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

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 agen.t 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 cost of the user. We compared the results of our model against a cost sensitive supervised learning algorithm. We then analyze the possible open problems in our model and try to provide some intuitions on the possible extensions to our model to handle these open problems.

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.* 2006), we considered 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. 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 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. 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.* 2006) had three major contributions: first was the formulation of the assistant POMDP which jointly models the application environment and the agent'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 two 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 predicting algorithm to predict the 3 most relevant folders (Bao, Herlocker, & Dietterich 2006). We try to model this as a decision-theoretic problem with repeated predictions as and when the user performs an action. Secondly, we identify some key open problems in this problem formulation and motivate these problems for the attention of the Intelligent Assistants community. We also attempt to extend our model 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 formal 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. 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 and the next couple of sections, we briefly present the formal definition of the problem and the solution. We refer the readers to (Fern *et al.* 2006) for details.

We model the agent'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 agent 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 agent chooses a goal and tries to achieve it. The assistant would observe the user's actions and the world states but does not observe 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 agent as a 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 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 agent goals and π is the unknown agent 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 the agent's goal 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 an agent 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 agent 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 agent 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 agent action selected according to π. 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 $\pi(\cdot|w, g)$. The **observation distribution** μ' is deterministic and reflects the fact that the assistant can only directly observe the world state and actions of the agent.

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.

Selecting Assistive Actions

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 agent 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 agent. Currently, the goal distribution is initialized by solving the user MDP and define the prior over the agent action using the Boltzmann distribution. This is justified as the assumption is that the agent 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 agent policy π, it is straightforward to incrementally update the posterior $P(g|O_t)$ upon each of the agent'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 agent action, then we leave the distribution unchanged. Otherwise, if o_t indicates that the agent 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 agent to execute action a in w. If the agent 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 agent 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 ambigous 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 agent 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.

Bao et.al viewed this problem of folder prediction as a supervised learning problem and used a cost sensitive prediction algorithm that aimed at minimizing the number of clicks of the user. The algorithm would choose the top three folders that would minimize the cost and then append them to the UI. The user then can choose one of these recommendations or navigate through the windows folder hierarchy if the recommendations are not relevant. They compared their results with the windows default recommendations and established that their algorithm outperforms the windows default predictions.

But, their algorithm does not take into account the response the user's reactions to the 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 cannot consider the user's actions in account and just optimizes a single cost. 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 use the user's response to our predictions to make further predictions. 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 heirarchy consists of 226 folders.

The state space consists of 4 parts: the current folder that the user is accessing and the three recommedations. This would correspond to a state space of size $226 \times \binom{226}{3}$. The action of the user is based on the recommendations of the assistant. In the worst case, the agent might have to navigate through all the folders. Hence the action space of the user would be $O(226 \times 3$. The action space of the assistant would be the three folders that it has to choose from and would be of the size $\binom{226}{3}$. The cost in our case was the number of user clicks to the correct folder. In this domain, the assistant and the agent's actions strictly alternate as the assistant revises its predictions after every user action.

We ran our decision theoretic model on 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 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 Figure 1. The histogram presented in the figure presents the cost required to reach the correct folder. Ideally, we would like to go to the correct folder in zero or one click (zero click case is when the top recommended folder is the right folder and the UI automatically opens the top recommended folder, while one click corresponds to opening one of the other 2 recommendations).

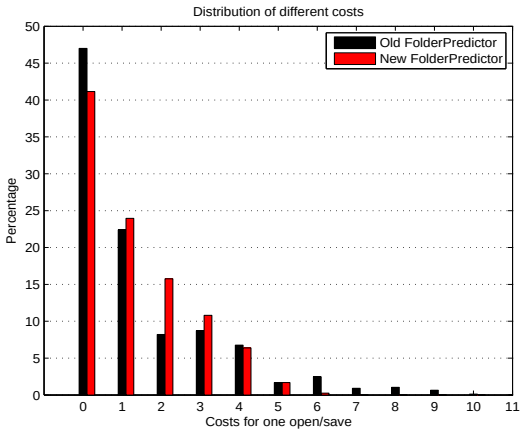

Figure 1: Results of Folder predictor.

It can be observed from the figure that when our model is used, the number of clicks more than 6 reduces to zero. This is due to the use of re-predictions in our model which is possible due to the decision-theoretic framework while Bao et.al's model makes a one-time prediction and hence cannot make use of the user's response to the recommendations. But, it can also be observed that their model had a higher number of zero clicks than ours. This is because, they consider only a subset of the folders for their predictions. They use information like the task the user is working on to recommend the folders, while we had to consider all the 226 folders for our hypotheses space to handle the possibility of a new folder. We are currently trying to incorporate the task specific information into the folder prediction.

We also compared average cost of using our model to that of Bao et.al. While the user on an average has to perform X clicks to reach the appropriate folder in our model, he had to perform X clicks on an average using their model[1]. This clearly demonstrates that our model was more effective in reducing the number of clicks for the user. This result is not surprising because when using our model the user does not have to perform more clicks to reach the folder as evident

[1]In their work, Bao et.al compared their method with the windows default prediction and established that their model was signifi cantly more effective than the default predictions in reducing the cost of the user.

from the figure. These results indicate that our decision-theoretic model is effective in reducing the effort of the user in searching for his files.

Open problems and Extensions

In this section, we try to present the list of assumptions of the current framework and attempt to remove 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. In this section, we attempt to motivate the problem of partial observability of the user and extend our formulation to handle the partial observability of the user.

Consider for example an 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. The current model has to be extended to capture the partial observability of 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 agent can assist the user in his exploration.

Currently, the system would try to infer the user's goals given his actions. If the user is opening some doors for exploration, 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 rational 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 agent'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 modelling 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 Assitant 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 fucntion 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 agent can observe the world states and the agent's actions. The source of the assitant's uncertainty is the user's goal.

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.* 2006) 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). We feel that it is 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 use 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 can be 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 an 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 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 the time of the day, 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 to achieve another goal. Our system currently computes $P(goal \mid s, a)$ 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 modelling. 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 handcoded 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 capture the goal change 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. Though we do not expect huge changes to our goal estimation and action selection procedures, at this point 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 agent 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.

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. Our current formulation assumes a simple goal structure. Typically, users in real-world decompose a larger problem into a set of smaller ones and aim to achieve them. For instance, 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 acieve these subgoals individually.

In our current framework, we are not considering hierarchical goals although we have noted in our earlier work

that it will not be hard to incorporate hierarchies in our setting. The major change would be to the goal estimation part where the goal structure would be hierarchical. We could use ideas similar to hierarchical HMMs or abstract HMMs to perform the goal estimation.

We feel that extending our framework to a hierarchical setting is critical for solving several of the problems that were listed above. Using goal hierarchies could make it possible for the user and the assistant to act in parallel. This can be achieved by allowing the user and the assistant to perform different subgoals simultaneously. There has been a substantial amount of research in the area of hierarchical reinforcement learning that propose several abstraction methods for handling large state spaces (Dietterich 2000; Sutton, Precup, & Singh 1999; Parr & Russell 1997). These abstraction methods can be used with the hierarchical goal structure to efficiently solve the POMDP. Also, it appears that the problem of user changing his goals could also be handled efficiently by the use of hierarchies for the goal structure. Another problem which we have not discussed in this paper but is very important to address is the possibility of the user forgetting subgoals. For instance, while submitting the proposal, the user might forget to attach his document. It would very useful if the assistant can infer that the user has forgotten to attach the document and prevent the email from being sent and assist the user in attaching the document. Including the hierarchical goal structure seems to make it natural to handle the possibility of user forgetting a subgoal.

Conclusion

In this work, we have two significant contributions: one to evaluate our decision-theoretic framework and the other to identify the open problems in our framework and present it to the Intelligent Agents community. We showed that our model was very effective in reducing the cost of the user while accesing the appropriate files. We are currently working on adopting some knowledge about the tasks in which the user is working on to increase the percentage of zero click predictions. We are also currently working on extending the model to incorporate hierarchies of the goal structure to handle the some of the problems that we outlined. One of our goals is to evaluate our myopic heuristics on more complex domains where solving the POMDPs would be impractical.

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*.

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

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. Tasktracer: A desktop environment to support multi-tasking knowledge workers. In *Proceedings of IUI*.

Fern, A.; Natarajan, S.; Judah, K.; and Tadepall1, P. 2006. 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.; J.McDermott; and Zabowski, D. 1994. Experience with a learning personal assistant. *Communications of the ACM* 37(7):80–91.

Parr, R., and Russell, S. 1997. Reinforcement learning with hierarchies of machines. In Jordan, M. I.; Kearns, M. J.; and Solla, S. A., eds., *Advances in Neural Information Processing Systems*, volume 10. The MIT Press.

Sutton, R. S.; Precup, D.; and Singh, S. P. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence* 112(1-2):181–211.

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