Spoken Language Interaction with Model Uncertainty: An Adaptive Human-Robot Interaction System

Finale Doshi and Nicholas Roy

Computer Science and Artificial Intelligence Laboratory,
Massachusetts Institute of Technology,
32 Vassar St., Cambridge, MA 02139.
{finale|nickroy}@mit.edu

Abstract

Spoken language is one of the most intuitive forms of interaction between humans and agents. Unfortunately, agents that interact with people using natural language often experience communication errors and do not correctly understand the user's intentions. Recent systems have successfully used probabilistic models of speech, language, and user behavior to generate robust dialog performance in the presence of noisy speech recognition and ambiguous language choices. However, decisions made using these probabilistic models are still prone to errors due to the complexity of maintaining a complete model of human intentions.

In this paper, we describe a decision-theoretic model for human-robot interaction using natural language. Our algorithm is based on the Partially Observable Markov Decision Process (POMDP), which allows agents to choose actions that are robust not only to uncertainty from noisy or ambiguous speech recognition but also unknown user models. Like most dialog systems, a POMDP is defined by a large number of parameters that may be difficult to specify *a priori* from domain knowledge, and learning these parameters from the user may require an unacceptably long training period. We describe an extension to the POMDP model that allows the agent to acquire a linguistic model of the user online, including new vocabulary and word choice preferences. Our approach not only avoids a training period of constant questioning, but also allows the agent to actively query for additional information when its uncertainty suggests a high risk of mistakes. We demonstrate our approach both in simulation and on a natural language interaction system for a robotic wheelchair application.

1 Introduction

Spoken language is one of the most intuitive forms of interaction between humans and agents. The ability for an agent to understand natural language commands can be especially useful when interacting with people who may not be trained to use more conventional robotic interfaces. For example, when intelligent robots are used as a medical assistive technology, both care-givers and care recipients rarely have experience with robots. In fact, as most people reach a comfortable level with technology, they often begin interacting with the technology as they would another person [Reeves and Nass, 1996]. As a result, natural language is an attractive way for robots to interact with users, minimizing the need for training and increasing the likelihood that the robot will be accepted by the intended users.

Spoken language interaction with a mobile robot is most often implemented as a process consisting of the following systems:

- A speech recognition system to recover the sequence of words uttered by the user.
- A dialog management system to infer the user intent (e.g., a command for the robot to perform some action).
- A motion planning system to respond to the user intent (e.g., go to some location).

For a robot to be able to use natural language to interact with people, it is critical that the robot react to the user's language in ways that are consistent with the user's expectations of reasonable behavior. In most human-robot interaction domains, and especially in medical assistive domains, a robot that violates user expectations and appears to make errors periodically will be quickly ignored as untrustworthy by the human users. As a result, natural language interaction must be robust to various errors in the models it uses to predict user intent.

One challenge is simply correctly identifying the spoken words, since speech recognition errors will make the input to the dialog system a noisy process. For example, the system may hear the words "coffee machine" when the user asks the robot to go to "copy machine." A dialog manager that can model the likelihood of recognition errors and act accordingly will have increased robustness in satisfying the user's request correctly. Secondly, the language itself may be ambiguous or confusing, and the system may not know what words, such as "kitchen," "kettle," and "microwave," may refer to the same location. In contrast, users may also use the same word to refer to multiple locations, such as "elevator" when there are multiple elevators. A dialog manager that can model confusion or ambiguity in the user request and act accordingly will have better success and robustness.

Another set of challenges arise because people may have different preferences in how a system responds to the interaction. For example, different users may have different tolerances of mistakes by the system, or have differing levels of patience toward confirmation questions by the system. Unfortunately, humans are challenging to model, and real-world interaction domains typically require a large number of parameters that are difficult to specify *a priori*, leading to performance errors. Even when the models are acquired from data of human behavior, the long training periods and the required amount of data may be frustrating to the user. It is therefore also desirable that a dialog manager be able to take an initial, potentially erroneous model and improve it online. A dialog manager that can represent its confidence in its model of the user and can choose actions appropriately will not only to avoid errors due to model uncertainty, but it will also be able take actions to learn more about the user and improve the model for the future.

If the dialog manager has an accurate probabilistic model of the word error rates and user preferences, then the system can infer a distribution over possible recognized utterances, user requests, and user models at each point during the interaction. Given such a distribution, the planning framework known as the Partially Observable Markov Decision Process (POMDP) can be used to compute decision-theoretic policies for the interaction. These policies have the desirable property of choosing actions (i.e., responses to the user) that minimize expected cost, as opposed to simply choosing the action that best matches the most likely user request and model. Additionally, POMDP policies can explicitly model the value of asking questions and gathering information; the optimal POMDP policy trades between asking questions to reduce uncertainty (thus avoiding errors), and fulfilling the user's request within a reasonable amount of time. The ability to reason about actions under uncertainty has led to the successful use of POMDPs in several assistive health-care [Roy et al., 2000, Hoey et al., 2005, Fern et al., 2007] and dialog management domains [Williams and Young, 2005, Litman et al., 2000].

In this paper, we describe how a POMDP model for human-robot interaction can be extended to allow the natural language model to be learned online. Although POMDPs have been used previously for human-robot interaction, the problem of online learning of decision-theoretic models of natural language interaction has not been addressed. In section 4 we describe an approach to passive learning of the interaction model, in which the system receives feedback after every action. We present an algorithm that allows the dialog manager to interact with the user while improving its model, an approach that is particularly important for a robotic wheelchair application that must adapt to users and new environments. The dialog manager adjusts its actions based on user feedback, and, over time, it learns how to maximize the reward it expects to receive. By simply analyzing the interaction, the dialog manager also updates its model of the voice recognition system. Since the agent only evolves a single model, the passive learner can easily be run in real-time settings, and we show that this approach is usually effective in converging to the optimal policy.

Unfortunately, the approach of passively learning a model from reward feedback suffers from several drawbacks. First, the dialog manager must make mistakes to learn about the consequences of a poor decision. The frequency of these mistakes quickly drops, but the high variance and perceived unreliability of the system during this learning phase quickly leads to user dissatisfaction with the overall system. Second, requiring the user to supply reward feedback after each action may be tedious, leading to frustration and inaccurate results. Even with a patient, well-intentioned user, people are notoriously bad at giving numerical feedback; inaccurate responses may lead to incorrect learning. Finally, without being explicitly aware of the quality of its knowledge, the dialog manager cannot know if its decisions are

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

based on a solid or unclear understanding of the world. To reduce the feedback requirements and to increase the planner robustness, in section 5 we extend our learning algorithm to allow the system to actively seek feedback about specific actions. We show how the dialog manager can explicitly maintain an estimate of the quality its knowledge about the world. This knowledge allows the dialog manager to assess its confidence in its own decision-making and decide when additional training is needed. We introduce the concept of a meta-query, that is, a question about an action that the dialog manager should take. These meta-queries take an intuitive form:

> "I think you definitely want me to go to the printer. Should I go to the printer?"

By introducing these meta-queries, the dialog manager can discover new vocabulary, word choices, and user preferences such as risk aversion. Although this active learning scheme introduces an additional computational cost, we present an approximation to our first POMDP that allows the dialog manager to determine when to ask for additional information, easing the training burden on the user and reducing the variance of the policy during learning.

Throughout this paper, we focus on natural language interaction for assistive technology such as the wheelchair shown in Figure 1. Our goal is to design a human-robot interaction system, or dialog manager, that allows both the user of the wheelchair and a caregiver to give natural instructions to the wheelchair, as well as ask the wheelchair computer for general information that may be relevant to the user's daily life.

2 The POMDP Model

Formally, a POMDP consists of the n-tuple $\{S, A, O, T, \Omega, R, \gamma\}$. The set S is a set of states, which, in the dialog management setting, correspond to what the user actually desires from the interaction. For example, in our robotic wheelchair scenario, the states may correspond to locations to which the user wishes to go or information the user wants from the system. The dialog manager cannot directly observe the user's desire and must infer it from a set of observations O. For our basic dialog manager, we take the observations to be the number of times certain keywords (such as those corresponding to locations or information requests) occur in an n-best list from the voice recognition system. However, the POMDP can easily accommodate observations that are comprised of more sophisticated natural language processing of the voice recognition output—our algorithms only require that the number of possible observations the dialog manager may receive is discrete and finite. The set A represents actions that the dialog manager may take in response to what it hears. The actions may include asking for a clarification, commanding the robot to perform some physical movement, or doing nothing.

In Figure 2, we show a cartoon of a simple dialog model. Initially, we model the user as being in the left-most node, a "start" state in which he or she does not desire anything from the dialog management system. Then, at some point in time, the user speaks to the robot to indicate that he or she wants it to perform some task. We denote this

step by the set of vertical stack of nodes in the center of the model. Each node represents a different task. The dialog manager must now interact with the user to determine what he wants. Once the task is successfully completed, the user transitions to the right-most "end" node, in which he or she again does not desire anything from the robot.

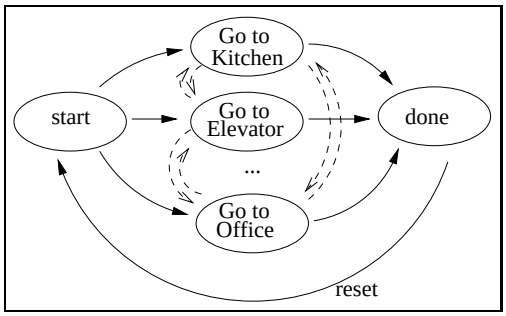


Figure 2: A toy example of a dialog POMDP. The nodes in the graph are different states of the dialog (i.e., user requests). Solid lines indicate likely transitions; we assume that the user is unlikely to change their request before their original request is fulfilled. The system automatically resets once we reach the end state.

The transition function T and observation function O of the POMDP model describe how the sets of states, actions, and observations behave. The transition function $T(s'|s, a)$ places a probability distribution over the states to which the user in state s may transition if the dialog manager takes action a. In our wheelchair scenario, the initial transition is fairly complex—the system must learn what requests are *a priori* more likely—but we assume that for the most part, a user will stick to his or her original request until the robot successfully completes the desired task. Transitions between different tasks, which would correspond to the user changing his mind about what he wants the system to do, are considered unlikely.

Similarly, the observation function $\Omega(o|s, a)$ places a probability distribution over the observation o that may be seen in state s after taking action a. The observation model encodes what keywords the dialog manager is likely to hear given its most recent action. For example, the model might encode that the keyword "coffee" is commonly heard if the dialog manager asks "Where would you like to go?" and the user wishes to go to the coffee machine. However, if the dialog manager had chosen to ask "Do you want to go to the coffee machine?" the observation model might state that the most likely observation is the word "yes." The POMDP model allows for different observations to be received in the same state (for example, both "coffee" and "machine" may be commonly heard if the user wishes to go to the coffee machine) but the assumption of the standard POMDP is that we do know the probabilities of hearing each keyword.

The reward function $R(s, a)$ specifies the immediate reward the dialog manager receives for taking action a when the user is in state s. In the wheelchair scenario, where the dialog manager is trying to determine where the user wishes to go, a clarification action such as "Do you want to go to the coffee machine?" may carry a small negative reward for causing a minor inconvenience, while incorrectly driving to the coffee machine may carry a large penalty for taking the user to the wrong location. To gain a high positive reward for completing a task and avoid large negative reward for taking an incorrect movement, the dialog manager may decide to accept small penalties for asking clarification questions. Finally, the discount factor $\gamma \in [0, 1)$ measures the relative importance of current and future rewards. A large γ indicates that future rewards are nearly as valuable as current rewards; thus the dialog manager will be more patient—and perhaps ask many questions—before committing to an action. Conversely, a small γ indicates that time is of the essence; thus the dialog manager may choose to risk a somewhat hasty decision rather than doing nothing and having time run out.

In making decisions, the dialog manager can only choose actions based on the knowledge of past actions it has tried and past observations it has received. It can never know what the user actually wanted. In general, the optimal action to take now will depend on *all* prior actions and observations; however, keeping a history of all conversations to date—and using that entire data set to make decisions—can become quite cumbersome. Fortunately, it can be shown that when it comes to making a decision, what really matters is the dialog manager's belief of what the user may desire. More formally, the belief is a probability distribution over states. If the agent takes some action a and hears

observation o from an initial belief b, it can easily update its belief using Bayes rule:

$$b^{a,o}(s) = \frac{\Omega(o|s',a)\sum_{s\in S} T(s'|s,a)b(s)}{\sum_{\sigma\in S}\Omega(o|\sigma,a)\sum_{s\in S} T(\sigma|s,a)b(s)} \quad (1)$$

This probability distribution will evolve as the dialog manager asks clarification questions and receives responses. Thus, at any point in time, the dialog manager knows how likely it is that the user is trying to request each possible task.

Intuitively, we can see how the belief can be used to select an appropriate action. For example, based on the voice recognition output, if the dialog manager believes that the user may wish to the coffee machine or the copy machine, then it may make sense for it to ask the user for clarification before commanding the wheelchair to one of the locations. However, if the dialog manager is almost certain that the user wants to go to the coffee machine, then the best course of action may be to command the robotic wheelchair to the coffee machine and avoid hassling the user with further questions. An optimal dialog manager will, for each belief b, choose an action that will maximize its long-term expected reward. We call the mapping from beliefs to actions a policy, and we represent this mapping using the concept of a value function $V(b)$. The value of a belief is defined to be the expected long-term reward the dialog manager will receive if it starts a user interaction in 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 also unique and satisfies the Bellman equation:

$$\begin{aligned} V(b) &= \max_{a\in A} Q(b,a), \\ Q(b,a) &= R(b,a) + \gamma \sum_{b'\in B} T(b'|b,a)V(b'), \\ Q(b,a) &= R(b,a) + \gamma \sum_{o\in O} \Omega(o|b,a)V(b_a^o), \end{aligned} \quad (2)$$

where $Q(b,a)$ represents the expected reward for starting in belief b, performing action a, and then acting optimally. The last equation follows 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 b after a Bayesian update of b using equation (1), and $\Omega(o|b,a)$, the probability of seeing o after performing a in belief b ($\sum_{s\in S}\Omega(o|s,a)b(s)$).

The Bellman equation may be solved iteratively:

$$V_n(b) = \max_{a\in A} Q_n(b,a), \quad (3)$$

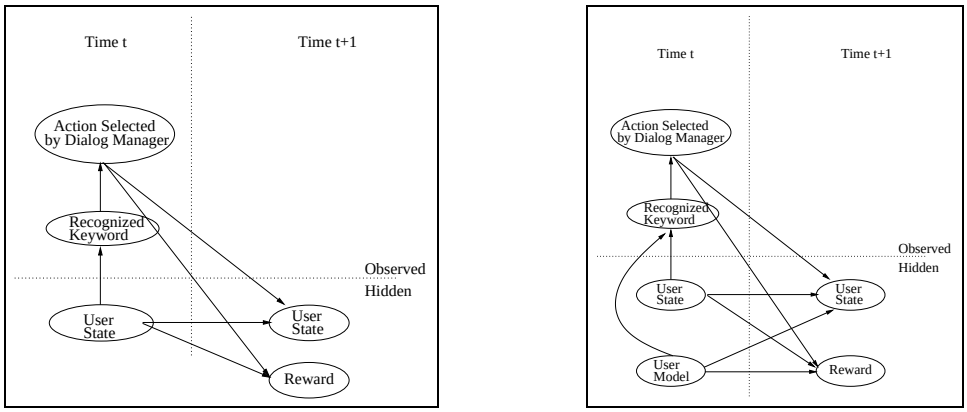
$$Q_n(b,a) = R(b,a) + \gamma \sum_{o\in O} \Omega(o|b,a)V_{n-1}(b_a^o). \quad (4)$$

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 (2).

The exact solution to equation (3) using an iterative backup approach is exponentially expensive, so we approximate the true backup operation by backing up at only a small set of beliefs. The choice of beliefs determines the quality of the approximation and thus the performance of the dialog manager. One approach is the "Point-Based Value Iteration" algorithm [Pineau et al., 2003], which involves starting with some belief b_0 (such as being in a "dialog-start" state). Then for each action a, we sample a user response o from the observation distribution and compute the updated belief state b_o^a (simulating the effect of one exchange between the user and the dialog manager). We add the farthest new beliefs to our set and repeat the process until we accumulate the desired number of beliefs. Since the beliefs represent the dialog manager's confusions over the user's request, picking beliefs reachable from the starting belief focuses our computation in situations the dialog manager is likely to experience.

3 Modeling POMDP Uncertainty

The behavior of the dialog manager derived from solving equations (3) and (4) depends critically on accurate choices of the transition probabilities, observation probabilities and the reward. For example, the observation parameters affect



(a) Standard POMDP model (b) Model-Uncertainty POMDP

Figure 3: (a) The standard POMDP model. (b) The extended POMDP model. In both cases, the arrows show which parts of the model are affected by each other from time t to $t+1$. Not drawn are the dependencies from time $t+1$ onwards, such as the user state and user model's effect on the recognized keyword at time $t+1$.

how the system associates particular keywords with particular requests. Similarly, the reward function affects how aggressive the dialog manager will be in assuming it understands a user's request, given limited and noisy information. An incorrect specification of the dialog model may lead to behavior that is either overly optimistic or conservative, depending on how accurately the model captures the user's expectations on the interaction.

At the same time time, learning all the parameters required to specify a complete dialog model can require a prohibitively large amount of data. However, while the model parameters may be difficult to specify exactly, we can often provide the dialog manager with an initial estimate of the model parameters that will generate a reasonable policy that can be executed while the model is improved. For example, even though we may not be able to attach an exact numerical value to driving a wheelchair user to the wrong location, we can at least specify that this behavior is undesirable. Similarly, we can specify that the exact numerical value is initially uncertain. As data about model parameters accumulate, the parameter estimates should converge to the correct underlying model with a corresponding reduction in uncertainty.

In order to formally specify such an adaptative interaction model, we extend the POMDP model to include the dialog parameters. Recall from the previous section we introduced the POMDP as a model of decision making conditioned on the current belief, that is, the probability of how likely a user is to be making a particular request. Figure 3(a) depicts this conventional model, where the arrows in the graph show which parts of the model are affect each other from time t to $t+1$. Although the variables below the "hidden" line in Figure 3(a) are not directly observed by the dialog manager, the parameters defining the model (i.e., the parameters in the function giving the next state) are fixed and known *a priori*. For instance, the reward at time t is a function of the state at the previous time and the action chosen by the dialog manager.

If the model parameters are not known *a priori* because the model is uncertain—for example, how much reward is received by the agent given the previous state and the action selected—then we extend the concept of the belief to also include the agent's uncertainty over possible models. In this new representation, which we call the *model-uncertainty POMDP*, both the user's request and the model parameters are hidden. Figure 3(b) shows this extended model, in which the reward at time t is still a function of the state at the previous time and the action chosen by the dialog manager, but the parameters are not known *a priori* and are therefore hidden model variables that must be estimated along with the user state. As the system designer, we can encode our knowledge of the system into the dialog manager's initial belief over what dialog models it believes are likely—a Bayesian prior over models—and let the agent improve upon this belief with experience.

We will restrict our discussion to inference of unknown transition, observation and reward function parameters and assume that the sets S, A, and O are known. This assumption is often reasonable because a specification of the state, action, and observation spaces are usually fixed in the engineering stage of an application. For example, if the dialog

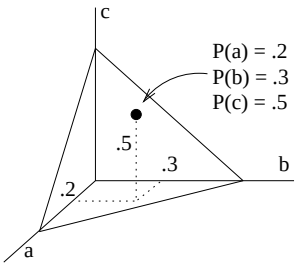


Figure 4: An example simplex for a multinomial that can take three different values (a,b,c). Each point on the simplex corresponds to a valid multinomial distribution; the Dirichlet distribution places a probability measure over this simplex.

manager is designed to service particular requests—be it driving a wheelchair to certain locations or automatically booking airline tickets—the set of states S is known when the system is designed. Similarly, we can pre-specify what kinds of statements and movements a robot can make during an interaction. Specifying the observation space—that is, all the keywords are important when interacting with humans—is more tricky, but we assume that the designer has sufficient knowledge of the application to create a list of keywords. We also note that we can always augment the list of keywords as the system interacts with users and discovers patterns in their vocabulary.

The need to represent the beliefs over models raises the question of how to represent beliefs over model parameters. The user state space is a discrete state space, so a standard histogram or multinomial distribution could be used. However, the rewards are continuous-valued, and parameters such as the transition functions T are continuous parameters of distributions themselves; a distribution over T is effectively a distribution over distributions. We describe how we model the uncertainty in different model parameters below.

3.1 Reward Uncertainty

Because each reward is a single scalar value conditioned on the state and action, we have considerable flexibility over distributions to use. To set an absolute scale for the reward values, we fix a large positive reward value for correctly satisfying a user's request, and a small penalty for confirming a correct request with the user (for the minor inconvenience of causing the user to communicate with the robot, instead of immediately satisfying the user's request). These two values set a scale and spread for the remaining reward values.

Our initial algorithm requires knowledge only of the expected reward values, so in theory, any distribution may be used. Our first approach employs Gaussian distributions, as they are easy to update. While it may be reasonable in some cases for a system designer to specify to the agent reasonably accurate expected reward values for each action, we relax this requirement in section 5, where we require the designer only to specify a range for each reward parameter, and we assume that the reward values are uniformly distributed between these ranges.

3.2 Transition and Observation Uncertainty

Since the state space is discrete, the T and Ω parameters define collections of histograms or multinomial distributions. A prior over the transition and observation parameters must therefore be a distribution over possible histograms. Figure 4 shows how one can visualize the space of all valid multinomial distributions for a discrete random variable X which can have three outcomes—consider a dialog manager that must choose to go left, right, or forward. Each possible distribution $p(X)$ over these three options corresponds to a point on the triangular simplex. We want the dialog manager to be able to update its belief efficiently, so we choose a Dirichlet prior over the transitions and observations. The Dirichlet provides a measure of the likelihood of each such distribution and is the conjugate prior to the multinomial, allowing the Dirichlet to be updated directly from the observed data. As the dialog manager's confidence in a particular model of user behavior increases, the probability mass of the Dirichlet distribution becomes increasingly concentrated around a single point.

More formally, given a set of parameters $\alpha_1 ... \alpha_m$, the likelihood of the discrete probability distribution $Pr[x = 1] = p_1, Pr[x = 2] = p_2, ..., Pr[x = m] = p_m$ is given by

$$P(\underline{p}; \underline{\alpha}) = \eta(\underline{\alpha}) \prod_i^m p_i^{\alpha_i - 1} \delta(1 - \sum_i^m p_i),$$

where η is a normalizing constant. The expected values of the Dirichlet distribution are given by

$$E[p_i|\underline{\alpha}] = \frac{\alpha_i}{\sum_j^m \alpha_j}, \tag{5}$$

and the mode is

$$\arg\max_{p_i}(p_i|\underline{\alpha}) = \frac{\alpha_i - 1}{\sum_j^m \alpha_j - m}. \tag{6}$$

Intuitively, the Dirichlet parameters correspond to counts of how often each event has occurred. Thus, the relative values of the α parameters indicate how likely each event is, and the overall magnitude of the α parameters indicates how many observations have been seen. Initially, an expert can specify an educated guess of the distribution—which we take to be the mode—and a count that represents the his overall confidence in his guess. As a simple illustration, consider the case where the agent asks the user a confirmation question such as "Would you like me to drive to the kitchen?" and we assume (1) the user does wish to go to the kitchen, and (2) there are only two possible responses, "yes" or "no." Suppose that the expert is fairly certain that the dialog manager will hear the word "yes" 90% of the time in this situation. Then he may specify $\alpha_{yes} = 90$ and $\alpha_{no} = 10$. This is equivalent to telling the agent that we have already tried to ask this question 100 times and observed "yes" 90 times. However, suppose the expert does not know very much about the noise in the voice recognition system. Then he may specify $\alpha_{yes} = 9$ and $\alpha_{no} = 1$. These parameters specify the same mean distribution—the expert thinks that the voice recognition error rate is around 10%—but now the agent treats this error rate as an estimate from 10 experiments rather than 100.

The counts analogy extends to the case where the dialog manager is now operating on its own. Every time it observes a certain event from a particular state, it increments the count of that event's corresponding α parameter. The counts grow monotonically with time, which is consistent with our notion that the system should get more confident about its model of the world as it gains experience. Unfortunately, a problem occurs when applying this approach in a real-world setting. Recall that the specific task that the user wants the agent to perform is hidden; the agent only receives observations that provide clues as to what the task may be. However, the α parameters correspond to what observations are likely to occur in a specific state s. To update the observation probability of an observation o, we must therefore know what the state was when o was observed. Fortunately, since we assume that the user has only one underlying request, which does not change until it is satisfied, it is often relatively easy to infer what the user state was throughout the dialog by recognizing what action successfully completed the interaction.[1]

4 Passive Learning for Natural Language Interaction

We have introduced the POMDP dialog model that allows the agent to interact with a user in noisy conditions, and we have described how the agent may express uncertainty in the dialog model itself. In this section, we describe how the agent, given its belief over possible dialog models, may choose actions to interact with a user. We first consider a scenario in which the dialog manager knows that its knowledge of the world is imperfect, but is unable to take any actions to improve its knowledge of the world. While unrealistic, this scenario is a useful baseline for more sophisticated algorithms. We also note that, even without explicitly taking actions to learn about the the dialog model, the agent can still improve its knowledge of how to interact with the user. For example, suppose that the dialog manager initially hears the word "tea," and user responds to the affirmative when the dialog manager asks if the user wishes to go to kitchen. Then the dialog manager can increase the probability that word "tea" is associated with the

[1] We can accomplish this task formally by constructing an HMM to represent the completed dialog. This update approach leads to a modified form of the standard Expectation-Maximization algorithm, and thus the prior will converge to some local optimal dialog model. HMMs and the forward-backward algorithm are well-developed approaches to statistical inference, especially in the speech-recognition community. A description of an HMM and forward-backward algorithm is beyond the scope of this article, but the reader may refer to [Rabiner, 1989] for more details.

kitchen. However, if the user responds to the negative, then the dialog manager can infer that either the word "tea" is not associated with the kitchen, or that "tea" is a commonly the output of a voice recognition error. Likewise, the dialog manager can discover what are the most popular user requests. However, other information cannot only be learned through user interactions. If the dialog manager is only listening for location keywords, it cannot determine the user's frustration due to a poor action or repeated questions. In the traditional reinforcement learning framework employed in this section, the dialog manager gains information about the user's preferences by asking the user to provide it with a reward ("reinforcement") after each interaction.

Recall that the standard dialog POMDP solution maximizes expected reward given the uncertainty over the user's request (the state); we now desire a solution that maximizes the expected reward given uncertainty in both the user's request and the user model. We begin by rewriting equation (4), which describes the value of performing action a given a belief b over the user's request:

$$\begin{aligned} q_{a,i}(s) &= R(s,a) + \gamma \sum_{o \in O} \sum_{s' \in S} T(s'|s,a)\Omega(o|s',a)V_{n-1,i}(s), \\ Q(b,a) &= \max_i \vec{q}_{a,i} \cdot b. \end{aligned}$$

The first equation averages over uncertainty caused by noisy voice recognition, ambiguity in the language or changes in user intent; this uncertainty is inherent in the interaction and cannot be reduced with additional interactions with the user. The second equation is an expectation over the agent's uncertainty in the user's request; this uncertainty can be reduced by asking questions to determine what the user actually wants. If we assume that the agent cannot learn about the true model, then uncertainty in the dialog model falls into the first category, that is, uncertainty that cannot be removed.

If the model parameters themselves are uncertain, then computing the vector $\vec{q_{a,i}}$—which is an average over the inherent uncertainty in the dialog model—now requires an additional expectation over possible dialog models, which we denote by the set M:

$$\begin{aligned} q_{a,i}(s) &= E_M[R(s,a) + \gamma \sum_{o \in O} \sum_{s' \in S} T(s'|s,a)\Omega(o|s',a)V_{n-1,i}(s)] \\ &= E_M[R(s,a)] + \gamma \sum_{o \in O} \sum_{s' \in S} E_M[T(s'|s,a)\Omega(o|s',a)V_{n-1,i}(s)] \\ &= E_M[R(s,a)] + \gamma \sum_{o \in O} \sum_{s' \in S} E_M[T(s'|s,a)]E_M[\Omega(o|s',a)]V_{n-1,i}(s), \end{aligned} \tag{7}$$

where $E_M[R(s,a)]$, $E_M[T(s'|s,a])$ and $E_M[\Omega(o|s',a)]$ are the means of the Dirichlet distributions as given by equation (5). The second line follows from the linearity of expectations, while the third line follows because the transitions T, the likelihood that the users will change their intended request, are independent of the noisy dialog observations Ω. The function $V_{n-1,i}$ is a fixed value from the previous iteration and does not require averaging. Thus, to find the optimal dialog policy given an uncertain user model, it is sufficient to solve the POMDP with the expected values of the model parameters.

Given a history of rewards, observations, and actions, the agent can update its belief over models as described in section 3. The updated belief provides it new expected values $E_M[R(s,a)]$, $E_M[T(s'|s,a)]$ and $E_M[\Omega(o|s',a)]$; it can now choose actions based on new expected model. To update the policy efficiently, we note that if the agent updates each belief over models after each successful dialog, its understanding of the dialog model—and its estimate of the average model—will not change greatly. Thus, we have the dialog manager refine its current value function instead of solving an entirely new policy. The refinement is performed via additional backups on our current value function with the updated user model. Since the backup operation of equation (4) is a contraction ([Gordon, 1995]), the additional backups will always bring the original dialog policy closer to the solution with the new user model. Table 1 summarizes the passive learning algorithm.

A key question is how much refinement is required after each model update—that is, how many backup operations should be performed—after updating the user model. Performing enough backups so that value function fully converges to a new value may require significant time. Especially early in learning process, when the agent has very little

PASSIVE DIALOG MODEL LEARNING

1. Initialize a prior distribution over dialog models and determine the policy of the mean model by solving equation 7.
2. Interact with the user using the policy in (1) until the dialog is complete, asking for numerical feedback at each step.
3. Use the utterances from the user to update the prior over the transition and observation models.
4. Use the numerical reinforcement signal from the user to update the prior over the reward model.
5. Refine the policy based on the new mean model.

Table 1: Passive dialog model learning.

knowledge of the true dialog model, such effort may not be useful because the updated average model may still be far from the true model. We compare three approaches to determine the how much to replan after each successful dialog:

1. Replan to completion. After each completed dialog, we perform enough backups for the new value function to converges. (In practice, we found that convergence sometimes required many backups even after only small changes in the parameters. To complete our simulations, we capped the number of backups per update step to 50.) This should lead to the best expected performance given the uncertainty in the user model. However, computing a POMDP policy to convergence can be also a slow process, leading to latencies in dialog manager responses.
2. Replan k times. Perform k backups, regardless of how much the updated model estimate differs from the previous estimate. This approach may prevent latencies in the dialog manager performance, but does not give the dialog manager time to compute better plans once it has confidence in the model.
3. Replan proportionally to model confidence. The sum of the variances on each parameter is a rough measure of the overall uncertainty in the dialog model. If an update reduces the overall variance by d_v, the agent backups $\lfloor k * d_v \rfloor$ times. The intuition is that the dialog manager should expend the most computational effort when it has the most reliable knowledge. For simulation purposes, we also capped the number of backups per update at 50.

4.1 Results

We used simulation to test the learning approach against a known ground truth optimal policy. In our test scenario, the dialog manager initially believed that its observations were more accurate than they actually were and that the consequences of going to the wrong place were not particularly severe (see table 2 for the model size and table 3 for initial guesses and true values). We attributed our estimates to two pseudo-observations per event per state-action pair to indicate a relatively low confidence in the initial parameter estimates. Finally, to account for the fact that the agent's estimate of the user model would be changing with time, we seeded the belief set used to solve for the dialog policy with common confusions (such as thinking two or three states were equally likely) to improve the accuracy of our approximate POMDP solver.

Parameter	Simulation
States	7
Actions	12
Observations	8

Table 2: Model Parameters for Simulation Tests.

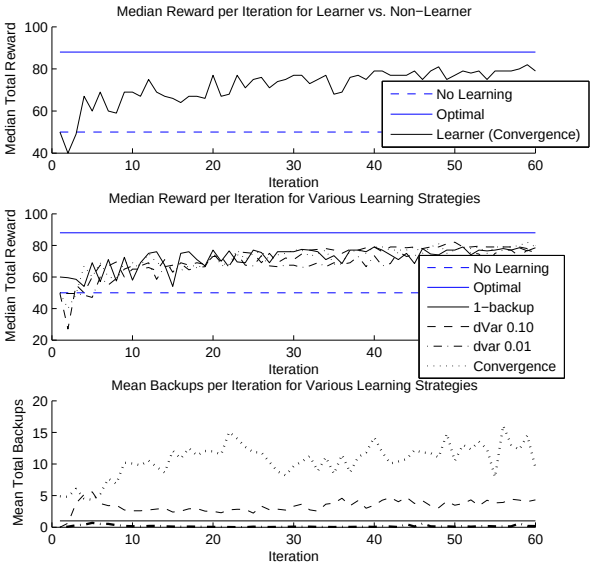


Figure 5: Performance and computation graphs. The learner outperforms the non-learner (top graph), and all of the replanning approaches have roughly the same increase in performance (middle graph), but replanning proportionally to the confidence of the model achieves that performance much less computation (and therefore faster response) than replanning to convergence (bottom graph).

As seen in Figure 5, all approaches achieved similar results, but replanning proportionally to the change in model confidence achieved that result with almost as little computation as replanning once per update (see Table 4; note that updates occur *after* a dialog is completed, while the robot is performing the requested task, and thus do not affect the flow of the conversation.) The model confidence approach focused replanning near the beginning of the trial when parameters first become certain. Each point represents 100 trials (except 'convergence'—in the interests of time, we completed only 37 trials).[2] We note that the k_{d_v}=0.01 approach completed fewer total backups than backing up once per iteration, but the larger number of backups near the beginning of the learning process helped stabilize the solution.

	Initial	True
P(self-transition)	.95	.95
P(correct obs if ask-which)	0.7	0.5
P(correct obs if confirm)	0.9	0.7
R(complete task)	100	100
R(ask-which)	-1	-10
R(correct confirm)	-1	-1
R(incorrect confirm)	-10	-2
R(incorrect destination)	-50	-500

Table 3: Initial and True Parameter Values for Simulation Tests. The initial model parameters assume a higher speech recognition higher accuracy and a user unforgiving of confirmation actions. This is an example of a very demanding user relative to the initial model specification.

[2]These tests used an oracle that provided the dialog manager with the user's true request to estimate the user state for model updating after the dialog was completed. We present these results because they show a clearer picture of the algorithm's performance. Tests using HMM-based state estimation for model updating performed slightly but not significantly worse.

Approach	Time (sec)
Convergence	135.80
1-backup	13.91
0.10-var	42.55
0.01-var	18.99

Table 4: Mean update times for each of the four approaches. Note that updates take place only after a dialog has been completed; when the dialog policy does not require any updates, the dialog manager's average response time is 0.019 seconds.

4.2 Discussion

The benefit of the simple approach presented in this section is that it requires very little computational effort; the backups are fast enough that the dialog manager can learn and interact with the user in real time. However, it suffers from several drawbacks. First, the user is forced to provide a reinforcement after every interaction within a dialog. The agent must also make mistakes to learn about a negative reinforcement. Furthermore, by only considering a point estimate of the dialog model when choosing actions, the agent is not fully aware in the limitations of its understanding of the dialog model. In the worst case scenario, this limitation can lead it to choose actions that will never allow it to learn the true dialog model. For example, consider a scenario in which there are only two actions: *ask* and *confirm*. Suppose that under some prior belief M over reward parameters, we have the following relationship between the true rewards and their expected values under the prior:

$$R_{ask} > R_{conf} = E_M[R_{conf}] > E_M[R_{ask}], \tag{8}$$

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, $E_M[R_{conf}]$ and $E_M[R_{ask}]$, it will never try the action *ask*. Thus, the dialog manager will never discover the mistake in its understanding. 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 remedy these issues in the next section.

5 Active Learning for Natural Language Interaction

In this section, we introduce a more sophisticated algorithm that allows the dialog manager to use the fact that it is unsure about the true dialog model more intelligently. By explicitly taking its uncertainty into account, the dialog manager behaves more robustly; even if the designer estimates the parameters incorrectly, the dialog manager will still be able to correct itself over time. Additionally, the meta-queries allow the learner to predict and avoid potential failures, reducing the variance of the policy performance during convergence to the correct model and corresponding optimal policy. However, we note that the full state space is continuous and high dimensional; to generate policies tractably, we assume that the parameters of the user model do not change over time, that is, the model m itself is stationary. Our agent also assumes that while it may use a series of actions to satisfy a user's request, only its next action can potentially improve its knowledge of the dialog model. This approximation simplifies planning under model uncertainty because the agent only needs to think one step ahead. (Of course, the next time it needs to select an action, it will also think that it has one more chance to learn about the true dialog model.)

In figure 6, we show typical simulation results for the model-uncertainty POMDP for a variety of priors. In each case, the overall performance is about the same, showing (as we expect) that the parameter POMDP approach is robust to the initialization. What is most interesting, however, is that if we start out with a conservative prior, that is, a prior that puts most of its weight on a "tough" set of rewards, the initial policy still performs relatively close to the optimal policy during the learning process. In this case, the system is robust because it is very cautious of doing the wrong thing, and the actions that allow it to actively learn about the model are relatively low-cost.

Choosing actions based on model uncertainty remedies one of the drawbacks of the previous approach, where the dialog manager might not adapt successfully if the expert estimated the system poorly. Recall that the passive learning

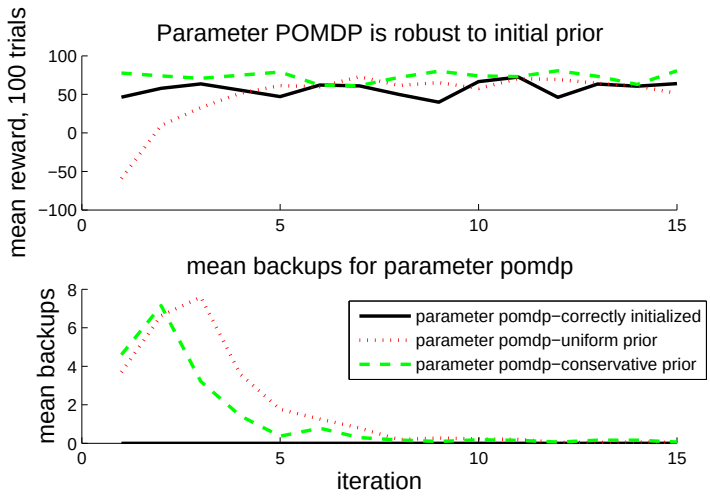


Figure 6: The three different priors—a uniform prior, a correct prior, and a conservative prior—converge to the same steady-state performance, but choosing a conservative prior allows us to do as well as if we had known the user's preference state initially. All solutions were backed up to convergence.

scheme also required numerical feedback after each interaction (Table 1, Step 2). Our new approach uses an alternative form of feedback: the meta-query. The meta-query asks the user to help the system by suggesting what action it should take next. For example, suppose that the dialog manager on our wheelchair is fairly certain that the user wishes to go to the printer. Then it might ask:

"I think you definitely want me to go to the printer. Should I go to the printer?"

On the contrary, if the wheelchair thinks that the user may want to go to the printer but is not very certain, it might ask:

"I think you may want me to go to the printer. Should I go to the printer?"

The choice of adverb gives the user an intuitive sense of the dialog manager's uncertainty. Thus, the user can advise the robotic wheelchair based on their internal preferences. For example, if the user is risk averse, they may respond "yes" to the first question but "no" to the second question. If the user answers a question to the negative, the wheelchair might follow up with further questions such as,

"In that case, I think I should confirm that you want to go to printer first. Is that correct?"

until it receives an affirmative response.[3]

We note that the meta-queries provide more ambiguous feedback regarding the dialog model than the explicit reinforcement signal in the previous approach. For example, in the wheelchair scenario, the dialog manager may need help because it does not know if the user's previous utterance corresponded to the printer location (a confusion in the observation model) or because it is simply not certain enough that it heard the user correctly (a confusion in the reward model). Fortunately, the source of the confusion does not matter greatly from the user's perspective; regardless of why the dialog manager is confused, the user provides information about the type of desired behavior, and the system determines what interaction models are consistent with the user's request. While meta-queries place a greater computational burden on the system, we believe they can provide a more natural way for the human to instruct the robot. The meta-query approach is also active in that the dialog manager chooses when it needs help instead of asking the user for feedback after each interaction.

[3] In our tests, we used an abbreviated form of the meta-queries for simulation speed.

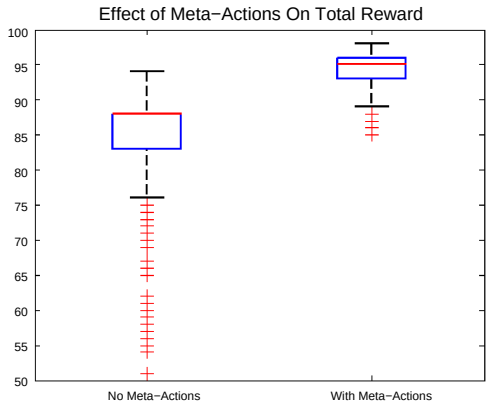


Figure 7: Boxplot of dialog manager performance with a discrete set of four possible models. In this case, the user is very intolerant to errors, but the learner does not initially know this. Although the medians of the two policies are not so different, the active learner (right) makes fewer mistakes than the passive learner (left), leading to overall much less user-annoying behavior.

5.1 Solving the Model-Uncertainty POMDP directly

In general, the transition, observation, and reward parameters are continuous-valued, with an infinite number of possible models, leading to computational intractability for conventional solvers. In special situations, however, uncertainty in the dialog model may be expressed as a small, discrete set of possible models rather than a continuous distribution, making the model-uncertainty POMDP much easier to solve.

For example, consider a scenario where we already have accurate transition and observation models (say, from some prior work with the voice recognition system), but a new user's preference model is unknown. The user's exact preference model may not matter as long as the dialog manager has roughly the appropriate pattern of behavior. In an extreme case, we may decide to only characterize the user's frustration with an incorrect movement as low or $high$, and similarly characterize the user's frustration with an incorrect confirmation as low or $high$. The user model can be described by two variables $\{WrongMovePenalty, WrongQuestionPenalty\}$. The two variables $WrongMovePenalty$ and $WrongQuestionPenalty$ can each take either values of $high$ or low, so that the model for a particular user might be $m = \{WrongMovePenalty = high, WrongQuestionPenalty = low\}$. This particular user would be conservative, with a preference to be asked questions repeatedly rather than risk being taken to the wrong location. With only four possible dialog models, the state space is still discrete and small, and we can now solve the model-uncertainty POMDP using a standard algorithm [Pineau et al., 2003].

We show simulated results with this very simple scenario of only four possible preference models in Figure 7. The figure compares the performance of the policy without using meta-queries (left column) to the performance of the policy *with* meta-queries. As expected, the system which has the ability to ask meta-queries can use the questions to gain information about the user's internal preference model. It is able to discern that the user is very sensitive about incorrect movements, and therefore it asks more confirmation questions before taking an action. The reduction in large negative mistakes is substantial—which is particularly important in dialog management, where users will likely find a system that regularly makes mistakes annoying. The passive learner does converge to close to the optimal policy—the difference in means between the two learners is not significant. What *is* significant is the variance of the passive learner before convergence.

Unfortunately, our approach of representing the user model as discrete values (such as $WrongQuestionPenalty$ = low) does not scale well. Experimentally we found that even a modest increase in the number of possible user models from 4 to 48 meant that the model-uncertainty POMDP could no longer be solved using standard solution techniques. While it may be possible to group the possible combinations of user preferences into a few representative models (since the effects of small changes to the preference model may not be apparent to the user), discretizing other parts of the user model such as vocabulary choices quickly produces an exponential number of states. For example, for each keyword the user might utter, we have to consider how likely it is to be heard in each goal location. We therefore

DIALOG MODEL LEARNING WITH BAYES RISK

1. Sample POMDPs from a prior distribution over dialog models.
2. Interact with the user:
   (a) Use the dialog model samples to compute the action with (approximately) minimum Bayes risk.
   (b) If the risk is larger than a given ϵ, perform a meta-query.
   (c) Update each dialog model sample's belief based on the observation received from the user.
3. Periodically resample from an updated prior over dialog models and return to step 2.

Table 5: Dialog model learning approach using Bayes risk and meta-queries.

turn to approximation techniques which will allow us to represent a larger class of models with continuous parameters.

5.2 Approximately Solving for a Dialog Policy

To simplify the problem of finding the dialog policy, we separate the problem into steps. In the first step, the dialog manager establishes a representative set of dialog models it believes are consistent with the user interactions it has experienced. Next, it solves for the optimal policy in each of these models. Thus, after any user utterance, it has a collection of actions—votes from each of the candidate models—to take next. These votes can be used to compute the risk of taking each action; if the safest action is too risky, the dialog manager will ask a meta-query. By providing information about the best action to take, the meta-query allows the dialog manager to prune away inconsistent models from its representative set. Table 5 outlines this approach.

Minimum Risk Action Selection

Given a dialog model m, the dialog manager can compute the loss of taking a particular action a compared to the best action a_{opt}. The loss is simply the difference $Q(b, a) - Q(b, a_{opt})$, where the Q-function denotes the value of taking action a in belief b (equation 2). Given a belief over models, the Bayes risk of an action is defined to be the loss the dialog manager expects to incur on average by choosing action a:

$$BR(a) = \int_M (Q_m(b_m, a) - Q_m(b_m, a_{opt,m}))p_M(m), \tag{9}$$

where M is the space of all possible dialog models, b_m is the current belief over possible user requests according to dialog model m, and $a_{opt,m}$ is the optimal action for the current belief b_m according to dialog model m. Let $a^* = \arg\max_{a \in A} BR(a)$ be the action with the least risk. If the risk $BR(a^*)$ is less than fixed cost of a meta-query, that is, if the least expected loss is still more than a certain threshold, the dialog manager will ask the meta-query. Intuitively, equation 9 computes the potential loss due to taking action a instead of the optimal action a_{opt} according to each possible dialog model m and weights that loss by the probability of model m. When the agent is sufficiently sure about the model, the risk will be low. When it is unsure about the model, the risk may be high but the series of meta-queries will lead it to choose the correct action.

We cannot compute the Bayes risk in close form, so we turn to numerical techniques to find an approximation. The agent's belief over user states and dialog models gives it the probability of each model $p(m)$; if it draws sample models from this distribution, it will draw many samples in regions where $p(m)$ is high and few samples from where $p(m)$ is low. The more samples it draws, the better the densities of the samples will represent the distribution from which they were drawn. Thus, it can approximate equation (9) with the sum:

$$BR(a) = \sum_i (Q_i(b_i, a) - Q_i(b_i, a_{opt,i}))w_i, \tag{10}$$

where Q_i provides the value of taking actions from belief according to dialog sample i. If model samples are drawn from $p(m)$, the weight w_i of each model is simply $\frac{1}{N}$, where N is the number of samples.

The dialog manager's belief over models will change as it interacts with the user, so the initial set of samples may not be representative of reasonable dialog models as the agent's knowledge improves. However, for computational reasons—since the dialog manager must solve every dialog model as it is sampled—it may be undesirable to resample models every time new information is received. In this case, the original sample set of models can be re-used by changing the weight of each model and representing the distribution $p_{new}(m)$ as a set of weighted samples. At each time step, the weight of each model should be adjusted to be proportional to the ratio of the previous likelihood of the sample and its likelihood given the new information. While it is possible to provide formal bounds on the number of samples needed to approximate the Bayes risk to a specified degree of accuracy, these bounds are loose and in practice we found that fifteen samples sufficed for our dialog management application.

POMDP Resampling

For practical applications, we use only a small set of samples to represent the agent's belief over dialog models. Even with reweighting, after some time, the initial sample set may no longer accurately represent the true distribution over models; it may occur that after several human-robot interactions, the system may discover that none of the dialog models it initially thought were representative are in fact reasonable. The need for resampling may also arise if one model's weight becomes very close to one and the remaining models have weights close to zero, causing the risk to appear small. While acceptable if the risk is truly small, we do not want the dialog manager to become over-confident due to a poor set of candidate models.

We have two sources of information to update our sample set of dialog models. One source is the history of the most recent dialog, which consists of action-observation pairs $h = \{a, o\}$. Another source is the set of meta-queries $Q = \{(q, r, h')\}$, where h' is the history of the dialog from the initial belief to the query, q is the query, and r is the user's response to the query. Given h and Q, the posterior probability $p_{M|h,Q}$ over models is:

$$p_{M|h,Q}(m|h, Q) = \eta p(Q|m)p(h|m)p_M(m), \tag{11}$$

where η is a normalizing constant. Note that if p_M is a Dirichlet distribution, then $\eta' p(h|m)p_M(m)$ is also a Dirichlet distribution since the likelihood $p(h|m)$ is product of multinomials. Recall that updating the Dirichlet distribution corresponded to adding counts. We can use the approach in section 3 to approximate how may counts to add to each Dirichlet parameter without knowing the true state history.

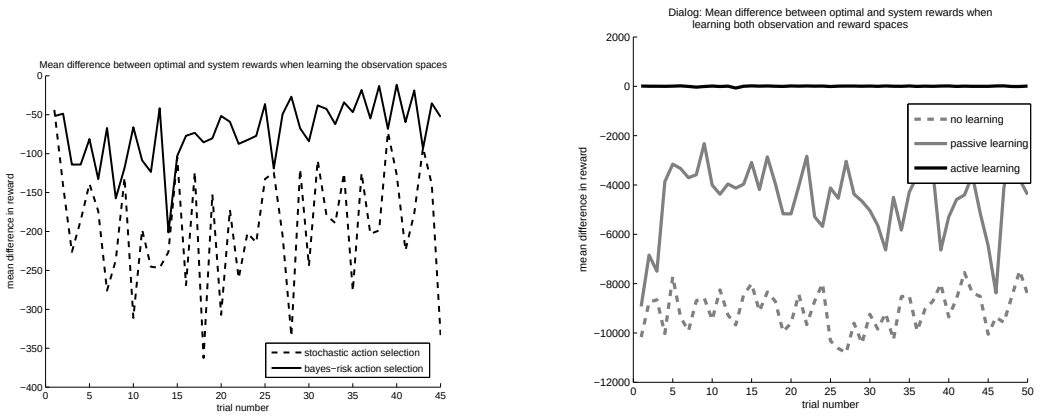
Incorporating meta-query information requires a different approach, since each specific meta-query response provides information about how the agent should behave, not the dialog model parameters. We do not have a closed-form expression for $p_{M|h,Q}$, so we must use sampling to draw dialog model samples that are consistent with all of the meta-queries that have been asked so far. Each query in the set Q provides a constraint on the feasible set of dialog models M. Dialog models are feasible if their policy is consistent with the responses in the meta-query. Computing this feasible set directly is intractable, however, given the set Q, we can check if a sampled dialog POMDP is consistent with the previous meta-query responses stored in Q. Thus, to sample POMDPs, we first generate models from the updated Dirichlet priors. Next, we solve for the optimal policy of each model and check if each dialog model's policy is consistent with the previous meta-query responses stored in Q.

5.3 Results

We now present results for this approach from simulated trials which allow us to test against a ground truth model, and also a small set of user studies.

Simulation

Figure 8 shows results from a simulated dialog manager for our robot wheelchair control application. Both figures show the difference between the learning policies and the ground truth optimal policy over time. In figure 8(a), we see the usefulness of the Bayes-risk approach (compared to stochastic actions selection based on the weights of the sampled models) when the reward model is known. In this case, the Bayes risk action selection allows us to choose



(a) Observation Model Learning

(b) Observation and Reward Model Learning

Figure 8: Dialog manager simulation results. (a) Results from learning only the observation model. (b) Benefits of active learning when learning both the observation and reward model.

First half of interactions	.79
Second half of interactions	.48

Table 6: Proportion of dialogs with meta-queries by location. The decrease in the number of meta-queries from the second to the third time the location was asked for is statistically significant at the $p = 0.05$ level.

non-risky actions. The usefulness of our approach is even more dramatic when the reward prior is uninformative (figure 8b). In this case, the dialog manager can improve somewhat by passively updating its priors based on what it has heard (solid gray line). However, simply listening cannot provide the dialog manager with information about the user's preferences. Moreover, since the active learning system asks the user for help whenever it is confused, this system does not suffer from poor performance while the model estimate converges. The meta-queries allow the active learner to learn while avoiding mistakes, thus maintaining a high level of performance.

Robotic Wheelchair

We also validated our approach on a dialog manager for a robotic wheelchair with a simple user study. The underlying POMDP, with 10 states, 38 observations, and 21 actions, used keywords from a voice recognition system output as observations. Initially, each state had one strongly mapped observation (such as 'printer' for the printer location). The remaining observations received uniform initial priors. Four users conducted 12-15 interactions (20-25 minutes) with the system.

By asking meta-queries, the dialog manager was able to successfully complete all 57 interactions without ever trying to drive the user to an incorrect location. Table 6 shows that the proportion of dialogs with meta-queries decreased significantly from the user's initial interactions to his or her final interactions with the system. The proportions are relatively high because the users, being new with the system, asked for a variety of locations or experimented with different vocabulary. In Table 7, we see that the system often asked a meta-query the first or second time a user asked to go to a particular location, but there was a significant drop in the number of meta-queries after the third time the user requested to go to the same place. We note that our active-learning approach allowed our dialog manager to automatically determine the length of its training phase. Our trials were too small to draw strong conclusions from these results (especially since we were unable to account for the effect of the novelty factor in the interactions), but our quantitative analysis does show statistically significant decreases in the number of meta-queries asked as the dialog manager adapted to each user.

Next we present some sample dialogs to illustrate our system's learning. Table 8 contains a sample dialog where

How often place was requested	First time	Second time	Third Time
Total number of requests	29	15	8
Number of requests with meta-queries	22	11	2
Proportion of requests with meta-queries	.76	.73	.25

Table 7: Proportion of dialogs with meta-queries. The decrease in the proportion of queries is significant at the $p = 0.05$ level.

the system used a meta-query to establish the meaning of a new word. In the initial dialog, the word 'forecast' is new to the system, that is, it has no prior on what that word may mean. The system determines that the Bayes risk is great enough to warrant asking for a policy clarification. As a result, future dialogs are biased toward choosing POMDP samples that satisfy the initial policy query request, which allows later dialogs to avoid the same kind of training.

Early Conversation
ROBOT: How can I help you?
USER: Give me the forecast.
ROBOT: I'm confused. What action should I take now?
The system pops up a meta-query window, and user indicates action "show weather" is correct.
ROBOT: Showers.
Later Conversation (with the same user)
ROBOT: How can I help you?
USER: What's the forecast for today?
ROBOT: Do you want the weather?
The system has partially the word "forecast" and confirms the correct state.
USER: Yup.
ROBOT: Showers.

Table 8: Learning a new word: The system learns the meaning of a new word through a meta-query.

In Table 9, the system adapts to a situation where two words, 'coffee' and 'copy' are often mistaken for each other in the voice recognition system. We stress that at the beginning of the dialog, the system was initialized to believe that the word 'coffee' was most likely if the user wished to go to the coffee machine, while the word 'copy' was most likely if the user wished to go to the copy machine. The other key words—"tea" and "kitchen"—-in this segment began with uniform priors. During the first conversation, the dialog manager is confused by the many new words. Even after asking a meta-query, it is confused by hearing the word "copy" since the user seems to want to go to the kitchen. After several interactions, the dialog manager goes to the correct location. We note that even in this tricky scenario, the use of the meta-queries helped the dialog manager focus on to the kitchen when the only word it recognized—"copy"—seemed to indicate that the user wished to go to another location. In the later dialog, we see the effect of the learning: upon hearing the words "kitchen" and "copy," the dialog manager correctly infers that the user wishes to drive to the kitchen.

5.4 Discussion

The approach presented in this section builds upon prior work in POMDP learning has also considered incorporating the parameters of the POMDP as additional hidden states into a larger model-uncertainty POMDP. Closest to our work are the Bayesian approaches in the Medusa [Jaulmes et al., 2005] and Beetle [Poupart et al., 2006] algorithms. Applying observations from Dearden et. al. ([Dearden et al., 1999]), the Medusa algorithm uses a candidate set of samples to represent a distribution over models. As in our approach, the agent updates the weight it places on each

Early Conversation
ROBOT: How can I help you?
USER: I want some tea from the kitchen.
ROBOT: How can I help you?
USER: The tea is in the kitchen near the coffee machine.
The system heard "copy" instead of "coffee" and is confused about the new word "kitchen."
ROBOT: How can I help you?
USER: Go to the kitchen.
ROBOT: I'm confused. What action should I take now?
The user indicates "Confirm the coffee machine area" is the correct action.
ROBOT: Do you want to go to the coffee machine area?
USER: Yup.
… *The system asks a few more (non-meta) questions before re-confirming the coffee machine state.*
ROBOT: Do you want to go to the coffee machine area?
USER: Yup.
ROBOT: Going to the coffee machine area.
Later Conversation (with same user)
ROBOT: How can I help you?
USER: Take me to the kitchen.
The system hears a noisy combination of "nick" and "kitchen" (mostly "kitchen").
ROBOT: How can I help you?
USER: I want some coffee.
The system hears "copy" instead of "coffee," but with its learned observation model, it confirms correctly.
ROBOT: Do you want to go to the coffee machine area?
USER: Yup.
ROBOT: Going to the coffee machine area.

Table 9: Learning about observation noise: the system learns new words for a noisy pair of states.

sample as new information arrives and changes its knowledge of the world. However, the Medusa algorithm chooses actions stochastically based on the weights of the samples. By choosing actions to minimize Bayes risk, our dialog manager can act robustly even with high uncertainty in the dialog model.

The Beetle algorithm takes a decision-theoretic approach to solving MDPs with uncertain parameters. Much like our model-uncertainty POMDP, Poupart et. al. treat the unknown MDP parameters as hidden state in a larger POMDP and derive an analytic solution (based on [Porta et al., 2006]) for a policy that will trade optimally between learning the MDP and maximizing reward. Unfortunately, these techniques do not extend tractably to the model-uncertainty POMDP, which is continuous in both the POMDP parameters (like the MDP) and the belief state (unlike the MDP). While approximate, our Bayes risk action selection criterion allows the dialog manager to function in this complex space of dialog models.

We also note that there are also non-Bayesian approaches for acting in uncertain environments. Some, targeted at industrial applications ([Nilim and Ghaoui, 2004],[Xu and Mannor, 2007]), aim to make the agent robust to (usually small) variations in its environment. While they provide guarantees for worst-case behavior, they do not describe how the agent should act if it has the ability to adapt to some of these variations through learning. Finally, a class of model-free approaches attempt to bypass the issue of model uncertainty by directly learning a policy ([Littman et al., 1995], [Even-Dar et al., 2005], citepsr). The model-free approaches require a typically large amount of data to produce reasonable behaviors, and, without a prior notion of possible pitfalls, they are likely to make many mistakes during the learning process. Our Bayesian approach avoids this problem by allowing the system designer to provide an indication of where potential pitfalls may be; by also letting the agent query the human for help, we ensure that the agent's policy is robust from the start.

While our active learning approach addresses many of the limitations in these previous works and resolves the drawbacks of the passive learning approach in section 4, there are two key areas for improvement and future work. First, the most computationally expensive part of this approach is finding dialog models consistent with the meta-query information that the user has provided; better sampling strategies will be required to allow the algorithm to scale to more complex dialogs. Second, we still are seeking effective ways for the user to aid the dialog management system when it is confused; we believe that meta-queries are only the first step toward more natural (and accurate) human-robot feedback. Resolutions to these issues will lead to even more robust and effective human-robot interactions.

6 Conclusion

Many approaches have been developed for human-robot interaction. In our work, we focused on robustly learning dialog models for effective human-robot interaction (unlike, for example, trying to learn an explicit vocabulary as in [Lopes and Teixeira, 2000, Lopes and Chauhan, 2007]. Many interaction systems provide the dialog manager with a set of rules to follow given particular outputs from a voice recognition system. For example, the Mercury system [Seneff and Polifroni, 2000] builds a network in which a flight reservation system keeps track of what information has already been provided and for what the user needs to be prompted. These rules can also help the agent monitor the quality of the interaction [Paek and Horvitz, 2004] and adapt to new users [Litman and Pan, 2002].

The drawback to rule-based systems is that they often have difficulty managing the many uncertainties that stem from noisy speech recognition or linguistic ambiguities. The ability to manage the trade-off between gathering additional information and servicing a user's request have made (PO)MDP planners particularly useful in dialog management; applications include a nursebot robot, designed to interact with the elderly in nursing homes[Roy et al., 2000], a vision-based system that aids Alzheimer's patients with basic tasks such as hand-washing [Hoey et al., 2005], an automated telephone operator ([Williams and Young, 2005]), and a tourist information kiosk ([Litman et al., 2000]).

POMDP-based dialog systems are faced with two difficulties. The first is that simply solving a POMDP can be quite computationally expensive. Fortunately, much research has focused on finding tractable dialog manager policies. For example, [Williams et al., 2005] derive methods to incorporate an explicit confidence output from the voice recognition system as an additional measurement, allowing the agent to reason about the possibility of speech recognition errors. In situations where only certain actions are relevant to certain states, the POMDP can be factored into hierarchies that reduce the overall amount of computation required [Pineau et al., 2001]. Certain dialogs also contain symmetries that lend themselves to efficient approximations [Williams and Young, 2005].

Unfortunately, the POMDP solution approaches typically assume a reasonably accurate user model. In domains where large amounts of data are available—for example, automated telephone operators—the user model may be relatively easy to obtain. For human-robot interaction, however, collecting sufficient user data to learn a statistically accurate model is usually expensive: trials take a lot of time from human volunteers. We developed an approach that allows the agent to leverage the benefits of POMDP-based dialog managers while addressing the difficulty of specifying the POMDP model parameters. An expert need only specify a prior belief describing his knowledge—and uncertainty—regarding the dialog model, and the agent refines this knowledge through interactions with the user.

To make the learning process robust, we introduced a risk-averse action selection criterion that allowed our dialog manager to behave robustly even when its knowledge of the true dialog model was uncertain. Instead of asking for feedback after every interaction, as many traditional reinforcement learning approaches require, our dialog manager uses an active learning scheme to only ask for help when it feels it does not have enough information to make a safe decision. Framed as meta-queries, or questions about actions the dialog manager is thinking of taking, our active learning scheme is also more intuitive than asking for numerical feedback. We demonstrated our approach both in simulation and on a real a dialog manager for a robotic wheelchair.

References

[Dearden et al., 1999] Dearden, R., Friedman, N., and Andre, D. (1999). Model based bayesian exploration. pages 150–159.

[Even-Dar et al., 2005] Even-Dar, E., Kakade, S. M., and Mansour, Y. (2005). Reinforcement learning in pomdps without resets. In *IJCAI*, pages 690–695.

[Fern et al., 2007] Fern, A., Natarajan, S., Judah, K., and Tedepalli, P. (2007). A decision-theoretic model of assistance. *IJCAI*.

[Gordon, 1995] 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 et al., 2005] Hoey, J., Poupart, P., Boutilier, C., and Mihailidis, A. (2005). Pomdp models for assistive technology. *IATSL Technical Report*.

[Jaulmes et al., 2005] Jaulmes, R., Pineau, J., and Precup, D. (2005). Learning in non-stationary partially observable markov decision processes. ECML Workshop.

[Litman et al., 2000] 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*, Seattle.

[Litman and Pan, 2002] Litman, D. J. and Pan, S. (2002). Designing and evaluating an adaptive spoken dialogue system. *User Modeling and User-Adapted Interaction*, 12(2-3):111–137.

[Littman et al., 1995] Littman, M. L., Cassandra, A. R., and Kaelbling, L. P. (1995). Learning policies for partially observable environments: Scaling up. In Prieditis, A. and Russell, S., editors, *Proceedings of the Twelfth International Conference on Machine Learning*, pages 362–370, San Francisco, CA, USA. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.

[Lopes and Chauhan, 2007] Lopes, L. S. and Chauhan, A. (2007). How many words can my robot learn? an approach and experiments with one-class learning. *Interaction Studies*, 8(1):53–81.

[Lopes and Teixeira, 2000] Lopes, L. S. and Teixeira, A. (2000). Human-robot interaction through spoken language dialogue. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems(IROS)*, pages 528–534.

[Nilim and Ghaoui, 2004] Nilim, A. and Ghaoui, L. (2004). Robustness in markov decision problems with uncertain transition matrices. *NIPS*.

[Paek and Horvitz, 2004] Paek, T. and Horvitz, E. (2004). Optimizing automated call routing by integrating spoken dialog models with queuing models. In *HLT-NAACL*, pages 41–48.

[Pineau et al., 2003] Pineau, J., Gordon, G., and Thrun, S. (2003). Point-based value iteration: An anytime algorithm for pomdps. *IJCAI*.

[Pineau et al., 2001] 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)*.

[Porta et al., 2006] Porta, J. M., Vlassis, N., Spaan, M., and Poupart, P. (2006). An point-based value iteration for continuous pomdp.

[Poupart et al., 2006] Poupart, P., Vlassis, N., Hoey, J., and Regan, K. (2006). An analytic solution to discrete bayesian reinforcement learning. In *ICML*, pages 697–704, New York, NY, USA. ACM Press.

[Rabiner, 1989] Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

[Reeves and Nass, 1996] Reeves, B. and Nass, C. (1996). *The Media Equation: How People Treat Computers, Television and New Media Like Real People and Places*. Cambridge Press.

[Roy et al., 2000] Roy, N., Pineau, J., and Thrun, S. (2000). Spoken dialogue management using probabilistic reasoning. In *Proceedings of the 38th Annual Meeting of the ACL*, Hong Kong.

[Seneff and Polifroni, 2000] Seneff, S. and Polifroni, J. (2000). Dialogue management in the mercury flight reservation system. In *ANLP/NAACL 2000 Workshop on Conversational systems*, pages 11–16, Morristown, NJ, USA. Association for Computational Linguistics.

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

[Williams et al., 2005] 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 and Mannor, 2007] Xu, H. and Mannor, S. (2007). The robustness-performance tradeoff in markov decision processes. In Schölkopf, B., Platt, J., and Hoffman, T., editors, *Advances in Neural Information Processing Systems 19*. MIT Press, Cambridge, MA.