum of all weights is preserved, and the dependencies between the weights are properly taken into account when we take the derivatives.

3.4.3 Gradient Derivation for the Mean Squared Error for Noisy-OR

We now derive the gradient equations for the mean-squared error function for the prediction of the target variable, when multiple FOCI-statements are combined by the Noisy-OR combining rule. Let the l^{th} training example e_l be denoted by $(\langle x^1_{l,1,1}, \ldots, x^{r_l}_{l,m_{l,r_l},k}\rangle, y_l)$, where $x^i_{l,j,p}$ is the p^{th} input value of the j^{th} instance of the i^{th} rule on the l^{th} example. The predicted probability of class y on e_l is given by

$$P(y=1|e_l) = 1 - \prod_i^{r_l} \frac{1}{m_{l,i}} \sum_j^{m_{l,i}} P_i(y=0|\mathbf{x}^i_{l,j}). \tag{3.8}$$

Here r_l is the number of rules the example satisfies, i is an index of the applicable rule, and $m_{l,i}$ is the number of instances of rule i on the l^{th} example. The squared error is given by

$$E = \frac{1}{2} \sum_{l=1}^n \sum_y (I(y_l, y) - P(y|e_l))^2$$

$$= \frac{1}{2} \sum_{l=1}^n \left[(I(y_l, y=0) - P(y=0|e_l))^2 + (I(y_l, y=1) - P(y=1|e_l))^2 \right] \tag{3.9}$$

$$= \frac{1}{2} \sum_{l=1}^{n} \left[(I(y_l, y = 0) - P(y = 0|e_l))^2 + (I(y_l, y = 1) - (1 - P(y = 0|e_l)))^2 \right] \tag{3.10}$$

Here y is a class label, and y_l is the true label of l^{th} example. $I(y_l, y)$ is an indicator variable that is 1 if $y_l = y$ and 0 otherwise. Taking the derivative of negative squared error with respect to $P(y|\mathbf{x}^i) = \theta_{y|\mathbf{x}^i}$, we get

$$\frac{-\partial E}{\partial \theta_{y|\mathbf{x}^i}} = \sum_{l=1}^{n} \left[(1 + (I(y_l, y = 1) - I(y_l, y = 1)) - 2P(y = 0|e_l))\delta(e_l) \right] \tag{3.11}$$

where $\delta(e_l)$ is given by

$$\delta(e_l) = \left[\frac{\#(\mathbf{x}^i|e_l)}{m_{l,i}} \Pi_{i' \neq i} P(y = 0|x_{l,j}^{i'}) \right]$$

3.4.4 Gradient Derivation for Loglikelihood for Weighted-Mean

As commented earlier, in the context of probabilistic modeling, it is more common to maximize the loglikelihood of the data given the hypothesis [8]. We now turn to gradient derivation for the loglikelihood in the presence of the Weighted-Mean combining rule.

From the definition of $P(y_l|e_l)$, we can see that this is

$$L = \sum_{l} \log P(y_l|e_l). \tag{3.12}$$

Taking the derivative of L with respect to $P(y|\mathbf{x}^i) = \theta_{y|\mathbf{x}^i}$, gives

$$\frac{\partial L}{\partial \theta_{y|\mathbf{x}^i}} = \sum_{l} \frac{1}{P(y_l|e_l)} \frac{1}{\sum_{i'} w_{i'}} \sum_{i}^{r_l} \frac{w_i}{m_{l,i}} \#(\mathbf{x}^i|e_l). \tag{3.13}$$

As before, the partial derivative of L with respect to the weights is given by

$$\frac{\partial L}{\partial w_i} = \sum_{l=1}^{n} \left[\delta(e_l, i) - \frac{1}{r_l} \sum_{r} \delta(e_l, r) \right]. \tag{3.14}$$

But now,

$$\delta(e_l, r) = \frac{1}{P_r(y_l|e_l)} \frac{1}{\sum_i w_i} \frac{1}{m_{l,r}} \sum_{j} P_r(y_l|\mathbf{x}^r_{l,j}). \tag{3.15}$$

3.4.5 Gradient Derivation for LogLikelihood for Noisy-OR

We now give the derivation of the gradient for loglikelihood with Noisy-OR as the combining rule. The loglikelihood L is given by,

$$L = \sum_{l} \log P(y_l|e_l). \tag{3.16}$$

where $P(y_l = 1|e_l)$ is given by,

$$P(y = 1|e_l) = 1 - \prod_{i}^{r_l} \frac{1}{m_{l,i}} \sum_{j}^{m_{l,i}} P_i(y = 0|\mathbf{x}^i_{l,j}). \tag{3.17}$$

Taking the derivative of the likelihood wrt $P(y|\mathbf{x}^i) = \theta_{y|\mathbf{x}^i}$ we get,

$$\begin{aligned} \frac{\partial L}{\partial \theta_{y|\mathbf{x}^i}} &= \frac{1}{P(y_l|e_l)} \frac{\partial P(y_l|e_l)}{\partial \theta_{y|\mathbf{x}^i}} \\ &= \sum_{l} \left[\frac{1}{P(y_l|e_l)} \left[\frac{\#(\mathbf{x}^i|e_l)}{m_{l,i}} \Pi_{i' \neq i} P(y = F|x^{i'}_{l,j}) \right] \right] \end{aligned} \tag{3.18}$$

3.4.6 Expectation-Maximization

Expectation-Maximization (EM) is a popular method to compute maximum likelihood estimates given incomplete data [18]. EM iteratively performs two steps: the *Expectation* step, where the algorithm computes the expected values of the missing data based on the current parameters, and the *Maximization* step, where the maximum likelihood of the parameters is computed based on the current expected values of the data. Expectation-Maximization avoids the slow update step of the gradient-based methods by directly maximizing the likelihood in each iteration. In our experiments, the Gradient descent algorithms take about twice as much time as the EM algorithm for convergence. We

Table 3.3: EM Algorithm for parameter learning in FOCIL

1. Take initial guesses for parameters θ and weights w_i
2. E Step: $\forall i$ and for each instantiation of each rule, compute the responsibilities
   $$\gamma^i_{l,j}(y) = \frac{(w_i)\frac{1}{m_{l,i}}\theta_{y|\mathbf{x}^i_{l,j}}}{\sum_{l,i',j}(w_{i'})\frac{1}{m_{l,i'}}\theta_{y|\mathbf{x}^{i'}_{l,j}}}$$
3. M Step: Compute the new parameters:
   $$\forall i, \mathbf{x}^i \quad \theta_{y|\mathbf{x}^i} = \frac{\sum_{l,j}\gamma^i_{l,j}(y)}{\sum_{y',l,j}\gamma^i_{l,j}(y')}$$
   and, if instantiations of at least two rules are present in l, compute $\quad \forall i \quad w_i = \left(\sum_{l,j}\gamma^i_{l,j}\right)/n_2$
   where n_2 is the number of examples with two or more rules instantiated.
4. Continue E and M steps until convergence.

first explain the EM algorithm for the case where we use the Weighted-Mean rule to combine the distributions due to different rules. We use the terminology of [32]. Consider n rules with the same resultant. Accordingly, there will be n distributions that need to be combined via the Weighted-Mean. Let w_i be the weight for rule i, such that $\sum_i w_i = 1$.

The EM algorithm for parameter learning in FOCIL is presented in Table 3.3. In the *expectation* step, it computes the responsibility of each instantiation of each rule. The responsibilities reflect the relative contribution of each rule instance to the conditional probability of the target variable. The numerator of the equation in Step 2 is the contribution of a particular instance of a particular rule to the conditional probability of the target. The denominator is the contribution of all instances of all applicable rules. Note that we consider the weight of the current rule and the number of instantiations of the current rule while computing the responsibilities.

In the *maximization* step, we use these responsibilities to update the CPTs treating them as fractional counts of examples that satisfy the influents. The numerator of Step 2 is the total responsibility of all instances of the i^{th} rule for a given value of the target variable. The denominator is the normalizing factor, which sums over all values of the target variable. The fraction approximates the parameter of the FOCI statement that represents the conditional probability of the target having the value y, given particular influent values in rule i.

Finally, the new weights of the combining rules are computed by the fraction of the responsibility assigned to each rule out of the sum of the responsibilities assigned to all rules over all a subset of examples in which at least 2 rules are instantiated. We only consider such examples because if an example matches only one rule, the distribution predicted by the single rule is going to be the final distribution. Hence, the weight does not impact the distribution and hence the example should not influence the weight.

The responsibilities can be thought of as conditional probabilities of the hidden node values given the target variable and the inputs in the corresponding value-based Bayesian network. Note that in our problem formulation, there are two levels of combining rules. As we discussed earlier, we can think of the Mean combining rule as resulting from randomly inheriting a value of the parent node as its own value. The responsibility of a particular influent instance can be thought of as the probability of choosing that instance to transmit its value to the final target node. Similarly, the weight of each rule can be thought of as representing the probability of choosing its target value as the final value.

The EM algorithm for the Noisy-OR is very similar to the one presented in Table 3.3. Since this combining rule is asymmetric with respect to the two binary values, in the E-step, there is a need for using seperate equations for the cases when the target variable takes a value 0 or 1. The responsibilities when $y = 1$ are computed using,

$$\gamma_{l,j}^{i}(y) = \frac{\frac{1}{m_{l,i}}\theta_{y|\mathbf{x}_{l,j}^{i}}}{1 - \sum_{l,i',j} \frac{1}{m_{l,i'}}\theta_{y=0|\mathbf{x}_{l,j}^{i'}}}$$

and when $y = 0$,

$$\gamma_{l,j}^{i}(y) = \frac{\frac{1}{m_{l,i}}\theta_{y|\mathbf{x}_{l,j}^{i}}}{\sum_{l,i',j} \frac{1}{m_{l,i'}}\theta_{y=0|\mathbf{x}_{l,j}^{i'}}}$$

Notice that $\gamma_{l,j}^{i}(y)$ gives the the contribution of each rule instance as a fraction of the total contribution to the value of the target variable. As before, once the responsibilities are computed, we can use them as fractional counts to compute the parameters of the influence statements.

3.5 Experiments and Results

In this section, we describe results on the data sets that we employed to test the learning algorithms. The first is based on the folder prediction task, where we applied two rules to predict the folder of a document. The second data set is a synthetic one that permits us to test how well the learned distribution matches the true distribution. We present the results for the experiments and compare them with the propositional classifiers. We do not evaluate the Noisy-OR algorithms in the folder data set as the target variable is not binary. Instead, we evaluate the Noisy-OR algorithms on a synthetic data set and present the results for the same. In this section, we first present the results on the synthetic data set for Noisy-OR and then follow it up with our experiments with the Weighted-Mean combining rule.

3.5.1 Synthetic Data set for Noisy-OR

To estimate the accuracy of the learned model using Noisy-OR, we constructed a synthetic data set. The data are generated using a synthetic target as defined by two FOCI statements, each of which has two influents and the same target attribute. The different instances of the same rule are combined using mean and the different rules are combined using the Noisy-OR combining rule. The two influents in each rule have a range of 10 and 2 values respectively. The target attribute can take 2 values. The probability values in the distribution of the synthetic target are randomly generated to be either between 0.9 and 1.0 or between 0.0 and 0.1. This is to make sure that the probabilistic predictions on examples are hard to predict and not too close to the default probability of $\frac{1}{2}$ each. Each example matches a rule with probability 0.5, and when it does match, it generates a number of instances randomly chosen between 3 and 10. This makes it imperative that the learning algorithm does a good job of inferring the hidden distributions both at the instance level and the rule level.

The goal is to evaluate the different versions of the Noisy-OR learning algorithms on this dataset to determine the accuracy of the learned distributions. We trained the learning algorithms on 15 sets of 2000 training examples and tested them on a set of 1000 test examples. The average absolute difference between corresponding entries in the true distribution and the predicted distribution was averaged over all the test examples. We flattened the data set by using the counts of the instances of the parents as features and used Weka to run Naive Bayes on this modified data set. We obtained the distribution over the target variable and present the results here.

The results are presented in Figure 3.6. The x-axis has the number of examples and y-axis has the average absolute error for the examples. As can be seen, all the algorithms

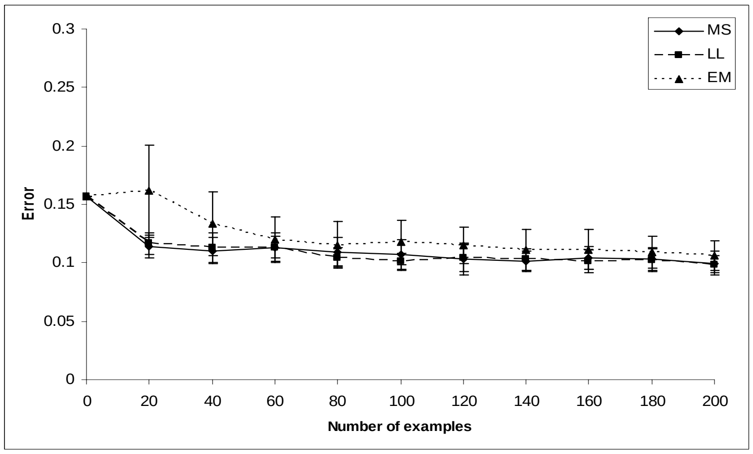


Figure 3.6: Learning curves for algorithms that use Noisy-OR on the synthetic data. EM: Expectation Maximization; GDMS: Gradient descent for Mean Square error; GDLL: Gradient descent for log likelihood

eventually converge to almost the same error rate (there is no statistically significant difference in error rates). Initially, EM seems to perform worse, but with more training data achieves comparable performance to the gradient descent methods. We also ran experiments using the Naive Bayes algorithm in weka. The Naive Bayes algorithm had a very poor performance and a very high error rate of close to 0.42 even with about 2000 training examples. Since the performance is very poor, we omit the propositional classifier from the learning curves.

3.5.2 Folder prediction

We employed the two rules that were presented earlier.

```
WeightedMean{
  If {role(Doc,Role,Task)} then  Task.id, Role.id Qinf (Mean) Doc.folder.
  If {source(Src,Doc)} then Src.folder Qinf (Mean) Doc.folder.
  }
```

As part of the Task Tracer project [22], we collected data for 500 documents and 6 tasks. The documents were stored in 11 different folders. Each document was manually assigned to a role with respect to each task with which it was associated. A document was assigned the *main* role if it was modified as part of the task. Otherwise, the document was assigned the *reference* role, since it was opened but not edited. A document is a *source* document if it was opened, edited, and then saved to create a new document or if large parts of it were copied and pasted into the new document. Since the documents could play several roles in several tasks, the number of $\langle Task, Role \rangle$ pairs vary[1].

We applied Gradient Descent and EM algorithms to learn both the parameters of the CPTs and the weights of the Weighted-Mean combining rule. We employed 10-fold cross-validation to evaluate the results. Within each fold, the learned network was applied to rank the folders of the current document and the position of the correct folder in this ranking was computed (counting from 1). The results are shown in Table 3.4, where the counts report the total number of times (out of 500) that the correct folder was ranked 1^{st}, 2^{nd}, etc. The final row of the table reports the mean reciprocal rank of the correct folder (the average of the reciprocals of the ranks). Mean reciprocal rank is a standard performance metric used in information retrieval literature and it is the higher the better. It is 1 if the true folder is always ranked as the top choice. If the true folder is always ranked second, the mean reciprocal rank falls to 1/2. If it is always ranked n, the mean reciprocal rank is $1/n$. Thus the score decreases monotonically with the amount of misranking, but suffers more at the top of the list than at the bottom of the list.

It is clear from the table that all the three relational algorithms performed very well: almost 90% of the documents had their correct folders ranked as 1 or 2 by all three algorithms[2].

[1]On average, each document participated in 2 $\langle Task, Role \rangle$ pairs, although a few documents participated in 5 to 6 $\langle Task, Role \rangle$ pairs.

[2]If the algorithm were to rank the folders at random, the score would be around 0.2745.

To compare these results with propositional learners, we flattened the data using as features the numbers of times each task-role pair and each source folder appears in each example. We then used Weka to run J48 and Naive Bayes algorithms on this new dataset to predict the class probabilities. J48 on the flattened data also performs as well as the relational classifiers while Naive Bayes does a little worse on the same data set. All the relational algorithms attributed high weights to the second rule compared to the first (see Table 3.5).

Rank	EM	GD-MS	GD-LL	J48	NB
1	349	354	346	351	326
2	107	98	113	100	110
3	22	26	18	28	34
4	15	12	15	6	19
5	6	4	4	6	4
6	0	0	3	0	0
7	1	4	1	2	0
8	0	2	0	0	1
9	0	0	0	6	1
10	0	0	0	0	0
11	0	0	0	0	5
Score	0.8299	0.8325	0.8274	0.8279	0.797

Table 3.4: Results of the learning algorithms on the folder prediction task. GD-MS: Gradient descent for Mean Square error; GD-LL: Gradient descent for log-likelihood; J48: Decision Tree; NB: Naive Bayes for loglikelihood.

To test the importance of learning the weights in the case with the Weighted-Mean, we altered the data set so that the folder names of all the sources were randomly chosen. As can be seen in Table 3.5, with this change, the source document rule is assigned low weight by the learning algorithms with a small loss in the score. In particular, the learning algorithms for the Weighted-Mean combining rule assigned a weight of 0.1 to the source rule and a weight of 0.9 to the task-role rule. The mean reciprocal rank was 0.832, so this shows that the learning algorithms were able to recover from the irrelevant inputs. We then repeated the learning process on this data set, but we held the weights on the two rules fixed at 0.5. The results were much worse, because the algorithms were forced to combine the useless source probability distribution with the informative task-role distribution. The mean reciprocal rank was 0.546.

		EM	GD-MS	GD-LL
Original data set	Weights	$\langle .15, .85\rangle$	$\langle .22, .78\rangle$	$\langle .05, .95\rangle$
	Score	.8299	.8325	.8274
Modified data set	Weights	$\langle .9, .1\rangle$	$\langle .84, .16\rangle$	$\langle 1, 0\rangle$
	Score	.7934	.8021	.7939

Table 3.5: Results of learning the weights in the original data set and the modified data set.

3.5.3 Synthetic data set experiment for weighted mean

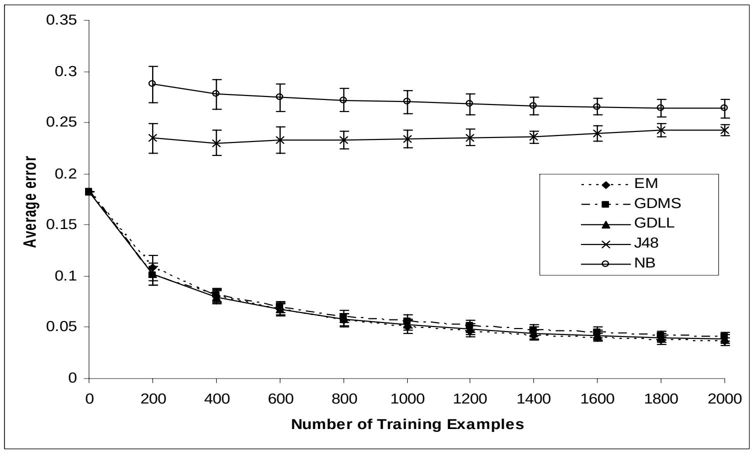


Figure 3.7: Learning curves for the synthetic data. EM: Expectation Maximization; GDMS: Gradient descent for Mean Square error; GDLL: Gradient descent for log likelihood; J48: Decision tree; NB: Naive Bayes.

The folder prediction experiment demonstrates good performance on an interesting real-world task, but it does not tell us how close the predicted probability distribution is to the true distribution on the test set. To realistically model a complex real-world domain, it is not enough to have a good classification accuracy on a single task. To use these predictions in complex inferences, it is important to accurately model the probability distributions. To estimate the accuracy of the learned model, we constructed a synthetic data set very similar to the one that was presented earlier. The data are generated using a synthetic target as defined by two FOCI statements, each of which has two influents and the same target attribute. The different instances of the same rule are combined using mean and the different rules are combined using the Weighted-Mean combining rule. The two influents in each rule have a range of 10 and 3 values respectively. The target attribute can take 3 values. The probability values in the distribution of the synthetic target are randomly generated to be either between 0.9 and 1.0 or between 0.0 and 0.1. As with previous experiment, this is to make sure that the probabilistic predictions on examples are hard to predict and not too close to the default probability of $\frac{1}{3}$ each. The rule weights are fixed to be 0.1 and 0.9 to make them far from the default, 0.5. Each example matches a rule with probability 0.5, and when it does match, it generates a number of instances randomly chosen between 3 and 10. This makes it imperative that the learning algorithm does a good job of inferring the hidden distributions both at the instance level and the rule level.

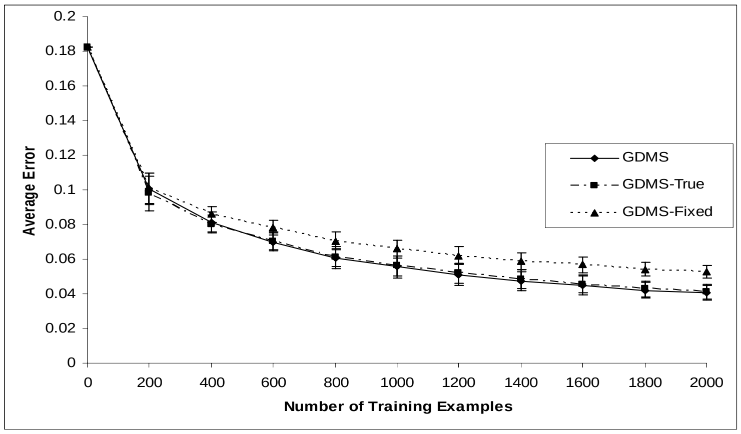


Figure 3.8: Learning curves for mean squared gradient descent on the synthetic data. GDMS: learning the weights; GDMS-True: gradient descent with true weights; GDMS-Fixed: gradient descent with weights fixed as $\langle 0.5, 0.5 \rangle$.

We trained the learning algorithms on 30 sets of 2000 training examples and tested them on a set of 1000 test examples. The average absolute difference between corresponding entries in the true distribution and the predicted distribution was averaged over all the test examples. Like the folder data set, we flattened the data set by using the counts of the instances of the parents as features and used Weka to run J48 and Naive Bayes on this modified data set. We obtained the distribution over the target variable from weka for these two algorithms.

The results are presented in Figure 3.7. All three relational algorithms that use the correct model have a very low average absolute error between the true and the predicted distribution. The overlapping of the error bars suggests that there is no statistically significant difference between the algorithms' performances. On the other hand, the propositional classifiers perform poorly on this measure compared to the relational algorithms. The performance of the relational algorithms is significantly better than the propositional classifiers. This demonstrates that though the propositional algorithms can perform reasonably well on a classification task, it is harder for them to learn the true distribution in a relational setting.

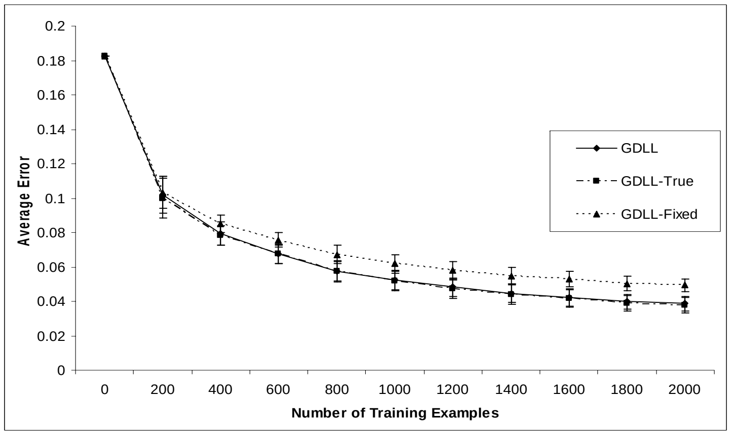


Figure 3.9: Learning curves for log-likelihood gradient descent on the synthetic data. GDLL: learning the weights; GDLL-True: gradient descent with true weights; GDLL-Fixed: gradient descent with weights fixed as $\langle 0.5, 0.5 \rangle$.

As with the folder data set, we wanted to understand the importance of learning the weights. Hence, for each learning algorithm, we compared three settings. The first setting is the normal situation in which the algorithm learns the weights. In the second setting, the weights were fixed at $\langle 0.5, 0.5 \rangle$. In the third setting, the weights were fixed to be their true values.

The results are presented in Figures 3.8, 3.9, and 3.10. There are three curves in each figure corresponding to the three settings. In all three algorithms, the first setting (weights are learned) gave significantly better error rates than the second setting (weights fixed at $\langle 0.5, 0.5 \rangle$) (Figures 3.8,3.9,3.10). This clearly demonstrates the importance of learning the weights. There was no significant difference between learning the weights and knowing the true weights. This shows that our algorithms effectively learn the weights of the combining rules with no additional cost.

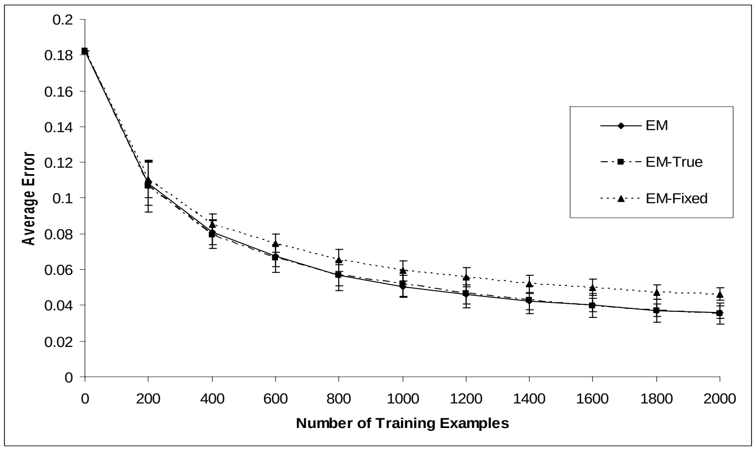


Figure 3.10: Learning curves for EM on the synthetic data. EM: learning the weights; EM-True: EM with true weights; EM-Fixed: EM with weights fixed as $\langle 0.5, 0.5 \rangle$.

3.6 Discussion

Learning the parameters of the first-order clauses have been explored by previous researchers. In [44], Koller and Pfeffer investigated the use of EM algorithm to learn the parameters of the first-order clauses in the presence of combining rules. They use Knowledge-Based Model Construction (KBMC) to construct a ground network for each data case and employ the EM algorithm to learn the parameters of the rules. The combining rules are also well-explored in other relational probabilistic settings such as probabilistic horn abduction [59] and Bayesian logic programs [42]. Indeed, the ability to tractably compose different influence statements or rules is necessary to build compact models. What is different in our work is that we learn the parameters in the presence of multiple influence statements each of which may have multiple instantiations. We show that both gradient descent and EM can not only learn the CPTs of the rules but also their parameters.

More recently Jaeger considers a weighted combination or a nested combination of the combining rules and uses a gradient ascent algorithm for optimizing the objective function [39]. This technique has been applied to his formalism of Relational Bayesian Networks(RBNs)[38]. The RBN is compiled into a likelihood graph that is then used for the various computations that are needed for the gradient ascent equation. The use of the likelihood graph greatly reduces the number of computations needed for the gradient computation.

3.7 Conclusion and Future Work

Combining rules help exploit causal independence in Bayesian networks and make representation and inference more tractable in the propositional case [33]. In first order languages, they allow succinct representation and learning of the parameters of networks where the number of parents of a variable varies from one instantiation to another. They make it possible to express pairwise interactions between influences and provide a natural representation for many models of influence.

We showed that we can employ classic algorithm schemes such as gradient descent and

EM to learn the parameters of the conditional influence statements as well as the weights of the combining rules. The performances of the three algorithms with the Weighted-Mean combining rule were quite similar on the folder data set. In the folder data set, the propositional classifiers performed as well as the relational ones. This is partly because the examples in this dataset often have only one or two task-role pairs, which makes it an easier problem. In the synthetic domain, all examples have at least 3 task-role pairs, and the propositional algorithms performed poorly. The experiments also show that learning the probability model is much more difficult than learning to classify. It would be interesting to extend this work to search over a set of combining rules to determine the one that best fits the data.

One of our goals is to extend this work to more general classes of combining rules and aggregators including tree-structured CPTs and noisy versions of other symmetric functions. The relationship between the aggregators and combining rules must be better understood and formalized. Efficient inference algorithms must be developed that take advantage of the decomposability of the combining rules as well as the flexibility of the first-order notation. Finally, we would like to develop more compelling applications in knowledge-rich and structured domains that can benefit from the richness of the first-order probabilistic languages. Extending the SRL languages to dynamic domains with actions and utility makes them much more appropriate for compelling real-world applications. Some work has already begun in this direction [56] and presented in the next chapter.

Chapter 4 – A Relational Hierarchical Model for Decision-Theoretic Assistance

4.1 Overview

In Chapter 2, a domain-independent decision-theoretic model of assistance was proposed, where the task is to infer the user's goal and take actions that minimize the expected cost of the user's policy. In this chapter, we extend this work to domains where the user's policies have rich relational and hierarchical structure. To this effect, we draw upon our experience in relational modeling which we presented in the previous chapter. Our results indicate that relational hierarchies allow succinct encoding of prior knowledge for the assistant, which in turn enables the assistant to start helping the user after a relatively small amount of experience.

4.2 Introduction

There has been a growing interest in developing intelligent assistant systems that help users in a variety of tasks ranging from washing hands to travel planning [10, 67, 2]. The emphasis in these systems has been to provide a well-engineered domain-specific solution to the problem of reducing the users' cognitive load in their daily tasks. A decision-theoretic model was proposed in Chapter 2 to formalize the general problem of assistantship as a partially observable Markov decision process (POMDP). In this framework, the assistant and the user interact in the environment to change its state. The goal of the assistant is to take actions that minimize the expected cost of completing the user's task [24]. In most situations, however, the user's task or goalis not directly observable to the assistant, which makes the problem of quickly inferring the user's goals from observed actions critically important. One approach to goal inference [24] is to learn a probabilistic model of the user's policy for achieving various goals and then to compute a posterior distribution over goals given the current observation history. However, for this approach to be useful in practice, it is important that the policy be learned as early in the lifetime of the assistant as possible. We call this the problem of "early assistance", which is the main motivation behind this work.

One solution to the early assistance problem, advocated in [24] and presented in Chapter 2, is to assume that (a) the user's policy is optimal with respect to their goals and actions, the so called "rationality assumption," and that (b) the optimal policy can be computed quickly by knowing the goals, the "tractability assumption." Under these assumptions, the user's policy for each goal can be approximated by an optimal policy, which may be quickly computed. Unfortunately in many real world domains, neither of these assumptions is realistic. Real world domains are too complex to allow tractable optimal solutions. The limited computational power of the user renders the policies to be locally optimal at best.

In this chapter, we propose a different solution to the early assistance problem, namely constraining the user's policies using prior domain knowledge in the form of hierarchical and relational constraints. Consider an example of a desktop assistant similar to CALO [13] that helps an academic researcher. The researcher could have some high level tasks like writing a proposal, which may be divided into several subtasks such as preparing the cover page, writing the project description, preparing the budget, completing the biography, etc. with some ordering relationships between them. We expect that an assistant that knows about this high level structure would better help the user. For example, if the budget cannot be prepared before the cover page is done, the assistant would not consider that possibility and can determine the user's task faster. In addition to the hierarchical structure, the tasks, subtasks, and states have a class and relational structure. For example, the urgency of a proposal depends on the closeness of the deadline. The deadline of the proposal is typically mentioned on the web page of the agency to which the proposal is addressed. The collaboration potential of an individual on a proposal depends on their expertise in the areas related to the topic of the proposal. Knowing these relationships and how they influence each other could make the assistant more effective.

This work extends the assistantship model to hierarchical and relational settings, building on the work in hierarchical reinforcement learning[19] and statistical relational learning (SRL).We extend the assistantship framework presented in Chapter 2 by including parameterized task hierarchies and conditional relational influences as prior knowledge of the assistant. We compile this knowledge into an underlying Dynamic Bayesian network and use Bayesian network inference algorithms to infer the distribution of user's goals given a sequence of their atomic actions. We estimate the parameters for the user's policy and influence relationships by observing the users' actions. Once the user's goal distribution is inferred, we determine an approximately optimal action by estimating the Q-values of different actions using rollouts and picking the action that has the least expected cost. The current work can be understood as augumenting the model presented in Chapter 2 with prior knowledge of the form presented in Chapter 3.

We evaluate our relational hierarchical assistantship model in two different toy domains and compare it to a propositional flat model, propositional hierarchical model, and a relational flat model. Through simulations, we show that when the prior knowledge of the assistant matches the true behavior of the user, the relational hierarchical model provides superior assistance in terms of performing useful actions. The relational flat model and the propositional hierarchical model provide better assistance than the propositional flat model, but fall short of the performance of the relational hierarchical approach.

The rest of the chapter is organized as follows: Section 2 summarizes the basic decision-theoretic assistance framework, which is followed by the relational hierarchical extension in Section 3. Section 4 presents the experiments and results, Section 5 outlines some related work and Section 6 concludes the chapter.

4.3 Decision-Theoretic Assistance

In this section, we briefly recall the decision-theoretic model of assistance presented in Chapter 2 which forms the basis the work in this chapter. In this setting, there is a user acting in the environment and an assistant that observes the user and attempts to assist him. The environment is modeled as an MDP described by the tuple $\langle W, A, A', T, C, I \rangle$, where W is a finite set of world states, A is a finite set of user actions, A' is a finite set of assistant actions, and $T(w, a, w')$ is a transition function that represents the probability of transitioning to state w' given that action $a \in A \cup A'$ is taken in state w. C is an action-cost function that maps $W \times (A \cup A')$ to real numbers, and I is an initial state distribution over W. An episodic setting is assumed, where the user chooses a goal and tries to achieve it. The assistant observes the user's actions and the world states but not the goal. After every user's action, the assistant gets a chance to take one or more actions ending with a **noop** action, after which the user gets a turn. The objective is to minimize the sum of the costs of user and assistant actions.

The user is modeled as a stochastic policy $\pi(a|w, g)$ that gives the probability of selecting action $a \in A$ given that the user has goal g and is in state w. The objective is to select an assistant policy π' that minimizes the expected cost given the observed history of the user. The environment is only partially observable to the assistant since it cannot observe the user's goal. It can be modeled as a POMDP, where the user is treated as part of the environment.

The assistant POMDP is solved approximately, by first estimating the goal of the user given the history of his actions, and then selecting the best assistive action given the posterior goal distribution. One of the key problems in effective assistantship is to learn the task quickly enough to start helping the user as early as possible. As pointed out in Chapter 2, this problem is solved by assuming that the user is rational, i.e., he takes actions to minimize the expected cost. Further, the user MDP is assumed to be tractably solvable for each goal. Hence, the earlier system solves the user MDP for each goal and uses it to initialize the user's policy.

Unfortunately the dual assumptions of tractability MDP and rationality make this approach too restrictive to be useful in real-world domains that are too complicated for any user to approach perfect rationality. We propose a knowledge-based approach to the effective assistantship problem that bypasses the above two assumptions. We provide the assistant with partial knowledge of the user's policy, in the form of a task hierarchy with relational constraints on the subtasks and their parameters. Given this strong prior knowledge, the assistant is able to learn the user's policy quickly by observing his actions and updating the policy parameters. We appropriately adopt the goal estimation and action selection steps of the previous model to the new structured policy of the user and show that it performs significantly better than the unstructured approach.

4.4 A Relational Hierarchical Model of Assistance

In this section, we propose a relational hierarchical representation of the user's policy and show its use for goal estimation and action selection.

4.4.1 Relational Hierarchical Policies

Typically, users solve difficult problems by decomposing them into a set of smaller ones with some ordering constraints between them. For example, proposal writing might involve writing the project description, preparing the budget, and then getting signatures from proper authorities. Also, the tasks have a natural class-subclass hierarchy, e.g., submitting a paper to ICML and IJCAI might involve similar parameterized subtasks. In the real world, the tasks are chosen based on some attributes of the environment or the user. For instance, the paper the user works on next is influenced by the closeness of the deadline. It is these kinds of relationships that we want to express as prior knowledge so that the assistant can quickly learn the relevant parameters of the policy. We model the user as a stochastic policy $\pi(a|w, T, O)$ that gives the probability of selecting action $a \in A$ given that the user has goal stack T and is in state w. O is the history of the observed states and actions. Learning a flat, propositional representation of the user policy is not practical in many domains. Rather, in this work, we represent the user policy as a *relational task hierarchy* and speed up the learning of the hierarchy parameters via the use of *conditional influence statements* that constrain the space of probabilistic dependencies.

Relational Task Hierarchies. A relational task hierarchy is specified over a set of variables, domain constants, and predicate symbols. There are predicate symbols for representing properties of world states and specifying task names. The task predicates are divided into primitive and abstract tasks. Primitive task predicates will be used to specify ground actions in the MDP that can be directly executed by the user. Abstract task predicates will be used to specify non-primitive procedures (that involve calling subtasks) for achieving high-level goals. Below we will use the term *task stack* to mean a sequence of ground task names (i.e. task predicates applied to constants).

A relational task hierarchy will be composed of relational task schemas which we now define.

Definition 1 (Relational Task Schema). *A relational task schema is either: 1) A primitive task predicate applied to the appropriate number of variables, or 2) A tuple $\langle N, S, R, G, P\rangle$, where the task name N is an abstract task predicate applied to a set of variables V, S is a set of child relational task schemas (i.e. the subtasks), R is a set of logical rules over state, task, and background predicates that are used to derive a candidate set of ground* *child tasks in a given situation, G is a set of rules that define the goal conditions for the task, and $P(s|T, w, O)$ is a probability distribution that gives the probability of a ground child task s conditioned on a task stack T, a world state w, and an observation history O.*

Each way of instantiating the variables of a task schema with domain constants yields a ground task. The semantics of a relational task schema specify what it means for the user to "execute to completion" a particular ground task as follows. As the base case, a primitive ground task is executed-to-completion by simply executing the corresponding primitive MDP action until it terminates, resulting in an updated world state.

An abstract ground task, can intuitively be viewed as specifying a stochastic policy over its child subtasks which is executed until its goal condition is satisfied. More precisely, an abstract ground task t is executed-to-completion by repeatedly selecting ground child tasks that are executed-to-completion until the goal condition G is satisfied. At each step given the current state w, observation history O, task stack T, and set of variable bindings B (that include the bindings for t) a child task is selected as follows: 1) Subject to the variable bindings, the rules R are used to derive a set of candidate ground child tasks. 2) From this set we draw a ground task s according to P, properly normalized to only take into account the set of available subtasks. 3) The drawn ground task is then executed-to-completion in the context of variables bindings B' that include the bindings in B along with those in s and a task stack corresponding to pushing t onto T.

Based on the above description, the set of rules R can be viewed as specifying hard constraints on the legal subtasks with P selecting among those tasks that satisfy the constraints. The hard constraints imposed by R can be used restrict the argument of the child task to be of a certain type or may place mutual constraints on variables of the child tasks. For example, we could specify rules that say that the document to be attached in an email should belong to the project that the user is working on. Also, the rules can specify the ordering constraint between the child tasks. For instance, it would be possible to say that to submit a paper the task of writing the paper must be completed first.

We can now define a relational task hierarchy.

Definition 2 (Relational Task Hierarchy)**.** *A relational task hierarchy is rooted acyclic graph whose nodes are relational task schemas that satisfy the following constraints: 1) The root is a special subtask called ROOT. 2) The leaves of the graph are primitive task schemas. 3) There is an arc from node n_1 to node n_2 if and only if the task schema of n_2 is a child of task schema n_1.*

We will use relational task hierarchies to specify the policy of a user. Specifically, the user's actions are assumed to be generated by executing the ROOT task of the hierarchy with an initially empty goal stack and set of variable bindings.

An example of a *Relational Task Hierarchy* is presented in the Figure 4.1 for a game involving resource gathering and tactical battles. For each task schema we depict some of the variable binding constraints enforced by the R as a logical expression. For clarity we do not depict the ordering constraints imposed by R. From the ROOT task the user has two distinct choices to either gathering a resource, *Gather(R)* or attacking an enemy, *Attack(E)*. Each of these tasks can be achieved by executing either a primitive action (represented with ovals in the figure) or another subtask. For example, to gather a resource, the user needs to collect the resource (denoted by *Collect(R)*) and deposit the resource at the storage (denoted by *Deposit(R,S)*, which indicates that R is to be deposited in S). Resources are stored in the storages of the same type (for example, gold in a bank, food in a granary etc.), which is expressed as the constraint $R.type = S.type$ in the figure. Once the user chooses to gather a resource (say $gold1$), the value of R in all the nodes that are lower than the node $Gather(R)$ is set to the value $gold1$. R is freed after $Gather$ is completed.

Conditional Influences: Often it is relatively easy to hand-code the rule sets R that encode hard-constraints on child tasks. It is more difficult to precisely specify the

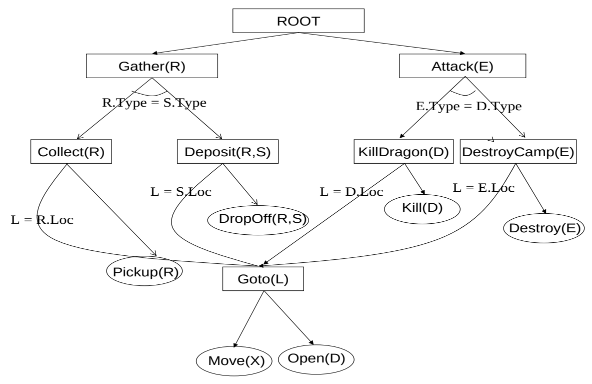


Figure 4.1: Example of a task hierarchy of the user. The inner nodes indicate subtasks while the leaves are the primitive actions. The tasks are parameterized and the tasks at the higher level will call the tasks at the lower level

probability distributions for each task schema. In this work, we take the approach of hand-coding a set of conditional influence statements that are used to constrain and hence speedup the learning of these probability distributions. The conditional influences describe the objects and their attributes that influence a subtask choice based on some condition, i.e., these statements serve to capture a distribution over the subtasks given some attributes of the environment ($P(subtask \mid worldstate)$). For example, since there could be multiple storage locations for a resource, the choice of a storage may be influenced by its distance to the resource. While this knowledge can be easily expressed in most SRL formalisms such as Probabilistic Relational Language [28] and Bayesian Logic Programs [42], we give an example in First-Order Conditional Influence Language (FOCIL) [55] presented in the previous chapter.

```
If {Goal(Gather(R)),Completed(Collect(R)),Equal(Type(R),Type(S))} then
Distance(Loc(R), Loc(S))) Qinf subgoal(Deposit(R,S))
```

Recall that a FOCIL statement of the form $If\{Z(\alpha)\} \quad then \quad Y_1(\alpha), \ldots, Y_k(\alpha) \quad Qinf \quad X(\alpha)$ means that $Y_1(\alpha)$, …, $Y_k(\alpha)$ influence $X(\alpha)$ when $Z(\alpha)$ is true, where α is a set of logical variables. The above statement captures the knowledge that if R is a resource that has been collected, and S is a storage where R can be stored, the choice of the value of S is influenced by the distance between R and S. The semantics and learning algorithms for such relational statements were discussed in the previous chapter. The influence statements also serve the role of "abstraction" in the MAXQ approach to hierarchical reinforcement learning [19]. The probability of choosing a subtask in a given state is determined solely by the attribute values of the objects mentioned in the conditional influence statement, which puts a strong constraint on the user's policy and makes it easier to learn.

The high level algorithm is presented in table 4.1. The parameters are updated at the end of the episode using MLE estimates. When an episode is completed, the set of completed tasks and the action trajectories are used to update the parameters of the nodes at different levels.

Table 4.1: Highlevel algorithm for assistance

- Iitialize DBNs as in Figure 4.2 incorporating all hard constraints into the CPTs
- For each episode
  - For each time step
    - Observe any task completed
    - Update the posterior distribution of goal stack based on the observation, the hard constraints, and FOCI statements
    - Observe the next action
    - Update the posterior distribution over the tasks in the task stack
    - Compute the best assistive action
  - Update the DBN parameters

4.4.2 Goal Estimation

In this section, we describe our goal estimation method, given the kind of prior knowledge described in the previous section, and the observations, which consist of the user's primitive actions. Note that the probability of the user's action choice depends in general on not only the pending subgoals, but also on some of the completed subgoals including their variable bindings. Hence, in general, the assistant POMDP must maintain a belief state distribution over the pending and completed subgoals. which we call the "goal structure."

We now define the assistant POMDP. The **state space** is $W \times \mathbf{T}$ where W is the set of world states and $\mathbf{T}$ is the user's goal structure. Correspondingly, the **transition probabilities** are functions between $(w, \mathbf{t})$ and $(w', \mathbf{t})$. Similarly, the **cost** is a function of $\langle$*state, action*$\rangle$ pairs. The **observation** space now includes the user's actions and their parameters (for example, the resource that is collected, the enemy type that is killed etc).

In this work, we make a simplifying assumption that there is no uncertainty about the completed subtasks. This assumption is justified in our domains, where the completion of each subtask is accompanied with an observation that identifies the subtask that has just completed. This would simplify the inference process as we do not need to maintain a distribution over the (possibly) completed subtasks. For estimating the user's goal stack, we use a DBN similar to the one used in [52] and present it in Figure 4.2. T^i_j refers to the task at time-step j and level i in the DAG. O^i refers to the completed subtask at level i. F^i_j is an indicator variable that represents whether T^i_j has been completed and acts as a multiplexer node. If the lower level task is completed and the current task is not completed, the transition function for the current task would reflect choosing an action for the current subtask. If the lower level task is not completed, the current task stays at its current state. If the current task is completed, the value is chosen using a prior distribution over the current task given the higher level tasks.

In the experiments reported in the next section, we compiled the FOCIL statements into a DBN structure by hand. The number of levels of the tasks in the DBN corresponds to the depth of the directed graph in the relational task hierarchy. The values of the different task level nodes will be the instantiated tasks in the hierarchy. For instance, the variable T_j^1 takes values corresponding to all possible instantiations of the second-level tasks. Once the set of possible values for each current task variable in the task is determined, the constraints are used to construct the CPT. For example, the constraint $R.Type = S.Type$ in the Figure 4.1 implies that a resource of one type can be stored in the storage of the same type. Assume that the user is gathering *gold*. Then in the CPT corresponding to $P(T_j^2 = Store(S, gold) \mid T_j^1 = Gather(gold)$, all the entries except the ones that correspond to a bank are set to 0. The rules R of the task schema determine the non-zero entries of the CPTs, while the FOCIL statements constrain the distributions further. Note that, in general, the subtasks completed at a particular level influence the distribution over the current subtasks at the same level through the hard constraints, which include ordering relationships. In our experiments, however, we have chosen to not explicitly store the completed subtasks at any stage since the ordering of subtasks has a special structure. The subtasks are partitioned into small unordered groups, where the groups are totally ordered. This allows us to maintain a small memory of only the completed subtasks in the current group.

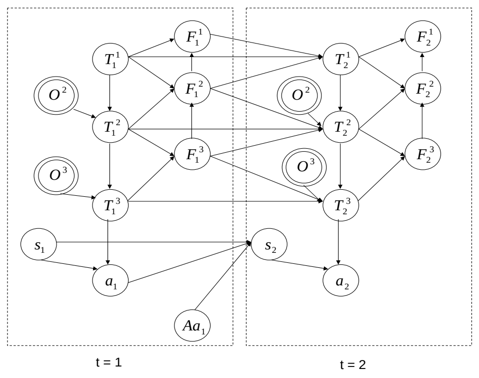


Figure 4.2: Dynamic Bayesian network that is used to infer the user's goal.

To illustrate the construction of the DBN given the hierarchy and influence statements better, let us consider the example presented in Figure 4.1. Assume that the user chooses to gather $g1$ (i.e., gold from location 1). Once the episode begins, the variables in the DBN are instantiated to the corresponding values. The task at the highest level T_j^1, would take values from the set $\langle$ $Gather(g1)$, $Gather(g2)$, $Gather(w1)$,$Gather(w2)$, $Destroy(e1)$,$Destroy(e2)$ $\rangle$, assuming that there are 2 gold and wood locations and 2 enemies. Similarly, the tasks at level n of the DBN would assume values corresponding to the instantiation of the nodes at the n^{th} level of the hierarchy. The conditional influence statements are used to obtain a prior distribution over the goal stack only after every subtask is finished or to minimize uncertainty and retain tractability. Once the prior is obtained, the posterior over the goal stack is updated after every user action. For example, once the user finishes the subtask of $collect(g1)$, the relational structure would restrict the set of subgoals to depositing the resource and the conditional influence statements would provide a prior over the storage locations. Once the highest level task of *Gather* is completed, the DBN parameters are updated using the complete set of observations. Our hypothesis that we verify empirically is that, the relational structure and the conditional influence statements together provide a strong prior over the task stack which enables fast learning.

Given this DBN, we need to infer the value of $P(T_j^{1:d} \mid T_{j-1}^{1:d}, F_{j-1}^{1:d}, a_j, O^{1:d})$, where d is the depth of the DAG i.e, infer the posterior distribution over the user's goal stack given the observations (the user actions in our case) and the completed goal stack. As we have mentioned, we are not considering the completed subgoals due to the fact that most of our constraints are total order and there is no necessity of maintaining them. Since we always estimate the current goal stack given the current action and state, we can approximate the DBN inference as a BN inference for the current time-step. The other issue is the learning of parameters of the DBN. At the end of every episode, the assistant updates the parameters of the DBN based on the observations in that episode using maximum likelihood estimates with Laplace correction. Since the model is inherently relational, we can exploit parameter tying between similar objects and hence accelerate the learning of parameters. The parameter learning in the case of relational models is significantly faster as demonstrated by our experiments.

It should be noted that in the earlier model, we solved the user MDP and used the values to initialize the priors for the user's action models. Though it seems justifiable, it is not always possible to solve the user MDP. We show in our experiments that even if we begin with an uniform prior for the action models, the relations and the hierarchical structure would enable the assistant to be useful even in the early episodes.

4.4.3 Action Selection

Given the assistant POMDP M and the distribution over the user's goal stack $P(T^{1:d} \mid O_j)$, where O_j are the observations, we can compute the value of assistive actions. Following the approach presented in Chapter 2, we approximate the assistant POMDP with a series of MDPs $M(t^{1:d})$, for each possible goal stack $t^{1:d}$. Thus, the heuristic value of an action a in a world state w given the observations O_j at time-step j would now correspond to,

$$H(w, a, O_j) = \sum_{t^{1:d}} Q_{t^{1:d}}(w, a) \cdot P(t^{1:d}|O_j)$$

where $Q_{t^{1:d}}(w, a)$ is the value of performing the action a in state w in the MDP $M(t^{1:d})$ and $P(t^{1:d}|O_j)$ is the posterior probability of the goal stack given the observations. Instead of sampling over the goals, we sample over the possible goal stack values. The relations between the different goals would restrict the number of goal-subgoal combinations. If the hierarchy is designed so that the subgoals are not shared between higher level goals, we can greatly reduce the number of possible combinations and hence making the sampling process practically feasible. We verify this empirically in our experiments. To compute the value of $Q_{t^{1:d}}(w, a)$, we use the policy rollout technique [7] where the assumption is that the assistant would perform only one action and assumes that the agent takes over from there and estimates the value by rolling out the user policy. Since the assistant has access to the hierarchy, it chooses the actions subjected to the constraints specified by the hierarchy.

4.5 Experiments and Results

In this section, we briefly explain the results of simulation of a user in two domains[1]: a gridworld doorman domain where the assistant has to open the right doors to the user's destination and a kitchen domain where the assistant helps the user in preparing food. We simulate a user in these domains and compare different versions of the decision theoretic model and present the results of the comparison. The different models that we compare are: the relational hierarchical model that we presented, a hierarchical model where the goal structure is hierarchical, a relational model where there are objects and relations but there is a flat goal structure and a flat model which is a very naive model with a flat goal structure and no notion of objects are relationships. Our hypothesis is that the relational models would benefit from parameter tying and hence can learn the parameters faster and would offer better assistance than their propositional counterparts at earlier episodes. Similarly, the hierarchical model would make it possible to decompose the goal structure thus making it possible to learn faster. We demonstrate through experiments that the combination of relational and hierarchical models would enable the assistant to be more effective than the assistant that uses either of these models.

[1]These are modification to the domains presented in Chapter 2

4.5.1 Doorman Domain

In this domain, the user is in a gridworld where each grid cell has 4 doors that the user has to open to navigate to the adjacent cell (see Figure 4.3. The hierarchy presented in Figure 4.1.*a* was used as the user's goal structure. The goals of the user are to *Gather* a resource or to *Attack* an enemy. To *gather* a resource, the user has to *collect* the resource and *deposit* it at the corresponding location. Similarly, to *destroy* an enemy, the user has to kill the *dragon* and *destroy* the castle. There are different kinds of resources, namely *food* and *gold*. Each resource can be stored only in a storage of its own type (i.e, *food* is stored in *granary* and *gold* is stored in *bank*). There are 2 locations for each of the resources and its storage. Similarly there are 2 kinds of enemy *red* and *blue*. The user has to kill the *dragon* of a particular kind and *destroy* the castle of the same kind. The episode ends when the user achieves the highest level goal. The actions that the user can perform are to move in 4 directions, open the 4 doors, pick up, put down and attack. The assistant can only open the doors or perform a *noop*. The door closes after one time-step so that at any time only one door is open. The goal of the assistant is to minimize the number of doors that the user needs to open. The user and assistant take actions alternately in this domain.

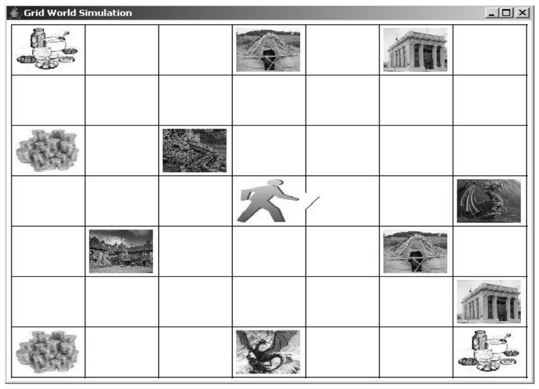


Figure 4.3: Doorman Domain. Each cell has 4 doors that the user has to open to navigate to the adjacent cell. The goal of the assistant is to minimize the number of doors that the user has to open.

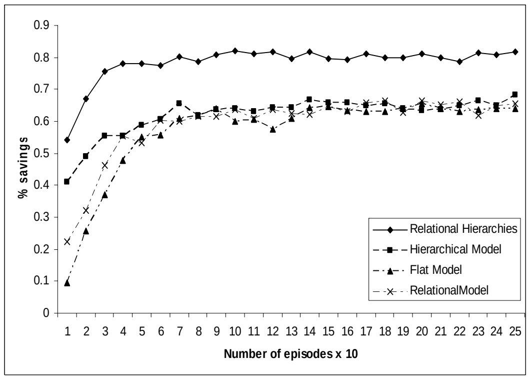


Figure 4.4: Learning curves for the 4 algorithms in the doorman domain. The y-axis presents the average savings for the user due to the assistant.

We employed four versions of the assistant that models the user's goal structure: one that models the structure as a relational hierarchical model, second which assumes a hierarchical goal structure but no relational structure (i.e., the model does not know that the 2 gold locations are of the same type etc and thus cannot exploit parameter tying), third which assumes a relational structure of user's goal but assumes flat goals and hence does not know the relationship between collect and deposit of subtasks, and the fourth that assumes a flat goal structure. A state is a tuple $\langle s, d \rangle$, where s stands for the the agent's cell and d is the door that is open. For the two flat cases, there is a necessity include variables such as *carry* that can take 5 possible values and *kill* that take 3 values to capture the state of the user having collected a resource or killed the dragon before reaching the eventual destination. Hence the state space of the 2 flat models is 15 times more than that of the hierarchical one.

To compare the 4 algorithms, we solved the underlying hierarchical MDP and then used the Q-values to simulate the user. For each episode, the higher level goals are chosen at random and the user attempts to achieve the goal. We calculate usefulness of the assistant as the ratio of the correct doors that it opens to the total number of doors that are needed to be opened for the user to reach his goal which is a worst-case measure of the cost savings of the user. We average the usefulness every 10 episodes. The user's policy is hidden from the assistant in all the algorithms and the assistant learns the user policy as and when the user performs his actions. The relational model captures the relationship between the resources and storage and between the dragon's type and the castle's type. The hierarchical model captures the relationship between the different goals and subgoals, for instance, that the user has to collect some resource in order to deposit it, etc. The hierarchical relational model has access to both the kinds of knowledge and also to the knowledge that the distance to the storage location influences the choice of the storage location.

The results are presented in Figure 4.4. The graph presents the average usefulness of the assistant after every 10 episodes. As can be seen from the figure, the relational hierarchical assistant is more useful than the other models. In particular, it can exploit the prior knowledge effectively as demonstrated by the rapid increase in the usefulness in earlier episodes. The hierarchical and relational models also exploit the prior knowledge and hence have a quicker learning rate than the flat model (as can be seen from the first few episodes of the figure). The hierarchical relational model outperforms the hierarchical model as it can share parameters and hence has to learn a smaller number of parameters. It outperforms the relational model as it can exploit the knowledge of the user's goal structure effectively and can learn quickly at the early stages of an episode.required for computing the best action of the assistant for all the four algorithms. This clearly demonstrates that the hierarchical relational model can be more effective without increasing the computational cost.

4.5.2 Kitchen Domain

The other experimental domain is a kitchen domain where the user has to cook some dishes. In this domain, the user has 2 kinds of higher-level goals: one in which he could prepare a recipe which contains a main dish and a side dish and the second in which, he could use some instant food to prepare a main dish and a side dish. There are 2 kinds of main dishes and 2 kinds of side dishes that he could prepare from the recipe. Similarly, there are 2 kinds of main dishes and 2 kinds of side dishes that he could prepare from instant food. The hierarchy is presented in Figure 4.5. The symbol $\in$ is used to capture the information that the object is part of the plan. For instance, the expression $I \in M.Ing$ means that the parameter to be passed is the ingredient that is used to cook the main dish. The plans are partially ordered. There are 2 shelves with 3 ingredients each. The shelves have doors that must be opened before fetching ingredients and only one door can be open at a time.

The state consists of the contents of the bowl, the ingredient on the table, the mixing state and temperature state of the ingredient (if it is in the bowl) and the door that is open. The user's actions are: open the doors, fetch the ingredients, pour them into the bowl, mix, heat and bake the contents of the bowl, or replace an ingredient back to the shelf. The assistant can perform all user actions except for pouring the ingredients or replacing an ingredient back to the shelf. The cost of all non-pour actions is -1. Unlike in the doorman domain, here it is not necessary for the assistant to wait at every alternative time step. The assistant continues to act until the **noop** becomes the best action according to the heuristic. The episode begins with all the ingredients in the shelf and the doors closed. The episode ends when the user achieves the goal of preparing a main dish and a side dish either with the recipe or using instant food. The savings is the ratio of the correct non-pour actions that the assistant has performed to the number of actions required for the goal. Similar to the other domain, we compared 4 different types of models of assistance. The first is the hierarchical relational model that has the knowledge of the goal-subgoal hierarchy and also has the relationship between the subgoals themselves. It knows that the type of the main dish influences the choice of the side dish. The second model is the hierarchical model, that has the notions of the goals and subgoals but no knowledge of

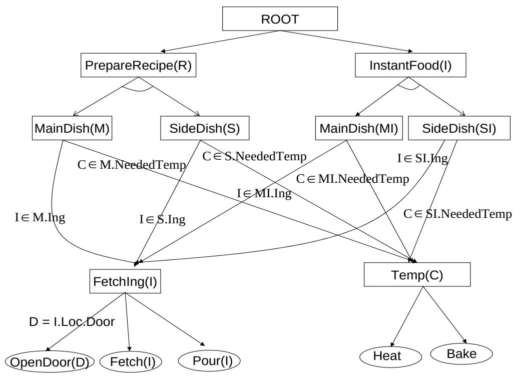


Figure 4.5: The kitchen domain hierarchy

the relationship between the main dishes and the side dishes and thus has more number of parameters to learn. The relational model assumes that there are two kinds of food namely the one prepared from recipe and one from instant food and does not possess any knowledge about the hierarchical goal structure. The flat model considers the preparation of each of the 8 dishes as a separate goal and assists the user. Both the flat model and the relational model assume that the user is always going to prepare the dishes in pairs but do not have the notion of main dish and side dishes or the ordering constraints between them.

The results are presented in Figure 4.6. As can be seen, the hierarchical models greatly dominate the flat ones. Among the models, the relational models have a faster learning rate than their propositional counterparts. They perform better in the earlier few episodes which clearly demonstrates that relational background knowledge accelerates learning. In this domain, the hierarchical knowledge seems to dominate the relational knowledge. This is due to the fact that all the subgoals are similar (i.e, each of them is preparing some kind of food) and the hierarchical knowledge clearly states the ordering of these subgoals. The relational hierarchical model has a better savings rate in the first few episodes as it has a fewer parameters to learn. Both the flat model and the relational model eventually converged on the same *savings* after 700 episodes. These results demonstrate that though all the models can eventually converge to the same value, the relational hierarchical model converges in early episodes.

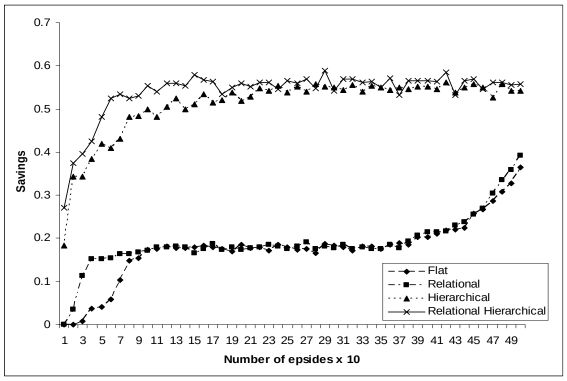


Figure 4.6: Learning curves of the different algorithms.

4.6 Related Work

Most of the decision-theoretic assistants have been formulated as POMDPs that are approximately solved offline. For instance, the COACH system helped people suffering from Dementia by giving them appropriate prompts as needed in their daily activities [10]. In this system, there is a single fixed goal of washing hands for the user. In *Electric Elves*, the assistant is used to reschedule a meeting should it appear that the user is likely to miss it [67]. These systems do not have a hierarchical goal structure for the user while in our system, the assistant infers the user's goal combinations and renders assistance.

Several plan recognition algorithms use a hierarchical structure for the user's plan. These systems would typically use a hierarchical HMM [25] or an abstract HMM [12] to track the user's plan. They unroll the HMMs to a DBN and perform inference to infer the user's plan. We follow a similar approach, but the key difference is that in our system, the user's goals are relational. Also, we allow for richer models and do not restrict the user's goal structure to be modeled by a HMM. We use the qualitative influence statements to model the prior over the user's goal stack. We observe that this could be considered as a method to incorporate richer user models inside the plan recognition systems. There has been substantial research in the area of user modeling. Systems that have been used for assistance in spreadsheets [35] and text editing [37] have used handcoded DBNs to infer about the user. Our system provides a natural way to incorporate user models into a decision-theoretic assistant framework.

In recent years, there have been several first-order probabilistic languages developed such as PRMs [27], BLPs [42], RBNs [38], MLNs [20] and many others. One of the main features of these languages is that they allow the domain expert to specify the prior knowledge in a succinct manner. These systems exploit the concept of parameter tying through the use of objects and relations. In this chapter, we showed that these systems can be exploited in decision-theoretic setting. We combined the hierarchical models typically used in reinforcement learning with the kinds of influence knowledge typically encoded in relational models to provide a strong bias on the user policies and accelerate learning.

4.7 Conclusions and Future Work

In this chapter, we proposed the incorporation of parameterized task hierarchies to capture the goal structure of a user in a decision-theoretic model of assistance. We used the relational models to specify the prior knowledge as relational hierarchies and as a means to provide informative priors. We evaluated our model against the non-hierarchical and non-relational versions of the model and established that combining both the hierarchies and relational models makes the assistant more useful.

The incorporation of hierarchies would enable the assistant to address several other problems in future. The most important one is the concept of parallel actions. Our current model assumes that the user and the assistant have interleaved actions and cannot act in parallel. Allowing parallel actions can be leveraged if the goal structure is hierarchical as the user can achieve a subgoal while the assistant can try to achieve another one. Yet another problem that could be handled due to the incorporation of hierarchies is the possibility of the user changing his goals midway during an episode. Finally, we can also imagine providing assistance to the user in the cases where he forgets to achieve a particular subgoal.

Chapter 5 – Logical Hierarchical Hidden Markov Models

5.1 Introduction

Activity recognition is a problem that has long been the focus of researchers and has a wide range of applications from surverillance to intelligent interfaces to assist the elderly. Accordingly, several kinds of solutions ranging from domain-specific hand-coded solutions to the more general Hidden Markov Models(HMM) have been proposed for this problem. Among these solutions, Hidden Markov Models and their several extensions are among the most popular methods for activity recognition.

The main advantage of HMMs and their extensions lies in the fact that they define a clear Bayesian semantics for the problem. Also, efficient algorithms that perform inference in linear time have been proposed for HMMs making them very attractive for this problem. The different extensions of HMM like the Hierarchical HMM (HHMM) [25] and abstract HMMs (AHMM) [12] are being widely used in activity recognition for a variety of applications.

The main drawback of HMMs is that they do not take into account the structure of the problem. The objects in the domain may be related by specific relationships and these relationships govern the action that the user performs in the current state. For example, the user is more likely to send a paper that he is writing to his co-authors and not to any random contact in his address book. Also, since the HMMs are inherently propositional, they do not allow for generalization among the objects of the domain. For instance, the models cannot be shared among multiple users of the desktop as a seperate HMM needs to be constructed for every user in a propositional setting. It would be prudent to think that most researchers follow the same pattern while submitting a proposal or most papers are being written by a similar methodology of running experiments, writing the paper, the abstract, send it to one's co-authors etc.

The goal of this work is to extend the HMM and its hierarchical version with a logical model that allows for generalization of the objects in the domain. The use of logical models will allow us to specify the models at an abstract level that can later be instantiated with specific instances and to perform inference on them. Also, the use of logical models will allow parameter sharing between the objects of the same type thus requiring smaller number of examples for learning.

In many real-world applications, the user chooses his action based on some conditions in the environment. For example, a user who goes for shopping might prefer the nearest store to shop from. If the product that he wants to buy is not available in that store, he might go to another one. If it was available, he might buy the product and return home. The fact that the product is available or not can be observed by looking at the inventory while the distance to the shop can be observed as well. But, it is not possible to observe what the user is thinking or what he wants to do. Hence, some parts of the state space are completely observed while some others are not. In our model, we consider the current state as having two components: an observable component (called the *world* state) and an unobservable component (called the *user state*). One of our critical assumptions is that the world state is completely observable and this would make it possible to perform tractable inference (as we show later in the chapter) in large domains.

In this chapter, we provide the syntax and semantics of the Logical HMMs and the Logical Hierarchical HMMs. We show that a ground HMM (correspondingly a ground HHMM) always exists given a Logical HMM (or a Logical Hierarchical HMM) and a set of values of the objects in the domain. We present the algorithm for grounding the Logical and the Logical Hierarchical HMM. We also present experimental results in a real-world intelligent desktop assistant and in a few synthetic domains.

One of the main motivations for this work is the goal estimation component of the

Relational Hierarchical Decision-Theoretic model of assistance presented in the previous chapter. As we had explained, the goal estimation is performed in the model using hand-coded DBNs. But as the number of objects in the domain increases, the inference problem will be intractable. We propose the use of Logical Hierarchical HMMs to perform the goal estimation in the assistance model. We have not yet integrated the LoHiHMMs with the assistance model. Hence in this chapter, we discuss LoHiHMMs as a seperate model in itself to perform activity (goal) recognition.

The rest of the chapter is organized as follows: the next section introduces the HMM, the hierarchical version of the HMM and previous work on logical HMMs. Section 3 presents the Logical HMM, its syntax and semantics and provides an algorithm for constructing a ground version of the Logical HMM. Section 4 repeats the same for the Logical Hierarchical HMM. Section 5 presents the experimental results on using the Logical HMM. The final section concludes the chapter by outlining some areas of future research.

5.2 Background

In this section, we briefly review the background work required for the chapter namely, the Hidden Markov Models, the Hierarchical HMMs, Particle Filtering and the previous work on Logical HMMs.

5.2.1 Hidden Markov Models

Hidden Markov models (HMMs) are used to model systems that follow a Markov process which is essentially a process with no memory. Specifically, in a markov process, the next state of the system is independent of the past states given its current state. HMMs are used to model stochastic markov processes. In HMMs, there are 2 kinds of variables: a *state* variable that follows a Markov chain and an *observation* variable whose value is generated by the current state. As the name implies, in a HMM the states are assumed to be unobserved(hidden). Hence, one of the important tasks is to infer the values of the current state conditioned on the values of the observations.

Formally, an HMM is defined using the 5-tuple $\langle S, \Pi, O, \Omega, I \rangle$, where S is a set of states, $\Pi(s'|s)$ is the next-state distribution, O is a set of observations,$\Omega(o|s)$ is the probability of observing o when in state s and I is the initial state distribution.

The HMM starts in one of the states s chosen according to $I(s)$, makes transitions to next states according to Π at every step, and emits observation o with probability $\Omega(o|s)$. It is clear that while considering the problem of activity recognition, the goal is to infer the distribution over the states given the current set of observations $p(s|o)$. Hence the activity recognition problem is posed as performing inference in an HMM.

HMM inference problems can be divided into four categories [53]:

- Filtering: Estimate the variables at current time-step given the observation history $P(S_t \mid O_{1:t})$
- Prediction: Estimate the variables in future (after c time-steps given the observation history $P(S_{t+c} \mid O_{1:t})$
- Smoothing: Obtain a better estimate of the state in the past given the observation history $P(S_{t-c} \mid O_{1:t})$
- Most Likely Explanation: Obtain the most likely state sequence that generated the sequence of observations $Argmax_{S_{1:t}} P(S_{1:t}|O_{1:t})$

In online settings, activity recognition is posed as the filtering problem which is to estimate the current state given the observation history. In this work, by inference we mean the filtering problem.

5.2.2 Hierarchical Hidden Markov Models

Although HMM seems to be a very good choice for activity recognition, there are some applications where it becomes difficult if not impossible to use HMMs. For example, a student researcher might email his/her advisor the current draft of the paper often during the paper submission. In this case, the same set of states of composing an email, attaching the document and sending it to the advisor is often visited in the trajectory. If we are to model this using a HMM, the number of states can be arbitrarily large as the states would have to capture the current subgoal, the higher level goals and the history of the completed subgoals. It is also difficult to model situations where some states take more than one time-step to be executed. For instance, editing a particular document might take several time-steps and it is not possible to fix the number of time-steps at the beginning of the task. Also, there are tasks that are performed recursively by the user. In addition, the user's taskes have a well-defined hierarhical structure that can be exploited if captured explicitly by the model. Though these situations can be captured by a HMM, the structure in the problem cannot be exploited efficiently.

Hierarchical HMMs[25] were developed for modeling such situations and they extend HMMs to include a hierarchical structure for the states. Each state in a HHMM is a self contained probabilistic model. One way to understand a Hierarchical HMM is to think of each state of the HHMM being an HHMM in itself. So when the HHMM transitions to the current state, the sub-HHMM is activated which in turn activates a HHMM at the lower level. This would mean that at each state, the HHMM will emit a sequence of observation symbols than a single symbol. The key difference to a normal HMM is the notion of *end* states. Each sub-HMM has a set of end states that can terminate the HMM at the current level and the control returns to the calling state of the parent HMM.

Any HHMM can be converted to a HMM[52]. The state of the HMMs correspond to the state stack of the HHMMs($S^{1:D}$), where D is the number of levels of the HHMM. If

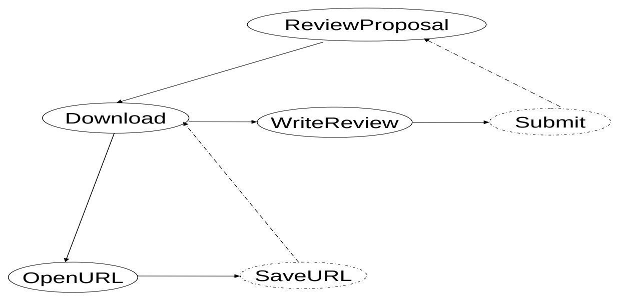


Figure 5.1: An example of a Hierarchical Hidden Markov Model.

the HHMM structure is a tree, it would imply that there are no repititive substructures. The HMM then contains one state for every combination of the parent and child states. If the HHMM has repeated substructures, they have to be duplicated in the HMM resulting in a large model. We refer to [52] and [25] for more details on converting a HHMM to a HMM.

An example of a HHMM is presented in Figure 5.1. The end states are presented by dashed ovals. As can be seen, to achieve the task of reviewing a proposal, the user has to download it, review it and submit the review. Once the review is submitted the control returns to the parent state. Similarly, to complete the task of downloading, the user needs to open the URL and save the proposal. Each level of the HHMM is a HHMM in itself with the property that each state of this HHMM can take more than one time-step to complete. There have been several extensions to the HHMM and it has been used extensively in activity recognition [57, 47]. We do not present these in detail in this chapter but point out the use of such models for activity recognition.

5.2.3 Particle Filtering

As we mentioned earlier, activity recognition has been formulated as the problem of filtering in the HMM (or the hierarchical HMM). Since in large HMMs, performing exact inference might be impractical, sampling methods became popular. The key innovation in these methods is to focus the set of samples on high probability regions of the state space i.e., generate more samples in high posterior regions. The main advantages of these approaches are:

- They are very easy to implement when compared to the exact methods. The data structures required are minimal when compared to exact algorithms like junction tree.
- These methods work on almost any distributions (including mixtures of distributions) while exact inference is almost impossible for continuous distributions
- They can handle variable-sized models and can easily adapt to the changes in the model structure
- No explicit unrolling is needed and in the presence of infinite samples these methods can give an exact distribution (asymptotic property)
- These methods make it possible to do a constant time update per time step

Though extremely popular, these methods are not without disadvantages. Some of the key disadvantages are:

- The sampling methods are slower than their other approximate counterparts (the deterministic methods)
- If there are not enough samples, these methods can have high variance

Among the sampling methods, *particle filtering* methods are very popular. The main idea in particle filtering is to represent the distribution as a set of samples of the state space. The general approach to particle filtering works as follows: Samples are generated according to the prior distribution and propagated according to the transition distribution. The samples are then re-weighed according to the observed evidence probabilities and new samples are generated.

Table 5.1: Particle filtering algorithm

```
function PF returns $\{x_t^i\}_{i=1}^N = PF(\{x_{t-1}^i\}_{i=1}^N, y_t)$
  1. For each sample $i$
     • Draw the sample $x_t^i$ using the distribution $x_t^i \sim P(x_t|x_{t-1}^i, y_t)$
     • Compute the weight of the current sample, $w^i \propto Pr(y_t|x_{t-1}^i)$
  2. Resampling
     • Resample the current state using, $x_t^i \sim \frac{w^i}{\sum_i w^i}$
     • Set the weights of the current sample, $w^i = 1$
```

The particle filtering algorithm is presented in Table 5.1. The state at time t is denoted by x_t and the observation at time step t is denoted by y_t. The main difference from the general approach mentioned earlier is that, the samples are drawn considering both the transition distribution and the observation distribution and the weights are the probability of the observation given the samples at the previous time-step. The distribution $P(x_t|x_{t-1}^i, y_t)$ is given by,

$$P(x_t|x_{t-1}^i, y_t) = \frac{P(y_t|x_t)P(x_t|x_{t-1})}{\sum_{x_t} P(y_t|x_t)P(x_t|x_{t-1})}$$

while,

$$P(y_t|x_{t-1}^i) = \sum_{x_t} P(y_t|x_t)P(x_t|x_{t-1})$$

The key thing to note is that the evidence (y_t) is being used while sampling for the new state. Recall that the most general approach uses the transition distribution for generating the samples and the observation distribution for generating the weights. This would mean that the observation (y_t) is not being considered during the sampling process leading to the requirement of more samples for converging on the true distribution. Using the observations for sampling would ensure that the number of samples required is small. Given the set of samples at any time-step, it is easy to obtain the distribution by counting. The main disadvantage of the particle filtering algorithm is that it performs poorly in high dimension spaces. This is due to the fact that the number of samples required to approximate the state distribution grows very rapidly with dimensionality. Hence, if the number of samples is not very large, this method could lead to high variance in the estimates.

5.2.4 Logical HMMs

Kersting et al. introduced Logical HMMs [41] as a method of combining ideas from ILP and dynamic models. In their framework, the states of the HMM are logical atoms (abstract states) rather than propositional states. A logical atom consists of a predicate name and a set of parameters that can be instantiated with constant ground values. The transition distribution in the logical HMM now consists of two kinds of distributions: an abstract transition distribution that specifies the probability of the next predicate given the current predicate and a selection distribution that specifies the probability of the instantiation of the parameters for the current state. In their work, the observations correspond to proof steps, e.g., the document that is currently being edited or the ls command issued at the command prompt. The selection distribution is specified using a Naive Bayes function. We contrast our formalism to theirs as we introduce our model.

5.3 Logical Hidden Markov Models

In this section, we present a logical extension to the Hidden Markov Model and outline the syntax and semantics of this model. The goal of this work is to formulate a hidden markov model that can be generalized across several objects in the domain.

In general, different users follow a similar pattern in achieving the same task. For instance, if the task is to review a proposal, the user would first open the document, print it, write the review and turn the review in. These tasks have to be performed irrespective of who the user is. Our goal is to parameterize the HMMs so that they can be generalized to several objects in the domain.

Consider the HMM presented in Figure 5.2. In this example, a user receives a document to edit through an email, edits the document and sends it back to the sender. The numbers on the arcs indicate the probability of the transition. Note the presence of logical conditions on the arcs. The semantics is that when the conditions are satisfied the HMM transitions to the next state with the given probability. For example, consider the node $Edit(D)$, which corresponds to the user editing a document D. If the user has not finished editing the document, he continues to edit it and is represented by the self-arc in the node. Once the user completes editing the document, he replies to the email in which he received the document.In this section, we first define the states and transitions of this HMM and then formally define the logical HMM.

The state in the model consists of two components, the state of the user and the state of the environment. The state of the user is assumed to be a single ground predicate that indicates the activity of the user. The user is restricted to be in only one state at any given instant of time. The state of the environment could be a conjunction of several predicates.

Definition 3 (State). *The state consists of two components: A user state which is a predicate that represents the current activity of the user and the environment state which*

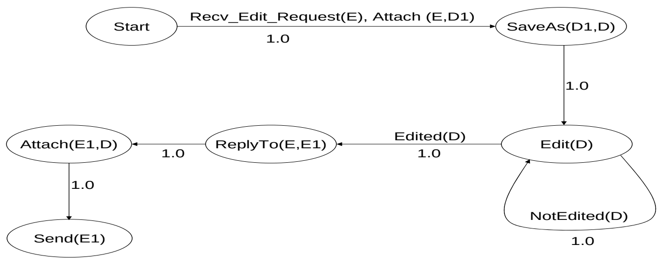


Figure 5.2: A Logical Hidden Markov Model for editing a document and sending it back to the sender. The logical conditions on the arcs indicate the conditions under which the transitions take place. The numbers on the arcs indicate the probability of the transition.

is a set of predicates of the environment.

An example of user state is $Edit(D)$, where $Edit$ is the logical predicate and D is the parameter (variable) associated with this state. The environment state could include information about the current time, the deadlines, the set of projects that the user is involved with and so on. Recall that the user state is a single predicate while the environment state is the conjunction of several predicates.

Given that we have defined the notion of state in our model, we need to define the transitions between the states. Our goal is to capture transitions between user states conditioned on some states of the environment. This will enable us to model the user's activity conditioned on the context of the execution. For example, if the document that the user is currently editing is a long document, the user might prefer to print the document, while if it is a short one, the user might just read it off the monitor. Though there are several ways of modeling the conditions in the transitions between the states, we use a model that is similar to decision lists or the case statements in programming languages.

The transitions in our model are called *logical transitions* and are of the form presented in Table 5.2. The transitions in our model define a transition between the current state of the user (called as *source* user state) and the next state of the user (called as *target*

Table 5.2: Structure of an Logical transition in our model. The current user state is $A(X,Y)$ and it transitions to one of the successor states based on the condition and logical transition probability.

$$A(X,Y) \mapsto \begin{cases} \text{if } C_1^1(X,Y) \wedge C_2^1(Y,Z)\text{... then} & \begin{cases} p_1^1: & B_1^1(X,Z) \\ p_2^1: & B_2^1(Y,Z) \end{cases} \\ \text{else if } C_1^2(X,Y) \wedge C_2^2(X,Y,Z)\text{... then} & \begin{cases} p_1^2: & B_1^2(X,Z) \\ p_2^2: & B_2^2(Y,Z) \end{cases} \\ \textit{else} & \ldots \end{cases}$$

user state) conditioned on some of the predicates of the environment. As shown in the figure, there could be several logical test predicates that can be conjunctions of several predicates (called hereafter as conditional predicates and shown as C in the figure) based on which the transition could take place.

The most important constraint is that exactly one branch of the transition will be always taken for the given states. Once a particular branch say k evaluates to true and the corresponding branch is taken, the target atom is chosen according to the transition distribution (called as logical transition distribution) $\vec{p^k}$ for the current branch k. $\vec{p^k}$ is a well defined distribution i.e, $\sum_{i \in S} p_i^k = 1$, where S is the set of target abstract states for the given logical transition. In the statement presented in table 5.2 for instance, $p_1^1 + p_2^1 = 1$.

An example transition is presented in Table 5.3. In this example, the user downloads a document P and if it is a long paper, the user prints the document with a probability 0.9 or reads it off the webpage with a probability 0.1. If the paper is not a long one, he reads of the webpage.

The branches can be understood similar to the decision lists in the programming languages. Given the instantiations of the current user state, the branch that first evaluates to true is taken. Once a branch is chosen, the logical transition distribution is used to choose the next user state predicate. In our model, the last branch has the key word *else*

Table 5.3: Example of an Logical transition. The user prints the document if it is a long document or reads it off the webpage if the URL is

$$Download(P) \mapsto \begin{cases} \text{if } LongPaper(P) \wedge URL(Z,P) \text{ then} & \begin{cases} 0.9: & Print(P) \\ 0.1: & Read(P,Z) \end{cases} \\ \text{else if } URL(Z,P) & \begin{cases} 1.0: & Read(P,Z) \end{cases} \\ \text{else} & \begin{cases} 1.0: & DoNothing \end{cases} \end{cases}$$

and has no conditions. It is a default one that will be taken if the other branch conditions fail to evaluate to true (for example see Table 5.3). This is to ensure that the conditions in atleast one branch would evaluate to true and hence one branch will always be taken for the given user state.

Logical Transition A logical transition consists of the current user state S as the source and a set of logical predicates of the environment. Corresponding to each predicate is a set of target user states and a corresponding set of probability distributions called the logical transition distributions. For each of the target user state, there is an associated instantiation distribution ($\vec{\mu}$) that specifies the probability of the values of the variables associated with the target state. We discuss the instantiation distribution later.

Definition 4 (Logical HMM)**.** *A Logical HMM consists of set of states S, a set of logical transitions Δ. The set of transitions define a markov chain at the logical level.*

Note that in the above definition, we ignore the observations which are predicates as well. In a later section, we discuss the observations. The above definition of the logical HMM is too general. If we assume that the logical predicates are unobserved along with the user states, the problem of user activity recognition would include performing inference over the environment states as well which is essentially a very complex problem.

Hence, we consider a restricted model in which we assume that the environment component of the state space is completely observed and the user state is assumed to be unobserved. This assumption is not unreasonable in many domains such as a desktop domain, video survillance etc where the effect of the user's actions in the environment is completely observable while the user's mental states are not known.

Definition 5 (Restricted Logical HMM)**.** *A Restricted Logical HMM consists of*

- *Set of states S which has two components: the user state S_u that is a single predicate and is unobserved and the environment state S_w that can possibly consist of many predicates and is completely observed*
- *a set of logical transitions Δ.*

Note that the assumption of the state being composed of the observable environment states and the unobserved user state predicate makes the inference process easier. This will enable us to ignore the changes to the environment predicates and just focus on the changes to the user state. The key thing to note here is that these logical conditions are the observable part of the state space. Hence given a current user state, it would be easy to evaluate the different logical predicates conditioned on the current user state.

Instantiation Distribution Once a user state is chosen as the target state for a transition, the instantiation of the user state must now be chosen. To this effect, the model of Kersting et.al specifies a distribution $\vec{\mu}$ (called as *instantiation distribution* henceforth) over the possible instantiations of each predicate. In our work, instead of directly specifying the distribution, we can specify some constraints. These constraints essentially serve to define the non-zero entries in the instantiation distribution and are part of the branch conditions. For example, the predicate $URL(Z, P)$ in the example, specifies the exact value of Z to be the URL of the document P. Yet another example of a hard constraint could be to specify that a paper can be sent only to the co-authors.

Though we use a uniform distribution for the instantiation distribution in our experiments, we do not make any assumptions regarding the nature of the distribution.

It is possible that the instantiation distribution can be specified as a function of some attributes of the environment. Consider an example from a resource gathering domain. Assume that the user has collected a particular resource say R. The user then has to store the resource in a storage say S. For instance, let R be *gold* while S be a *bank*. Since there could be many banks in the domain, the user could choose a bank based on the distance to the gold location. The instantiation distribution of S in this case is a function of the location of the bank.

Definition 6 (User Transition Disribution)**.** *User Transition Distribution defines the distribution of the next user state given the current state, i.e.,* $P(u_t|u_{t-1}, w_t)$ *where* w_t *is the description of the current world state,* u_{t-1} *is the previous user state and* u_t *is the current user state.*

Given the logical transitions of the form shown in Table 5.2, let $BP_i(X, Y)$ be the first branch predicate which evaluates to true in state $\langle u, w\rangle$. Let the current user state u be $A(x, y)$ and $CP_i^j(x, y, z)$ be true. Then $P(u' = B_i^j(x, z)|A(x, y), w) = p_i^j * \mu_i^j(w, x, y, z)$. In other words, since the i^{th} branch predicate is true, the corresponding branch is taken, and with probability p_i^j, $B_i^j(x, Z)$ is chosen and is initialized with a probability $\mu_i^j(w, x, y, z)$.

We can define the *World Transition Distribution* ($P(w_t \mid u_{t-1}, w_{t-1})$ analogously for each predicate in the environment state. However, since we do not address planning in our work, we do not need these definitions and do not define them formally.

Global Variables There could be some variables in the LoHMM whose value should not change as the HMM evolves. For instance, consider the example in Figure 5.2. The user, after editing the document, has to reply to the email in which he received the request to edit. This means that the value of the variable E once set to the value of the email that has the request should not be changed. In our model, some of the variables are declared as global and their values do not change once set.

Substitution Substitution in LoHMMs is defined exactly as in first-order logic. A substitution θ is defined as assignment of values to all the variables in the LoHMM. The set of variables is the union of all the variables in all the predicates of all the user states and the environment states. We denote the substitution of a set of predicates P as $P(\theta)$.

Ground HMM A ground HMM is a HMM constructed by substituting the values for all the variables of a Logical HMM. We refer the user state that has been substituded for its variablse as an instantiated user state. The state-space of this ground HMM consists of two components: the instantiated user states and the global list. The transition distribution specifies a distribution over the next ground user states and the values of the global variables.

Theorem 1 (Existence of a ground HMM)**.** *A ground HMM can be constructed for every Logical HMM M and a substitution θ.*

Proof. Let $u_t = U_t(\theta)$ denote the instantiated (ground) user state at current timestep t. Given u_t, it is clear that a single branch will be chosen in the logical transition corresponding to U_t. Once the logical conditions in a branch are satisified, the corresponding user transition distribution would define a coherent distribution over the possible next ground states. The probability of the next ground state u_{t+1} given that the branch i is true, is given by,

$$P(u_{t+1}) = p_i^j * \mu_i^j$$

where j is the next user state predicate. Since for every current ground user state, there is always a next user state due to the fact that atleast one branch always evaluates to true. Hence, for any given ground user state, there is a well-defined distribution over the possible next ground user states. Thus there exists a ground HMM for the logical HMM M and the substitution θ.

The motivation behind proving the existence of a HMM is that, since HMM defines a unique distribution over the trajectories, it follows that the LoHMM also defines a unique distribution over the trajectories. This would ensure that there is a well-defined model-theoretic semantics for the LoHMM. In a similar spirit, in the next section, we show that a ground Hierarchical HMM exists given a Logical Hierarchical HMM.

5.3.1 Summary

To summarize, in Logical HMMs,

- The state consists of the user state and the environment states
- The user state atoms do not have any functional symbols
- The transitions take place based on certain predicates of the environment
- The next user state predicate is chosen using the logical transition distribution
- The instantiation distribution is specified as hard constraints
- There is a global list of variables whose values once set do not change

We discuss about the observations later in the chapter. The Logical HMM defines a markov chain over the ground atoms when the variables are instantiated. In fact, HMMs are a special case of the Logical HMMs with: the states having an arity of zero, no conditional branches and no instantiation probability. Hence Logical HMMs generalize HMMs.

Let us consider the LoHMMs of Kersting et.al [41] and try to understand the relationship between the two models. In their model, there was an abstract transition distribution $\vec{p}$, which chooses the target predicate to transition to. In our model, first the logical conditions are evaluated, the corresponding branch is taken and then the target predicate is chosen according to the distribution $\vec{p^k}$. Since the logical conditions are mutually exclusive, only one branch can be chosen for any particular transition. Hence the two models behave similarly once the logical conditions are evaluated for the current instantiation.

Specific Instances One potential issue is that there could exist several transitions that can be matched with the current instance. For example, ignoring the conditions there could be 2 different abstract transitions defined as follows:

$$A(X,Y) \longrightarrow B(X,Z)$$
$$A(x,Y) \longrightarrow C(Y,Z)$$

The second transition is more specific when compared to the first transition. If we use a substitution $\{\theta = X/x, Y/y\}$, we could match the two rules and there is a question of which transition to choose. Kersting et.al, handle this by saying that the more specific transition is always preferred. In our work, we require that these 2 rules are ordered and combined to a single decision list as follows

$$A(X,Y) \begin{cases} \mathrm{I}f \quad (X = x) \quad 1.0 : C(Y,Z) \\ \\ \mathrm{e}lse \quad 1.0 : B(X,Z) \end{cases}$$

The idea is to encode the more specific instances as well in the transition. This is natural in our formalism as we always assume that only one branch can be true for any ground instantiation. This means that no two different logical transitions can exist with the same predicate name as its source.

5.4 LoHiHMMs

In many real-world situations, users solve difficult problems by decomposing them into a set of smaller ones. For example, proposal writing might involve writing the project description, preparing the budget, and then getting signatures from proper authorities. Thus the tasks to be completed by the user have a natural hierarchical structure. To capture this kind of knowledge using LoHMMs, we could make the state predicate include the states at all the levels of the hierarchy. This is not a very good option mainly due to a couple of reasons: first, this is not very elegant and hence it would be difficult for the domain expert to specify this kind of LoHMM; and second the inference process can be very difficult due to the explosive state space. A representation that could capture such a hierarchical structure cleanly can be used to perform efficient inference. The special structure can make the inference process easier. In this section, we incorporate hierarchies into LoHMMs. We term the user states as tasks for the purposes of the LoHiHMMs.

Recall that in a LoHMM, there are logical (horizontal) transitions between user states while in the Logical Hierarchical HMM (LoHiHMM) in addition to the horizontal transitions, there are vertical (task-subtask) transitions between parent and child tasks [1] and hence there is a necessity for an additional distribution that we call as the selection distribution to capture this transition. It should be mentioned that LoHiHMMs impose the hierarchical structure on the user's states and not on the environment states since our goal is to capture the user's task structure using hierarchies.

Thus, in the LoHiHMM, there are three kinds of distributions: in addition to the transition and the instantiation distributions of Logical HMMs there is a selection distribution which chooses the child user state of the parent user state. The execution semantics of the LoHiHMM is as follows: When control reaches the current user state, it chooses a child state to transition to based on the selection distribution. Once all the lower level

[1] We refer to the states of the LoHiHMM as tasks and subtasks in this section

HHMM terminates, the control reaches back to the current state and it chooses a horizontal transition as with the case of Logical HMMs. The main intuition is that at each level we have a LoHiHMM and the selection distribution chooses the LoHiHMM to execute. Each vertical transition (selection of a child) also has logical conditions that need to be evaluated to choose the child. LoHiHMMs are to LoHMMs what HHMMs are to HMMs. They can be viewed as either incorporating hierarchies to LoHMMs or adding logical models to Hierarchical HMMs.

5.4.1 Instantiation Probability and Global Variables

The *Instantiation Distribution* μ is similar to the LoHMMs in that they are constrained by the hard constraints in the transitions. But, in a hierarchical setting, there is another important issue: *variable binding.* The variable instantiation of the parent state should stay the same in all transitions made by its child states, i.e., the variables of the parent state should be "global". Their values cannot change once instantiated until that state finishes its execution. Hence we require that during instantiation, if the variable is in the global list, the instantiation distribution would retain its value.

5.4.2 Selection Probability

The selection probability specifies the probability distribution over the child tasks given the parent tasks. We call the transition between a parent task and a child task as a *selection transition* and the syntax is presented in Table 5.4. The syntax is similar to the LoHMM syntax except that we now use special keywords namely, *parent* and *child* to identify the special kind of rule. The rule that is presented in table 5.4 is interpreted as: *"If the current user task is $A(X,Y)$, then it chooses one of the possible next child tasks by evaluating the branch predicates (shown as C_i^j in the rule) and then chooses the child*

Table 5.4: The syntax of the selection transition statements. The keywords *parent* and *child* denote the parent and child task respectively. The other parts of the statements are similar to LoHMM

$$parent:\ A(X,Y) \mapsto \begin{cases} \text{if } C_1^1(X,Y) \wedge C_2^1(X,Y,Z) \text{... then} & \begin{cases} sp_1^1 : child: & B(X,Z) \\ sp_2^1 : child: & C(Y,Z) \end{cases} \\ ... & \\ else & ... \end{cases}$$

user task based on the selection distribution". This rule is similar to the transition rule in LoHMMs except that in this case, it defines a task-subtask transition of a HMM. The branch predicates are handle in a manner similar to a decision list such that the first satisfied branch is always taken. It is important to mention that the selection probability is well defined, i.e., the sum of the probabilities of the different child abstract tasks for a given branch is 1 ($\sum_i sp_i^j = 1$). Note the presence of a default branch that is chosen if all the other branch conditions fail to evaluate to true.

5.4.3 Transition Probability

Horizontal transition is more complicated in LoHiHMMs than in LoHMMs, due to issues with variable binding not present in the latter. Since we have used global lists and modified the instantiation distribution to accomodate for the global variables, we now proceed to define the horizontal transitions and the associated probabilities. The transitions are similar to the LoHMMs with one difference: the transitions must include the task of the parent. Hence, the horizontal transition is conditioned on the instantiated value of the parent tasks.

The syntax of the horizontal transition is presented in Table 5.5. The statement is similar to the ones presented for the LoHMMs except that these statements include the

Table 5.5: The syntax of the horizontal transition statements. The keywords *pa*, *ch* and *nch* denote the parent the current child and the next child abstract tasks respectively. The other parts of the statements are similar to LoHMM

$$pa:\ A(X,Y);\ ch:\ B(X,Z) \mapsto \begin{cases} \text{if } C_1^1(X,Y,Z) \wedge C_2^1(X,Y,Z)\text{... then} & \begin{cases} p_1^1 : nch : C(X,Y,Z,W) \\ p_2^1 : nch : D(X,Y,Z,W) \end{cases} \\ \text{...} & \\ else & \text{...} \end{cases}$$

parent tasks (shown explicitly using the keyword *pa*). These statements can be interpreted as follows: *"If the current parent task is A(X,Y) and the current child task is B(X,Z), then it chooses one of the possible logical transitions by evaluating the branch predicates (shown as C_i^j in the rule) and chooses the abstract child task based on the transition distribution"*. For instance, the next child could be of the form $C(x,y,z,w)$ with probability $p_1^1.\mu(w)$, where p_1^1 is the abstract transition probability for $C(X,Y,Z,W)$ and $\mu(w)$ is the instantiation probability for w. Note that the variables in the parent (X,Y) will be "global", since its binding will stay the same until the current parent terminates.

As with the selection transition of LoHiHMM, the set of branch predicates for a particular parent-child combination transition rule is part of a decision-list so that the first branch that evaluates to true is taken. We could imagine adding hard constraints on the instantiation distribution in the conditions of the branches. Similar to the LoHMMs, the transition distribution is well defined i.e., $\sum_i p_i^j = 1$.

5.4.4 Ending Probability

If the LoHMM at a lower level terminates at some tasks, the control would return to the parent task. For example, it is possible that the user might decide to take a break from running the experiments and might instead decide to work on the abstract while at other times, he might want to complete running the experiments. It is also possible that the current task might terminate in more than one task.

Hence there is a necessity to define a distribution over the possible end tasks at a particular level. We specify the ending probability is a function β from the set of ground tasks (at any given level) to $[0, 1]$.

Theorem 2 (Existence of a ground HHMM)**.** *A ground HHMM can be constructed for every LoHiHMM M and a substitution θ for the predicates of M.*

Proof Sketch. The proof follows from Theorem 1. Here, we provide a high-level intuition for the proof. Recall that when the LoHiHMM execution reaches the current state, the state chooses a child state to transition to. Once the control returns to the current state, it chooses to make a horizontal transition. Since at every time-step, there is only one transition (either horizontal or vertical) is chosen due to the substitution there exists always a single next state distribution. If the next state is chosen using a horizontal transition, the probability of the next state is the product of the transition and instantiation distributions. This is to say that

$$p(u_{t+1}) = p_i^j * \mu_i^j$$

where u_{t+1} is the user state at the next time-step, i is the current branch chosen in the horizontal transition and j is the predicate that is being chosen. If the next state is chosen using a vertical transition, the probability is the product of the selection and instantiation distributions, i.e.,

$$P(u_{t+1}) = sp_i^j * \mu_i^j$$

where u_{t+1} is the user state at the next time-step, i is the current branch chosen in the vertical transition and j is the predicate that is being chosen. Also, the HHMM at the current level can choose to terminate based on the ending probability. This shows that there exists an unique HHMM for every LoHiHMM whose states consists of the state task and the global value list.

5.5 Inference

In this section, we outline a particle filtering algorithm for performing inference in a LoHMM. We then show how to extend this algorithm to the case with hierarchies. It is not practical to perform exact inference over this model due to the fact the ground HMM generated from this model can have potentially a very large number of states and hence the transition and the observation CPTs can be very large.

There are a few ways by which we can perform efficient inference: the first two methods are based on the observation that the transition and observation matrices are very sparse and that some of the states are not possible (for example, a state in which a global variable takes a value different from that of the global list). The third method is to avoid explicit unrolling and try to use particle filtering method using an oracle that can generate a sampled next state given the current state and then use the instantiation distribution over the possible values of the variables. We first briefly present the two methods that build a ground HMM but exploit the sparsity and then explain the particle filter with logical querying in detail.

In the first method, while performing inference, we can maintain a distribution over all the next states whose probability is greater than some ϵ. Let us call this set S' and the entire state space as S. The idea is to treat all the other elements of the state space (i.e., the elements of the set $\langle S - S' \rangle$) as having zero probability of transition. So in this case, we need to keep only a small number of states for the transition. This would render it possible for us to perform exact inference on the model. Note that this is slightly different from keeping track of the non-zero probability elements alone as some non-zero probability elements that are less than ϵ are ignored. Siddiqui and Moore [63] use a similar approach. But, they keep track of top k elements of the transition distribution and treat the rest as having constant transition probability.

The second method is the use of the well-known decision-tree representation of the CPTs. More specifically, it is well-known that a tree-structured CPT can represent the context-specific independence between the causes of a particular target variable. It is possible that we could use a tree-structured representation for the CPTs where at each internal node a logical condition is evaluated. Such a representation would be able to handle very large structured HMMs as the tree stores only the non-zero elements. If needed, we could approximate the CPTs by storing only the entries of the CPT whose values are greater than ϵ. This would further reduce the size of CPTs making inference on the ground HMMs practical.

5.5.1 Particle Filter with Logical querying

Particle filter algorithm was introduced earlier. To recall, the particle filter samples the next state based on the evidence and computes the weight of the samples. Then it resamples the states based on the weights of the samples. We now show that this algorithm could be used for the LoHMMs as well. The main bottleneck is computing the summation over all the next states (x_t) in the equation,

$$P(x_t|x_{t-1}^i, y_t) = \frac{P(y_t|x_t)P(x_t|x_{t-1})}{\sum_{x_t} P(y_t|x_t)P(x_t|x_{t-1})}$$

In a logical model, this is a critical issue as the number of possible ground states can be very large. Hence the number of particles needed can be very large. Also, it might not even be possible to enumerate all the states for the summation to be computed. For instance, in a desktop there could be a very large number of files and emails. It is impractical to enumerate all possible combinations of files that can be attached to an email.

However there are only a small number of documents that are possible candidates at any time and hence we probably need to consider only a few of those files. This would correspond to only a few non-zero entries in the transition matrix of the ground HMM. As pointed out earlier, a good representation of the sparse matrices might help the cause. But even for a sparse matrix representation, we need to construct a ground HMM which might not be possible in several situations.

Hence in this work, we propose to use a method that avoids enumeration of the complete state space. In our model, given the value of the previous sample and the values of the elements in the global list, a selection procedure would return the set of possible next states. The hypothesis is that the conditions and the constraints in the abstract transitions would greatly reduce the set of possible next states. We verify this hypothesis empirically.

Given, a particular ground state and the values of the global list, the selection procedure would first determine the set of logical transitions that have the current user state predicate as the head and obtain the set of the possible next user state predicates based on the logical conditions. For each of the next predicates, possible instantiations are considered using the instantiation distribution. The probability of the next ground state is the product of the selection probability and the instantiation probability. In some cases, if we are only interested in the current user state predicate (for instance, whether the user is composing an email or editing a document), we ignore the instantiation distribution and sample only from the selection distribution.

For example, consider the example presented in Figure 5.2. Let as assume that the user just completed editing the document say $d1$. So when the particle filter queries for possible next states, the condition $Edited(d1)$ would be satisfied. Since the email E is in the global list, the selection procedure would just return the next state as replying to email E. This would greatly reduce the computation that is otherwise needed to sum over all the states.

Our inference procedure can be understood as a lazy evaluation procedure where the HMM is constructed on the fly based on the values that have been assigned to the variables and the conditions that are satisfied. The main goal of this procedure is to avoid the explicit construction of a ground HMM. This is because as we have pointed out earlier, the number of objects in the real-world is very large which in turn leads to huge transition and observation matrices. But considering the objects lazily based on the current state and the values of the variables, we are able to perform effective inference in real time. We demonstrate this using experiments.

Infinite variables: In many relational domains, there exist variables whose set of possible values cannot be specified in advance. For example, it is not possible to predict the fileName that the user is going to use to save the current document as the number of file names can be very large. It is clear that in these cases the summation over the possible next states cannot be performed. There are several possible ways of handling this situation. One simple and naive solution is to only consider the predicates and ignore the variables. A second method would be to use an idea similar to that of Hierarchical Dirichlet process mixtures [6] for infinite HMMs. In this case, we can maintain the set of values that have been encountered explicitly and use a prior distribution over to model the possibility of the next state being from this set or a new state. This distribution can slowly decay the probability of choosing a new value. This would mean that initially there is a high probability of seeing a new value and this probability decreases with time.

In our work, we explore a simpler solution for this problem. Our assumption is that at some point in the trajectory, the observation would exactly specify the value of the infinite variable. Till such observation, we treat the infinite variable as unknown and sample only the predicates and the finite variables. This assumption is not unreasonable in many domains. For instance, in an intelligent desktop assistant, at some point of execution, we can directly observe the filename that the user uses to save the current document. Similarly, we can observe the subject of an email once the user has composed the email. The intuition is that the uncertainty over the infinite variables need not be modeled as they would be removed by the user himself fairly early in the trajectory. It is an interesting future direction to use the dirichlet process idea in the particle filter.

5.5.2 Observations

So far in the discussion, we have not explained clearly how the observations are handled in the inference. It should be mentioned that the observations can be easily handled in the LoHMM framework. Associated with each state predicate we can define a set of observation predicates and an associated observation distribution. The observations in our model are similar to the state predicates. The observation consists of a predicate name and a set of variables. The observation may or may not consist of the same variables that are present in the state predicate. While performing inference, the algorithm would check for the observation variables in the state predicate. If the observation variable is present in the state predicate the oracle would set the value of the corresponding variable in the sample to the same value. If not, it considers the possible set of values for the state predicate and returns them. While computing the next samples and the weights, the observation distribution is used. This is to say that $P(y_t|x_t) = 1$, when the arguments of the observation and the predicates match and is 0, when they do not. Hence when sampling, the values of the state predicates are chosen so that they take the same value as that of the observation and the other samples are rejected.

5.6 Experiments

In this section we present some experiments using the Logical HMMs. Experiments were conducted in two domains: an intelligent desktop domain, where the goal is to track the user's progress in receiving a document, editing it and sending it back to the sender and a synthetic domain that would allow for testing several scenarios. It has to be mentioned that in our experiments, we use uniform distribution for instantiation and do not update the parameters after a trajectory. All the experiments are performed in an offline mode.

5.6.1 CALO Domain Experiment

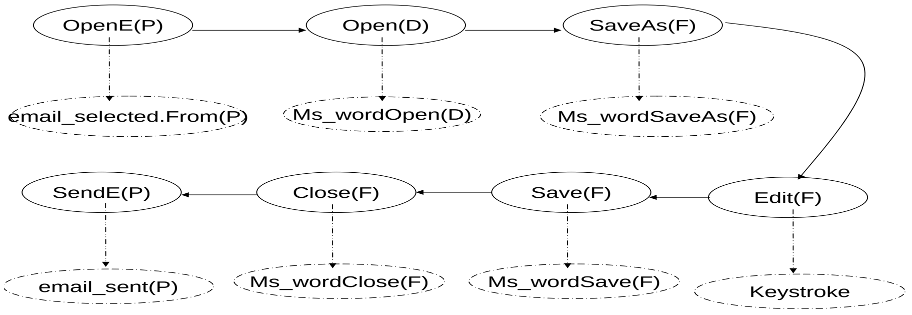


Figure 5.3: The HMM for the CALO scenario. The user receives a document, saves it as a file, edits it and sends it back to the sender

One of the goals of the Logical HMM is to use it in real-time desktop assistants (ex.CALO [13]) to track the user's progress towards his goal. As a first step, we considered a scenario where the user receives a request to edit a document from a person say P, saves the document as another file, edits it and sends it back to P. The Logical HMM for this is presented in Figure 5.3. We present the example graphically.

The goal of this experiment is to track the user's state as he performs the task in the scenario. In Figure 5.3, the dashed ovals indicate the observations for CALO. The problem then is to infer the abstract state of the user and the value of the parameters given the observations. We use the particle filter with selection procedure for this purpose. Currently, the experiment is performed in a batch mode (offline). So the trajectories are

Figure 5.4: Results of the inference in the CALO domain.

```
Time step = 1
Current Predicate OpenEmail probability = 1.0
values = 2 Probability = 1.0

Time step = 2
Current Predicate Open probability = 1.0
values = 0 Probability = 1.0

Time step = 3
Current Predicate SaveAs probability = 1.0
values = 1 Probability = 1.0

Time step = 4
Current Predicate Save probability = 1.0
values = 1 Probability = 1.0

Time step = 5
Current Predicate Close probability = 1.0
values = 1 Probability = 1.0

Time step = 6
Current Predicate SendE probability = 1.0
values = 1 Probability = 1.0
```

collected as the user performs a task and the corresponding events are extracted and given as input to the inference engine. The results are presented in Figure 5.6.1. As can been seen, the predictions include the distribution over the user state predicates at the current time-step and the distribution over the instantiations at the current time-step. The output is a distribution over the predicate names at the current state and a distribution over the values of the variables conditioned on the predicate names. Only the non-zero probabilities are displayed.

In the above scenario, the user dutifully follows the workflow and performs the task without any interruptions. But in real-world, the user rarely works without interruptions. In particular, while working with a desktop, there are several sources of interruptions including but not restricted to emails, phone calls, breaks etc. Our LoHMM in the purest form cannot handle the interruptions. In order to consider them, we introduce the notion of background state. Basically, we explicitly model the fact that the user's current task might be in the background when the user is working on a task that is different from the current task (for example attending a phone call). We assume that there is a small probability that the background state is possible at every time-step and if the observations are not consistent with the current model, the distribution skews towards the background state. An example output is given in Figure 5.6.1. In this example, the user is performing some other tasks in timesteps 2 and 5. The LoHMM inference engine retains the samples of the states from the previous time step (1 and 4) and uses them to predict the states at time steps 3 and 6.

In this scenario, there is a need for global list to retain the value of the person from whom the email was received (P). Also, the name of the file that user saves the current document can possibly take infinitely many values and hence we do not make predictions about the name of the file until it is actually observed. It is not clear how to represent this scenario as a ground HMM due to the fact that the file name can take infinitely possible values. Currently, this inference engine is being integrated with CALO to perform activity recognition and tested on more sophisticated workflow models.

5.6.2 Synthetic experiment

The real world dataset showed that the LoHMM inference engine was able to perform well in the realworld domains in the presence of large data. Although the problem is interesting, the observation model exactly provided information about the states and hence the probability of the true state was always 1. Hence we constructed a few synthetic experiments where the probability of the true state of the user cannot be determined easily. Also, the use of synthetic data can make it possible for us to change the settings in order to rigorously test our algorithms. Yet another reason for the synthetic data is the fact that the correct state of the user can be easily obtained as against a real world domain where it is harder to exactly determine whether the prediction is correct or not. In this section, we present the synthetic experiments and their results.

In the first experiment, there were 4 predicates and probabilistic transitions between them. For instance, there were transitions of the form, $A(X,Y)$ to $C(Y,Z)$ and $D(Z)$ with probabilities 0.4 and 0.6 respectively. In addition both C and D generate the same observation. The goal of this experiment is thus to evaluate the LoHMM in presence of stochastic transitions and shared observations between predicates. There were 2 kinds of scenarios created in this experiment. One in which the user completes the trajectory without any interruptions and the second in which the user has a few interruptions while executing the task.

The results are presented in Figure 5.6 as $Exp1$ and $Exp2$. $Exp1$ does not have any interruptions while $Exp2$ has a few interruptions. On the $y-axis$, we have presented the mean reciprocal rank(MRR) of the "true"' state of the user. MRR was defined earlier in chapter 3. To obtain the MRR, we rank the states by their probabilities and obtain the inverse of the rank of the true state. This is then averaged over a few runs. In our experiments, we used about 30 runs for each scenario. It is easy to see that higher the MRR (closer to 1) better is the performance.

It can be observed from the graph that in both cases, eventually the algorithm converges on the true state (MRR = 1). In the second case with many background states, the algorithm takes a few more time steps to converge to the true state. If we are to model this as a ground HMM and obtain a similar performance, we would need about 540 states (as we need about 45 states and the global list can take about 12 values). This would mean that the ground transition matrix can take 540^2 entries and this requires the use of the techniques that we discussed earlier.

The second domain is a synthetic grid world domain. Here, the goal of the user is to pick up a resource (gold,wood or food) and deposit in the corresponding storage location. The key thing to note in this domain is that the resource that is picked up is not directly observed. The resource must be in inferred by the path that is taken to the storage and by the type of storage location. For instance, when the user goes to the bank it is highly probable that he has collected gold. The goal of this experiment is to test whether the particle filter can track the user's state correctly under the circumstances when the observations do not directly provide information about the state.

The results of this experiment without and with the background states are presented as $Exp3$ and $Exp4$ respectively in Figure 5.7. Similar to the earlier experiment, the results are averaged over 30 runs. As can be seen from the figure, in this setup, the LoHMM takes longer to converge on the true state of the user as the uncertainty persists for longer time in this domain. Ultimately the LoHMM is able to infer the true state of the user. Similar to the earlier experiment, if we need to represent this domain using a ground HMM, the ground HMM will have around 600 states making the process of performing inference extremely tedious.

5.7 Discussion and Conclusion

Zettlemoyer et al. [70] consider the problem of filtering in relational HMMs and propose the idea of logical particle filter for this case. Recall that in propositional PF, each particle is a sample of the current state. In their method, each particle is a logical formula that specifies a set of states. In their model, the states are conjunction of propositions and/or functions. The observations and the actions are propositions (i.e., unparameterized). The basic idea is to construct a set of partitions (hypotheses) where each of the partitions represents a set of states such that every state in a hypothesis has the same transition probability. The hypotheses at the current time step are split into a set of mutually exclusive ones, the transition probability is then applied and the observations are used to specialize the hypothesis. The expectation over states and hypothesis is decomposed to $E_{hyp}[E_{state}(f)]$. The inner expectation is computed exactly while the outer expectation is sampled. Since all the states in the hypothesis are considered to have the same transition probability, the inner expecation can be computed easily. The main bottleneck lies in the construction of these hypothesis. It is not clear how to construct these mutually exclusive hypothesis in the presence of function symbols for the state space. We sacrifice expressiveness for efficiency in our work and consider only predicate symbols for the state space, hence making the particle filter simpler.

In this chapter, we motivated the need for the logical representations of HMMs for performing activity recognition. The models that are used and learned by these LoHMMs can be generalized across several users. The models allow for conditional branching where the branching between states takes place based on certain conditions of the attributes of the environment. We then extended this model to the case where the user's tasks have a hierarchical structure. We also presented a particle filter algorithm to perform efficient inference on these models. We presented the need for handling variables that can take infinite values and outlined a simple lazy technique for handling this case. We evaluated the LoHMM on two environments and showed that this model can track the user's goal effectively. It is clear that using a propositional HMM will not be feasible in domains with a large number of objects. The current LoHMM model has been integrated into CALO and several tests are being conducted currently. The LoHMM model is proposed to be used for activity recognition in CALO replacing the HMM model that is currently being employed.

One of the possible future directions is to integrate the LoHiHMMs with Relational Hierarchical Decision-Theoretic model of assistance for the goal estimation. The idea is to use LoHMMs in the goal recogniser component of the model. There is also a need to evaluate the LoHiHMM model in real world applications such as CALO. Yet another possible direction could be to construct a model that given a workflow of the user can convert it into a LoHiHMM and can perform inference given observations. There is also the need for investigating techniques for inference in the LoHiHMM that can exploit both the logical and hierarchical structure of the HMM. Another open problem is to research better solutions to handling variables that can take infinite values. This can possibly be done by the use of hierarchical dirichlet processes but the approach and the solution need to be well formalized.

Figure 5.5: Results of the inference in the CALO domain with interruptions. The user is interrupted in time-steps 2 and 5 and is performing some other task.

```
Time step = 1
Current Predicate OpenEmail probability = 1.0
values = 2 Probability = 1.0

Time step = 2 Background State
Current Predicate OpenEmail probability = 1.0
values = 2 Probability = 1.0

Time step = 3
Current Predicate Open probability = 1.0
values = 0 Probability = 1.0

Time step = 4
Current Predicate SaveAs probability = 1.0
values = 1 Probability = 1.0

Time step = 5 Background State
Current Predicate SaveAs probability = 1.0
values = 1 Probability = 1.0

Time step = 6
Current Predicate Save probability = 1.0
values = 1 Probability = 1.0

Time step = 7
Current Predicate Close probability = 1.0
values = 1 Probability = 1.0

Time step = 8
Current Predicate SendE probability = 1.0
values = 0 Probability = 1.0
```

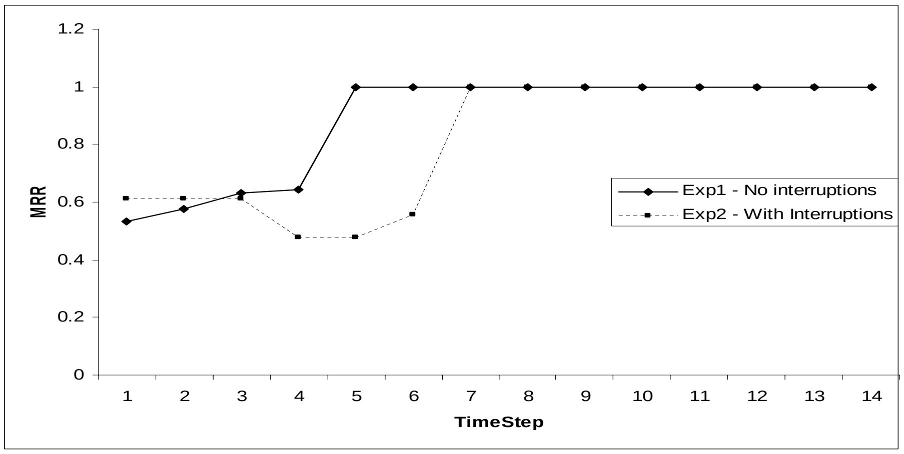


Figure 5.6: Results of the LoHMM inference on synthetic experiments. Experiment 1 does not have any interruptions, while experiment 2 has interruptions.

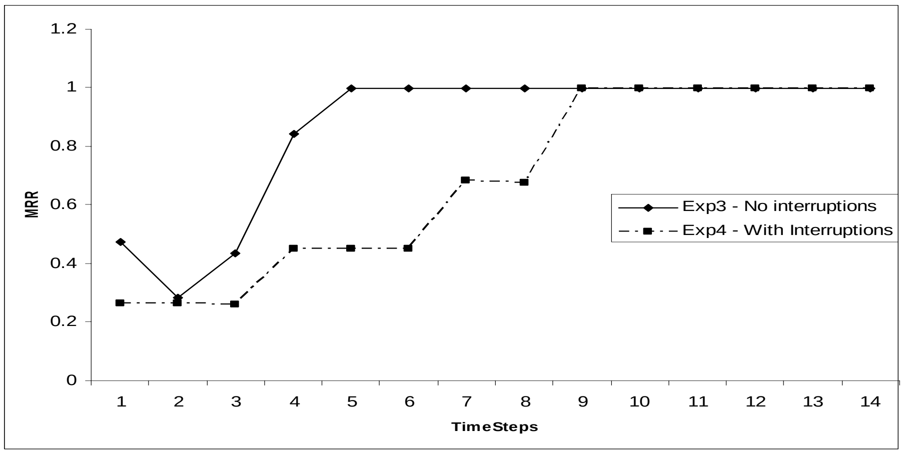


Figure 5.7: Results of the LoHMM inference on second synthetic experiment. Experiment 3 does not have any interruptions, while experiment 4 has interruptions.

Chapter 6 – Conclusions and Future Work

6.1 Contributions of thesis

In this work, we initially introduced a decision-theoretic model of assistance that could be used in several assistant systems. We modeled the assistant as a POMDP to select the assistive actions. We presented an approximate solution to the POMDP and outlined two myopic heuristics to solve the POMDP. We evaluated the model on three domains: two game-like environments and a real world folder predictor domain. Our evaluation using human subjects in two game-like domains show that the approach can significantly help the user. We also demonstrated in a real world folder predictor that the decision-theoretic framework was more effective than the state of the art techniques for folder prediction. It should be mentioned that this framework can be naturally extended to the case where the environment is partially observable to the agent and/or to the assistant. This requires recognizing actions taken to gather information, e.g., opening the fridge to decide what to make based on what is available. Also, our method can handle cases where there are varying cost functions for the user and the assistant.

We also presented algorithms based on classic algorithm schemes such as gradient descent and EM to learn the parameters of the relational influence statements.We presented a language for representing the relational influence statements and represented several formalisms using this language, thus establishing that the learning methods are applicable to several formalisms that employ the use of combining rules. In our work, we used two levels of combining rules with *mean* as the combining rule at one level and one of *Noisy-OR* or *weighted mean* as the combining rule at the second level. Our experiments indicate that the algorithms based on the relational models perform much better than the propositional algorithms. It would be interesting to extend this work to search over a set of combining rules to determine the one that best fits the data.

We then presented the relational hierarchical model of assistance which in essence was the incorporation of parameterized task hierarchies to capture the goal structure of a user in a decision-theoretic model of assistance. In this work, the relational hierarchical models served as prior knowledge about the user's goal structure and placed hard constraints on the user's policy. The relational models were also used as informative priors for the user's goal distribution. The models were evaluated against several versions of the decision-theoretic model in 2 domains. The results indicate that the use of relational hierarchical model made the assistance very effective in the first few episodes where the other models had to learn the user's policy. These results reinforce the fact that the relational hierarchies serve as a strong bias on the user policy that enables the assistant to quickly adapt to the user.

This model combines the strengths of three different approaches: hierarchies for plan recognition, decision-theory for assistance and relational models for succinct representation of knowledge. This model is very general (due to the use of POMDPs) and can be used in several domains. Also POMDPs allow us to handle uncertainty and varying costs in the domain. The use of relational models enables this model to be generalized across several users and objects in the domain. Hierarchies allow us to realistically model the goal structure of the user and perform efficient inference of the user's goals.

Finally, we presented a model for performing efficient goal estimation/activity recognition. This model can be understood as extending the standard Hidden Markov Models to logical and hierarchical settings. We presented the syntax and semantics of the logical HMMs and the hierarchical extension (LoHiHMM). We also presented an inference algorithm based on particle filtering that would avoid the complete enumeration of the state space. The major advantage of this model is that being a relational model, it can exploit parameter tying that would enable easier learning of parameters. We evaluated the model in a real-time assistant (CALO) and presented the results for the same. We also tested the model on a few synthetic domains and the results indicate that the model is able to track the user's progress towards his goals effectively. This model is currently integrated with the CALO desktop assistant and further tests are being performed on this model.

6.2 Future work

Currently, the user policy is being tracked in the relational hierarchical model of assistance using a hand-coded DBN. It is easy to imagine that in large domains, the number of variables would render the inference in DBN intractable. In such cases, the CPTs tend to be very large and sparse and cannot be handled efficiently. One of the future directions is to use the LoHiHMMs to perform inference over the user's goal stack. This would avoid the explicit unrolling of the DBN and handcoding of the parameters. Also, it would be an interesting research direction to extend the LoHiHMMs to handle variables with infinitely many values. For example, the set of file names cannot be possibly restricted. But the assistant should still be able to render effective assistance if it can make some reasonable assumptions about the file names such as considering that the file name could contain the project name and/or the version number. One possible future direction lies in exploring the use of relational models for activity recognition in personal assistant systems. The LoHiHMMs would be a way of specifying the abstractions in the decision-theoretic model. Also, currently the parameters of the LoHiHMM are not learned from data. The real power of the relational models can be exploited while learning the parameters. Hence, yet another research direction would be to develop algorithms for parameter learning of the LoHiHMMs.

User modeling has been explored extensively and Bayesian user models have been recently used in several domains such as medical, intelligent desktop etc. Statistical relational learning provides a natural way of generalizing across objects in the domain. One of the research problems is to explore the use of statistical relational learning in developing sophisticated user models that can be generalized. For instance, given a workflow of an user, an intelligent desktop assistant could build a user model that can be generalized across all the users who use the desktop.

Another possible direction would be in making the assistant useful even in abnormal situations such as user forgetting to achieve a subgoal (for example, forgetting to attach a document to an email), or change his goal mid-way (for example, when a more urgent task comes at hand) or terminating a task abruptly. To achieve this, we need to exploit the research in other areas to incorporate a more sophisticated model of the user to perform effective assistance. For instance, there has been extensive research on why humans forget in the cognitive science literature and several reasons have been provided. It is interesting to model these reasons explicitly to predict when the user forgets the goal. The use of statistical relational models for user modeling will help us generalize the models across different users.

Yet another possible direction for future research includes formalizing the relationship between the aggregators and combining rules. Also, it is important to extend our learning algorithms to more general classes of combining rules and aggregators including tree-structured CPTs and noisy versions of other symmetric functions. Finally, we would like to develop more compelling applications in knowledge-rich and structured domains that can benefit from the richness of the first-order probabilistic languages.

Bibliography

[1] Eric E. Altendorf, Angelo C. Restificar, and Thomas G. Dietterich. Learning from sparse data by exploiting monotonicity constraints. In *Proceedings of UAI 05*, 2005.

[2] J. L. Ambite, G. Barish, C. A. Knoblock, M. Muslea, J. Oh, and S. Minton. Getting from here to there: Interactive planning and agent execution for optimizing travel. In *IAAI*, pages 862–869, 2002.

[3] D. Andre and S. J. Russell. State abstraction for programmable reinforcement learning agents. In *Eighteenth national conference on Artificial intelligence*, pages 119–125, 2002.

[4] C. G. Atkeson and S. Schaal. Learning tasks from a single demonstration. *IEEE Transactions on Robotics and Automation*, pages 1706–1712, 1997.

[5] X. Bao, J. L. Herlocker, and T. G. Dietterich. Fewer clicks and less frustration: reducing the cost of reaching the right folder. In *In Proceedings of IUI*, pages 178–185, 2006.

[6] M. J. Beal, Z. Ghahramani, and C. Edward Rasmussen. The infinite hidden markov model. In *NIPS*, pages 577–584, 2001.

[7] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, 1996.

[8] John Binder, Daphne Koller, Stuart Russell, and Keiji Kanazawa. Adaptive Probabilistic networks with hidden variables. *Machine Learning*, 29(2-3):213–244, 1997.

[9] N. Blaylock and J. F. Allen. Statistical goal parameter recognition. In *ICAPS*, 2004.

[10] J. Boger, P. Poupart, J. Hoey, C. Boutilier, G. Fernie, and A. Mihailidis. A decision-theoretic approach to task assistance for persons with dementia. In *IJCAI*, 2005.

[11] C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *JAIR*, 11:1–94, 1999.

[12] H. Bui, S. Venkatesh, and G. West. Policy recognition in the abstract hidden markov models. *JAIR*, 17, 2002.

[13] CALO. Cognitive agent that learns and organizes, http://calo.sri.com., 2003.

[14] A. R. Cassandra. *Exact and approximate algorithms for partially observable markov decision processes.* PhD thesis, 1998.

[15] J. A. Clouse and P. E. Utgoff. A teaching method for reinforcement learning. In *Proceedings of the Ninth International Workshop on Machine Learning*, pages 92–110, 1992.

[16] W. W. Cohen, V. R. Carvalho, and T. M. Mitchell. Learning to classify email into speech acts. In *Proceedings of Empirical Methods in NLP*, 2004.

[17] A. Cypher. *Watch What I Do: Programming by Demonstration.* MIT Press, 1993.

[18] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm (with discussion). *Journal of the Royal Statistical Society*, B.39, 1977.

[19] T.G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

[20] Pedro Domingos and Mark Richardson. Markov logic: A unifying framework for statistical relational learning. In *Proceedings of the SRL Workshop in ICML*, 2004.

[21] P. Doshi. A particle filtering algorithm for interactive pomdps. In *In Proceedings of Conference on Modeling Other Agents from Observations*, 2004.

[22] A. N. Dragunov, T. G. Dietterich, K. Johnsrude, M. McLaughlin, L. Li, and J. L. Herlocker. Tasktracer: A desktop environment to support multi-tasking knowledge workers. In *Proceedings of IUI*, 2005.

[23] K. Driessens. Adding guidance to relational reinforcement learning. In *Third Freiburg-Leuven Workshop on Machine Learning*, 2002.

[24] A. Fern, S. Natarajan, K. Judah, and P. Tadepalli. A decision-theoretic model of assistance. In *Proceedings of the International Joint Conference in AI*, 2007.

[25] S. Fine, Y. Singer, and N. Tishby. The hierarchical hidden markov model: Analysis and applications. *Machine Learning*, 32:41–62, 1998.

[26] H. Geffner and B. Bonet. Solving large pomdps using real time dynamic programming. In *Working notes. Fall AAAI symposium on POMDPs*, 1998.

[27] L. Getoor, N. Friedman, D. Koller, and A. Pfeffer. Learning probabilistic relational models. *Invited contribution to the book Relational Data Mining, S. Dzeroski and N. Lavrac, Eds.*, 2001.

[28] Lise Getoor and John Grant. Prl: A probabilistic relational language. *Mach. Learn.*, 62(1-2):7–31, 2006.

[29] Lise Getoor and Ben Taskar. *Introduction to Statistical Relational Learning.* MIT Press, 2007.

[30] M. L. Ginsberg. GIB: Steps Toward an Expert-Level Bridge-Playing Program. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 584–589, 1999.

[31] C. Guestrin, D. Koller, R. Parr, and S. Venkataraman. Efficient solution algorithms for factored MDPs. *JAIR*, pages 399–468, 2003.

[32] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning.* Springer, 2001.

[33] David Heckerman and John S. Breese. Causal independence for probability assessment and inference using bayesian networks. Technical Report MSR-TR-94-08, Microsoft Research, 1994.

[34] David Heckerman, Christopher Meek, and Daphne Koller. Probabilistic models for relational data. Technical Report MSR-TR-2004-30, March 2004.

[35] E. Horvitz, J. Breese, D. Heckerman, D. Hovel, and K. Rommelse. The lumiere project: Bayesian user modeling for inferring the goals and needs of software users. In *In Proceedings of UAI*, pages 256–265, Madison, WI, July 1998.

[36] B. Hui and C. Boutilier. Who's asking for help?: a bayesian approach to intelligent assistance. In *In Proceedings of IUI*, pages 186–193, 2006.

[37] B. Hui and C. Boutilier. Who's asking for help?: a bayesian approach to intelligent assistance. In *Proceedings of IUI*, 2006.

[38] Manfred Jaeger. Relational Bayesian networks. In *Proceedings of UAI-97*, 1997.

[39] Manfred Jaeger. Parameter learning for relational bayesian networks. In *Proceedings of the International Conference in Machine Learning*, 2007.

[40] M. J. Kearns, Y. Mansour, and A. Y. Ng. A sparse sampling algorithm for near-optimal planning in large markov decision processes. In *IJCAI*, 1999.

[41] K. Kersting, L. De Raedt, and T. Raiko. Logical hidden markov models. *JAIR*, 25:425–456, 2006.

[42] Kristian Kersting and Luc De Raedt. Bayesian logic programs. In *Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming*, 2000.

[43] Kristian Kersting and Luc De Raedt. Adaptive bayesian logic programs. In *Proceedings of the ILP '01*, pages 104–117, 2001.

[44] Daphne Koller and Avi Pfeffer. Learning probabilities for noisy first-order rules. In *IJCAI*, pages 1316–1323, 1997.

[45] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th International Conf. on Machine Learning*, pages 282–289, 2001.

[46] T. Lau, S. Wolfman, P. Domingos, and D. Weld. Programming by demonstration using version space algebra. *Machine Learning*, 53(1-2):111–156, 2003.

[47] L. Liao, D. J. Patterson, D. Fox, and H. Kautz. Learning and inferring transportation routines. *Artif. Intell.*, 171(5-6):311–331, 2007.

[48] S. Mahadevan, T. M. Mitchell, J. Mostow, L. I. Steinberg, and P. Tadepalli. An apprentice-based approach to knowledge acquisition. *Artif. Intell.*, 64(1):1–52, 1993.

[49] T. M. Mitchell, R. Caruana, D. Freitag, J.McDermott, and D. Zabowski. Experience with a learning personal assistant. *Communications of the ACM*, 37(7):80–91, 1994.

[50] A. W. Moore and C. G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13:103–130, 1993.

[51] S. Muggleton. Stochastic logic programs. In *Advances in Inductive Logic Programming*, pages 254–264, 1996.

[52] K. Murphy and M. Paskin. Linear time inference in hierarchical HMMs. In *Proceedings of Neural Information Proceesing Systems*, 2001.

[53] Kevin Patrick Murphy. *Dynamic bayesian networks: representation, inference and learning*. PhD thesis, 2002. Chair-Stuart Russell.

[54] K. Myers, P. Berry, J. Blythe, K. Conleyn, M. Gervasio, D. McGuinness, D. Morley, A. Pfeffer, M. Pollack, and M. Tambe. An intelligent personal assistant for task and time management. In *AI Magazine*, 2007.

[55] S. Natarajan, P. Tadepalli, E. Altendorf, T. G. Dietterich, A. Fern, and A. Restificar. Learning first-order probabilistic models with combining rules. In *Proceedings of the International Conference in Machine Learning*, 2005.

[56] Sriraam Natarajan, Prasad Tadepalli, and Alan Fern. A relational hierarchical model for decision-theoretic assistance. In *Proceedings of 17th Annual International Conference on Inductive Logic Programming*, 2007.

[57] N. T. Nguyen, D. Q. Phung, S. Venkatesh, and H. Bui. Learning and detecting activities from movement trajectories using the hierarchical hidden markov models. In *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2*, pages 955–960, 2005.

[58] R. Parr and S. Russell. Reinforcement learning with hierarchies of machines. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, 1997.

[59] D. Poole. Probabilistic Horn abduction and bayesian networks. *Artificial Intelligence, Volume 64, Numbers 1, pages 81-129*, 1993.

[60] D. Precup and R. S. Sutton. Multi-time models for temporally abstract planning. In *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, 1998.

[61] D. V. Pynadath and M. P. Wellman. Probabilistic state-dependent grammars for plan recognition. In *UAI*, pages 507–514, 2000.

[62] Taisuke Sato and Yoshitaka Kameya. Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research*, pages 391–454, 2001.

[63] S. Siddiqi and A. Moore. Fast inference and learning in large-state-space hmms. In *Proceedings of the 22nd International Conference on Machine Learning*, 2005.

[64] S. P. Singh, D. J. Litman, M. J. Kearns, and M. A. Walker. Optimizing dialogue management with reinforcement learning: Experiments with the njfun system. *Journal of Artificial Intelligence Research*, 16:105–133, 2002.

[65] C. Skaanning, F. V. Jensen, and U. Kjaerulff. Printer troubleshooting using bayesian networks. In *IEA/AIE '00: Proceedings of the 13th international conference on Industrial and engineering applications of artificial intelligence and expert systems*, pages 367–379, 2000.

[66] P. Tadepalli, R. Givan, and K. Driessens. Relational reinforcement learning: An overview. In *In Proceedings of ICML Workshop on Relational Reinforcement Learning*, 2004.

[67] P. Varakantham, R. Maheswaran, and M. Tambe. Exploiting belief bounds: Practical pomdps for personal assistant agents, 2005.

[68] P. Varakantham, R. T. Maheswaran, and M. Tambe. Exploiting belief bounds: practical pomdps for personal assistant agents. In *AAMAS*, 2005.

[69] M. A. Walker. An application of reinforcement learning to dialogue strategy selection in a spoken dialogue system for email. *Journal of Artificial Intelligence Research*, 12:387–416, 2000.

[70] L. S. Zettlemoyer, H. M. Pasula, and L. P. Kaelbling. Logical particle filtering. In *Proceedings of the Dagstuhl Seminar on Probabilistic, Logical, and Relational Learning*, 2007.