

A Decision-Theoretic Model of Assistance - Evaluation, Extensions and Open Problems

Sriraam Natarajan and Kshitij Judah and Prasad Tadepalli and Alan Fern

School of EECS, Oregon State University

Corvallis, OR 97331 USA

{natarasr,judahk,tadepall,afern}@eecs.oregonstate.edu

Abstract

There is a growing interest in intelligent assistants for a variety of applications from organizing tasks for knowledge workers to helping people with dementia. In our earlier work, we presented a decision-theoretic framework that captures the general notion of assistance. The objective was to observe a goal-directed agent and to select assistive actions in order to minimize the overall cost. We employed the use of POMDPs to model the assistant whose hidden state was the goal of the agent. In this work, we evaluate our model of assistance on a real world domain and establish that our model was very effective in reducing the efforts of the user. We compared the results of our model against a cost-sensitive supervised learning algorithm. We also describe our current work on extending the model to include relational hierarchies. We then analyze some problems in our model and suggest possible extensions to handle them.

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 (Boger *et al.* 2005) and desktop work management (CALO 2003). However, most of this work has been fine-tuned to the particular application domains. Also, much of the prior work on intelligent assistants did not take a sequential decision making or decision-theoretic approach. For example, email filtering is typically posed as a supervised learning problem (Cohen, Carvalho, & Mitchell 2004), while travel planning combines information gathering with search and constraint propagation (Ambite *et al.* 2002). There have been other personal assistant systems that are explicitly based on decision-theoretic principles (Boger *et al.* 2005; Varakantham, Maheswaran, & Tambe 2005). Most of these systems have been formulated as POMDPs that are approximately solved offline.

In the previous work (Fern *et al.* 2007), we considered a model where the assistant observes a goal-oriented user and must select assistive actions in order to best help the user

achieve its goals. We described and evaluated a comprehensive decision-theoretic framework for intelligent assistants. To perform well the assistant must be able to accurately and quickly infer the goals of the user 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 user, to reason about varying action costs, to handle unforeseen situations, and to adapt to the user over time. We considered 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 previous work (Fern *et al.* 2007) had three major contributions: first was the formulation of the assistant POMDP which jointly models the application environment and the user's hidden goals, second was the presentation of an approximate solution approach based on explicit goal estimation and myopic heuristics and third was the evaluation of the framework on two game-like environments.

In this short paper, we extend the previous work in three ways: Firstly, we evaluate our framework on a more realistic domain, the folder predictor (Bao, Herlocker, & Dietterich 2006) of the Task Tracer project. In this setting, the user is searching for a file and the assistant would try to recommend a set of 3 folders for the ease of access. Bao et al. considered the problem as a supervised learning problem and applied a cost-sensitive algorithm to predict the 3 most relevant folders (Bao, Herlocker, & Dietterich 2006). We model this as a decision-theoretic problem with repeated predictions as and when the user performs an action. Secondly, we try to extend the model to incorporate relational hierarchies. Thirdly, we identify some key open problems in this framework and suggest extensions to handle the issues that we have identified.

The remainder of this paper is organized as follows. In the next section, we briefly introduce our problem setup, followed by a definition of the assistant POMDP and present our approximate solution technique based on goal estimation and online action selection. We then give an empirical evaluation of the approach in the folder predictor domain. Next, we outline our current work on extending the model to incorporate relational hierarchies. We follow this with a discussion of the key improvements that can be made to the

model and present some changes in the model that handle these improvements.

Problem Setup

In this section, we briefly present the formal definition of the problem and the solution. We refer the readers to (Fern *et al.* 2007) for details.

We model the user’s environment 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 distributions that represents the probability of transitioning to state w' given that action $a \in A \cup A'$ is taken in state w . 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 assume that we are in an episodic setting, 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. The assistant after observing the user perform an action executes a set of actions ending in **noop** action after which the user may perform an action. Our objective is to minimize the sum of the costs of user and assistant actions. We model the user 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 and the assistant as a history-dependent stochastic policy $\pi'(a|w, t)$. We assume that we have at our disposal the environment MDP and the set of possible goals G . Our objective is to select an assistant policy π' that minimizes the expected cost given by $E[C(I, G_0, \pi, \pi')]$, where G_0 is an prior distribution over user goals and π is the unknown user policy.

Assistant POMDP

A POMDP extends an MDP in a partially observable environment by including a finite set of observations O , and a distribution $\mu(o|s)$ over observations $o \in O$ given the current state s . A POMDP policy can be viewed as a mapping from belief states to actions. We will capture the uncertainty of the user’s goal by including it as a hidden component of the POMDP state. Given an environment MDP $\langle W, A, A', T, C, I \rangle$, a goal distribution G_0 , and a user policy π we now define the corresponding *assistant POMDP*.

The **state space** is $W \times G$ so that each state is a pair (w, g) of a world state and user goal. The **initial state distribution** I' assigns the state (w, g) probability $I(w)G_0(g)$, which models the process of selecting an initial state and goal for the user at the beginning of each episode. The **action set** is equal to the assistant actions A' , reflecting that assistant POMDP will be used to select actions.

The **transition function** T' assigns for any action a except for **noop**, the state transitions from (w, g) to (w', g) with probability $T(w, a, w')$. For the **noop** action, T' simulates the effect of executing an user action selected according to π . The **cost model** C' reflects the costs of user 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 $C'((w, g), \text{noop}) = E[C(w, a)]$, where a is a random variable distributed according to $\pi(\cdot|w, g)$. The **observation**

distribution μ' is deterministic and reflects the fact that the assistant can only directly observe the world state and actions of the user.

A policy π' for the assistant POMDP maps state-action sequences to assistant actions. In our problem setup the assistant POMDP is not directly at our disposal as we are not given π or G_0 . Rather we are only given the environment MDP and the set of possible goals. As described in the next section our approach will approximate the assistant POMDP by estimating π and G_0 based on observations and select assistive actions based on this model.

Approximately Solving the Assistant POMDP

Since solving the exact assistant POMDP will be impractical, we proposed an approximation to the POMDP in our earlier work, which we present here. We consider the selection of assistant’s actions as a two-step process. First, the assistant needs to estimate the goal of the user by observing the user’s actions and then select its best action.

To estimate the goal of the user, the assistant learns the goal distribution G_0 and policy π of the user by storing the goal achieved by the user at the end of the episode along with the set of world state-action pairs for the user. Currently, the goal distribution is initialized by solving the user MDP and define the prior over the user action using the Boltzmann distribution. This is justified as the assumption is that the user is reasonably close to optimal. The prior needs to be updated after every episode and eventually will converge to the user policy.

Goal Estimation Given the user policy π , it is straightforward to incrementally update the posterior $P(g|O_t)$ upon each of the user’s actions. 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 user action, then we leave the distribution unchanged. Otherwise, if o_t indicates that the user selected action a in state w , then we update the distribution 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 user to execute action a in w . If the user is close to optimal, this approach can lead to rapid goal estimation, even early in the lifetime of the assistant.

Action Selection In our earlier work, we had approximated the assistant POMDP M by a set of assistant MDPs for each goal g which is denoted by $M(g)$. An optimal policy for $M(g)$ gives the optimal assistant’s action when the user tries to achieve goal g . The Q-value of $M(g)$ is denoted by $Q_g(w, a)$, which is the expected cost of executing action a and then following the optimal policy. Given these MDP’s the heuristic value of executing an action a in state w is

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

The actions are then selected greedily according to H which measures the utility of taking an action under the assumption that the goal ambiguity is resolved in one step. If the goal distribution is highly ambiguous the assistant will select **noop** action.

In our experiments with the folder predictor, we resorted to two kinds of approximations for the H values. One is to replace the user policy with a fixed default policy and not updating the goal posterior after every episode. The second approximation is to compute the Q_g values using the simulation technique of *policy rollout* (Bertsekas & Tsitsiklis 1996). The main idea here is that we compute the value of $Q_g(w, a)$ by assuming that the assistant performs only a single action and the user takes over. 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. In $H(w, a, O_t)$ we replace $Q_g(w, a)$ with the expectation $\sum_{w' \in W} T(w, a, w') \cdot \bar{C}(\pi, w', g)$. We combine the two approximations and use this heuristic in our experiments with the folder predictor.

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 (Dragunov *et al.* 2005), researchers developed a file location system called *folder predictor* (Bao, Herlocker, & Dietterich 2006). The idea behind folder predictor is that if we have knowledge about the user’s file access patterns, we could help the user with his file accesses by predicting the folder in which the file has to be accessed or saved.

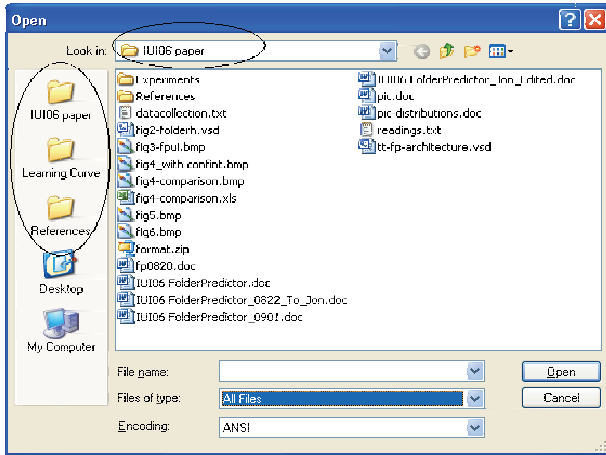


Figure 1: Folder predictor (Bao, Herlocker, & Dietterich 2006).

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 1). 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.

Bao *et al.*(2006) considered the problem as a supervised learning problem and implemented a cost-sensitive algo-

rithm for the predictions with the cost being the number of clicks of the user. 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 in a task as possible destinations. This precludes handling the possibility of user accessing a new folder.

We used our model for folder prediction. 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 UI and used our model to make predictions. We used 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.

$$P(f) = \mu_0 P_0(f) + (1 - \mu_0) P_1(f)$$

where P_0 is the probability according to Bao *et al.*’s algorithm, P_1 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, 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 by Bao *et al.*(2006).

We applied the decision theoretic model to the data set. For each request, our assistant would make the prediction 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 consid-

ered 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 Figure 2.

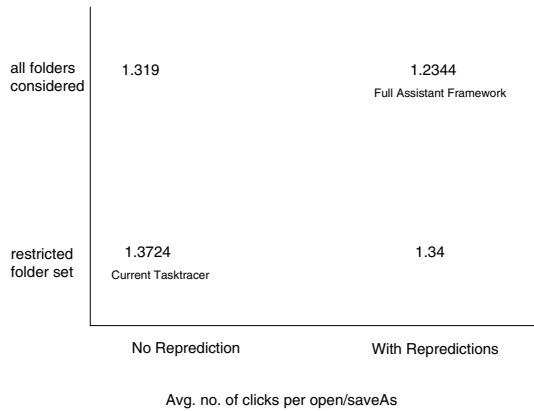


Figure 2: Results of Folder Navigation. There are four algorithms corresponding to the combinations of with and without predictions and considering the entire folder hierarchy or a restricted set.

The figure 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 yield a lower cost than the original algorithm, but combining the two changes is significantly more effective.

Relational Hierarchical Goal structure

We are currently working on incorporating hierarchical goal structure of the user in our formulation. We are also focussing on making the hierarchies relational. In this section, we briefly outline the changes to our formulation in order to account for the hierarchical goal structure. There have been several plan recognition algorithms that use a hierarchical structure for the user’s plan. These systems would typically use a hierarchical HMM (Fine, Singer, & Tishby 1998) or an abstract HMM (Bui, Venkatesh, & West 2002) etc to track the user’s plan. In our current formulation, we assumed that the user had a set of flat goals that he was trying to achieve. Typically, users in real-world decompose a larger problem into a set of smaller ones and aim to achieve them. For in-

stance, the higher level goal of the user might be to cook a particular recipe while the subgoals might be to have a certain set of ingredients heated in one bowl and a different set baked in another bowl and then mix both and heat. The user would then try to achieve these subgoals individually.¹

As another example, consider the task of the user trying to submit a proposal. He might consider first writing up a document, look up references and then turning in the proposal through email. Hence, he would divide the task into a set of sub-tasks that he will try and achieve individually. Also, the tasks and sub-tasks are performed exactly in a similar fashion for many documents. For instance, to submit a paper to *IJCAI* or *ICML*, the user will run experiments and edit the document, send it to his co-authors and turn it in by the deadline. If a propositional hierarchy is employed, there is a necessity to duplicate the same hierarchy for each of these papers.. Also in many real-world domains, there are several different objects and relations can exist between these objects. For instance, only the co-authors of a paper would receive an email with the paper attached. Hence there is a necessity to capture the relational nature of the domain in the hierarchies that model the user’s goal-subgoal structure.

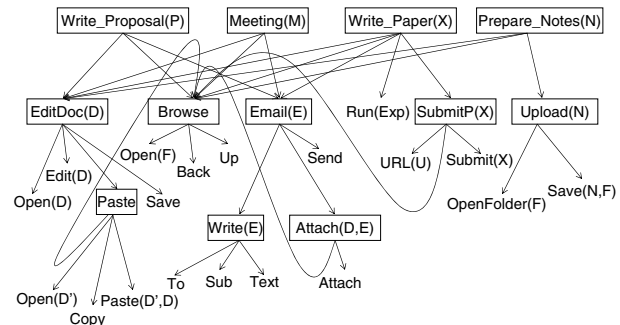


Figure 3: Example of a task hierarchy of the user. The inner nodes indicate subtasks while the leaves are the primitive actions. The tasks are parameterized and the tasks at the higher level will call the tasks at the lower level

An example of the user’s task hierarchy is presented in Figure 3. The task hierarchy forms a DAG with a single “root” node. The nodes that are the children of this root node are the tasks that the user needs to perform. In our example, they correspond to writing a proposal (*Write_Proposal(P)*), preparing for a meeting (*Meeting(M)*), writing a paper (*Write_Paper(X)*) and preparing class notes (*Prepare_Notes(N)*). It can be observed that these tasks are parameterized, the parameter being the object to be used for the task to be achieved. Each of these tasks can be achieved by executing either a primitive action (represented without boxes in the figure) or another subtask. For example, for writing a proposal the user needs to edit a document

¹In this section, we use the terms task and goal interchangeably.

(*EditDoc(D)* in the figure). The hierarchy would also include tasks such as *Email(E)* and *Attach(E,D)* which means that *E* is an email and *D* is attached to *E*.

In our formulation, there is a single hierarchy (similar to the one in Figure 3) that would be accessible both to the user and the assistant. The user and assistant could potentially have different sets of actions. There will also be ordering constraints of the form $A \prec B$, which means that the subtask *A* should be completed before *B*. The constraints obey the principle of transitivity. i.e., if $A \prec B$ and $B \prec C$, then $A \prec C$. The subtasks are partially ordered. For instance, one could either attach a document to an email and then type the text in the email and vice-versa. In addition, certain subtasks might be optional and they would be specified along with the constraints. For instance, attaching a document to an email is not always a necessary subtask. The ordering constraints could include the optional subtasks as well. A task is considered as completed iff all the necessary (non-optional) subtasks are completed. Once a subtask is completed, the control returns to the parent task, which then chooses the next “legal” task i.e., the next subtask that satisfies all the constraints and starts executing it. When the parent task chooses a child task to execute, it passes the appropriate parameters to the child task. For instance, if the parent task is *Email(E)* and a document has to be attached to the email, it will call the child task *Attach(D,E)* with the parameters *E* which is the email and *D* which is the document that has to be attached to *E*.

Let us now consider the changes that need to be incorporated in our framework for handling the hierarchical goal structure of the user.

- The user is modeled as a stochastic policy. The user has a set of goals that he chooses to achieve and tries to achieve it. In our setting, we model the user as choosing one of the highest level goals at random. It is not a critical assumption and a more complex mechanism for the selection of goals can be employed. We also assume that the parameter at the highest level goal is observed. This is important as it simplifies the underlying process of inferring the user’s goals. More precisely, it will enable us to use a DBN for performing inference on the user’s goals.
- The assistant seeks to minimize the total cost of the user actions and the assistant actions. Note that as with the previous work, the action set of the assistant need not be the same as that of the user. It could include actions like *noop* or prompts for the user. The user is provided with an *undo* action that allows him to retract the undesirable actions of the assistant. In this setting, the assistant and the user’s actions strictly alternate, an assumption which we will relax in the next section.
- This leads to using hierarchical POMDP to select the assistive actions. The hierarchy captures the user’s goal structure. The partial observability is due to the fact that the user’s goals are not directly observed by the assistant. The only observations are the user’s primitive actions and the assistant has to infer the user’s goal-subgoal combinations based on its observations. Hence the belief state would now correspond to the current goal stack of the

user.

With the above observations, we now define the assistant POMDP. The **state space** is $W \times \vec{G}$ where W is the set of world states and \vec{G} is the user’s goal stack. Note that in our earlier work G was a single goal state. In this case, it is a goal stack corresponding to the task-subtask combinations. The size of this goal stack would correspond to the maximum depth in the DAG starting from the *Root* node. Correspondingly, the transition probabilities are functions between (w, \vec{g}) and (w', \vec{g}') . Similarly, the cost is now a function of the new $\langle state, action \rangle$ pairs. The observation space now includes the user’s actions and their parameters (for example, the document that is edited, the recipients of the email etc). The policy of the assistant is now a hierarchical policy which consists of a policy for each subtask that indicates how it would be solved.

For estimating the user’s goal stack, we use a DBN similar to the one used in (Murphy & Paskin 2001). The tasks at different level are parameterized. Since we observe the parameter of the task at the highest level (for example, the proposal or the paper that is being written, the meeting that the user is preparing for or the class notes that the user is working on), we could use the DBN to infer the distribution over the tasks given that we observe the argument of the parameterized task. We exploit the power of relational models when we instantiate the objects. For instance, if we declare that the documents that are in the same directory are the ones that the user would refer, we could restrict the space of documents that we have to consider for reference. Similarly once we know the paper that the user is attempting to write, the set of experiments that need to be performed can be deduced. Yet another example where we would exploit the power of relations is when we could use relational statements that might include aggregators for setting the prior distributions. As an example, the prior over a particular folder being used for reference could depend on the number of documents that are used in the same task as the one we are currently editing. We are currently working on incorporating such statements into the model for initializing the distributions.

In our earlier work, we used policy rollout to select the assistant action. The main idea is that, we sample user actions for every goal and compute the expected reward of performing an action. We used the goal distribution to compute the expectation. We could use the same principle to select the action in this setting as well. Instead of sampling over all goals, we need to sample goals at all levels and compute the expectations over all the levels. The heuristic value of executing an action in state w will now be,

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

To compute the value of $Q_{g^{1:d}}(w, a)$, we could sample all the possible combinations of the goals and compute the expectation assuming that the assistant will perform an action and the user will complete the rest of the goal.

Advantages of Hierarchies

An obvious question is the need for extending the model to capture relational hierarchies. Primary reason for this is that it is more natural to handle a more sophisticated goal structure than the simple goal structure that we have focused earlier and the power of the relational models can be effectively exploited. Also, this would enable the assistant to address several other issues. 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. We motivate this further in the next section. The other advantage is exploiting the idea of state abstraction. Though we have not made this explicit in our discussion, we plan to exploit the state abstraction in our action selection procedures. We would consider only parts of the state space that are relevant to the current goal while selecting an action. 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 (for instance, the user might forget to attach a file to an email and the assistant could help him avoid sending the email without the attachment).

Open problems and Extensions

In this section, we try to present the list of assumptions of the current framework and suggest ways to relax these assumptions.

Partial Observability of the user

In our current model, we assume that the user can completely observe the environment. This is sometimes an unrealistic assumption. In real world, the user cannot sense the environment fully. Consider for example a household assistant and the goals of the user are to cook some dishes. The ingredients are present in shelves and cabinets that have opaque doors. The user does not have complete knowledge of the environment (i.e., he does not know which ingredient is in which shelf) while the assistant completely observes the environment (i.e., it knows the exact location of each ingredient). The user has certain beliefs about the presence of the ingredients in the shelves and quite naturally he will start exploring. If the assistant knows that the user is exploring, it could either point to the right doors to open or open the right doors for the user. As another example, we could consider searching for a particular webpage. We can imagine this domain to be similar to the folder predictor domain, except that the entire web structure is not known to the user while the assistant can have the complete knowledge of the structure. So the user would tend to explore to access the appropriate webpage and the assistant can assist the user in his exploration. The current model has to be extended to capture the partial observability of the user.

Currently, the system would try to infer the user's goals given his actions. If the user is opening some doors for ex-

ploration, the system would try to estimate the goal after every opening of the door. The assumption in our current framework is that the user is optimal w.r.t the goal. This assumption would mean that the actions of user opening different doors without fetching an ingredient would confuse the assistant as it will not correspond to any optimal policy of the user. It is more correct to think that the user is optimal with respect to the goal and his belief states. In the current example, the user does not know where a particular ingredient is present and the most optimal thing to do to fetch this ingredient is to open the doors of different shelves till he finds it. The present framework does not capture the belief state explicitly and hence cannot deal with the case where the user is performing exploratory actions. Incorporating the belief states of the user in the assistant's state space would enable the assistant to make useful inferences about the user's goals and his belief states given his actions. In (Doshi 2004), the authors introduce the setting of interactive POMDPs, where each agent models the other agent's beliefs. Note that in our setting, since the user is oblivious of the assistant, there is no necessity of modeling the assistant's beliefs by the user.

The extension to our model involves two changes. First, the environment is now modeled as a POMDP as opposed to modeling as an MDP where the belief states are those of the user. The environment POMDP extends the earlier defined MDP by including the set of user observations (O_u) and the observation distribution of the user (μ_u). Next, the user's beliefs must be captured in the Assistant POMDP. Currently, the state space of the assistant consists of the world states and the goals of the user. To model user's partial observability, we include the user's belief states as part of the state space of the assistant. Thus, the new state space of the assistant is $S = W \times G \times B_u$, where W is the set of world states, G is the set of goal states and B_u is the set of belief states of the user. The transition function, cost function and the observation functions would correspondingly reflect the change. The Transition function T' is a mapping from $\langle w, g, b_u \rangle$ to $\langle w', g, b'_u \rangle$ for all non-noop actions. For noop actions, T' simulates the user policy according to the environment's transition function. The Observation function is still deterministic and the assistant can observe the world states and the user's actions. The sources of the assistant's uncertainty are the user's goal and his belief states.

Large state space

One of the important features of our model is that the POMDP of the assistant needs to be solved online and the user policy needs to be updated after every user trajectory to the goal. The state space of the POMDP has to be very small to facilitate solving of the POMDP online. Even for moderate state spaces, it is not feasible to solve the POMDP online. In our earlier experiments in (Fern *et al.* 2007) with the cooking domain, the state space had about 140,000 states and it was not computationally feasible to solve the user MDP or the assistant MDP and we had to resort to approximations. It is easy to see that in our current example of household assistant, the number of states grows exponentially with the number of ingredients and it is not feasible

to solve the POMDP online. There have been many approaches to handle the problem of large state spaces such as function approximation, abstraction and model minimization techniques. In *Electric Elves*, 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 (Varakantham, Maheswaran, & Tambe 2005). It seems necessary to explore the use of such techniques for the development of intelligent assistants as many situations involve a large number of state features.

Also, in the current model, we are assuming that the user MDP can be solved and used it for initializing the goal distribution. But, this may not be possible in many cases as the state space could be very large. In the cooking example, there could be a very large number of ingredients and each of them may be in the bowl, shelf or on the cooking table etc and hence the state space is huge. In these cases, it is not possible to solve the user MDP and use it to initialize the distribution. An obvious choice is to use uniform distribution over the goals and update the distribution after every episode. The problem with this method is that the assistant will not be useful in the early stages until the distribution has skewed towards a particular goal.

Another alternative that we have used in one of our experiments is to use the optimal policy of the user based on domain knowledge (though the underlying MDP is very large, the policy can be deterministic and can be defined by a set of rules) and then initialize the goal distribution according to the optimal policy. For instance in the cooking domain, there are only a few optimal ways a particular dish can be cooked. Hence, the user has only a certain number of optimal policies for cooking these dishes. We can use these optimal policies to initialize the goal distribution.

Yet another possibility is to leverage the earlier work on learning apprentice systems and learning by observation (Mitchell *et al.* 1994). The user's actions provide training examples to the system which can be used for learning the user policy and the prior distributions.

Changing goals

In many real world situations, the user might change his mind while trying to achieve his goal. For instance, the user can decide to change the dish he was trying to cook based on what is available, changes in weather etc (if it is cold outside, the user might not prefer a cold salad for dinner though he might have opened the bag of veggies already). There is thus a possibility that the user can change his goal midway while trying to achieve another goal. Our system currently computes $P(goal | s, a)$ using all the state-action pairs. Eventually, it will converge to the right goal. But, if we observe that the user is doing a small set of actions that takes him away from the current goal, we can quickly infer that he is going after a different goal. The system currently would converge to the correct goal distribution slowly. Hence, there is a necessity for the framework to model explicitly the possibility of the user changing his goals while trying to achieve another.

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 (Horvitz *et al.* 1998). Hui and Boutilier used a similar idea for assistance with text editing (Hui & Boutilier 2006). They use DBNs with hand-coded parameters to infer the type of the user and computed the expected utility of assisting the user. It would be interesting to explore these kind of user models in our system to determine the user's intentions and then compute the optimal policy for the assistant. We believe that incorporating a more richer model of the user that would explicitly model the goal changes would make it possible for our system to handle the case of user changing his goals without having to modify the solution methods.

Expanding the set of goals

In our setting, we assume a fixed set of goals that the user is trying to achieve. For instance, in the household assistant domain, the assumption would be that the user can only cook a few set of dishes. This is sometimes an unrealistic assumption and would seriously limit the effectiveness of the assistant. This fixed set of goals would mean that the assistant is only capable of assisting only for this set of goals. It is possible that though the user might be trying to cook a new recipe but the assistant can still afford to help. There is no possibility of handling this expanding set of goals in our present formulation. One of the interesting directions is to extend our formulation to handle new goals as they come and learn about them once the user has achieved them. It is not clear at this point as to what extensions to the model will enable the model to handle this problem of expanding set of goals.

Parallel Actions

Our current model assumes that the user and the assistant have interleaved actions and cannot act in parallel. Though this is not an unrealistic assumption, it would be more useful to make it possible for both the assistant and the user to perform actions in parallel. For example, while the user is trying to prepare one part of the recipe, the assistant could try and prepare the other part of the recipe. This would save the user the time of waiting for the assistant to complete his action and then continue with his actions. The amount of time required to achieve the goal can be reduced drastically by allowing for parallelism. 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. It is not clear yet what extensions to the current model are required for parallel actions as the user and the assistant might try to achieve very distinct subgoals. There are several potential issues like the amount of parallelism allowed, the extent to which the assistant can be allowed to be autonomous (as we do not want the assistant to continue to perform erroneous actions, for instance, cook something that is not part of the recipe), user's adaptability to the assistant's actions etc. It is not clear how feasible it would be to solve the Assistant POMDP with parallel actions, but nevertheless parallelizing the actions seems to be an interesting area of research.

Forgetting Subgoals/Actions

The current model does not explicitly capture the possibility of the user forgetting to achieve a subgoal while trying to achieve the goal. For instance, if the user is to cook a recipe, he might forget to add a particular ingredient to the cooking bowl. Currently, the system would consider this as a possibility of some other goal and cannot understand that he is trying to achieve the goal of cooking a dish but forgot to perform a subgoal in the process. If the assistant can explicitly model this possibility of forgetting to perform an action or a subgoal, it could remind the user to add the particular ingredient. A possible solution is to add a richer user model on top of the current DBN to model the possibility of forgetting explicitly.

Conclusion

In this paper, we presented new results for our decision-theoretic assistant framework in the domain of folder prediction. We also discussed a number of open problems that still exist in our framework. Our folder-prediction results showed that the framework can be effective in a real-world problem where a significant reduction in user cost would be widely appreciated. Many of the open problems discussed appear to be partially addressable by incorporating goal hierarchies into the framework, which is currently one of our major focuses. We are also considering the possibility of incorporating a more sophisticated user model and action space for the assistant to handle several key issues that we discussed in this work.

Acknowledgement

We would like to express our gratitude to Thomas.G. Dietterich and Xinlong Bao for their suggestions and discussions regarding the folder predictor. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA), through the Department of the Interior, NBC, Acquisition Services Division, under Contract No. NBCHD030010. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA.

References

- Ambite, J. L.; Barish, G.; Knoblock, C. A.; Muslea, M.; Oh, J.; and Minton, S. 2002. Getting from here to there: Interactive planning and agent execution for optimizing travel. In *IAAI*, 862–869.
- Bao, X.; Herlocker, J. L.; and Dietterich, T. G. 2006. Fewer clicks and less frustration: reducing the cost of reaching the right folder. In *IUI '06*, 178–185.
- Bertsekas, D. P., and Tsitsiklis, J. N. 1996. *Neuro-Dynamic Programming*. Athena Scientific.
- Boger, J.; Poupart, P.; Hoey, J.; Boutilier, C.; Fernie, G.; and Mihailidis, A. 2005. A decision-theoretic approach to task assistance for persons with dementia. In *IJCAI*.
- Bui, H.; Venkatesh, S.; and West, G. 2002. Policy recognition in the abstract hidden markov models. *JAIR* 17.

- CALO. 2003. Cognitive agent that learns and organizes, <http://calo.sri.com>.
- Cohen, W. W.; Carvalho, V. R.; and Mitchell, T. M. 2004. Learning to classify email into speech acts. In *Proceedings of Empirical Methods in NLP*.
- Doshi, P. 2004. A particle filtering algorithm for interactive pomdps.
- Dragunov, A. N.; Dietterich, T. G.; Johnsrude, K.; McLaughlin, M.; Li, L.; and Herlocker, J. L. 2005. Task-tracer: A desktop environment to support multi-tasking knowledge workers. In *Proceedings of IUI*.
- Fern, A.; Natarajan, S.; Judah, K.; and Tadepalli, P. 2007. A decision-theoretic model of assistance. In *IJCAI (To appear)*.
- Fine, S.; Singer, Y.; and Tishby, N. 1998. The hierarchical hidden markov model: Analysis and applications. *Machine Learning* 32(1):41–62.
- Horvitz, E.; Breese, J.; Heckerman, D.; Hovel, D.; and Rommelse, K. 1998. The lumiere project: Bayesian user modeling for inferring the goals and needs of software users. In *In Proc UAI*, 256–265.
- Hui, B., and Boutilier, C. 2006. Who's asking for help?: a bayesian approach to intelligent assistance. In *IUI*, 186–193.
- Mitchell, T. M.; Caruana, R.; Freitag, D.; JMcDermott; and Zabowski, D. 1994. Experience with a learning personal assistant. *Communications of the ACM* 37(7):80–91.
- Murphy, K., and Paskin, M. 2001. Linear time inference in hierarchical HMMs.
- Varakantham, P.; Maheswaran, R. T.; and Tambe, M. 2005. Exploiting belief bounds: practical pomdps for personal assistant agents. In *AAMAS*.