Efficient Model Learning for Dialog Management

Finale Doshi and Nicholas Roy
MIT Computer Science and Artificial Intelligence Laboratory
32 Vassar Street
Cambridge MA, 02139 USA

Abstract

Intelligent planning algorithms such as the Partially Observable Markov Decision Process (POMDP) have succeeded in dialog management applications (Roy, Pineau, & Thrun 2000; Williams & Young 2005; Williams, Poupart, & Young 2005) because they are robust to uncertainties in human-robot interaction. Like all dialog planning systems, POMDPs require an accurate model of what the user might say and how he wishes to interact with the robot. In the POMDP framework, the user's vocabulary and preferences are generally specified using a large probabilistic model with many parameters. While it may be easy for an expert to specify reasonable values for these parameters, gathering data to specify the parameters accurately *a priori* is expensive.

In this paper, we take a Bayesian approach to learning the user model while simultaneously refining the dialog manager's policy. First, we show how to compute the optimal dialog policy with uncertain parameters (in the absence of learning), along with a heuristic that allows the dialog manager to intelligently replan its policy given data from recent interactions. Next, we present a pair of approaches which explicitly consider the robot's uncertainty about the true user model when taking actions; we show these approaches can learn user preferences more robustly. A key contribution of this work is the use of "meta-actions," queries about what the robot should have done, to discover a user's dialog preferences without making mistakes that may potentially annoy the user.

Introduction

Spoken dialog managers allow for natural human-robot interaction, especially in environments where the user has limited mobility. However, several issues can make it difficult to determine how the dialog manager should react to a user. Voice recognition technology produces noisy outputs, and even with perfect voice recognition, the dialog manager must decipher the user's intent in the face of linguistic ambiguity. As the dialog manager attempts to discover what the user wants, it must also be sensitive to the types of queries it makes: each user will have different preferences on what type of questioning they will tolerate before they become frustrated with the system.

Partially Observable Markov Decision Processes (POMDPs) are a planning technique that have led to more robust dialog management because they effectively handle uncertainty and ambiguity in the dialog (Roy, Pineau, & Thrun 2000; Williams & Young 2005; Williams, Poupart, & Young 2005). The typical POMDP model operates by assuming that the user's intent, or "state" is hidden from the robot. Instead, the user's intent is inferred from a stream of noisy or ambiguous observations, such as utterances from a speech recognition system. Given a reward model of the user's interaction preferences, the dialog manager chooses actions to maximize the expected long-term reward. By choosing actions based on how clearly it understands the user, the system can model the value of asking additional confirmation questions now to avoid doing an incorrect action later.

Figure 1: Our dialog manager allows more natural human communication with a robotic wheelchair.

The ability to manage the trade-off between gathering information and gathering rewards have made POMDP dialog managers particularly effective in health care domains (Roy, Pineau, & Thrun 2000; Hoey *et al.* 2005). Especially in these populations, the observed utterances may be quite noisy, and the cost of dialog errors may be high because the dialog may involve large motions of assistive devices. One example is our our robotic wheelchair (Figure 1). The cost of misunderstanding a desired destination—and having the wheelchair start going to the wrong location—is greater than the annoyance of having the user repeat a request.

The POMDP has been shown to manage the dialog well if it has an accurate user model, and most POMDP dialog-management research (Williams, Poupart, & Young 2005;

Pineau, Roy, & Thrun 2001; Williams & Young 2005) assumes that such a model is available. In domains where large amounts of data are available—for example, automated telephone operators—an accurate user model may be relatively easy to obtain. For human-robot interaction, however, collecting sufficient user data to learn a statistically accurate model may take may require a lot of time from users, and specifying the model from expert knowledge may also be difficult. For example, how does one specify the probability that the user appeared to ask for the time when they actually wanted the weather? Just as tricky is specifying a user's dialog preferences: how often should the robot confirm a user's utterances before going to the specified location?

Some learning algorithms have designed systems to train parameters from scratch while maintaining reasonable dialogs (Litman *et al.* 2000), and in general there are several reinforcement learning approaches for planning in unknown environments. These approaches, however, do not take advantage of the fact that while specifying an accurate model may be difficult, it is often possible to specify a reasonable model from domain knowledge. We take a Bayesian approach to reinforcement learning, which allows us to include any domain knowledge we may have as part of our initial belief over possible user models.

Another issue with using the standard reinforcement learning model in a dialog manager is that the user must provide the robot with reward feedback at every stage of the training process; in this way the dialog manager builds a model of the user's preferences. This training process can be tedious, and people often find it difficult to quantify their preferences. We explore the use of "meta-actions," queries about actions that the dialog manager should have taken, as a more natural way for the dialog manager to gain information about the user. For example, by asking the wheelchair user if he would like to hear a list of locations when choosing a destination, the dialog manager can determine whether the user prefers directed confirmation questions or a more free-form conversation.

Combining the ideas of priors and meta-actions, the following sections present three Bayesian extensions to the basic POMDP model that allow a robot to learn a user's dialog model online:

- I. Maximizing Expected Performance. This section describes how an robot should act in the face of parameter uncertainty if it does not have the opportunity to learn, and then extends this approach to incorporate an efficient form of online learning.
- II. Improving Dialog Manager Robustness. This section describes how we can increase robustness by making the dialog manager explicitly aware of the uncertainty of its parameters (so it can take actions as needed to reduce that uncertainty).
- III. Meta-Actions. This section describes how we can reduce our reliance on explicit user feedback (required in the first two approaches) while still learning the user's preferences robustly.

POMDP Dialog Manger Formulation

A POMDP consists of the n-tuple $\{S,A,O,T,\Omega,R,\gamma\}$. S, A, and O are sets of states, actions, and observations. For our dialog manager, the states represent the user's (hidden) intent, in this case the places where the user would like the robotic wheelchair to go. The observations are the actual utterances that the dialog manager hears, and the actions include queries to get additional information and physical movements to destinations.

We assume that the state, action and observation sets are discrete and finite to make learning the parameters and updating the POMDP tractable for real-time dialogs (although work has extended dialog managers to large state spaces (Williams & Young 2005) and continuous observation spaces (Williams, Poupart, & Young 2005)). In our tests, user could choose from one of five goal locations, and our basic dialog manager chose from three action types: it could ask a general question, such as "Where would you like to go?"; it could confirm a specific location; or it could drive to a specific location. The observations were keywords filtered from the output of voice recognition software (Glass 2003). Figure 2 shows an example dialog flow.

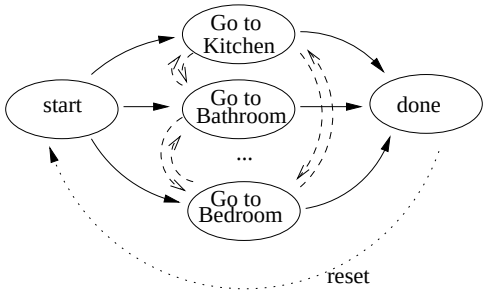

Figure 2: A toy example of a dialog POMDP. Solid lines represent more likely transitions; we assume that user is unlikely to change their intent before their original request is fulfilled (dashed lines). The system automatically resets once we enter the 'done' state.

The remaining functions $R(s,a)$, $T(s'|s,a)$, and $\Omega(o|s,a)$ together model the user's communication style and preferences. The transition function $T(s'|s,a)$ gives the probability $P(s'|s,a)$ of transitioning from state s to s' if the robot takes action a. In our model, we assume that the user is unlikely to change where he wishes to go during the course of a single dialog. The observation function $\Omega(o|s,a)$ gives the probability $P(o|s,a)$ of seeing observation o from state s after taking action a. The reward $R(s,a)$ specifies the immediate reward for taking action a in state s. The rewards specify what actions the user prefers in different states; the dialog manager uses this information to determine how much the user is willing to tolerate clarification questions before becoming frustrated.

Finally, the discount factor $\gamma \in [0,1]$ allows us to bias the dialog manager towards satisfying user intents more quickly. A discount factor of 0 means that we only value current rewards; thus we will not take low-yield actions (such as additional clarifications) now that may result in higher user satisfaction in the future. A discount factor of 1 means that future rewards are just as valuable as current rewards; such a robot

may spend a long time asking clarification questions before taking an action.

Since the POMDP's state is hidden, the dialog manager must decide its next action based on its previous history of actions and observations. Defining a policy in terms of the history can get quite cumbersome, so we typically keep a probability distribution over the states (known as the belief b), which is a sufficient statistic for the previous history of actions and observations. Given a new action and observation, we can update the belief b using Bayes rule:

$$b_n(s) = \eta\Omega(o|s',a)\sum_{s\in S} T(s'|s,a)b_{n-1}(s) \quad (1)$$

where η is a normalizing constant. The goal of the POMDP dialog manager is to find a policy mapping the set of beliefs B to actions A to maximize the expected reward $E[\sum_n \gamma^n R(s_n,a_n)]$.

In order to find a good dialog manager, that is, to find a policy that maximizes our reward for each belief, we use the concept of a value function to represent our policy. Let the value function $V(b)$ represent the expected reward if we start with belief b. The optimal value function is piecewise-linear and convex, so we represent V with the vectors V_i; $V(b) = max_i V_i \cdot b$. The optimal value function is unique and satisfies the Bellman equation:

$$V(b) = \max_{a\in A} Q(b,a), \quad (2)$$

$$Q(b,a) = R(b,a) + \gamma \sum_{b'\in B} T(b'|b,a)V(b'), \quad (3)$$

$$Q(b,a) = R(b,a) + \gamma \sum_{o\in O} \Omega(o|b,a)V(b_a^o), \quad (4)$$

$$Q(b,a) = \max_i \vec{q_a} \cdot b. \quad (5)$$

$$q_a(s) = R(s,a) + \gamma \sum_{o\in O}\sum_{s'\in S} T(s'|s,a)\Omega(o|s',a)\alpha_i(s) \quad (6)$$

The $Q(b,a)$ values represent the total expected reward if we start from b, do a, and then act optimally. Equation 2 chooses the action that maximizes the expected reward to derive the optimal dialog policy. Equation 4 follows from equation 3 if we note that there are only $|O|$ beliefs that we can transition to after taking action a in belief b (one corresponding to each observation). The belief b_a^o is the new belief after a Bayesian update of b using equation 1. $\Omega(o|b,a)$, the probability of seeing o after performing a in belief b, is $\sum_{s\in S}\Omega(o|s,a)b(s)$. The linear properties of the value function further decompose the Q function into equations 5 and 6; this decomposition allows us to see that the Q function is an expectation over the belief (uncertainty in user's intent, equation 5) and the user model (ambiguities in the user's communication, equation 6).

These equations may be solved iteratively:

$$V_n(b) = \max_{a\in A} Q_n(b,a), \quad (7)$$

$$Q_n(b,a) = R(b,a) + \gamma \sum_{o\in O} \Omega(o|b,a)V_{n-1}(b_a^o). \quad (8)$$

Each iteration, or *backup*, brings the value function closer to its optimal value (Gordon 1995). Once the value function has been computed, it is used to choose actions. After each observation, we update the belief using equation 1 and then choose the next action using $\arg\max_{a\in A} Q(b,a)$ with $Q(b,a)$ given in equation 3.

Note that the exact solution to equation 7 using an iterative backup approach is exponentially expensive in computation time, so we must approximate the solution. The Point-Based Value Iteration, PBVI, (Pineau, Gordon, & Thrun 2003) is an approximation technique for solving POMDPs that backs up the POMDP solution only at certain belief points. The quality of the solution depends on the density of the belief points in the reachable portion of the belief space. PBVI suggests a heuristic that begins with an initial belief (or belief set). For each action a, we sample a user response o from the observation distribution, compute the updated belief state b_o^a (simulating the effect of one exchange between the user and the dialog manager), and add the farthest new beliefs to our set. In this way, we focus computation on confusions that the dialog manager is likely to experience.[1]

I. Maximizing Expected Performance

In the previous section, we described the basic POMDP model. Here we describe some of our work initially presented in (Doshi & Roy 2007) to show how this model changes when the parameters that govern the user model (T, Ω, and R) are uncertain. We demonstrate a simple update heuristic on our robotic wheelchair.

Acting with Uncertain User Models

If we do not know the model parameters, then a reasonable approach is to assume a distribution over both the state and the model itself. Once we have a distribution over user parameters, we must add an additional expectation in Equation 6:

$$\begin{aligned} q_a(s) &= E[R(s,a) + \\ &\quad \gamma \sum_{o\in O}\sum_{s'\in S} T(s'|s,a)\Omega(o|s',a)\alpha_i(s)] \\ &= E[R(s,a)] + \\ &\quad \gamma \sum_{o\in O}\sum_{s'\in S} E[T(s'|s,a)]E[\Omega(o|s',a)]\alpha_i(s)] \end{aligned}$$

The second line follows from the linearity of expectations and the independence of parameter distributions T and Ω. Thus, we show that, in the absence of learning, solving the POMDP with the expected parameter values will maximize the expected reward.

In our experiments, we place Dirichlet priors, which provide a probability measure over valid multinomial distributions, over the observation and transition distribution parameters and Gaussian priors over the reward parameters. Updating the model distributions as we observe transitions and utterances is straightforward, but we need to know the user's

[1] For our tests, we vary the PBVI algorithm and belief sampling heuristics to optimize for the symmetries in our problem.

(hidden) state to update the correct $R(s,a)$, $T(s'|s,a)$, and $\Omega(o|s,a)$ distributions. The simple dialog structure allows us to infer the user's state history from the action that ended the exchange.

After each update, we have an improved understanding of the model, and we can update the dialog manager's policy to reflect the new model. Unfortunately, recomputing a new policy after an update can require large amounts of computation, and computing a fully-converged policy may be wasted effort if the parameters are fairly uncertain. Therefore, we tested two termination heuristics in addition to backing up to convergence: backing up once, and backing up proportionally to the variance reduction in the parameters. The intuition behind the final heuristic is that we wish to expend more computational effort when parameters become more certain.

Simulation Results

We first tested our approach on a simulated dialog manager. The user's states corresponded to five goal locations; actions included clarification queries (low cost) and motions (high cost if incorrect). The most likely model under our prior distribution over models (used to compute the initial dialog management policy) assumed observations had half the true error rate and that the user was more forgiving of mistakes. Averaged over 100 simulated trials, all heuristics improved performance similarly, but backing up proportionally to variance reduction did so with less computation.

Wheelchair Results

We also tested our algorithm on a real wheelchair dialog manager. More complex than the simulation model, the wheelchair's vocabulary included not only keywords associated with particular locations, but also several words—such as elevator—not initially mapped to any goal location. Our adaptive dialog manager learned associations for the unmapped keywords and the user's preferences (see example in Table 1). As seen in Figure 3, the learning dialog manager generally out-performed the non-learner.

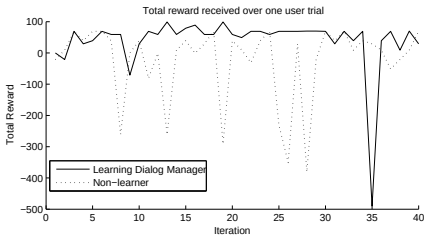

Figure 3: The learner (solid) generally outperforms the non-learner (dashed), only making one serious mistake (far right).

II. Improving Dialog Manager Robustness

The planning model in the previous section maximized the expected reward over the uncertain user model, but for more robust performance in a learning context, we would like to consider the uncertainty of the model itself in making decisions. For example, consider a scenario in which there are only two actions: *ask* and *confirm*. Suppose that under some prior belief p over reward parameters, we have the following relationship between the true rewards and their expected values under the prior:

$$R_{ask} > R_{conf} = E_p[R_{conf}] > E_p[R_{ask}], \qquad (9)$$

where R_{ask} is the reward for asking a general query and R_{conf} is the reward for asking a confirmation question. If the dialog manager attempted action *ask*, it would discover that its belief about R_{ask} was incorrect. However, if the dialog manager only makes decisions based on the rewards it expects to receive, $E_p[R_{conf}]$ and $E_p[R_{ask}]$, it will never try the action *ask*. Thus, the dialog manager will be stuck with a suboptimal policy. This situation will occur if the domain expert estimates the reward means incorrectly, even if the expert states that he is very unsure about some of the values he chose. We can avoid such mishaps by considering uncertainty in the user model when making decisions.

Table 1: Part of a wheelchair-user dialog. The dialog specification failed to specify how to handle the word 'elevator.' The learner realizes that while there are multiple elevators, the user most often uses it in the context of the Gates Tower.

Without learning
USER: Take me to the elevator.
The system does not know what 'elevator' means.
ROBOT: Where did you want to go?
USER: The Gates elevator please.
ROBOT: Do you want to go to the Gates Tower?
USER: Yes.
ROBOT: Going to the Gates Tower.
After learning:
USER: Take me to the elevator.
ROBOT: Do you want to go to the Gates Tower?
USER: Yes.
ROBOT: Going to the Gates Tower.

Parameter POMDP Formulation

To make the learning process more robust, we include the parameters of the POMDP as explicit additional hidden state in our model. Thus, the dialog manager can trade-off between actions that provide information about the user, provide information about the user's desired destination, or travel to a particular location. The dialog manager "learns" about the user's model as necessary, and learning does not require heuristics outside of the POMDP solver. Since large POMDPs can be computationally challenging to solve, in this section we focus only on learning user preferences, that is, the reward values associated with various actions.

Figure 4 shows how we incorporate the unknown reward parameters into the POMDP model to create a more robust dialog manager. Our new state space consists of a vector $\{s_u, \vec{s_r}\}$, where s_u is the user's state (where he wishes to go) and $\vec{s_r}$ is vector of the user's preferences. We use the word "user state" to refer to s_u and "preference state" to refer to $\vec{s_r}$. Our dialog manager contains five rewards for general queries, correct confirmations, incorrect confirmations, cor-

rect movements, and incorrect movements. Of the five unknown reward values, we can arbitrarily set two of them (in our case, we set the values of correct movements and confirmations); these values fix the translation and scale factor of the reward set. The remaining values must be learned.

We assume that we have a discrete set of reward values from which to choose and that these reward values are stationary over time. While assuming a discrete set of reward values may seem like a severe limitation, we note that policies are surprisingly robust to the choice of values. Intuitively, this should make sense because the changes in the policy—such as how many times to confirm a goal before performing a physical movement—are discrete, encompassing a range of possible user preference states.

For this approach, we require the user to give the dialog manager explicit reward feedback after every action. We extend our observation space to be $\{o_d, o_r\}$, where o_d is the speech to the dialog manager and o_r is a reward entered by the user. Our new belief b($s_u, \vec{s_r}$) represents the probability that the user is in state s_u *and* the rewards are given by $\vec{s_r}$.

Including the reward parameters in the POMDP increases the size of the state space and thus increases the computation required. As seen in Figure 4, however, independencies between the user's current goal (s_u) and his overall reward preferences ($\vec{s_r}$) allow us to factor transition and observation probabilities in the POMDP. For example, consider the probability of observing a particular $\{o_d, o_r\}$ in a state $\{s_u, \vec{s_r}\}$. The observed speech input does not depend on the user's reward model, so the observation probability factors as:

$$P(o_d, o_r | s_u, \vec{s_r}) = P(o_d | s_u) \cdot P(o_r | s_u, \vec{s_r})$$

Similar factorizations exist for the other transition and observation probabilities.

Simulation Results

We tested our approach on a sample problem with 48 preference states (for 336 total states) and 9 discrete reward values (making 72 possible observations). While the factorizations help when solving the POMDP, this POMDP was still too large to solve efficiently. Not only are many more belief points required, but the matrix computations for solving the POMDP are nearly cubic in the size of the state space. The computations become more tractable, however, if we note that our dialog manager's initial policy need not be refined for all the possible users it may encounter: at the beginning, all the dialog manager really needs to know is how to behave when it knows little about the user's preferences.

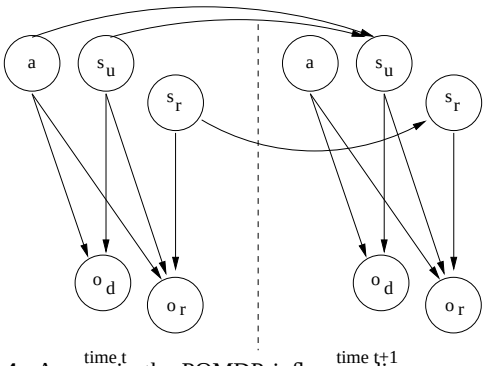

Figure 4: Arrows in the POMDP influence diagram connect elements that affect other elements. The robot's action a and the previous user state affect the current user state s_u, while the user's preferences s_r never changes. The observed reward o_r depends on the user's state and his preferences and the observed dialog o_d depends only on the user's state.

Although the dialog manager knows little about the user in its starting belief, the user's reward inputs will quickly make some preference states highly unlikely. As we gain additional information about the user's preferences, the dialog manager can resample beliefs in the relevant parts of the belief space: for example, once it is fairly sure about how the user feels about the wheelchair driving to the wrong location, it can focus on how the user may feel about general queries or incorrect confirmation questions. By performing additional backups on the new belief set, we can then refine the dialog policy in the parts of the belief space where it is most likely to matter. Thus, our overall policy improves as we discover the user's true set of preferences. (Regarding how much we should refine a policy given a new observation, we can choose either to backup to convergence or follow the variance-reduction heuristic from the first approach.)

Recall that one pitfall of using the expected values of the parameters is that we can get stuck in an incorrect user model because the dialog manager is afraid of trying a certain action. Figure 5 shows how incorporating the parameters into the model prevents this issue; as long as our prior places a non-zero probability over all possible users preferences, we will still converge to the correct dialog policy.

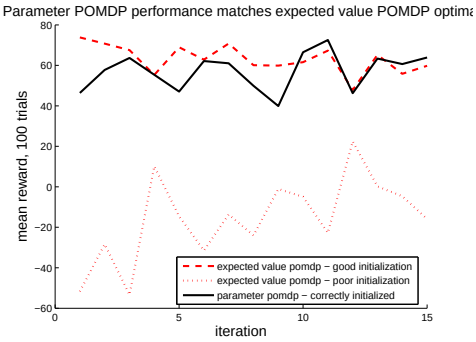

Figure 5: Performance of parameter POMDP initialized with a good prior compared to different initializations of the expected value POMDP. The parameter POMDP achieves the higher (correct) level of performance without sensitivity to initialization. The parameter POMDP seems to do slightly worse than the well-initialized expected value POMDP, but the difference is not statistically significant: on the final trial, the expected value POMDP has median 85 and IQR 16.5, and the parameter POMDP has median 86 and IQR 11. The poorly-initialized expected value POMDP reaches a median of 26.5 and IQR of 207 after 120 trials.

The question remains of what starting belief we should choose. An obvious choice, if we had no knowledge of the user's preferences, would be a prior that gave equal weight to each preference state. This approach will converge to the correct model, but since our initial policy was not fully refined, we may do better by being more conservative in our

initial prior (that is, biasing our prior to preference states with more severe penalties). If this prior also has full support, we will still converge to the user's actual preference state. However, since we will initially act more conservatively, we will often suffer from fewer major failures during the learning process. Figure 6 shows simulation results in which a conservative prior aids the learning process.

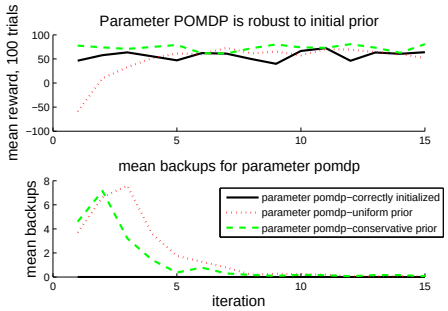

Figure 6: All three priors converge to the same steady-state performance, but choosing a conservative prior allows us to do as well as if we had known the user's preference state initially. All solutions were backed up to convergence.

III. Applying Meta-Actions

In the planning and learning models described so far, we required the user to give the agent explicit feedback after each interaction and encoded the rewards as part of the observation space. While requiring explicit feedback is a standard element of the reinforcement learning framework, it is unsatisfactory in an assistive robotics setting for two reasons. First, providing feedback after each action may be tiresome to the user. Second, the robot must make mistakes to learn about consequences of a poor decision. In our final approach, we remedy these issues by introducing "meta-actions," or queries about what the dialog manager should have done.

In the context of learning a user's preference model, meta-actions may take the form of questions such as "If I am 90% certain that you wish to go to some destination, should I confirm this with you or proceed directly to the goal location?" If the user tells dialog manager to confirm the action in the future, we can infer that the user places a high penalty on incorrect decisions *without experiencing its effects*. However, if the user tells the wheelchair to proceed without confirmation, we have learned that the user is willing to risk an incorrect decision to avoid being asked multiple questions.

The user's response to the meta-action allows us to prune inconsistent preference states $\vec{s_r}$ from the belief space. For example, to determine which $\vec{s_r}$ are feasible for the previous question, we first find the optimal POMDP policy without including the meta-actions. Next, for all $\vec{s_r}$ and $b(s_u)$ with 90% probability mass in one state, we check if predicted action matches the user's response. Although we have to solve the dialog manager POMDP in two stages—without and with meta-actions—all computations may be performed before user interactions begin. We also note that in order to know the policy for a specific user preference state, we do not need to solve the entire large POMDP—we only need to solve our small dialog model (as described in Figure 2) for a specified user reward model.

Note that our meta-actions are not of the form, "My current belief is b, what should I have done?" Instead, we phrase them as hypothetical questions, independent of the dialog manager's current beliefs of the user's goals and preferences. Phrasing the questions in such a way helps make solving the larger POMDP tractable—if the results of a meta-action depended on the agent's belief, instead of the underlying state, the resulting POMDP would have to solved over a continuous belief space instead of over the discrete state space.

Since the meta-actions are hypothetical, and not grounded to the dialog manager's current belief, one may be concerned that the robot will ask meta-actions during "random" times in the conversation and frustrate the user. However, the POMDP will ensure that the meta-actions are only used when the robot has reached a point in the conversation where it is unsure about the user's preference model, and the different possible preference models have different optimal policies. For example, at the beginning of a conversation, the correct action for the dialog manager may be to ask "Where do you want to go?" If the robot receives a somewhat noisy utterance that might be "cafe," then the next correct action may depend on the user's preferences: if they don't mind occasional mistake, perhaps the robot should go to the cafe, but if the user finds incorrect movements extremely frustrating, the robot should confirm what it thinks it heard. Thus, the dialog manager will now ask a meta-action to determine the user's tolerance to movement mistakes.

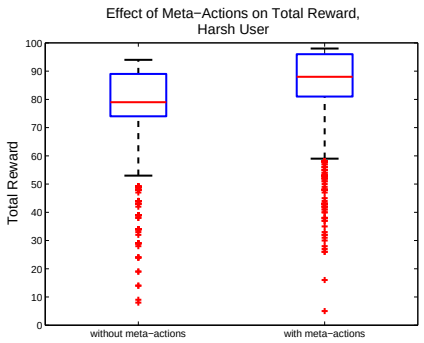

Figure 7: Box-plot of total rewards for the scenario where the "real" user has reward -50 for incorrect confirmations and -500 for incorrect movements. By asking about the user's preferences, the dialog manager with meta-actions is able to avoid actions that will frustrate the user. Each test had 100 trials of 30 dialogs.

Simulation Results

Solving the large POMDP still poses computational challenges, and because the meta-actions have effects spanning many preference states, we cannot use the resample-and-refine method from the previous approach. We focus on a single example to demonstrate the effectiveness of meta-actions. As in the last approach, we assumed that the transition and observation models were known. General queries cost -5, correct confirmations -1, and correct movements 100. Incorrect movements cost either -500 or -100, and incorrect confirmation cost either -10 or -5. (The optimal pol-

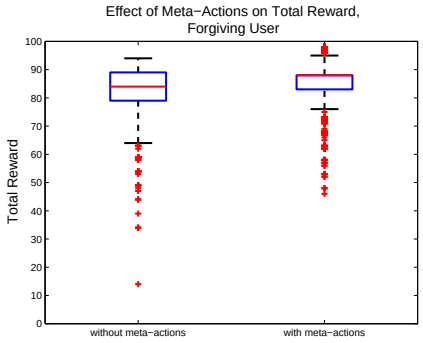

Figure 8: Box-plot of total rewards for the scenario where the "real" user has reward -5 for incorrect confirmations and -100 for incorrect movements. The agent still takes meta-actions, but the information is less useful since there reward for an incorrect confirmation, which has a larger impact on the policy, does not contain a major pitfall to avoid. Each test had 100 trials of 30 dialogs.

icy is fairly insensitive to the user's reward model, so we chose these somewhat extreme values to demonstrate how meta-actions affect the dialog.) In this scenario, the dialog manager could ask three meta-action queries:

1. If I am 90% certain of where you want to go, should I still confirm with you before going there?
2. If I'm clueless, should I list options for you to confirm?
3. If I'm only 40% sure that you want to go to a certain location, should I still try confirming that location with you?

Figure 7 shows a box-plot of the total rewards received with and without meta-actions where the "real" user has reward -10 for incorrect confirmations and -500 for incorrect movements. Questions about the user's preferences allow the dialog manager to avoid asking confirmation questions that are likely to be incorrect. (As it turns out, the reward for incorrect confirmation, which effectively trades between whether the robot should ask a general query and try to confirm a possible goal location, plays a much larger role in the policy than the reward for the incorrect movement.) We also note that even the dialog manager without meta-actions, although suboptimal, does not fare too badly because its belief is always split between the four possible reward states. Thus it is still more conservative than a policy that believed itself to be in an incorrect user state.

Meta-actions provide the most improvements when there are pitfalls to avoid, but they can also safely discover that the user is actually more forgiving than the robot previously considered. Figure 8 shows a box-plot of the total rewards received with and without meta-actions where the "real" user has reward -100 for incorrect movements and -5 for incorrect confirmations. While both dialog policies have similar performance, the policy without meta-actions fares slightly worse because part of its belief always thinks that the simulated user's preference state might be more severe than it actually is (and thus it is more conservative).

Wheelchair Results

We implemented the meta-action version of our parameter POMDP on our robotic wheelchair using the model de-

Table 2: The following three dialogs (collected from a volunteer in our lab), demonstrate how a typical conversation changes meta-actions are introduced.

Typical conversation
USER: Hello.
ROBOT: Where do you want to go?
USER: I want to go to Bryt's desk.
ROBOT: Do you want to go to Bryt's desk?
USER: Yes.
ROBOT: Going to Bryt's desk.
With Meta-Actions: Conversation One
This user preferred to be asked confirmation questions.
USER: Hello.
ROBOT: If I don't know where you want to go, should I list options?
The system uses this query to determine if the user is tolerant to incorrect confirmation questions.
USER: Yes.
ROBOT: Do you want to go to Seth's office?
USER: No, I want to go to Nick's office.
ROBOT: Do you want to go to Nick's office?
USER: No, I mean yes.
Unfortunately, the user's response is ambivalent enough that the system does not recognize the user's intent.
ROBOT: Do you want to go to Bryt's desk?
USER: No.
ROBOT: Do you want to go to the group meeting?
USER: No.
ROBOT: Do you want to go to coffee machine?
USER: No.
ROBOT: Do you want to go to Nick's office?
The robot continues to operate by listing locations, but it picks the most likely one to repeat.
USER: Yes.
ROBOT: Going to Nick's office.
With Meta-Actions: Conversation Two
This user preferred to be asked more general questions.
USER: Hello.
ROBOT: If I don't know where you want to go, should I list options?
USER: No, now take me to the coffee machine.
ROBOT: If I'm 95% sure about where you want to go, should I still confirm with you first?
Knowing the user's tolerance to going to the wrong place didn't matter in the previous context, but now it does affect how the conversation proceeds.
USER: Yes.
ROBOT: Where do you want to go?
The robot has some idea of where the user wishes to go from the first utterance, but it needs additional information. It also knows that the user is not tolerant to incorrect confirmations.
USER: The coffee machine.
ROBOT: Going to the coffee machine.

scribed above. While it is difficult to provide quantitative results without explicit reward feedback, Table 2 contains snippets of dialogs between the wheelchair and a human that show how the dialog manager's actions change after it uses meta-actions to learn more about the human's preferences.

Related Work

As we mentioned before, most POMDP-based dialog managers assume that an accurate user model is available (Williams, Poupart, & Young 2005; Pineau, Roy, & Thrun 2001; Williams & Young 2005). In the field of POMDP learning, the Medusa algorithm (Jaulmes, Pineau, & Precup 2005) places priors over the POMDP parameters, samples and solves many POMDPs from these prior distributions, and uses this sample to vote for a good policy. While guaranteed to converge, Medusa does not have an explicit method for choosing actions to reduce uncertainty in the parameters. Our first method shares Medusa's Bayesian approach; however, we begin with only one POMDP dialog manager instead of several samples.

The Beetle algorithm (Poupart *et al.* 2006), another Bayesian approach to planning with uncertain models, plans in uncertain MDPs by incorporating the MDP parameters as hidden state in a POMDP. As in our later two approaches, this allows the robot to trade between actions that will improve its knowledge of the model and actions that will gain rewards. While Beetle focuses on learning system dynamics (which are readily available in the MDP setting, since the state is fully observable), we focus on learning the user's preference or reward model.

While we focus exclusively on Bayesian approaches to planning with uncertain models, there do exist minimax approaches to finding robust MDP policies when transition matrices are uncertain (Nilim & Ghaoui 2004; Xu & Mannor 2007). Geared toward applications where variations are unavoidable, such as tolerances on factory machinery, these approaches do not incorporate learning into their model.

Conclusions

We presented three POMDP-based approaches to dialog management on a robotic wheelchair. Our first approach learned all aspects of the POMDP user model—transitions, observations, and rewards—and based its decisions only the expected values of the uncertain parameters. To improve the quality of this approach, we noted that replanning only corrects for loss in performance due to incomplete convergence; by estimating the variance due to the uncertainty in both the dialog and the user model we can judge the effectiveness of additional planning. While efficient, our first approach was not necessarily robust.

In our second two approaches, we folded the unknown parameters of the user model into an even larger POMDP. We also introduced the idea of meta-actions, or questions about what the dialog manager should have done. Since the large POMDPs are difficult to solve, we focused only on learning the user's reward model. We showed that we were able to learn the user's preferences robustly and that we could meta-actions as a form of implicit user feedback. In our future work, we plan to extend the use of meta-actions to also gaining information about other aspects of the user model, such as what words they tend to use when referring to particular locations.

References

Doshi, F., and Roy, N. 2007. Efficient model learning for dialog management. In *HRI '07: Proceedings of the 2nd Conference on Human-Robot Interaction (to appear)*.

Glass, J. 2003. A probabilistic framework for segment-based speech recognition. *Computer Speech and Language* (17):137–152.

Gordon, G. J. 1995. Stable function approximation in dynamic programming. In *Proceedings of the Twelfth International Conference on Machine Learning*. San Francisco, CA: Morgan Kaufmann.

Hoey, J.; Poupart, P.; Boutilier, C.; and Mihailidis, A. 2005. Pomdp models for assistive technology.

Jaulmes, R.; Pineau, J.; and Precup, D. 2005. Learning in non-stationary partially observable markov decision processes. Workshop on Non-Stationarity in Reinforcement Learning at the ECML.

Litman, D.; Singh, S.; Kearns, M.; and Walker, M. 2000. NJFun: a reinforcement learning spoken dialogue system. In *Proceedings of the ANLP/NAACL 2000 Workshop on Conversational Systems*.

Nilim, A., and Ghaoui, L. 2004. Robustness in markov decision problems with uncertain transition matrices.

Pineau, J.; Gordon, G.; and Thrun, S. 2003. Point-based value iteration: An anytime algorithm for pomdps.

Pineau, J.; Roy, N.; and Thrun, S. 2001. A hierarchical approach to pomdp planning and execution. In *Workshop on Hierarchy and Memory in Reinforcement Learning (ICML)*.

Poupart, P.; Vlassis, N.; Hoey, J.; and Regan, K. 2006. An analytic solution to discrete bayesian reinforcement learning. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, 697–704. New York, NY, USA: ACM Press.

Roy, N.; Pineau, J.; and Thrun, S. 2000. Spoken dialogue management using probabilistic reasoning. In *Proceedings of the 38th Annual Meeting of the ACL*.

Williams, J., and Young, S. 2005. Scaling up pomdps for dialogue management: The "summary pomdp" method. In *Proceedings of the IEEE ASRU Workshop*.

Williams, J. D.; Poupart, P.; and Young, S. 2005. Partially observable markov decision processes with continuous observations for dialogue management. In *Proceedings of SIGdial Workshop on Discourse and Dialogue 2005*.

Xu, H., and Mannor, S. 2007. The robustness-performance tradeoff in markov decision processes. In Schölkopf, B.; Platt, J.; and Hoffman, T., eds., *Advances in Neural Information Processing Systems 19*. Cambridge, MA: MIT Press.