Journal of Artificial Intelligence Research 50 (2014) Submitted 10/13; published 05/14

A Decision-Theoretic Model of Assistance

Alan Fern AFERN@EECS.OREGONSTATE.EDU
School of EECS, Oregon State University, Corvallis, OR USA

Sriraam Natarajan NATARASR@INDIANA.EDU
SoIC, Indiana University, Bloomington, IN USA

Kshitij Judah JUDAHK@EECS.OREGONSTATE.EDU
School of EECS, Oregon State University, Corvallis, OR USA

Prasad Tadepalli TADEPALL@EECS.OREGONSTATE.EDU
School of EECS, Oregon State University, Corvallis, OR USA

Abstract

There is a growing interest in intelligent assistants for a variety of applications from sorting email to helping people with disabilities to do their daily chores. In this paper, we formulate the problem of intelligent assistance in a decision-theoretic framework, and present both theoretical and empirical results. We first introduce a class of POMDPs called hidden-goal MDPs (HGMDPs), which formalizes the problem of interactively assisting an agent whose goal is hidden and whose actions are observable. In spite of its restricted nature, we show that optimal action selection for HGMDPs is PSPACE-complete even for deterministic dynamics. We then introduce a more restricted model called helper action MDPs (HAMDPs), which are sufficient for modeling many real-world problems. We show classes of HAMDPs for which efficient algorithms are possible. More interestingly, for general HAMDPs we show that a simple myopic policy achieves a near optimal regret, compared to an oracle assistant that knows the agent's goal. We then introduce more sophisticated versions of this policy for the general case of HGMDPs that we combine with a novel approach for quickly learning about the agent being assisted. We evaluate our approach in two game-like computer environments where human subjects perform tasks, and in 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

Personalized AI systems that interactively assist their human users have received significant attention in recent years (Yorke-Smith, Saadati, Myers, & Morley, 2012; Lieberman, 2009; Myers, Berry, Blythe, Conleyn, Gervasio, McGuinness, Morley, Pfeffer, Pollack, & Tambe, 2007). However, an overarching formal framework of interactive assistance that captures all these different systems and provides a theoretical foundation is largely missing. In this paper we address this lacuna by introducing a general framework of decision-theoretic assistance, analyzing the problem complexity under different assumptons, proposing different heuristic solutions, and evaluating their effectiveness.

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. In real applications, this requires that the assistant be able to handle uncertainty about the environment and the agent, to reason about varying

©2014 AI Access Foundation. All rights reserved.

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 a class of partially observable Markov decision processes (POMDPs) called Hidden Goal MDPs (HGMDPs), 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 the 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 preconceived application scenario. Instead, when using our framework, the burden on the developer is to provide a model of the application domain and the agent, or alternatively a mechanism for learning one or both of these models from experience. Our framework then uses those models in an attempt to compute, in any situation, whether assistance could be beneficial and if so what assistive action to select.

The second contribution of this work is to analyze the properties of this formulation. Despite the restricted nature of HGMDPs, the complexity of determining if an HGMDP has a finite-horizon policy of a given value is PSPACE-complete even in deterministic environments. This motivates a more restricted model called Helper Action MDP (HAMDP), where the assistant executes a helper action at each step. The agent is obliged to accept the helper action if it is helpful for its goal and receives a reward bonus (or cost reduction) by doing so. Otherwise, the agent can continue with its own preferred action without any reward or penalty to the assistant. We show classes of this problem that are complete for PSPACE and NP. We also show that for the class of HAMDPs with deterministic agents there are polynomial time algorithms for minimizing the expected and worst-case regret relative to an oracle assistant that knows the goal of the agent. Further, we show that the optimal worst case regret can be characterized by a graph theoretic property called the tree rank of the corresponding all-goals policy tree and can be computed in linear time.

In principle, given a HGMDP, 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 HGMDPs, each of which needs to be solved. A third contribution of our work is a set of myopic action selection mecahnisms that approximate the optimal policy. For HAMDPs, we analyze a myopic heuristic and show that it has a regret which is upper bounded by the entropy of the goal distribution for HAMDPs. Furthermore we give a variant of this policy that is able to achieve worst-case and expected regret that is logarithmic in the number of goals without any prior knowledge of the goal distribution. We also describe two other approaches that are based on a combination of explicit goal estimation, myopic heuristics, and bounded search and are more generally applicable to HGMDPs.

In order for the above approach to be useful, the HGMDP must incorporate a reasonably accurate model of the agent being assisted. A fourth 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.

The final contribution of our work is to evaluate our framework in three domains. First we consider two game-like computer environments with 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 this 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 job of the assistant is to predict the user's destination folder and take actions that provide "short cuts" to reach it. The results show that our generic assistant framework compares favorably to the hand-coded solution of Bao et al.

The remainder of this paper is organized as follows. In the next section, we introduce our formal problem setup in terms of HGMDPs, followed by an analysis of computational complexity and guarantees for myopic heuristics in the case of HAMDPs. Next, we present our approximate solution approach for general HGMDPs 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. Hidden Goal Markov Decision Processes

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 consider an episodic problem setting where at the beginning of each episode the agent begins in some world state and selects a goal from a finite set of possible goals. 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 agent may possibly navigate to. Importantly, the assistant can fully observe the world state and the agent's actions, but cannot observe the goal of the agent. We model the interaction between the agent and assistant as sequential where the agent and assistant alternate turns, taking a single action per turn (possibly noop).[1] The episode ends when either the agent's or the assistant's action leads to a goal state. An immediate reward is accumulated after each action during the episode. The total reward of an episode is equal to the sum of the rewards obtained during the episode. Note that the available actions for the agent and assistant need not be the same and may have varying rewards. Since the assistant and the agent share the same objective, all rewards are viewed from the perspective of the agent. The objective of the assistant is to behave in a way that maximizes the expected total reward of an episode.

More formally, we model the above interaction via Hidden Goal Markov Decision Processes (HGMDPs). An HGMDP is an MDP in which the goal of the user is not observed while the rest of the environment is completely observed. An HGMDP is a tuple $\langle S, G, A, A', T, R, \pi, I_S, I_G \rangle$ where S is a set of states, G is a finite set of possible agent goals, A is the set of agent actions, and A' is the set of assistant actions. Typically A' will include a noop action, which allows the assistant to decide not to provide any assistance at a particular decision epoch. T is the transition function where $T(s, g, a, s')$ is the probability of a transition to state s' from s after taking action $a \in A \cup A'$ when the agent's goal is g. R is the reward function which maps $S \times G \times (A \cup A')$ to real values. π

1. We consider this strictly alternating turn model for simplicity. However, it is straightforward to use this model to capture interactions that are not strictly alternating, e.g. allowing the assistant or agent to take multiple actions in a row.

is the agent's policy that maps $S \times G$ to distributions over A and need not be optimal in any sense. I_S (I_G) is an initial state (goal) distribution.

An assistant policy for an HGMDP defines a distribution over actions given the sequence of preceding observations, i.e., the sequence of state action pairs. It is important that the assistant policy depends on the history rather than only the current state since the entire history can potentially provide evidence about the goal of the agent, which may be necessary for selecting an appropriate action. We must now define an objective function that will be used to evaluate the value of a particular assistant policy. We consider a finite-horizon episodic problem setting, where each HGMDP episode begins by drawing an initial state $s \sim I_S$ and a goal $g \sim I_G$. The process then alternates between the agent and the assistant executing actions (including noops) in the environment until the horizon or a terminal state is reached. The agent is assumed to select actions according to π. In many domains, a terminal goal state will be reached within the horizon, though in general, goals can have arbitrary impact on the reward function. The reward for the episode is equal to the sum of the rewards of the actions executed by the agent and assistant during the episode. The objective of the assistant is to reason about the HGMDP and observed state-action history in order to select actions that maximize the expected (or worst-case) total reward of an episode.

Before proceeding further it is worth reviewing some of the assumptions of the above formulation and the potential implications.

- **Partial Observability:** The definition of the HGMDP is very similar to the definition of a Partially Observable Markov Decision Process (POMDP). In fact, the HGMDP is a special case of the POMDP where the unobserved part of the state space consists of a single component which corresponds to the goal of the user. For simplicity we have assumed that the world state is fully observable. 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. In Section 3, we shed some light on the hardness of HGMDPs and describe some specialized heuristic solutions with performance guarantees.

- **Agent Policy:** 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 been explored previously (Natarajan, Tadepalli, & Fern, 2007).

- **Sequential Interaction:** 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, filter spam, sort email, etc., and "smart homes" that open doors, switch on appliances, and so on. Many opportunities for assistance in these domains are of the sequential variety. 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.

- **Goal-Dependent Transitions:** The dependence of the reward and policy on the goal allows the model to capture the agent's desires and behavior under each goal. The dependence of T on the goal is less intuitive and in many cases there will be no dependence when T is used only to model the dynamics of the environment. However, we allow goal dependence of T for generality of modeling. For example, it can be convenient to model basic communication actions of the agent as changing aspects of the state, and the result of such actions will often be goal dependent.

There are two main obstacles to solving the problem of intelligent assistance in our framework. First, in many scenarios, initially the HGMDP will not be directly at our disposal since we will lack accurate information about the agent policy π and/or the goal distribution I_G. This is often due to the fact that the assistant will be deployed for a variety of initially unknown agents. Rather, the assistant will find itself in an environment and be only given the possible set of goals. As described in Section 5, our approach to this difficulty is to learn an approximate HGMDP by estimating the agent policy π and the goal distribution I_G. Furthermore, we will also describe a bootstrapping mechanism for learning these approximations quickly. The second obstacle to solving the HGMDP is the generally high computational complexity of solving HGMDPs. To deal with this issue Section 4 considers various approximate techniques for efficiently solving the HGMDPs.

It should be mentioned that it is possible to provide assistance using simpler domain-specific engineered solutions. In particular, the domains we consider later could be solved using several less expensive solutions and do not require the machinery of HGMDPs. In fact, in one of our domains, we compare our model against an existing supervised learning method. The goal of our work is to provide a domain-independent framework that can potentially encapsulate several such assistant systems with the hope that it gives rise to a robust understanding and methodology of building such systems with much less human effort in the future.

3. Theoretical Analysis

In this section, we will analyze the hardness of solving HGMDPs, and show that despite being a special case of POMDPs, they are just as hard. This motivates a new model called the Helper-Action MDP (HAMDP) which is more restricted and is more amenable to approximate solutions. We will then introduce a myopic heuristic to solve HAMDPs and analyze its performance. We will also analyze some special cases of HAMDPs which permit more efficient solutions.

3.1 Complexity Results on Hidden Goal MDPs

Given knowledge of the agent's goal g in an HGMDP, the assistant's problem reduces to solving an MDP over assistant actions. The MDP transition function captures both the state change due to the assistant action and also the ensuing state change due to the agent action selected according to π given g. Likewise the reward function on a transition captures the reward due to the assistant action and the ensuing agent action conditioned on g. The optimal policy for this MDP corresponds to an optimal assistant policy for g. However, since the real assistant will often have uncertainty about the agent's goal, it is unlikely that this optimal performance will be achieved.

We can view an HGMDP as a collection of $|G|$ MDPs that share the same state space, where the assistant is placed in one of the MDPs at the beginning of each episode, but cannot observe which one. Each MDP is the result of fixing the goal component of the HGMDP definition to one of the goals. This collection can be easily modeled as a restricted type of partially observable MDP (POMDP) with a state space $S \times G$. The S component is completely observable, while the G component is unobservable but only changes at the beginning of each episode (according to I_G) and remains constant throughout an episode. Furthermore, each POMDP transition provides observations of the agent action, which gives direct evidence about the unchanging G component. From this perspective HGMDPs appear to be a significant restriction over general POMDPs. However, our first result shows that despite this restriction the worst-case complexity is not reduced even for deterministic dynamics.

Given an HGMDP M, a horizon $m = O(|M|)$, and a reward target r^*, the *short-term reward maximization problem* asks whether there exists a history-dependent assistant policy that achieves an expected finite horizon reward of at least r^*. For general POMDPs this problem is PSPACE-complete (Papadimitriou & Tsitsiklis, 1987; Mundhenk, 2001), and for POMDPs with deterministic dynamics, it is NP-complete (Littman, 1996). However, we have the following result.

Theorem 1. *Short-term reward maximization for HGMDPs with deterministic dynamics is PSPACE-complete.*

Proof. Membership in PSPACE follows from the fact that any HGMDP can be polynomially encoded as a POMDP for which policy existence is in PSPACE. To show PSPACE-hardness, we reduce the PSPACE-complete problem TQBF (truth of quantified Boolean formula) to the problem of the existence of a history-dependent assistant policy of expected reward $\geq r$.

Let Φ be a quantified Boolean formula in the form $\forall x_1 \exists x_2 \forall x_3 \ldots \exists x_n \, \{C_1(x_1, \ldots, x_n) \wedge \ldots \wedge C_m(x_1, \ldots, x_n)\}$, where each C_i is a disjunctive clause. For us, each goal g_i is a clause, The agent chooses a goal uniformly randomly from the set of goals formed from Φ and hides it from the assistant. The states consist of pairs of the form (v, i), where $v \in \{0, 1\}$ is the current value of the goal clause, and i is the next variable to set. The actions of the assistant are to set the existentially quantified variables. The agent simulates setting the universally quantified variables by choosing actions from the set $\{0, 1\}$ with equal probability. The episode terminates when all the variables are set, and the assistant gets a reward of 1 if the value of the clause is 1 at the end and a reward of 0 otherwise.

Note that the assistant does not get any useful informtion about the goal until the termination of the episode. If Φ is satisfiable, then there is an assistant policy that leads to a reward of 1 over all goals and all choices of agent actions, and hence has an expected value of 1 over the goal distribution. If not, then at least one of the goals will not be satisfied for some setting of the universal quantifiers, leading to an expected value < 1. Hence the TQBF problem reduces to deciding if a HGMDP has a policy of expected reward ≥ 1. $\square$

This result shows that any POMDP can be encoded as an HGMDP with deterministic dynamics, where the stochastic dynamics of the POMDP are captured via the stochastic agent policy in the HGMDP. However, the HGMDPs resulting from the PSPACE-hardness reduction are quite pathological compared to those that are likely to arise in realistic assistant domains. Most importantly, the agent's actions provide practically no information about the agent's goal until the end of an episode, when it is too late to exploit the knowledge. This suggests that we search for restricted classes of HGMDPs that will allow for efficient solutions with performance guarantees.

3.2 Helper Action MDPs

The motivation for Helper Action MDPs (HAMDPs) is to place restrictions on the agent and assistant that avoid the following three complexities that arise in general HGMDPs: 1) the agent can behave arbitrarily poorly if left unassisted and as such the agent actions may not provide significant evidence about the goal; 2) the agent is free to effectively "ignore" the assistant's help and not exploit the results of assistive action, even when doing so would be beneficial; and 3) the assistant actions have the possibility of negatively impacting the agent compared to not having an assistant. HAMDPs will address the first issue by assuming that the agent is competent at (approximately) maximizing reward without the assistant. The second and the third issues will be addressed by assuming that the agent will always "detect and exploit" helpful actions and that the assistant actions never hurt the agent even when they are unhelpful.

Informally, the HAMDP provides the assistant with a *helper action* for each of the agent's actions. Whenever a helper action h is executed directly before the corresponding agent action a, the agent receives a bonus reward of 1. However, the agent will only accept the helper action h (by taking a) and hence receive the bonus, if a is an action that the agent considers to be good for achieving the goal without the assistant. Thus, the primary objective of the assistant in an HAMDP is to maximize the number of helper actions that get accepted by the agent.[2]

While simple, this model captures much of the essence of assistive domains where assistant actions cause minimal harm and the agent is able to detect and accept good assistance when it arises.

An HAMDP is an HGMDP $\langle S, G, A, A', T, R, \pi, I_S, I_G \rangle$ with the following constraints:

- The agent and assistant actions sets are $A = \{a_1, \ldots, a_n\}$ and $A' = \{h_1, \ldots, h_n\}$, so that for each a_i there is a corresponding *helper action* h_i.

- The state space is $S = W \cup (W \times A')$, where W is a set of *world states*. States in $W \times A'$ encode the current world state and the previous assistant action.

- The reward function R is 0 for all assistant actions. For agent actions the reward is zero unless the agent selects the action a_i in state (s, h_i) which gives a reward of 1. That is, the agent receives a bonus of 1 whenever it takes an action directly after the corresponding helper action.

- The assistant always acts from states in W, and T is such that taking h_i in s deterministically transitions to (s, h_i).

- The agent always acts from states in $S \times A'$, resulting in states in S according to a transition function that does not depend on h_i, i.e. $T((s, h_i), g, a_i, s') = T'(s, g, a_i, s')$ for some transition function T'.

- Finally, for the agent policy, let $\Pi(s, g)$ be a function that returns a set of actions and $P(s, g)$ be a distribution over those actions. We will view $\Pi(s, g)$ as the set of actions that the agent

2. Note that here we have assumed all bonus rewards are 1 for simplicity. Most of our results below are easily extended to non-uniform positive bonus rewards. In particular for our main result concerning bounding the regret of the myopic policy, the analogous result simply includes a constant factor equal to the maximum possible reward bonus. The exceptions are Theorems 7 and 8, which we do not currently have analogous results for non-uniform reward bonuses.

considers acceptable (or equally good) in state s for goal g. The agent policy π always selects a_i after its helper action h_i whenever a_i is acceptable. That is, $\pi((s, h_i), g) = a_i$ whenever $a_i \in \Pi(s, g)$. Otherwise the agent draws an action from $P(s, g)$.

In a HAMDP, the primary impact of an assistant action is to influence the reward of the following agent action. Also notice that HAMDPs do not have rewards that are inherent to the underlying environment. Rather, the only rewards are the bonuses received whenever the agent accepts a helper action. While we could have defined the model to include environmental reward in addition to helper bonuses, this unnecessarily complicates the model (as the following hardness result shows). Instead, we assume that the inherent environmental reward is already captured by the agent policy via $\Pi(s, g)$, which is considered to contain actions that approximately optimize this reward.

As an example, the HAMDP model captures both the "doorman domain," and the "desktop domain" in our experiments. In the doorman domain, the helper actions correspond to opening doors for the agent, which reduces the cost of navigating from one room to another. In the desktop domain, the helper actions correspond to offering shortcuts to a user's destination folders. Importantly opening an incorrect door or offering an incorrect shortcut does not increase the (physical) cost to the agent over having no assistant at all, which is a key property of HAMDPs.

Despite the apparent simplification of HAMDPs over HGMDPs, it turns out that somewhat surprisingly the worst case computational complexity is not reduced.

Theorem 2. *Short-term reward maximization for HAMDPs is PSPACE-complete.*

Proof. Membership in PSPACE follows easily since HAMDPs are a specialization of HGMDPs. The proof of PSPACE-hardness is identical to that of Theorem 1 except that here, instead of the agent's actions, the stochastic environment models the universal quantifiers. The agent accepts all actions until the last one and sets the variable as suggested by the assistant. After each of the assistant's actions, the environment chooses a value for the universally quantified variable with equal probability. The last action is accepted by the agent if the goal clause evaluates to 1, otherwise not. There is a history-dependent policy whose expected reward is greater than or equal to the number of existential variables if and only if the quantified Boolean formula is satisfiable. $\square$

Unlike the case of HGMDPs, the stochastic dynamics are essential for PSPACE-hardness as will be shown in a later section. Despite this negative result, the following sections demonstrate the utility of the HAMDP restriction by giving performance guarantees for simple policies and improved complexity results. So far, there are no analogous results for general HGMDPs.

3.3 Myopic Heuristic Analysis

HAMDPs are closely related to the sequential prediction framework of Littlestone (1988). In this framework, in each round the learner is shown a new instance for which it predicts a binary label. If the prediction is incorrect, a mistake is made, and the learner is given the true label. In the realizable setting, the true labels are determined by a hypothesis in a target class. An optimal prediction algorithm minimizes the upper bound on the number of mistakes over all possible hypotheses. We can view the helper action in HAMDP as a prediction of the user action. Maximizing the bonus reward in HAMDP is equivalent to minimizing the number of mistakes in sequential prediction. Unlike sequential prediction, here the predictions (actions) need not be binary. While in sequential prediction the sequence of states are arbitrarily chosen, here we assume that they are generated by a Markov process. In spite of these differences, some of the results of sequential prediction can be adapted to HAMDPs albeit using a different terminology. However, we derive all our results from first principles for consistency.

Given an assistant policy π', the regret of a particular episode is the extra reward that an oracle assistant with knowledge of the goal would achieve over π'. For HAMDPs the oracle assistant can always achieve a reward equal to the finite horizon m, because it can always select a helper action that will be accepted by the agent. Thus, the regret of an execution of π' in a HAMDP is equal to the number of helper actions that are not accepted by the agent, which we will call *mispredictions*. From above we know that optimizing regret is PSPACE-hard and thus here we focus on bounding the expected and worst-case regret of the assistant. We now introduce our first myopic heuristic and show that it is able to achieve regret bounds that are logarithmic in the number of goals.

Coarsened Posterior Heuristic. Intuitively, our myopic heuristic will select an action that has the highest probability of being accepted with respect to a "coarsened" version of the posterior distribution over goals. The myopic policy in state s given history H is based on the *consistent goal set* $C(H)$, which is the set of goals that have non-zero probability with respect to history H. It is straightforward to maintain $C(H)$ after each observation (observations include the world state and the agent's actions). The myopic policy is defined as:

$$\hat{\pi}(s, H) = \arg\max_a I_G(C(H) \cap G(s, a))$$

where $G(s, a) = \{g \mid a \in \Pi(s, g)\}$ is the set of goals for which the agent considers a to be an acceptable action in state s. The expression $I_G(C(H) \cap G(s, a))$ can be viewed as the probability mass of $G(s, a)$ under a coarsened goal posterior which assigns goals outside of $C(H)$ probability zero and otherwise weighs them proportional to the prior.

Theorem 3. *For any HAMDP the expected regret of the coarsened posterior heuristic is bounded above by the entropy of the goal distribution $\mathcal{H}(I_G)$.*

Proof. The main idea of the proof is to show that after each misprediction of the myopic policy (i.e. the selected helper action is not accepted by the agent) the uncertainty about the goal is reduced by a constant factor, which will allow us to bound the total number of mispredictions on any trajectory.

Consider a misprediction step where the coarsened posterior heuristic selects helper action h_i in state s given history H, but the agent does not accept the action and instead selects $a^* \neq a_i$. By the definition of the myopic policy we know that $I_G(C(H) \cap G(s, a_i)) \geq I_G(C(H) \cap G(s, a^*))$, since otherwise the assistant would not have chosen h_i. From this fact we now argue that $I_G(C(H')) \leq I_G(C(H))/2$ where H' is the history after the misprediction. That is, the probability mass under I_G of the consistent goal set after the misprediction is less than half that of the consistent goal set before the misprediction. To show this we will consider two cases: 1) $I_G(C(H) \cap G(s, a_i)) < I_G(C(H))/2$, and 2) $I_G(C(H) \cap G(s, a_i)) \geq I_G(C(H))/2$. In the first case, we immediately get that $I_G(C(H) \cap G(s, a^*)) < I_G(C(H))/2$. Combining this with the fact that $C(H') \subseteq C(H) \cap G(s, a^*)$ we get the desired result that $I_G(C(H')) \leq I_G(C(H))/2$. In the second case, note that

$$\begin{aligned} C(H') &\subseteq C(H) \cap (G(s, a^*) - G(s, a_i)) \\ &\subseteq C(H) - (C(H) \cap G(s, a_i)) \end{aligned}$$

Combining this with our assumption for the second case immediately implies that $I_G(C(H')) \leq I_G(C(H))/2$.

The above shows that if a misprediction is made between histories H and H' then $I_G(C(H')) \leq I_G(C(H))/2$. This implies that for any episode, after n mispredictions resulting in a history H_n, $I_G(C(H_n)) \leq 2^{-n}$. Now consider an arbitrary episode where the true goal is g. We know that $I_G(g)$ is a lower bound on $I_G(C(H_n))$, which implies that $I_G(g) \leq 2^{-n}$ or equivalently that $n \leq -\log(I_G(g))$. Thus for any episode with goal g the maximum number of mistakes is bounded by $-\log(I_G(g))$. Using this fact we get that the expected number of mispredictions during an episode with respect to I_G is bounded above by $-\sum_g I_G(g)\log(I_G(g)) = \mathcal{H}(I_G)$, which completes the proof. □

Since $\mathcal{H}(I_G) \leq \log(|G|)$, this result implies that for HAMDPs the expected regret of the myopic policy is no more than logarithmic in the number of goals. Furthermore, as the uncertainty about the goal decreases (decreasing $\mathcal{H}(I_G)$) the regret bound improves until we get a regret of 0 when I_G puts all mass on a single goal. It turns out that this logarithmic bound is asymptotically tight in the worst case.

Theorem 4. *There exists a HAMDP such that for any assistant policy the expected regret is at least* $\log(|G|)/2$.

Proof. Consider a deterministic HAMDP such that the environment is structured as a binary tree of depth $\log(|G|)$, where each leaf corresponds to one of the $|G|$ goals. By considering a uniform goal distribution it is easy to verify that at any node in the tree there is an equal chance that the true goal is in the left or right sub-tree during any episode. Thus, any policy will have a 0.5 chance of committing a misprediction at each step of an episode. Since each episode is of length $\log(|G|)$, the expected regret of an episode for any policy is $\log(|G|)/2$. □

Resolving the gap between the myopic policy bound and this regret lower bound is an open problem.

3.3.1 APPROXIMATE GOAL DISTRIBUTIONS.

Suppose that the assistant uses an approximate goal distribution I'_G instead of the true underlying goal distribution I_G when computing the myopic policy. That is, the assistant selects actions that maximize $I'_G(C(H) \cap G(s,a))$, which we will refer to as the myopic policy relative to I'_G. The extra regret for using I'_G instead of I_G can be bounded in terms of the KullbackLeibler (KL) divergence (Kullback & Leibler, 1951) between these distributions KL($I_G \parallel I'_G$), which is zero when I'_G equals I_G.

Theorem 5. *For any HAMDP with goal distribution I_G, the expected regret of the myopic policy with respect to distribution I'_G is bounded above by $\mathcal{H}(I_G) +$ KL$(I_G \parallel I'_G)$.*

Proof. The proof is similar to that of Theorem 3, except that since the myopic policy is with respect to I'_G rather than I_G, we derive that, on any episode, the maximum number of mispredictions n is bounded above by $-\log(I'_G(g))$. Using this fact, the average number of mispredictions is given by:

$$\begin{aligned}
&\textstyle\sum_g \quad I_G(g)\log(\frac{1}{I'_G(g)}) = \\
&\textstyle\sum_g \quad I_G(g)\left(\log(\frac{1}{I'_G(g)}) + \log(I_G(g)) - \log(I_G(g))\right) = \\
&\textstyle\sum_g \quad I_G(g)\log(\frac{I_G(g)}{I'_G(g)}) - \sum_g I_G(g)\log(I_G(g)) \\
&= \quad \mathcal{H}(I_G) + \mathrm{KL}(I_G \parallel I'_G).
\end{aligned}$$

□

Note that for any random variable X with distribution P over a finite domain of size N, we have KL$(P \parallel U) = \log(N) - \mathcal{H}(P)$, where U is the uniform distribution. Thus, a consequence of Theorem 5 is that the myopic policy with respect to the uniform goal distribution has expected regret bounded by $\log(|G|)$ for any HAMDP, showing that logarithmic regret can be achieved without knowledge of I_G. This can be strengthened to hold for worst case regret.

Theorem 6. *For any HAMDP, the worst case and hence expected regret of the myopic policy with with respect to the uniform goal distribution is bounded above by* $\log(|G|)$.

Proof. The proof of Theorem 5 shows that the number of mispredictions on any episode is bounded above by $-\log(I'_G)$. In our case $I'_G = 1/|G|$ which shows a worst case regret bound of $\log(|G|)$. This immediately implies that the expected regret bound of the uniform myopic policy is bounded by $\log(|G|)$. □

3.4 Deterministic Agent Policies

We now consider several special cases of HAMDPs. First, we restrict the agent's policy to be deterministic for each goal, i.e. $\Pi(s, g)$ has at most a single action for each state-goal pair (s, g).

Theorem 7. *The myopic policy achieves the optimal expected reward for HAMDPs with deterministic agent policies.*

The proof is given in the Appendix. It is sometimes desirable to minimize the worst possible regret compared to an oracle assistant who knows the agent's goal. As we show below, this can be captured by a graph-theoretic notion of tree rank that generalizes the rank of decision trees (Ehrenfeucht & Haussler, 1989).

Definition 1. *The rank of a rooted tree is the rank of its root node. If a node is a leaf node then rank(node) =* 0*, else if a node has at least two distinct children* c_1 *and* c_2 *with equal highest ranks among all children, then rank(node) =* $1+$ *rank*(c_1)*. Otherwise rank(node) = the rank of the highest ranked child.*

The optimal trajectory tree (OTT) of a HAMDP in deterministic environments is a tree where the nodes represent the states of the HAMDP reached by the prefixes of optimal action sequences for different goals starting from the initial state.[3] Each node in the tree represents a state and a set of

3. If there are multiple initial states, we build an OTT for each initial state. Then the rank would be the maximum of the ranks of all trees.

goals for which it is on the optimal path from the initial state. Since the agent policy is deterministic, there is at most one trajectory per goal in the tree. Hence the size of the optimal trajectory tree is bounded by the number of goals times the maximum length of any trajectory, which is at most the size of the state space in deterministic domains. The following Lemma follows by induction on the depth of the optimal trajectory tree. The proof is in the Appendix.

Lemma 1. *The minimum worst-case regret of any policy for an HAMDP for deterministic environments and deterministic agent policies is equal to the tree rank of its optimal trajectory tree.*

This leads to the following.

Theorem 8. *If the agent policy is deterministic, the problem of minimizing the maximum regret in HAMDPs in deterministic environments is in* **P**.

Proof. We first construct the optimal trajectory tree. We then compute its rank in linear time while simultaneously computing the optimal minimax policy using the recursive definition of tree rank. The result then follows from Lemma 1. □

3.5 Bounded Branching Factor Policies

The assumption of a deterministic agent policy may be too restrictive in many domains. We now consider agent policies which may have a constant number of possible actions in $\Pi(s, g)$ for each state-goal pair as defined below.

Definition 2. *The branching factor of a HAMDP is the largest number of possible actions in* $\Pi(s, g)$ *by the agent in any state for any goal and any assistant's action.*

The Doorman domain of Section 6.1 has a branching factor of 2 since there are at most two optimal actions to reach any goal from any state.

Theorem 9. *Minimizing the worst-case regret in finite horizon HAMDPS in deterministic environments with a constant branching factor* k *is NP-complete.*

The proof is in the appendix. We can also show that minimizing the expected regret for a bounded k is NP-hard. We conjecture that this problem is also in NP, but this question remains open.

4. Solving Practical HGMDPs

Although HAMDPs offer a theoretically elegant framework, the requirements of practical assistant systems are not easily satisfed by its assumptions. In this section, we consider the more general problem of solving HGMDPs and offer some practical heuristic solutions that are inspired by our theoretical analysis.

In principle we could use a general purpose POMDP solver to solve HGMDPs. While the POMDP solvers based on point-based methods and search-based methods have become more efficient over the years, they are still too inefficient to be used in an interactive setting, where the parameters of the POMDP are continually being updated. Moreover, as the analysis in the previous section suggests, simple myopic heuristics based on the knowledge of the goal distribution and the optimal policies given the goals appear promising to yield respectable performance. For this reason, we adopt an approach based on Bayesian goal estimation followed by heuristic action selection, and evaluate it in three different domains. Below, we first give an overview of our solution algorithm and then describe each of the components in more detail.

4.1 Overview

In this section, we will assume that we are given an HGMDP M, delegating the problem of learning M to Section 5. Let $O_t = o_1, ..., o_t$ be an observation sequence observed by the assistant from the beginning of the current trajectory until time t. 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 HGMDP. 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. In such cases, this suggests that the state/goal estimation problem for the HGMDP 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, in such cases, we can expect 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 HGMDP to gain tractability by limiting or completely avoiding such reasoning. Of course, as shown by our PSPACE-hardness results, goals will not always be rapidly revealed and non-myopic reasoning will be essential.

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.

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 HGMDP 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.

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 show that this approach is particularly well suited to our setting and to evaluate some efficiently computable heuristics specifically designed for solving HGMDPs. Below we describe the goal estimation and action selection operations in more detail.

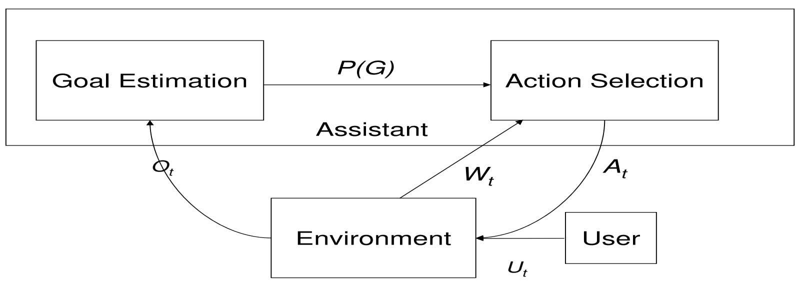

Figure 1: Depiction of the assistant architecture. The agent has a hidden goal and selects actions U_t that cause the environment to change world state W_t, typically moving closer to the goal. The assistant (upper rectangle) is able to observe the world state along with the observations generated by 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.

4.2 Goal Estimation

Given an HGMDP with agent policy π and initial goal distribution I_G, 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.

At the beginning of each episode we initialize the goal distribution $P(g|O_0)$ to I_G. 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 s, we update the posterior according to $P(g|O_t) = (1/Z) \cdot P(g|O_{t-1}) \cdot \pi(a|s,g)$, 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 s. The accuracy of goal estimation relies on how well the policy π learned by the assistant reflects the true agent policy. As described in Section 5.2, 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 s to s' we can use the MDP transition function T to marginalize over possible agent actions yielding the update,

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

4.3 Action Selection

Given the HGMDP 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 HGMDPs, 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 review how to utilize sparse sampling to obtain non-myopic action selection.

4.3.1 ACTION SELECTION 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 the goal g is achieved. Note that the state transition after an assistant action a' in state s is the result of successive state transitions, first due to the assistant action and then due to the ensuing agent action, which is selected based on the agent policy and 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 $Q_g(s,a)$, which is the expected cost of executing action a in state s and then following the optimal policy.

We now consider a second heuristic for action selection, which accounts for non-uniform rewards and the true goal posterior, unlike the coarsened posterior heuristic introduced in Section 3.3. It is simply the expected Q-value of an action over assistant MDPs, and has also been called the Q_{MDP} method by Cassandra (1998). The heuristic value for assistant action a in state s given observations O_t is

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

Intuitively $H(s,a,O_t)$ 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. 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 s, rather than a state-action pair, by maximizing over all actions $\max_a H(s,a,O_t)$.

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 (see Section 5.2 for policy learning). 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 describe in Section 5.2. 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 $Q_g(s,a)$ in the expression for H. This is done by first simulating the effect of taking action a in state s and then using π to estimate the expected cost for the agent to achieve g from the resulting state. That is, we approximate $Q_g(s,a)$ by assuming that the assistant will only select a single initial action followed by only agent actions. More formally, let $\bar{C}_n(\pi,s,g)$ be a function that simulates n trajectories of π achieving the goal from state s and then averaging the trajectory costs. The heuristic H_r is identical to $H(s,a,O_t)$ except that we replace $Q_g(s,a)$ with the expectation $\sum_{s' \in S} T(s,a,s') \cdot \bar{C}(\pi,s',g)$. We can also combine both of these heuristics, using a fixed default user policy and policy rollouts, which we denote by $H_{d,r}$.

4.3.2 SPARSE SAMPLING

All of the above heuristics are somewhat myopic in the sense that they do not take into account potentially persistent ambiguity about the agent's goal and do not consider the use of information gathering actions to resolve the ambiguity. In cases where it is beneficial to consider some 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 (s_t, O_t), denoted by $Q^d(s_t, a, O_t)$. 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 s_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.

The base case $Q^0(s_t, a, O_t)$ 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:

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

$$V^d(s, O) = \max_a Q^d(s, a, O) \tag{2}$$

where g is a random variable distributed according to the goal posterior $P(g|O)$ and (s', O') is a random variable that represents the belief state after taking action a in belief state (s, O). In particular, s' is the 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 reward of the assistant action a when the goal is g.

Sparse sampling approximates the above expectation by averaging a set of b samples of successor belief states. The sparse-sampling pseudo-code is presented in Table 4.3.2. Given an input belief state (s, O), assistant action a, heuristic H, depth bound d, and sampling width b the algorithm returns (an approximation of) $Q^d(s, a, O)$. 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 (s, O) are generated. The observations will be of the form $o_i = (s'_i, a_i, s_i)$, where s'_i is the state resulting from taking action a in state s, a_i is the ensuing agent action selected in s'_i based on a goal drawn from the goal posterior, and s_i is the result of taking action a_i in state s'_i. Each observation o_i corresponds to a new belief state $(s_i, [O; o_i])$ 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 maximizing 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.

5. Learning HGMDPs

In this section, we tackle the problem of learning the HGMDP while interacting with the environment to assist the agent. We assume that the set of goals G is known to the agent. 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, for possibly different agents. For example, in a desktop environment, the environment MDP corresponds to a description of the various desktop functionalities, which remains fixed across users. If one is not

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

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

Table 1: Pseudo-code for Sparse Sampling in the HGMDP

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 saw in Section 4, our solution methods 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, allows for useful assistance to be provided.

5.1 Maximum Likelihood Estimates

It is straightforward to estimate the goal distribution G_0 and agent policy π by simply observing the agent's actions, possibly while being assisted, and to compute empirical estimates of the relevant quantities. This can be done by storing the goal achieved at the end of each episode along with the set of world state-action pairs observed for the agent during the episode. The estimate of I_G can then be based on observed frequency of each goal (usually with Laplace correction to avoid extreme values of the probabilities). Likewise, the estimate of $\pi(a|s, g)$ is simply the frequency for which action a was taken by the agent when in state s and having goal g. While in the limit these maximum likelihood estimates will converge to the correct values of the true HGMDP, 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.

5.2 Model-Based Bootstrapping

We will leverage our environment MDP model in order to bootstrap the learning of the agent policy. In particular, we assume that the agent is near optimal in the sense that, for a particular goal and world state, he is more likely to select actions that are close 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 are quite tedious, e.g., navigating through the directory structure of a computer desktop. Performing optimally in such tasks is not difficult for humans.

Given the "near rationality assumption", we initialize the estimate of the agent's policy to a prior that is biased toward optimal agent actions. To do this we will consider the environment MDP with the assistant actions removed and solve for the Q-function $Q(a, s, g)$ using MDP planning techniques. The Q-function gives the expected cost of executing agent action a in world state s 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,

$$\hat{\pi}(a|s, g) = \frac{1}{Z(s, g)} \exp(K \cdot Q(a, s, g)) \tag{3}$$

where $Z(s, g)$ 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 $\alpha_i = \alpha_0 \cdot \hat{\pi}(a_i|s, g)$. 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 $\pi(\cdot|s, g)$, it is easy to update the posterior over $\pi(\cdot|s, g)$ 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 HGMDP.

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 among 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 be 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.

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 the user studies in the two game-like domains, for each episode, the user's and the assistant's actions were recorded. These user studies were performed using 12 human subjects (graduate students in CS department at Oregon State University) over a single session. 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 her desired folder. Two of these three domains, namely, the doorman domain and the folder predictor domain, fall under the category of HAMDPs since the assistive actions can be merely viewed as helper actions that the agent can ignore. The kitchen domain on the other hand needs a slightly more general formulation since the agent and the assistant do not strictly alternate, and the assistant's actions cannot be ignored by the agent.

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 2). 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 total number of states is 245 (49 squares with 5 possibilities for the door). 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. The agent's and the assistant's actions strictly alternate in this domain, satisfying the definition of HAMDPs. There is a reward of -1 (or a cost of 1) if the user has to open the door and no reward to the assistant's action. The trial ends when the agent picks up the desired object. Note that while we have included a **noop** action for the assistant, in this domain the action is never selected, since the cost of opening a wrong door and noop are the same, while there is no potential benefit of selecting **noop**.

In this experiment, we evaluated two heuristics: one where we fixed the user policy to the default policy in the HGMDP creation (H_d) avoiding the need for repeated computation of the HGMDP at every step 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.

The results of the user studies for the doorman domain are presented in Tabe 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. Rather than reporting the negtive rewards, the table shows the total number of

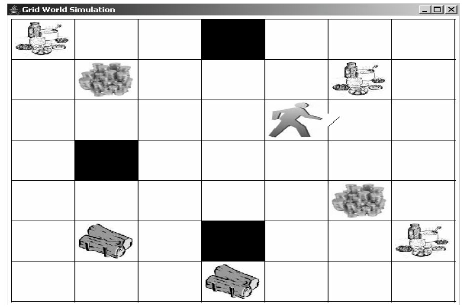

Figure 2: Doorman Domain. The agent's goal is to fetch a resource. The grid cells are separated by doors that must be opened before passing through.

actions for all trials across all users without the assistant *N*, and the total number of actions with the assistant *U*, and the average of percentage savings (*1-(U/N)*) over all trials and over all the users.[4]

As can be seen, both the methods reduce the number of actions by more than 50%. Note that an assistant that selects among the four doors at random would reduce the number of actions by only 25% in comparison. An omniscient assistant who knows the user's goal reduces the number of actions by 78%. This is not 100% because the first door is always opened by the user. In our experiments, if we do not count the user's first action, the number of actions reduces by 65%.[5] It can be observed that H_r appears to have a slight edge over H_d. One possible reason for this is that while using H_d, we do not re-solve the MDP after updating the user policy, while H_r is always using the updated user policy. Thus, rollout is reasoning with a more accurate model of the user.

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

Table 2: 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.

4. 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.
5. Note that this first action requirement is easily aviodable. It is simply the equivalent of an "on" switch to indicate that the user is ready to move about in the grid. We can also replace this requirement by explicitly adding an on button to the interface to start a new episode.

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-zero 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 and effort required for humans in such studies, but simulated the human users by choosing actions according to policies learned from their observed actions from the previous user study. The results are presented in the last 5 rows of Table 2. Note that the absolute numbers of actions in the user studies and the simulations are not comparable as they are based on different numbers of trajectories. The human users were tested on fewer trajectories to minimize their fatigue. We see that sparse sampling increased the average run time (last column) 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 ingredients (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. We restricted the assistant from pouring ingredients as it is an irreversible action. The reward for all non-pour actions is -1. Experiments were conducted on 12 human subjects who are computer science graduate students. Unlike in the doorman domain, here we allowed the the assistant to take multiple consecutive actions. The turn switches to the user when the assistant executes the **noop** action.

This domain has a large state space and hence it is not possible to update the user policy after every trajectory. Hence, 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. As in the doorman

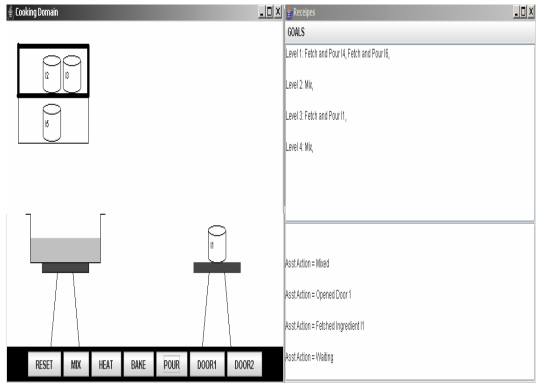

Figure 3: The kitchen domain. The user is to prepare the dishes described in the recipes on the right. The assistant's actions are shown in the bottom frame.

domain, the total number of agent actions with and without the assistant, and the percentage reduction due to the assistant are presented. The number of user actions was summed over 12 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 H_d, which would wait until the goal distribution is highly skewed toward a particular goal.

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

Table 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). Again, the absolute numbers of actions of the user studies are not comparable to that of simuations due to different numbers of trajectories in each case. Since sparse sampling considers a larger number of trajectories than the other methods, the policies learned are sometimes better than those learned from other heuristics, although they took more time to execute. However, 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 4). 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.

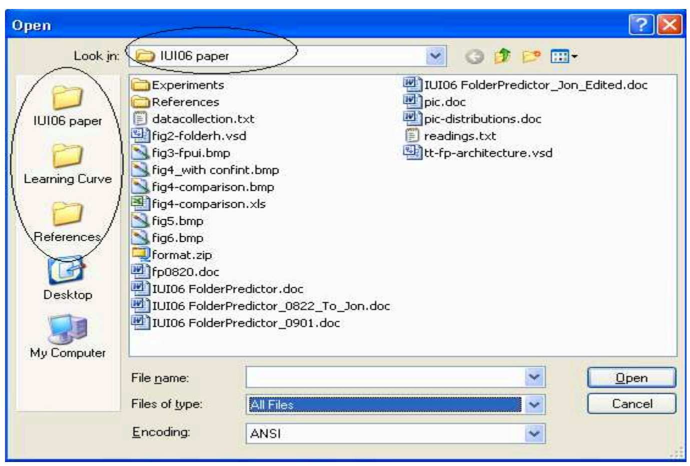

Figure 4: Folder predictor (Bao et al., 2006).

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 (2006), 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 the 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 increases the value of μ_0 increases and would eventually converge to 1, 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 reward in our case was the negative of the number of user clicks. 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 rewards computed by the model developed by Bao et al. (2006).

We applied the decision theoretic model to the data set. For each request, our assistant would make the prediction 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, and otherwise 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 numbers indicate the average number of clicks required by the agent to reach his/her correct folder. 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. Bao et. al have shown that their algorithm performs significantly better than the windows default prediction which has an average of 2.6 clicks per folder navigation. 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, considering all folders in the hierarchy for prediction including the possibility of the user accessing a new folder is found to be useful. 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 (Yorke-Smith et al., 2012; Lieberman, 2009; Myers et al., 2007). Some of this effort is focused on building desktop assistants that help with tasks such as calendar scheduling (Refanidis, Alexiadis, & Yorke-Smith, 2011), email filtering (Cohen, Carvalho, & Mitchell, 2004), on-line diagnostics (Skaanning, Jensen, & Kjaerulff, 2000), and travel planning (Ambite, Barish, Knoblock, Muslea, Oh, & Minton, 2002). 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 et al., 2004), travel planning combines information gathering with search and constraint propagation (Ambite et al., 2002), and printer diagnostics is formulated as Bayesian network inference (Skaanning et al., 2000). Other approaches focus on socially assistive robots setting where a robot is designed to aid human agents in achieving their goals (Johnson, Cuijpers, Juol, Torta, Simonov, Frisiello, Bazzani, Yan, Weber, Wermter, et al., 2013). 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 (Johnson, 2014; Konidaris, Kuindersma, Grupen, & Barto, 2012; 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 are solved offline. For instance, the COACH system helped people suffering from dementia by giving them appropriate prompts as needed in their daily activities (Boger, Poupart, Hoey, Boutilier, Fernie, & Mihailidis, 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 which is either low or $high$. Rather, in our formulation the goal is a random variable that is hidden to the assistant. Since their state-action space is significantly smaller (1280 as against 2×10^8 states in our folder predictor domain), it is possible for them to solve the POMDP exactly. Given that we need to re-solve the POMDP after every user action, it becomes prohibitively expensive. Yet another difference is that the length of the trajectory to the goal is small in their case and hence a plan graph would suffice to capture the user policy. In our model, we do not restrict to a plan graph and instead solve the user MDP to bootstrap the policy. They have mentioned learning the user policy as a future direction. In our work, even though we start with an initial estimate of the user policy, we update it after every goal is achieved. This can be considered as online learning of user policy with a reasonably good prior. We note that a combination of these two frameworks (one for modeling user's responsiveness and the other for modeling user's goal) 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 a meeting 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 that 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 & Gmytrasiewicz, 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.

There have been several dialogue systems proposed and many of them are based on decision-theoretic principles (Walker, 2000; Singh, Litman, Kearns, & Walker, 2002). For instance, the NJFun system is designed as an MDP to provide assistance to the user by interacting with the user and providing the answer to the user's questions. It uses an automatic speech recognizer (ASR) to interpret the human dialogues and uses a dialogue policy to choose the best action (the response). The goals of the user could be a set of standard queries such as locations of restaurants, wineries, shopping centers etc. The state space would be the dialogue states, i.e., the current state of the dialogue between the user and assistant (such as greeting, choice state etc). The observations are the interpretations of the dialogues of the human by the ASR. The NJFun system can be usefully modeled as an HGMDP, where the goal of the assistant is to infer the user's query given the observations and provide appropriate response. The initial assistant policy can be learned from training data, in a manner similar to the dialogue policy of the NJFun system.

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 his actions and attributes and provided assistance as needed 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 the use of ideas from plan recognition (Charniak & Goldman, 2013; Gal, Reddy, Shieber, Rubin, & Grosz, 2012; Chu, Song, Kautz, & Levinson, 2011) in our system to take into account the user's intentions and attitudes while computing the optimal policy for the assistant.

Recently, there have been methods proposed for solving POMDPs called *point based* methods (Pineau, Gordon, & Thrun, 2003; Porta, Vlassis, Spaan, & Poupart, 2006; Kurniawati, Hsu, & Lee, 2008; Shani, Pineau, & Kaplow, 2013). An example of this method is point based value iteration (PBVI) (Pineau et al., 2003; Porta et al., 2006) that takes a set of belief points B as input and maintains a set of POMDP α-vectors at each iteration. Each iteration produces a new set of α-vectors that are optimal for each belief point with respect to the α-vectors in the previous iteration. The approximation made by PBVI when compared to value iteration is that there is no guarantee that the set of α-vectors is optimal for the entire belief space. By omitting some α-vectors, PBVI maintains a constant run time per iteration. Application of efficient point based methods such as

PBVI to the decision-theoretic assistance problem and evaluation of their performance compared to the policy rollout and sparse sampling methods remains a promising research direction.

8. Summary and Future Work

We introduced a decision-theoretic framework for assistant systems and described the HGMDP as an appropriate model for selecting assistive actions. The computational complexity of HGMDPs motivated the definition of a simpler model called HAMDP, which allows efficient myopic heurstics and more tractable special cases.

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 Contract No. NBCHD030010. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA. Alan Fern and Prasad Tadepalli gratefully acknowledge the following grants: NSF IIS-0964705 and ONR N00014-11-1-0106. Sriraam Natarajan thanks Army Research Office grant number W911NF-13-1-0432 under the Young Investigator Program.

Appendix A. Proof of Theorem 7

According to the theory of POMDPs, the optimal action in a POMDP maximizes the sum of the immediate expected reward and the value of the resulting belief state (of the assistant) (Kaelbling, Littman, & Cassandra, 1998). When the agent policy is deterministic, the initial goal distribution I_G and the history of agent actions and states H fully captures the belief state of the agent. Let $V(I_G, H)$ represent the value of the current belief state. Then the value function of the belief state is given by by the following Bellman equation, where H' stands for the history after the assistant's action h_i and the agent's action a_j.

$$V(I_G, H) = \max_{h_i} E(R((s, h_i), g, a_j)) + V(I_G, H') \tag{4}$$

Since there is only one agent's action in $\Pi(s, g)$, the agent action a_j, the subsequent state s' in H' and its value do not depend on h_i. Hence the best helper action h^* of the assistant is given by:

$$\begin{aligned} h^*(I_G, H) &= \arg\max_{h_i} E(R((s, h_i), g, a^*(s, g))) \\ &= \arg\max_{h_i} \sum_{g \in C(H)} I_G(g) I(a_i \in \Pi(s, g)) \\ &= \arg\max_{h_i} I_G(C(H) \cap G(s, a_i)) \end{aligned}$$

where $C(H)$ is the set of goals consistent with the current history H, and $G(s, a_i)$ is the set of goals for which a_i is good in state s. $I(a_i \in \Pi(s, g))$ is an indicator function which is $= 1$ if $a_i \in \Pi(s, g)$. Note that h^* is exactly the myopic policy. □

Appendix B. Proof of Lemma 1

The worst-case regret of any pair (s, G) in the HAMDP is given by the following Bellman equation. assuming that G is the set of possible goals of the agent and the current state is s. Regret$(s, G) = 0$ if s is a terminal state or all goals in G are satisfied in s. Otherwise, Regret(s, G) = $\min_i \max_{j \neq i}$ {Regret(s_i, G_i), $1 +$ Regret(s_j, G_j))} where (s_i, G_i) and (s_j, G_j) are in $Children((s, G))$.

Here the outer $\min$ is due to the assistant picking a helper action for a goal in G_i to maximize the reward and the inner $\max$ is due to the agent either accepting it, or picking a different goal to minimize the reward. The proof is by induction. Each node in the trajectory tree represents a state and a set of goals G for which that state is on the optimal path.

Basis: If (s, G) is a leaf node, it is either a terminal state or all goals in G are satisfied in s. Hence its rank equals its reward which is 0.

Inductive step: Suppose that the induction is true for all children of (s, G). We will consider two cases.

Case 1. There is a unique child of (s, G) representing (s_1, G_1) which has the highest regret among all its children. By inductive hypothesis, rank$((s_1, G_1))$ = regret(s_1, G_1). If the assistant chooses the helper action that corresponds to (s_1, G_1), the agent can only choose actions that yield lower regret in the worst case. Choosing any other helper action would increase the regret, since the agent could then choose a_1 and add 1 to the regret. So we have, regret(s, G)= regret(s_1, G_1) = rank$((s_1, G_1))$ = rank$((s, G))$.

Case 2. There are at least two children (s_1, G_1) and (s_2, G_2) of (s, G) which have the highest rank among all its children. By inductive hypothesis, rank$((s_1, G_1))$ = rank$((s_2, G_2))$ = regret(s_1, G_1) = regret(s_2, G_2). Here the agent can increase the regret by 1 more by choosing a goal in G_2 if the assistant chooses G_1 and vice versa. Hence, regret(s, G)= 1+regret(s_1, G_1) = 1+rank$((s_1, G_1))$ =

rank((s, G)).
Hence in both cases, we have shown that regret(s, G) is rank((s, G)). □

Appendix C. Proof of Theorem 9

We first show that the problem is in NP. We build a tree representation of a history-dependent policy for each initial state. Every node in the tree is represeted by a triple (s, i, G), where s is a state, G is a set of goals for which it is on a good path, and i is the index of the helper action chosen by the policy at that node. The root node corresponds to a possible initial state and the initial goal set I_G. The children of a node in the tree represent possible successor nodes (s_j, j, G_j) reached by the agent's response to h_i, whether by accepting h_i and executing a_i or by executing other actions. The children resulting from a_i are called "accepted," and the latter are called "rejected." Note that multiple children can result from the same action because the dynamics are a function of the agent's goal.

We now guess a policy tree and check that its maximum regret, i.e. the maximum number of rejected children in any path from the root to a leaf, is within bounds. To verify that the optimal policy tree is of polynomial size we note that the number of leaf nodes is upper bounded by $|G| \times \max_g N(g)$, where $N(g)$ is the number of leaf nodes generated by the goal g. To estimate $N(g)$, we start from the root and navigate downwards. For any node that contains g in its goal set, if some accepted child contains g, then it will be the only child that will be reached for g. If not, there is a misprediction and there are at most k children reached. Hence, the number of nodes reached for g grows geometrically with the number of mispredictions. From Theorem 6, since there are at most $\log |G|$ mispredictions in any such path, $N(g) \leq k^{\log_2 |G|} = k^{\log_k |G| \log_2 k} = |G|^{\log_2 k}$. Hence the total number of all leaf nodes of the tree is bounded by $|G|^{1+\log k}$, and the total number of nodes in the tree is bounded by $m|G|^{1+\log k}$, where m is the number of steps to the horizon. Since this is polynomial in the problem parameters, the problem is in NP.

To show NP-hardness, we reduce 3-SAT to the given problem. We consider each 3-literal clause C_i of a propositional formula Φ as a possible goal. The rest of the proof is identical to that of Theorem 1 except that all variables are set by the assistant since there are no universal quantifiers. The agent only rejects the setting of the last variable in the clause if the clause evaluates to 0. The worst regret on any goal is 0 iff the 3-SAT problem has a satisfying assignment. □

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 *Proceedings of the Fourteenth Conference on Innovative Applications of Artificial Intelligence*, pp. 862–869.

Atkeson, C. G., & Schaal, S. (1997). Learning tasks from a single demonstration. In *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 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 *Proceedings of the Eleventh International Conference on Intelligent User Interfaces*, 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 *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling*, pp. 297–305.

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 *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pp. 1293–1299.

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

Bui, H., Venkatesh, S., & West, G. (2002). Policy recognition in the abstract hidden markov models. *Journal of Artificial Intelligence Research*, *17*, 451–499.

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

Charniak, E., & Goldman, R. (2013). Plan recognition in stories and in life. *CoRR*, *abs/1304.1497*.

Chu, Y., Song, Y., Kautz, H., & Levinson, R. (2011). When did you start doing that thing that you do? interactive activity recognition and prompting. In *Proceedings of the Twenty-Fifth AAAI Conference Workshop on Artificial Intelligence and Smarter Living*, pp. 15–21.

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 the Conference on Empirical Methods in Natural Language Processing*, pp. 309–316.

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

Doshi, P., & Gmytrasiewicz, P. (2004). A particle filtering algorithm for interactive POMDPs. In *Proceedings of the Workshop on Modeling Other Agents from Observations*, pp. 87–93.

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 the Tenth International Conference on Intelligent User Interfaces*, pp. 75–82.

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

Ehrenfeucht, A., & Haussler, D. (1989). Learning decision trees from random examples. *Information and Computation*, *82*(3), 231–246.

Gal, Y., Reddy, S., Shieber, S., Rubin, A., & Grosz, B. (2012). Plan recognition in exploratory domains. *Artificial Intelligence*, *176*(1), 2270–2290.

Geffner, H., & Bonet, B. (1998). Solving large POMDPs using real time dynamic programming. In *Proceedings of AAAI Fall Symposium on POMPDs*.

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

Guestrin, C., Koller, D., Parr, R., & Venkataraman, S. (2003). Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research*, *19*, 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 *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pp. 256–265.

Hui, B., & Boutilier, C. (2006). Who's asking for help?: a Bayesian approach to intelligent assistance. In *Proceedings of the Eleventh International Conference on Intelligent User Interfaces*, pp. 186–193.

Johnson, D., Cuijpers, R., Juol, J., Torta, E., Simonov, M., Frisiello, A., Bazzani, M., Yan, W., Weber, C., Wermter, S., et al. (2013). Socially assistive robots: A comprehensive approach to extending independent living. *International Journal of Social Robotics*, *6*(2), 195–211.

Johnson, M. (2014). *Inverse optimal control for deterministic continuous-time nonlinear systems*. Ph.D. thesis, University of Illinois at Urbana-Champaign.

Kaelbling, L., Littman, M., & Cassandra, A. (1998). Planning and acting in partially bservable stochastic domains. *Artificial Intelligence*, *101*(1-2), 99–134.

Kearns, M. J., Mansour, Y., & Ng, A. Y. (1999). A sparse sampling algorithm for near-optimal planning in large markov decision processes. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pp. 1324–1331.

Konidaris, G., Kuindersma, S., Grupen, R., & Barto, A. (2012). Robot learning from demonstration by constructing skill trees. *The International Journal of Robotics Research*, *31*(3), 360–375.

Kullback, S., & Leibler, R. (1951). On information and sufficiency. *The Annals of Mathematical Statistics*, *22*(1), 79–86.

Kurniawati, H., Hsu, D., & Lee, W. (2008). Sarsop: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Proceedings of Robotics: Science and Systems IV*.

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

Lieberman, H. (2009). User interface goals, AI opportunities. *AI Magazine*, *30*(3), 16–22.

Littlestone, N. (1988). Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, *2*(4), 285–318.

Littman, M. L. . (1996). *Algorithms for Sequential Decision Making*. Ph.D. thesis, Brown University.

Mahadevan, S., Mitchell, T. M., Mostow, J., Steinberg, L. I., & Tadepalli, P. (1993). An apprentice-based approach to knowledge acquisition.. *Artificial Intelligence*, *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.

Mundhenk, M. (2001). *The complexity of planning with partially-observable Markov Decision Processes*. Ph.D. thesis, Friedrich-Schiller-Universitdt.

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*, Vol. 28, pp. 47–61.

Natarajan, S., Tadepalli, P., & Fern, A. (2007). A relational hierarchical model for decision-theoretic assistance. In *Proceedings of the Seventeenth Annual International Conference on Inductive Logic Programming*, pp. 175–190.

Papadimitriou, C., & Tsitsiklis, J. (1987). The complexity of Markov Decision Processes. *Mathematics of Operations Research*, *12*(3), 441–450.

Pineau, J., Gordon, G., & Thrun, S. (2003). Point-based value iteration: An anytime algorithm for POMDPs. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pp. 1025 – 1030.

Porta, J., Vlassis, N., Spaan, M., & Poupart, P. (2006). Point-based value iteration for continuous POMDPs. *Journal of Machine Learning Research*, *7*, 2329–2367.

Pynadath, D. V., & Wellman, M. P. (2000). Probabilistic state-dependent grammars for plan recognition. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pp. 507–514.

Refanidis, I., Alexiadis, A., & Yorke-Smith, N. (2011). Beyond calendar mashups: Intelligent calendaring. In *Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling System Demonstrations*.

Shani, G., Pineau, J., & Kaplow, R. (2013). A survey of point-based POMDP solvers. *Autonomous Agents and Multi-Agent Systems*, *27*(1), 1–51.

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 *Proceedings of the Thirteenth 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 *Proceedings of the Fourth Internation Conference on Autonomous Agents and Multiagent Systems*, pp. 978–985.

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.

Yorke-Smith, N., Saadati, S., Myers, K., & Morley, D. (2012). The design of a proactive personal agent for task management. *International Journal on Artificial Intelligence Tools*, *21*(1), 90–119.