A Decision-Theoretic Model of Assistance

Alan Fern AFERN@EECS.OREGONSTATE.EDU
Sriraam Natarajan NATARASR@EECS.OREGONSTATE.EDU
Kshitij Judah JUDAHK@EECS.OREGONSTATE.EDU
Prasad Tadepalli TADEPALL@EECS.OREGONSTATE.EDU
School of EECS, Oregon State University
Corvallis, OR 97331 USA

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

1. 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 (Myers et al., 2007). However, most of this work has been fine-tuned to the particular application domains. In this paper, we describe and evaluate a more comprehensive framework for designing intelligent assistants.

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

The first contribution of this work is to formulate the problem of selecting assistive actions as an assistant POMDP, which jointly models the application environment along with the agent's policy and hidden goals. A key feature of this approach is that it explicitly reasons about the environment

and agent, which provides the potential flexibility for assisting in ways unforeseen by the developer as new situations are encountered. Thus, the developer need not design a hand-coded assistive policy for each of a set of preconceived application scenarios. Instead, when using our framework, the burden on the developer is to provide a model of the application domain and agent, or alternatively a mechanism for learning one or both of these models from experience. Our framework then utilizes those models in an attempt to compute, in any situation, whether assistance could be beneficial and if so what assistive action to select.

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

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

A fourth contribution of our work is to evaluate our framework in three domains. First we consider two game-like computer environments using 12 human subjects. The results in these domains show that the assistants resulting from our framework substantially reduce the amount of work performed by the human subjects. We also consider a more realistic domain, the folder navigator (Bao, Herlocker, & Dietterich, 2006) of the Task Tracer project. In the folder-navigator domain, the user navigates the directory structure searching for a particular location to open or save a file, which is unknown to the assistant. The assistant has the ability to take actions that provide "short cuts" by showing the user a set of three potential destination folders. Bao et al. (2006) formulated the problem in a supervised learning framework and applied a cost-sensitive algorithm to predict the 3 most relevant folders at the beginning of the navigation process. We model this problem in our framework, which allows for the assistant to recommend folders at any point during navigation, not just at the beginning. The results show that our generic assistant framework compares favorably to the hand-coded solution of Bao et al. (2006).

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

2. Problem Setup

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

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

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

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

Before proceeding it is worth reviewing some of the assumptions of the above formulation and the potential implications. For simplicity we have assumed that the environment is modeled as an MDP, which implies full observability of the world state. This choice is not fundamental to our framework and one can imagine relatively straightforward extensions of our techniques that model the environment as a partially observable MDP (POMDP) where the world states are not fully observable. We have also assumed that the agent is modeled as a memoryless/reactive policy that gives a distribution over actions conditioned on only the current world state and goal. This assumption is also not fundamental to our framework and one can also extend it to include more complex models of the user, for example, that include hierarchical goal structures. Such an extension has recently been explored (Natarajan, Tadepalli, & Fern, 2007).

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

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

3. The Assistant POMDP

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

dealing with both the problem of resolving uncertainty about the underlying MDP state based on observations and selecting actions to help better identify the state and/or make progress toward the goal.

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

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

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

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

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

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

- The **cost model** C' reflects the costs of agent and assistant actions in the MDP. For all actions a except for **noop** we have that $C'((w, g), a) = C(w, a)$. Otherwise we have that

$C'((w,g),\textbf{noop}) = E[C(w,a)]$, where a is a random variable distributed according to $\pi(\cdot|w,g)$. That is, the cost of a **noop** action is the expected cost of the ensuing agent action.

- The **initial state distribution** I' assigns the state (w,g) probability $I(w)G_0(g)$, which models the process of independently selecting an initial state and goal for the agent at the beginning of each episode.
- The **observation set** O' is equal to $W \times (A \cup A')$, that is all pairs of world states and agent/assistant actions.
- The **observation distribution** μ' is deterministic and reflects the fact that the assistant can only directly observe the world state and actions. For the **noop** action in state (w,g) leading to state (w',g), the observation is (w',a) where $a \in A$ is the action executed by the agent immediately after the **noop**. Note that the observations resulting from **noop** actions are informative for inferring the agent's goal since they show the choice of action selected by the the agent in a particular state, which depends on the goal. For all other assistant actions the observation is equal to the W component of the state and the assistant action, i.e. $\mu'((w,g),a,(w',g)) = (w',a)$. Note that the observations resulting from non-noop actions are not informative with respect to inferring the agent's goals.

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

There are two main obstacles to solving the assistant POMDP and in turn the assistant design problem. First, in many scenarios, initially the assistant POMDP will not be directly at our disposal since we will lack accurate information about the agent policy π and/or the goal distribution G_0. This is often due to the fact that the assistant will be deployed for a variety of initially unknown agents. Rather, we will often only be given a definition of the environment MDP and the possible set of goals. As described in the next section, our approach to this difficulty is to utilize an approximate assistant POMDP by estimating π and G_0. Furthermore, we will also describe a bootstrapping mechanism for learning these approximations quickly. The second obstacle to solving the assistant POMDP is the generally high computational complexity of finding policies for POMDPs. To deal with this issue Section 5 considers various approximate techniques for efficiently solving the assistant POMDP.

4. Learning the Assistant POMDP

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

Relaxing the assumption that we are provided with the set of possible goals is more problematic in our current framework. As we will see in Section 5, our solution methods will all depend on knowing this set of goals and it is not clear how to learn these from observations, since the goals, unlike states and actions, are not directly observable to the assistant. Extending our framework so that the assistant can automatically infer the set of possible user goals, or allow the user to define their own goals, is an interesting future direction. We note, however, it is often possible for a designer to enumerate a set of user goals before deployment that while perhaps not complete, allow for useful assistance to be provided.

4.1 Maximum Likelihood Estimates

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

4.2 Model-Based Bootstrapping

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

Given the "near rationality assumption", we initialize the estimate of the agent's policy to a prior that is biased toward more optimal agent actions. To do this we will consider the environment MDP with the assistant actions removed and solve for the Q-function $Q(a,w,g)$ using MDP planning techniques. The Q-function gives the expected cost of executing agent action a in world state w and then acting optimally to achieve goal g using only agent actions. In a world without an assistant,

a rational agent would always select actions that maximize the Q-function for any state and goal. Furthermore, a close-to-rational agent would prefer actions that achieve higher Q-values to highly suboptimal actions. We first define the Boltzmann distribution, which will be used to define our prior,

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

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

In our experiments, we found that this prior provides a good initial proxy for the actual agent policy, allowing for the assistant to be immediately useful. Further updating of the posterior tunes the assistant better to the peculiarities of a given agent. For example, in many cases there are multiple optimal actions and the posterior will come to reflect any systematic bias for equally good actions that an agent has. Computationally the main obstacle to this approach is computing the Q-function, which needs to be done only once for a given application domain since the environment MDP is constant. Using dynamic programming this can accomplished in polynomial time in the number of states and goals. When this is not practical, a number of alternatives exist including the use of factored MDP algorithms (Boutilier et al., 1999), approximate solution methods (Boutilier et al., 1999; Guestrin et al., 2003), or developing domain specific solutions.

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

5. Solving the Assistant POMDP

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

5.1 Overview

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

the assistant or agent). Given O_t and M our goal is to compute an assistant action whose value is close to optimal.

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

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

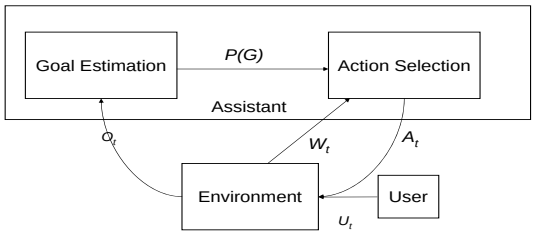

Figure 1: Depiction of the assistant architecture. The user/agent has a hidden goal and selects actions U_t that cause the environment to change world state W_t, typically moving closer to the goal. The assistant (upper rectangle) is able to observe the world state along with the observations generated by the environment, which in our setting contain the user/agent actions along with the world state. The assistant is divided into two components. First, the goal estimation component computes a posterior over agent goals $P(G)$ given the observations. Second, the action selection component uses the goal distribution to compute the best assistive action A_t via a combination of bounded search and myopic heuristic computation. The best action might be **noop** in cases where none of the other assistive actions has higher utility for the user.

With the above motivation, our assistant architecture, depicted in Figure 1, alternates between *goal estimation* and *action selection* as follows:

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

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

The key element of the architecture is the computation of the myopic heuristics. On top of this heuristic, we can optionally obtain non-myopic behavior via search by building a look-ahead sparse-sampling tree. Our experiments show that such search can improve performance by a small margin at a significant computational cost. We note that the idea of utilizing myopic heuristics to select actions in POMDPs is not new, see for example (Cassandra, 1998; Geffner & Bonet, 1998), and similar methods have been used previously with success in applications such as computer bridge (Ginsberg, 1999). The main contribution here is to note that this approach seems particularly well suited to the assistant POMDP and also in later sections to suggest some efficiently computable heuristics that are specifically designed for the assistant POMDP framework. Below we describe the goal estimation and action selection operations in more detail.

5.2 Goal Estimation

Given an assistant POMDP with agent policy π and initial goal distribution G_0, our objective is to maintain the posterior goal distribution $P(g|O_t)$, which gives the probability of the agent having goal g conditioned on observation sequence O_t. Note that since we have assumed that the assistant cannot affect the agent's goal, only observations related to the agent's actions are relevant to the posterior. Given the agent policy π, it is straightforward to incrementally update the posterior $P(g|O_t)$ upon each of the agent's actions by referring to the Bayesian network of Figure 2. The node g refers to the current goal of the user and is the only hidden variable that is distributed according to the current goal posterior, A_t denotes the agent's action at time t, and W_t refers to the world state at time-step t. The agent action A_t is distributed according to the agent's policy π.

At the beginning of each episode we initialize the goal distribution $P(g|O_0)$ to G_0. On timestep t of the episode, if o_t does not involve an agent action, then we leave the distribution unchanged. Otherwise, when the agent selects action a in state w, we update the posterior according to $P(g|O_t) = (1/Z) \cdot P(g|O_{t-1}) \cdot \pi(a|w,g)$, where Z is a normalizing constant. That is, the distribution is adjusted to place more weight on goals that are more likely to cause the agent to execute action a in w. The accuracy of goal estimation relies on how well the policy π learned by the assistant reflects the true agent policy. As described above, we use a model-based bootstrapping approach for estimating π and update this estimate at the end of each episode. Provided that the agent is close to optimal, as in our experimental domains, this approach can lead to rapid goal estimation, even early in the lifetime of the assistant.

We have assumed for simplicity that the actions of the agent are directly observable. In some domains, it is more natural to assume that only the state of the world is observable, rather than the actual action identities. In these cases, after observing the agent transitioning from w to w' we can

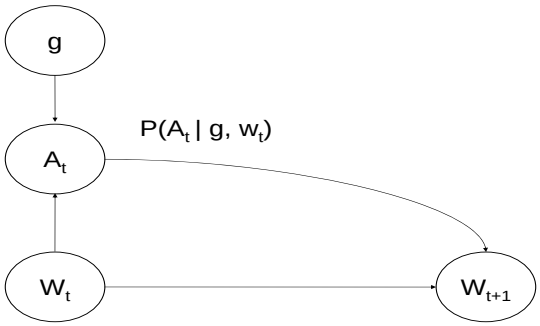

Figure 2: The Bayes net used to compute the posterior distribution over the agent's goals. The node A_t represents the agent action that is conditioned on the agent's goal G and current world state W_t. In the simplest setting G is the only hidden variable with both the agent actions and states being observable. In other cases, the agent action may also be hidden, with only the state transition being directly observable.

use the MDP transition function T to marginalize over possible agent actions yielding the update,

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

5.3 Action Selection

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

5.3.1 MYOPIC HEURISTICS

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

the Q-function of $M(g)$ by $Q_g(w, a)$, which is the expected cost of executing action a and then following the optimal policy.

Our first myopic heuristic is simply the expected Q-value of an action over assistant MDPs. This heuristic has also been called the Q_{MDP} method in (Cassandra, 1998). The heuristic value for assistant action a in state w given observations O_t is

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

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

The primary computational complexity of computing H is to solve the assistant MDPs for each goal in order to obtain the Q-functions. Technically, since the transition functions of the assistant MDPs depend on the approximate agent policy π, we must re-solve each MDP after updating the π estimate at the end of each episode. However, using incremental dynamic programming methods such as prioritized sweeping (Moore & Atkeson, 1993) can alleviate much of the computational cost. In particular, before deploying the assistant we can solve each MDP offline based on the default agent policy given by the Boltzmann bootstrapping distribution described earlier. After deployment, prioritized sweeping can be used to incrementally update the Q-values based on the learned refinements we make to π.

When it is not practical to exactly solve the assistant MDPs, we may resort to various approximations. We consider two approximations in our experiments. One is to replace the user's policy to be used in computing the assistant MDP with a fixed default user policy, eliminating the need to compute the assistant MDP at every step. We denote this approximation by H_d. Another approximation uses the simulation technique of *policy rollout* (Bertsekas & Tsitsiklis, 1996) to approximate $Q_g(w, a)$ in the expression for H. This is done by first simulating the effect of taking action a in state w and then using π to estimate the expected cost for the agent to achieve g from the resulting state. That is, we approximate $Q_g(w, a)$ by assuming that the assistant will only select a single initial action followed by only agent actions. More formally, let $\bar{C}_n(\pi, w, g)$ be a function that simulates n trajectories of π achieving the goal from state w and then averaging the trajectory costs. The heuristic H_r is identical to $H(w, a, O_t)$ except that we replace $Q_g(w, a)$ with the expectation $\sum_{w' \in W} T(w, a, w') \cdot \bar{C}(\pi, w', g)$. We can also combine both of these heuristics, using a fixed default user policy and policy rollouts, which we denote by $H_{d,r}$.

5.3.2 SPARSE SAMPLING

All of the above heuristics can be used to greedily select assistant actions, resulting in purely myopic action-selection strategies. In cases where it is beneficial to include some amount of non-myopic reasoning, one can combine these heuristics with shallow search in the belief space of the assistant MDP. For this purpose we utilize depth d bounded sparse sampling trees (Kearns, Mansour, & Ng,

1999) to compute an approximation to the Q-function for a given belief state (w_t, O_t), denoted by $Q^d(w_t, a, O_t)$. Given a particular belief state, the assistant will then select the action that maximizes Q^d. Note that for convenience we represent the belief state as a pair of the current state w_t and observation history O_t. This is a lossless representation of the belief state since the posterior goal distribution can be computed exactly from O_t and the goal is the only hidden portion of the POMDP state.

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

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

$$V^d(w, O) = \min_a Q^d(w, a, O) \quad (2)$$

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

Sparse sampling approximates the above equations by averaging a set of b samples of successor belief states to approximate the expectation. The sparse-sampling pseudo-code is presented in Table 1.Given an input belief state (w, O), assistant action a, heuristic H, depth bound d, and sampling width b the algorithm returns (an approximation of) $Q^d(w, a, O)$. First, if the depth bound is equal to zero the heuristic value is returned. Otherwise b samples of observations resulting from taking action a in belief state (w, O) are generated. In the case that the action is not **noop**, the observation is created by simulating the effect of the assistant action in the environment MDP and then forming an observation from the pair of resulting state and action a (recall that observations are pairs of states and previous actions). In the case that the assistant action is **noop**, the observation is generated by sampling an agent goal and an action and then simulating the environment MDP for the agent action. The observation is simply the pair of resulting state and agent action. Each observation $o_i = (w_i, a_i)$ corresponds to a new belief state $(w_i, [O; o_i])$ where $[O; o_i]$ is simply the concatenation of o_i to O. The code then recursively computes a value for each of these belief states by minimizing Q^d over all actions and then averages the results.

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

6. Experimental Results

In this section, we present the results of conducting user studies and simulations in three domains: two game-like environments and a folder predictor domain for an intelligent desktop assistant. In

Table 1: Pseudo-code for Sparse Sampling in the Assistant POMDP

- **Given:** heuristic function H, belief state (w, O), action a, depth bound d, sampling width b
- **Return:** an approximation $Q^d(w, a, O)$ of the value of a in belief state (w, O)

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

the user studies in the two game-like domains, for each episode, the user's and the assistant's actions were recorded. The ratio of the cost of achieving the goal with the assistant's help to the optimal cost without the assistant was calculated and averaged over the multiple trials for each user. We present similar results for the simulations as well. The third domain is a folder predictor domain, where we simulated the user and used one of our heuristics to generate the top 3 recommended folders for the user. We present the number of clicks required on an average for the user to reach his desired folder.

6.1 Doorman Domain

In the doorman domain, there is an agent and a set of possible goals such as collect *wood*, *food* and *gold*. Some of the grid cells are blocked. Each cell has four doors and the agent has to open the door to move to the next cell (see Figure 3). The door closes after one time-step so that at any time only one door is open. The goal of the assistant is to help the user reach his goal faster by opening the correct doors.

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

In this experiment, we evaluated the two heuristics: one where we fixed the user policy to the default policy in the assistant POMDP creation (H_d) and the second where we use the policy rollout

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

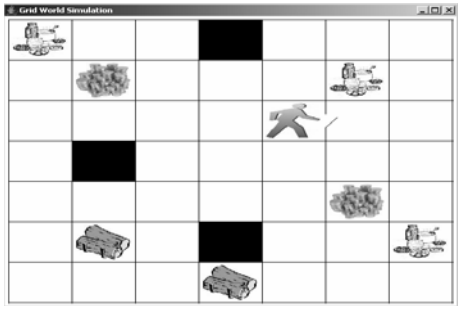

Figure 3: Doorman Domain. The agent's goal is to fetch a resource. The grid cells are separated by doors that must be opened before passing through.

The results of the user studies for the doorman domain are presented in Figure 2. The first two rows give cumulative results for the user study when actions are selected greedily according to H_r and H_d respectively. The table presents the total optimal costs (number of actions) for all trials across all users without the assistant *N*, and the costs with the assistant *U*, and the average of percentage cost savings (*1-(U/N)*) over all trials and over all the users[1].

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

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

We also conducted experiments using sparse sampling with non-trivial depths. We considered depths of $d = 1$ and $d = 2$ while using sampling widths of $b = 1$ or $b = 2$. The leaves of the sparse sampling tree are evaluated using H_r which simply applies rollout to the user policy. Hence sparse sampling of $d = 0$ and $b = 1$, would correspond to the heuristic H_r. For these experiments, we did not conduct user studies, due to the high cost of such studies, but simulated the human

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

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

Table 2: Results of experiments in the Doorman Domain. The first two rows of the table present the results of the user studies while the rest of the table presents the results of the simulation.

users by choosing actions according to policies learned from their observed actions. The results are presented in the last 5 rows of Table 2. We see that sparse sampling increased the average run time by an order of magnitude, but is able to produce a reduction in average cost for the user. This result is not surprising in hindsight, for in the simulated experiments, sparse sampling is able to sample from the exact user policy (i.e. it is sampling from the learned policy, which is also being used for simulations). These results suggest that a small amount of non-myopic reasoning can have a positive benefit with a substantial computation cost. Note, however, that the bulk of the benefit realized by the assistant can be obtained without such reasoning, showing that the myopic heuristics are well-suited to this domain.

6.2 Kitchen Domain

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

There are 8 different recipes. The state consists of the location of each of the ingredient (bowl/shelf/table), the mixing state and temperature state of the ingredient (if it is in the bowl) and the door that is open. The state also includes the action history to preserve the ordering of the plans for the recipes. The user's actions are: open the doors, fetch the ingredients, pour them into the bowl, mix, heat and bake the contents of the bowl, or replace an ingredient back to the shelf. The assistant can perform all user actions except for pouring the ingredients or replacing an ingredient back to the shelf. The cost of all non-pour actions is -1. Experiments were conducted on 12 human subjects. Unlike in the doorman domain, here it is not necessary for the assistant to wait at every alternative time step. The assistant continues to act until the **noop** becomes the best action according to the heuristic.

This domain has a large state space and hence it is not possible to update the user policy after every trajectory. The two heuristics that we compare both use the default user policy. The second heuristic in addition uses policy rollout to compare the actions. In other words, we compare H_d and $H_{d,r}$. The results of the user studies are shown in top part of the Table 3. The total cost of

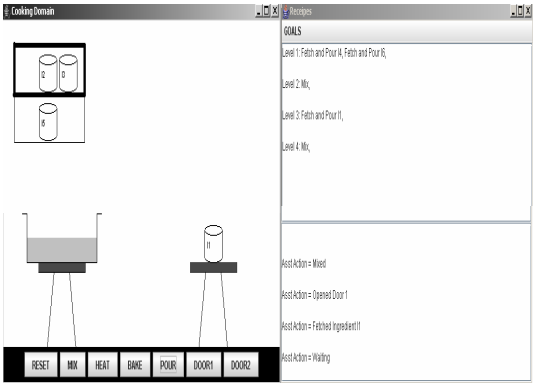

Figure 4: The kitchen domain. The user is to prepare the dishes described in the recipes on the right. The assistant's actions are shown in the bottom frame.

user's optimal policy, the total cost when the assistant is present, and the average ratio of the two are presented. The number of user actions was summed over 15 users and the cumulative results are presented. It can be observed that $H_{d,r}$ performs better than H_d. It was observed from the experiments that the $H_{d,r}$ technique was more aggressive in choosing non-noop actions than the H_d, which would wait until the goal distribution is highly skewed toward a particular goal.

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

Table 3: Results of experiments in the Kitchen Domain. The first two rows of the table present the results of the user studies while the last 5 rows present the results of the simulation.

We compared the use of sparse sampling and our heuristic on simulated user trajectories for this domain as well (see the last 5 rows of Table 3). It can be observed that the total number of user actions is much higher for the simulations than the user studies due to the fact that user studies are costly. The simulations had to consider a much larger state-space than the one used in the user studies. Because of this, the policies learned on simulations are significantly cheaper than in the user studies, although they took more time to execute. There is no significant difference between the solution quality of rollouts and sparse sampling on simulations, showing that our myopic heuristics

are performing as well as sparse sampling with much less computation. Sparse sampling with higher depths requires an order of magnitude more computation time when compared to the rollout.

6.3 Folder Predictor

In this section, we present the evaluation of our framework on a real-world domain. As a part of the Task Tracer project (Dragunov, Dietterich, Johnsrude, McLaughlin, Li, & Herlocker, 2005), researchers developed a file location system called *folder predictor* (Bao et al., 2006). The idea behind the folder predictor is that by learning about the user's file access patterns, the assistant can help the user with his file accesses by predicting the folder in which the file has to be accessed or saved.

In this setting, the goal of the folder predictor is to minimize the number of clicks of the user. The predictor would choose the top three folders that would minimize the cost and then append them to the UI (shown in ovals in Figure 5). Also, the user is taken to the first recommended folder. So if the user's target folder is the first recommended folder, the user would reach the folder in zero clicks and reach the second or the third recommended folder in one click. The user can either choose one of the recommendations or navigate through the windows folder hierarchy if the recommendations are not relevant.

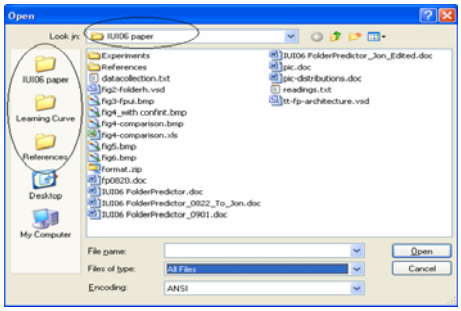

Figure 5: Folder predictor (Bao et al., 2006).

Bao et al. considered the problem as a supervised learning problem and implemented a cost-sensitive algorithm for the predictions with the cost being the number of clicks of the user (Bao et al., 2006). But, their algorithm does not take into account the response of the user to their predictions. For instance, if the user chooses to ignore the recommended folders and navigates the folder hierarchy, they do not make any re-predictions. This is due to the fact that their model is a one-time prediction and does not consider the user responses. Also, their algorithm considers a restricted set of previously accessed folders and their ancestors as possible destinations. This precludes handling the possibility of user accessing a new folder.

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

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

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

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

The state space consists of 4 parts: the current folder that the user is accessing and the three recommendations two of which are unordered. This would correspond to a state space of size $226 \times 225 \times \binom{224}{2}$. The action of the user is either to choose a recommended folder or select a different folder. The action of the assistant corresponds to choosing the top 3 folders and the action space is of size $225 \times \binom{224}{2}$. The cost in our case was the number of user clicks to the correct folder. In this domain, the assistant and the user's actions strictly alternate as the assistant revises its predictions after every user action. The prior distribution was initialized using the costs computed by the model developed in (Bao et al., 2006).

We applied the decision theoretic model to the data set. For each request, our assistant would make the prediction using the $H_{d,r}$ heuristic (which uses the default user policy and the rollout method) and then the user is simulated. The user would accept the recommendation if it shortens his path to the goal else would act according to his optimal policy. The user here is considered close to optimal, which is not unrealistic in the real world. To compare our results, we also used the model developed by Bao et al. in the data set and present the results in Table 4.

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

Table 4: Results of the experiments in the folder predictor domain. The entry in the top left hand cell is the performance of the current Task Tracer, while the one in the bottom right hand cell is the performance of the decision-theoretic assistant

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

new folder. It can be observed that either of the modifications yields a lower cost than the original algorithm, but combining the two changes is significantly more effective.

7. Discussion and Related Work

Our work is inspired by the growing interest and success in building useful software assistants (Myers et al., 2007). Some of this effort is focused on building desktop assistants that help with tasks such as email filtering, on-line diagnostics, and travel planning. Each of these tasks typically requires designing a software system around specialized technologies and algorithms. For example, email filtering is typically posed as a supervised learning problem (Cohen, Carvalho, & Mitchell, 2004), travel planning combines information gathering with search and constraint propagation (Ambite, Barish, Knoblock, Muslea, Oh, & Minton, 2002), and printer diagnostics is formulated as Bayesian network inference (Skaanning, Jensen, & Kjaerulff, 2000). Unfortunately the plethora of systems and approaches lacks an overarching conceptual framework, which makes it difficult to build on each others' work. In this paper, we argue that a decision-theoretic approach provides such a common framework and allows the design of systems that respond to novel situations in a flexible manner reducing the need for pre-programmed behaviors. We formulate a general version of the assistantship problem that involves inferring the user's goals and taking actions to minimize the expected costs.

Earlier work on learning apprentice systems focused on learning from the users by observation (Mahadevan, Mitchell, Mostow, Steinberg, & Tadepalli, 1993; Mitchell, Caruana, Freitag, J.McDermott, & Zabowski, 1994). This work is also closely related to learning from demonstration or programming by demonstration (Atkeson & Schaal, 1997; Cypher, 1993; Lau, Wolfman, Domingos, & Weld, 2003). The emphasis in these systems is to provide an interface where the computer system can unobtrusively observe the human user doing a task and learn to do it by itself. The human acts both as a user and as a teacher. The performance of the system is measured by how quickly the system learns to imitate the user, i.e., in the supervised learning setting. Note that imitation and assistance are two different things in general. While we expect our secretaries to learn about us, they are not typically expected to replace us. In our setting, the assistant's goal is to reduce the expected cost of user's problem solving. If the user and the assistant are capable of exactly the same set of actions, and if the assistant's actions cost nothing compared to the user's, then it makes sense for the assistant to try to completely replace the human. Even in this case, the assistantship framework is different from learning from demonstration in that it still requires the assistant to *infer* the user's goal from his actions before trying to achieve it. Moreover, the assistant might learn to solve the goal by itself by reasoning about its action set rather than by being shown examples of how to do it by the user. In general, however, the action set of the user and the assistant may be different, and supervised learning is not appropriate. For example, this is the case in our Folder predictor. The system needs to decide which set of folders to present to the user, and the user needs to decide which of those to choose. It is awkward if not impossible to formulate this problem as supervised learning or programming by demonstration.

Taking the decision-theoretic view helps us approach the assistantship problem in a principled manner taking into account the uncertainty in the user's goals and the costs of taking different actions. The assistant chooses an action whose expected cost is the lowest. The framework naturally prevents the assistant from taking actions (other than noop) when there is no assistive action which is expected to reduce the overall cost for the user. Rather than learning from the user how to behave,

in our framework the assistant learns the user's policy. This is again similar to a secretary who learns the habits of his boss, not so much to imitate her, but to help in the most effective way. In this work we assumed that the user MDP is small enough that it can be solved exactly given the user's goals. This assumption may not always be valid, and it makes sense in those cases to learn from the user how to behave. It is most natural to treat this as a case where the user's actions provide exploratory guidance to the system (Clouse & Utgoff, 1992; Driessens, 2002). This gives an opportunity for the system to imitate the user when it knows nothing better and improve upon the user's policy when it can.

There have been other personal assistant systems that are based on POMDP models. However, these systems are formulated as domain-specific POMDPs and solved offline. For instance, the COACH system helped people suffering from Dementia by giving them appropriate prompts as needed in their daily activities (Boger et al., 2005). They use a plan graph to keep track of the user's progress and then estimate the user's responsiveness to determine the best prompting strategy. A distinct difference from our approach is that there is only a single fixed goal of washing hands, and the only hidden variable is the user responsiveness. Rather, in our formulation the goal is a random variable that is hidden to the assistant. We note, that a combination of these two frameworks would be useful, where the assistant infers both the agent goals and other relevant hidden properties of the user, such as responsiveness.

In *Electric Elves*, the assistant takes on many of the mundane responsibilities of the human agent including rescheduling meetings should it appear that the user is likely to miss it. Again a domain-specific POMDP is formulated and solved offline using a variety of techniques. In one such approach, since the system monitors users in short regular intervals, radical changes in the belief states are usually not possible and are pruned from the search space (Varakantham, Maheswaran, & Tambe, 2005). Neither exact nor approximate POMDP solvers are feasible in our online setting, where the POMDP is changing as we learn about the user, and must be repeatedly solved. They are either too costly to run (Boger et al., 2005), or too complex to implement as a baseline, e.g., Electric Elves (Varakantham et al., 2005). Our experiments demonstrate simple methods such as one-step look-ahead followed by rollouts would work well in many domains where the POMDPs are solved online. In a distinct but related work (Doshi, 2004), the authors introduce the setting of interactive POMDPs, where each agent models the other agent's beliefs. Clearly, this is more general and more complex than ordinary POMDPs. Our model is simpler and assumes that the agent is oblivious to the presence and beliefs of the assistant. While the simplified model suffices in many domains, relaxing this assumption without sacrificing tractability would be interesting.

Reinforcement Learning has been explored before in specific interactive settings such as in dialogue management in spoken dialogue systems (Singh, Litman, Kearns, & Walker, 2002; Walker, 2000). For example, the goal of the NJFun system in (Singh et al., 2002) is to learn a policy for optimally interacting with the user who is trying to query a database in spoken natural language. In particular the system decides when to take initiative in the dialogue, and whether to confirm its understanding of the speaker's utterance. The user's speech is processed through an automatic speech recognition (ASR) system which produces a noisy belief state which is compressed into a set of derived features. A dialogue policy is then learned over this set of derived features. The approach taken is to first learn a user's model through an exploratory phase, and then use offline value-iteration on the model to learn an optimal policy. This work can be viewed as a specific instance of our assistantship framework to dialogue management where the state/goal estimation is done by a separate ASR system and the action selection is done by offline reinforcement learning.

Our work is also related to on-line plan recognition and can be naturally extended to include hierarchies as in the hierarchical versions of HMMs (Bui, Venkatesh, & West, 2002) and PCFGs (Pynadath & Wellman, 2000). Blaylock and Allen describe a statistical approach to goal recognition that uses maximum likelihood estimates of goal schemas and parameters (Blaylock & Allen, 2004). These approaches do not have the notion of cost or reward. By incorporating plan recognition in the decision-theoretic context, we obtain a natural notion of optimal assistance, namely maximizing the expected utility.

There has been substantial research in the area of user modeling. Horvitz et.al took a Bayesian approach to model whether a user needs assistance based on user actions and attributes and used it to provide assistance to user in a spreadsheet application (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 compute the expected utility of assisting the user. It would be interesting to explore these kind of user models in our system to take into account the user's intentions and attitudes while computing the optimal policy for the assistant.

8. Summary and Future Work

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

One future direction is to consider more complex domains where the assistant is able to do a series of activities in parallel with the agent. Another possible direction is to assume hierarchical goal structure for the user and do goal estimation in that context. Recently, the assistantship model was extended to hierarchical and relational settings (Natarajan et al., 2007) by including parameterized task hierarchies and conditional relational influences as prior knowledge of the assistant. This prior knowledge would relax the assumption that the user MDP can be solved tractably. This knowledge was compiled into an underlying Dynamic Bayesian network, and Bayesian network inference algorithms were used to infer a distribution of user's goals given a sequence of her atomic actions. The parameters for the user's policy were estimated by observing the users' actions.

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

Acknowledgements

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 Con-

tract 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., & Minton, S. (2002). Getting from here to there: Interactive planning and agent execution for optimizing travel. In *IAAI*, pp. 862–869.

Atkeson, C. G., & Schaal, S. (1997). Learning tasks from a single demonstration. *IEEE Transactions on Robotics and Automation*, 1706–1712.

Bao, X., Herlocker, J. L., & Dietterich, T. G. (2006). Fewer clicks and less frustration: reducing the cost of reaching the right folder. In *In Proceedings of IUI*, pp. 178–185.

Bertsekas, D. P., & Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Athena Scientific.

Blaylock, N., & Allen, J. F. (2004). Statistical goal parameter recognition. In *ICAPS*.

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

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

Bui, H., Venkatesh, S., & West, G. (2002). Policy recognition in the abstract hidden markov models. *JAIR*, *17*.

Cassandra, A. R. (1998). *Exact and approximate algorithms for partially observable markov decision processes*. Ph.D. thesis.

Clouse, J. A., & Utgoff, P. E. (1992). A teaching method for reinforcement learning. In *Proceedings of the Ninth International Workshop on Machine Learning*, pp. 92–110.

Cohen, W. W., Carvalho, V. R., & Mitchell, T. M. (2004). Learning to classify email into speech acts. In *Proceedings of Empirical Methods in NLP*.

Cypher, A. (1993). *Watch What I Do: Programming by Demonstration*. MIT Press.

Doshi, P. (2004). A particle filtering algorithm for interactive pomdps. In *In Proceedings of Conference on Modeling Other Agents from Observations*, Columbia University USA.

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

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

Geffner, H., & Bonet, B. (1998). Solving large pomdps using real time dynamic programming. In *Working notes. Fall AAAI symposium on POMDPs*.

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

Guestrin, C., Koller, D., Parr, R., & Venkataraman, S. (2003). Efficient solution algorithms for factored MDPs. *JAIR*, 399–468.

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

Hui, B., & Boutilier, C. (2006). Who's asking for help?: a bayesian approach to intelligent assistance.. In *In Proceedings of IUI*, pp. 186–193.

Kearns, M. J., Mansour, Y., & Ng, A. Y. (1999). A sparse sampling algorithm for near-optimal planning in large markov decision processes. In *IJCAI*.

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

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

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

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

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

Natarajan, S., Tadepalli, P., & Fern, A. (2007). A relational hierarchical model for decision-theoretic assistance. In *Proceedings of 17th Annual International Conference on Inductive Logic Programming*.

Pynadath, D. V., & Wellman, M. P. (2000). Probabilistic state-dependent grammars for plan recognition. In *UAI*, pp. 507–514.

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

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

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

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