A Relational Hierarchical Model for Decision-Theoretic Assistance

Sriraam Natarajan, Prasad Tadepalli, and Alan Fern

School of EECS,
Oregon State University

Abstract. Building intelligent assistants has been a long-cherished goal of AI and many were built and fine-tuned to specific application domains. In recent work, 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 paper, we extend this work to domains where the user's policies have rich relational and hierarchical structure. 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.

1 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 [2, 6, 3]. 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 recently 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 [9]. In most situations, however, the user's task or goal[1] is 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 [9] 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 [9], 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

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

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 paper, 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 [4] 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.

The current paper extends the assistantship model to hierarchical and relational settings, building on the work in hierarchical reinforcement learning[10] and statistical relational learning.We extend the assistantship framework of [9] 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.

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 paper 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 paper.

2 Decision-Theoretic Assistance

In this section, we briefly describe the decision-theoretic model of assistance of [9] which forms the basis of our work. 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.

In [9], 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. In [9], 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, their 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 [9] to the new structured policy of the user and show that it performs significantly better than the unstructured approach.

3 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.

3.1 Relational Hierarchical Policies

Users in general, 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 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

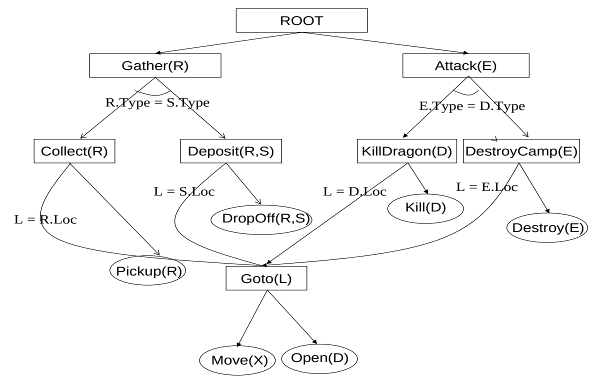

Fig. 1. Example of a task hierarchy of the user. The inner nodes indicate subtasks while the leaves are the primitive actions. The tasks are parameterized and the tasks at the higher level will call the tasks at the lower level

in S) in that order. 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 only 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 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. Though this knowledge is not tied to any particular formalism, the example is provided in probabilistic relational language (PRL) [18].

$$\begin{gathered}DependsOn(subgoal(Deposit(R,S)), Distance(Loc(R), Loc(S))) \\ \longleftarrow Goal(Gather(R)), Completed(Collect(R)), Equal(Type(R), Type(S))\end{gathered}$$

$DependsOn(X(\alpha), Y_1(\alpha), \ldots, Y_k(\alpha)) \longleftarrow Z(\alpha)$ means that $Y_1(\alpha)$, $\ldots, Y_k(\alpha)$ influence $X(\alpha)$ when $Z(\alpha)$ is true, where α is a set of 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. This statement can then be used to learn the exact parametric relationship between the location of the storage and the probability of it being chosen to store the resource. While the structure of these statements is inspired by languages such as PRLs [18] and Bayesian logic programs [15], the influence statements also serve the role of "abstraction" in the MAXQ approach to hierarchical reinforcement learning [10]. The probability of choosing a subtask in a given state is determined solely by the attribute values of the objects mentioned in the conditional influences, which puts a strong constraint on the user's policy when the distribution is not specified directly.

3.2 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 $\langle \textit{state, action} \rangle$ 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 a subtask is accompanied with observations that remove any uncertainty about it.This would enable the inference process to be much simpler 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 [16] and present it in Figure 2. T_j^i refers to the task at time-step j and level i in the DAG. O^i refers to the completed subtask at level i. There is no necessity of tracking the completed tasks at the highest level as the episode ends when the highest level task is completed. Note that the subtasks completed at a particular level influences the distribution over the current subtasks at the same level. This would enable us to factor the completed goal stack efficiently. It should be mentioned that in our experiments, we have chosen to ignore the completed goal stack for the main reason that most of our subtasks are totally ordered and hence there is no necessity for explicitly maintaining the history. F_j^i is an indicator variable that represents whether T_j^i 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. For a task at level i to be completed, all the subtasks at level k, $k > i$ must have been completed. a_j refers to the user's actions at time step j and s_j

refers to the world state at time step j. Aa_j refers to the assistant action at time step j.

Consider the problem of unrolling the DBN given the relational task hierarchy and the conditional influence statements. 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 the possible instantiations of the top-level goals. Once all the set of values that are possible for each variable are determined, the constraints are used to construct the CPT. For example, the constraint $R.Type = S.Type$ in the Figure 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. Hence, the constraints are used to determine the non-zero entries of the CPTs. The conditional influence statements are then used to determine the probabilities for the non-zero entries. In our model, we use the conditional influence statements after a particular subtask is completed. The conditional influence statements specify an informative prior over the choice of the next legal subtask given the completed goal stack and some attributes of the environment.

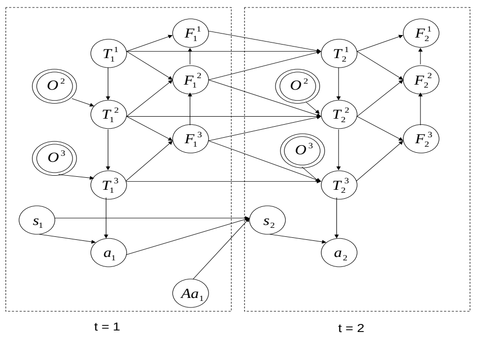

Fig. 2. Dynamic Bayesian network that is used to infer the user's goal.

To illustrate the construction of the DBN given the hierarchy and influence statements better, let us consider the example presented in Figure 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 at the beginning of the episode 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 goal stack due to the fact that most of our constraints are total order and there is no necessity of maintaining the completed stack. 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 laplace estimates. More specifically, the parameters of the different subtasks, the action models and that of the influence statements would be updated based on the trajectory over user actions in that episode. Since the model is inherently relational, we are able to 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 Fern et.al 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.

3.3 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 of [9], 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 [5] 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 Experiments and Results

In this section, we briefly explain the results of simulation of a user in two domains[2]: 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.

4.1 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 3.a). The hierarchy presented in Figure 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

[2] These are modification to the domains presented by Fern et.al[9]

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.

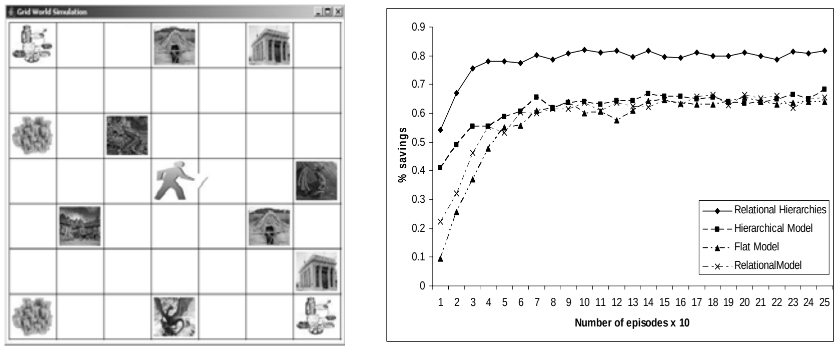

Fig. 3. (a)Doorman Domain. Each cell has 4 doors that the user has to open to navigate to the adjacent cell. The goal of the assistant is to minimize the number of doors that the user has to open. (b)Learning curves for the 4 algorithms in the doorman domain. The y-axis presents the average savings for 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 3.*b*. 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.

4.2 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.*a*. 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.

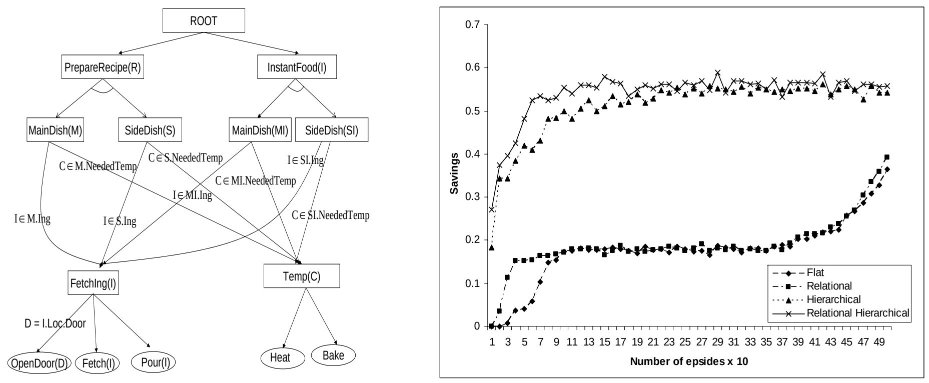

Fig. 4. (a)The kitchen domain hierarchy. (b)Learning curves of the different algorithms in the Kitchen Domain.

The savings in this domain 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 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.*b*. 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.

5 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 [2]. 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 [6]. 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 [17] or an abstract HMM [1] 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 [7] and text editing [8] 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 [14], BLPs [15], RBNs [12], MLNs [13] 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 paper, 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.

6 Conclusions and Future Work

In this work 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.

References

1. H. Bui and S. Venkatesh and G. West: Policy recognition in the Abstract Hidden Markov Models, JAIR, volume = 17, 2002
2. Jennifer Boger and P. Poupart and J. Hoey and C. Boutilier and G. Fernie and A. Mihailidis: A Decision-Theoretic Approach to Task Assistance for Persons with Dementia., IJCAI, 2005, 1293-1299
3. J. L. Ambite and G. Barish and C. A. Knoblock and M. Muslea and J. Oh and S. Minton: Getting from Here to There: Interactive Planning and Agent Execution for Optimizing Travel, IAAI, 862–869, 2002
4. CALO - Cognitive Agent that Learns and Organizes, http://calo.sri.com., 2003,
5. D. P. Bertsekas and J. N. Tsitsiklis: Neuro-Dynamic Programming, 1996, Athena Scientific,
6. P. Varakantham and R. Maheswaran and M. Tambe: Exploiting belief bounds: Practical POMDPs for personal assistant agents, AAMAS, 2005
7. E. Horvitz and J. Breese and D. Heckerman and D. Hovel and K. Rommelse: The Lumiere project: Bayesian user modeling for inferring the goals and needs of software users, UAI, 256–265, 1998
8. B. Hui and C. Boutilier: Who's asking for help?: a Bayesian approach to intelligent assistance, IUI,2006,186-193
9. A. Fern and S. Natarajan and K. Judah and P. Tadepalli: A Decision-Theoretic Model of Assistance, IJCAI, 2007
10. T. G. Dietterich: Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition, JAIR, volume 13, 227-303, 2000,
11. P. Tadepalli and R. Givan and K. Drissens: Relational Reinforcement Learning - An Overview, Workshop on Relational Reinforcement Learning, ICML,2004
12. M. Jaeger: Relational Bayesian Networks,UAI-97
13. P. Domingos and M. Richardson: Markov logic networks, Mach. Learn., volume 62, number 1-2, 2006, pages 107-136
14. L. Getoor and N. Friedman and D. Koller and A. Pfeffer: Learning Probabilistic Relational Models, Invited contribution to the book Relational Data Mining, S. Dzeroski and N. Lavrac, Eds. 2001
15. K. Kersting and L. De Raedt: Bayesian Logic Programs,ILP 2000
16. K. Murphy and M. Paskin: Linear time inference in hierarchical HMMs, NIPS, 2001
17. S. Fine and Y. Singer and N. Tishby: The Hierarchical Hidden Markov Model: Analysis and Applications, Machine Learning, 32, 1, pages 41-62, 1998
18. L.Getoor and J. Grant: PRL: A Probabilistic Relational Language, Machine Learning Journal, 2005. Machine Learning, 62, 1/2,7-33, 2006