Efficient Model Learning for Dialog Management

Finale Doshi
CSAIL MIT
32 Vassar Street
Cambridge, MA 02139
finale@mit.edu

Nicholas Roy
CSAIL MIT
32 Vassar Street
Cambridge, MA 02139
nickroy@mit.edu

ABSTRACT

Intelligent planning algorithms such as the Partially Observable Markov Decision Process (POMDP) have succeeded in dialog management applications [10, 11, 12] because they are robust to the inherent uncertainty of human interaction. Like all dialog planning systems, however, POMDPs require an accurate model of the user (e.g., what the user might say or want). POMDPs are generally specified using a large probabilistic model with many parameters. These parameters are difficult to specify from domain knowledge, and gathering enough data to estimate the parameters accurately *a priori* is expensive.

In this paper, we take a Bayesian approach to learning the user model simultaneously with dialog manager policy. At the heart of our approach is an efficient incremental update algorithm that allows the dialog manager to replan just long enough to improve the current dialog policy given data from recent interactions. The update process has a relatively small computational cost, preventing long delays in the interaction. We are able to demonstrate a robust dialog manager that learns from interaction data, out-performing a hand-coded model in simulation and in a robotic wheelchair application.

Categories and Subject Descriptors: I.2.6 Learning: Parameter learning **General Terms:** Algorithms, Human Factors **Keywords:** Human-robot interaction, decision-making under uncertainty, model learning

1. INTRODUCTION

Spoken dialog managers allow for natural human-robot interaction, especially for public use in everyday environments. However, several issues make it difficult to decipher the user's intent. First, voice recognition technology produces noisy outputs, a problem exacerbated by ambient noise. Voice recognition also varies widely in quality among different users. Finally, even with perfect voice recognition, we still must determine the user's intent in the face of linguistic ambiguity. Partially Observable Markov Decision Processes (POMDPs) are a planning technique that have led to more robust dialog management because they can effectively handle uncertainty and ambiguity in the dialog ([10, 11, 12]). The typical POMDP model operates by assuming that the user intent, or "state" is hidden from the robot. Instead, the user intent is inferred from a stream of noisy or ambiguous observations, such as the utterances from a speech recognition system, visual gestures, etc..

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

The POMDP dialog manager compensates for state uncertainty with explicit information gathering actions, and it also balances the costs of taking an incorrect action against the cost of asking further clarification questions. The ability to manage the information gathering trade-off have made POMDP dialog managers particularly effective in health care domains [10]. The cost of dialog errors on health-care robots is relatively high, because the dialog often involves large motions of assistive devices. For example, with the robotic wheelchair (shown in Figure 1), the cost of misunderstanding a desired destination is much more than the mild annoyance of the user needing to repeat themselves; the wheelchair may already be taking the user to the wrong location.

Much POMDP dialog-management research has focused on developing factored models and other specialized structures to improve performance and algorithmic complexity ([12], [7], [11]). These 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.

Specifying the model from expert knowledge is also difficult. For example, how does one specify the probability that the user appeared to ask for the time when they actually wanted the weather? Even linking the occurrences of keywords to states poses a tricky problem: types of speech recognition errors are difficult to predict (see Section 4 for examples), and modeling mistakes can severely impact the quality of the dialog.

Some learning algorithms have designed systems to train parameters from scratch while maintaining reasonable dialogs ([4]), but we wish to take advantage of any domain knowledge we may have in creating initial parameter estimates. In this paper, we take a Bayesian approach to parameter uncertainty: we maintain distributions—not point estimates—of the model parameters. Unlike [4], our system also learns online (instead of from a training set) and uses a POMDP approach that truly handles uncertainty both in the dialog and in the model. We will demonstrate a POMDP-based dialog manager that initially performs with some basic competence and steadily improves without paying a large computational penalties. The planner smoothly converges to the best plan given the data while preserving good responsiveness in the overall system.

2. PROBLEM FORMULATION

2.1 POMDP overview

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 intent, in this case the places where the user would like a robotic wheelchair to go. Actions include queries to the user and physical movement. Unlike many conventional dialog managers, however, the POMDP model assumes that the current state (the user's intent) is hidden and must be inferred from a set of probabilistic observations. The likelihood of any observation is conditioned on the current (hidden) state. In the dialog management setting, the observations correspond to the utterances that the dialog manager hears (more details in Section 4.2), and the observation is emitted stochastically according to word error probabilities and likelihoods of user error. We will assume that the state, action and observation sets are all discrete, finite, and relatively small; thus, learning the parameters and updating the POMDP is tractable for real-time dialogs (although work has extended dialog managers to large state spaces [11] and continuous observation spaces [12]).

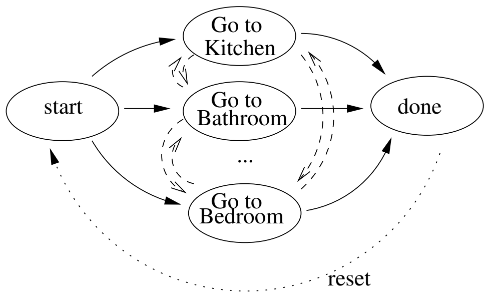

Figure 2: A toy example of a dialog POMDP. Solid lines represent more likely transitions; we assume that user is unlikely to change their intent before their original request is fulfilled (dashed lines). The system automatically resets once we enter the 'done' state.

The remaining components of the POMDP tuple describe how the world behaves. 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, that is, what the user is likely to want next, given the state they were just in and the action the system just took. 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 reward gives us a way to specify what the "right" actions are in different states and how much the user is willing to tolerate clarification questions before becoming frustrated. The discount factor $\gamma \in [0,1]$ allows us to bias the dialog manager towards satisfying user intents more quickly; γ weighs how much we value future rewards to current rewards: a discount factor of 0 means that we only value current rewards, while $\gamma = 1$ implies that future rewards are just as valuable as current rewards.

Recall that the POMDP's state is hidden; after taking an action, we receive an observation to use when choosing our next action. In general, our estimate of the user's current state, and therefore the best next action to take, will depend on the entire history of actions and observations that the dialog manager has experienced. Since defining a policy in terms of the history can get quite cumbersome, we typically keep a probability distribution over the states, known as the belief. The belief 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:

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

where η is a normalizing constant. Thus, 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 also unique and satisfies the Bellman equation:

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

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. $\Omega(o|b,a)$, the probability of seeing o after performing a in belief b, is $\sum_{s\in S}\Omega(o|s,a)b(s)$.

This equation may be solved iteratively:

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

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

Each iteration, or *backup*, brings the value function closer to its optimal value[2]. 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.

Note that 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[6]. 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[6], 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 confusions over the user's intent, picking beliefs reachable from the starting belief focuses our computation in situations the dialog manager is likely to experience.

2.2 Uncertainty in Parameters

The policy that results from solving equations 3 and 4 depends critically on accurate choices of the transition probabilities, observation probabilities and the reward—these parameters will strongly effect how the system associates different utterances with different dialog states or how aggressive the system will be about assuming a correct interpretation of the users' desires. Capturing the underlying space of possible user states and system actions is usually not difficult, but specifying the probabilistic dynamics and perception models is difficult to do with certainty. We can often write down *reasonable* models, but rarely the *best* model. As a result, the dialog manager may seem "fussy," conservative, or badly behaved, depending on how accurately the model captures our own internal expectation of behavior.

What we would ideally like to do is to actually learn and improve the model as the dialog progresses with each user, assuming our reasonable parametric model as a starting point. The approach we will use is to assume that our model parameters are initially uncertain, and we will find a POMDP dialog management policy that models this uncertainty. We will then improve the model from experience, reducing the uncertainty of the parameters and recomputing the dialog manager policy over time.

The specific model of uncertainty we use is to represent T, Ω, and R by probability distributions with a collective set of hyperparameters Θ (for simplicity, we assume that the discount γ is known). The rewards are modeled as Gaussians; the model designer initializes a mean, variance, and a pre-observation count. The pre-observation count, a measure of the expert's confidence, corresponds the size of an imaginary sample-set from which the expert based his conclusions.

The uncertainty in the transition and observation distributions is captured with Dirichlet distributions. Recall from Section 2.1 that T and Ω are given by multinomial distributions; as a result the Dirichlet distribution is a natural choice because it places a probability measure over the simplex of valid multinomials. Figure 3 shows an example of a simplex for a discrete random variable that can take three different values; each transition probability distribution $p(\cdot|s,a)$ is some point on this simplex, and the Dirichlet gives a measure of the likelihood of each such distribution.

Given a set of parameters $\alpha_1 ... \alpha_m$, the likelihood of the discrete probability distribution $p_1 ... p_m$ 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, the normalizing constant, is the multinomial beta function. The expected values of the Dirichlet distribution are given by

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

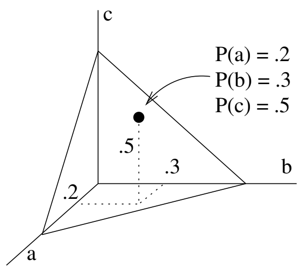

Figure 3: An example simplex for a multinomial that can take three different values (a,b,c). Each point on the simplex corresponds to a valid multinomial distribution; the Dirichlet distribution places a probability measure over this simplex.

and the mode is

$$E[p_i|\underline{\alpha}] = \frac{\alpha_i - 1}{\sum_j^m \alpha_j - m}. \tag{6}$$

Note that Dirichlet distributions have an additional, desirable property in that they are easy to update. For example, suppose we are given a set of observation parameters $\alpha_1 ... \alpha_{|O|}$ corresponding to a particular s,a. If we observe observation o_i, then a Bayesian update produces new parameters $(\alpha_1, \dots, \alpha_i + 1, \dots, \alpha_{|O|})$. Thus, we can think of quantity $\alpha_i - 1$ as a count of how many times observation o_i has been seen for the (s,a) pair. Initially, the expert can make an educated guess of the parameters—which we take to be the mode of the Dirichlet distribution—and a pre-observation total that represents the expert's confidence in his guess.

Knowing what α parameters to increment after a dialog sequence requires knowing what states occurred during the dialog. After each interaction, we know what observations were received and what actions were taken but not the actual states encountered. We do, however, have the estimate of the state sequence provided by the beliefs during the dialog execution, calculated by equation 1. At the end of the dialog, we can improve this estimate of the state sequence by constructing a Hidden Markov Model (HMM) from our expected-value POMDP and applying the Viterbi algorithm to determine the most likely complete state sequence for that interaction[1]. In general, the estimate of the state sequence will have many errors if the model parameters are not well known. However, the structure of the dialog POMDP (Figure 2) alleviates the HMM issues: since the most likely trajectory for a single interaction is through a single 'desire'-state until the user intent is satisfied by the dialog manager, there are fewer places for the HMM to err. To increase accuracy, we only update parameters after a successful interaction and pad the end of the history with 'done' observations. Our simulation results compare the performance of the HMM estimator with an oracle that gives us the true state history.

3. LEARNING NEW DIALOG POLICIES

We propose first solving for a dialog policy for the uncertain model using the expected values of the model parameters. After each dialog, we update our user model and apply additional replanning, or backups, to $Q(b, a)$ to refine the current dialog policy.

3.1 Solving the POMDP

We first consider the problem of finding an optimal dialog policy when the user model is uncertain and the dialog manager does

[1]HMMs and the Viterbi algorithm are well-developed approaches to statistical inference, especially in the speech-recognition community. A description of an HMM and Viterbi is beyond the scope of this paper, but the reader may refer to [8] for more details.

not expect to learn from interactions. A standard POMDP solution maximizes expected reward given the uncertainty over the user's intent (the state); we now desire a solution that maximizes the expected reward given uncertainty in both the user's intent 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 intent:

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

The first equation is an expectation over our uncertainty in the user's intent. The second equation averages over the stochasticity in the user model using our knowledge of how likely the user is to have changed their mind and imperfections in the voice recognition system. Since we are trying to compute the Q-value for a particular belief b, the value of b is unaffected by our uncertainty in the model. However, computing the vector $\vec{q_{a,i}}$—which is an average over the stochasticity in the user model—now requires an additional expectation over our uncertainty in the user model:

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

where $E_\Theta[R(s,a)]$, $E_\Theta[T(s'|s,a])$ and $E_\Theta[\Omega(o|s',a)]$ 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 mind, are independent of the noisy dialog observations Ω. The $V_{n-1,i}$ 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.

Recall that the solution to equation 4 is exponentially expensive, and so we approximate the solution polynomially with a fixed set of sample belief points using the PBVI algorithm [6]. This same approach holds for approximating the solution to equation 7. However, when the user model is uncertain, sampling beliefs using the expected value of model the may leave gaps in the belief space that are critical to the quality of approximation. We found these gaps were not an issue for our initial simulations, which used a simpler user model, but for our user tests we seeded the initial belief set with common dialog scenarios—pairwise and three-way confusions between user intents—in addition to the 'pre-dialog' belief b_0. These beliefs helped ground the dialog policy solution. Next we sampled single dialog interactions as before and added a random new belief to the set until we had sampled 500 total beliefs. This approach significantly improved our results in practice.

3.2 Updating the POMDP Solution

Updating the user model on R, T, and Ω gives us new expected values $E_\Theta[R(s,a)]$, $E_\Theta[T(s'|s,a)]$ and $E_\Theta[\Omega(o|s',a)]$ that we can use for solving a dialog policy. Instead of solving a new POMDP from scratch, however, we apply the intuition that the information from a single conversation likely causes only small changes to the dialog policy. We perform additional backups on our current value function with the updated user model. Since the backup operation of equation 4 is a contraction ([2]), the additional backups will always bring the original dialog policy closer to the solution with the new user model. Thus, the dialog policy will converge to the optimal policy if our distribution over user models converges to the true user model. Unfortunately, the distributions may not converge: because new data depends on the last action taken, the current policy causes us to miss certain areas of the model space. Even with an infinite number of trials, if the dialog manager always takes what it currently thinks is the best action—instead of risking an exploratory action—it may miss opportunities to improve. Although we saw significant dialog improvements in practice, we are currently building explicitly exploratory dialog into our approach.

3.3 Update Heuristics

A key question is how much replanning is required—that is, how many backup operations should be performed—after updating the user model. Iterating on a value function to convergence is expensive. Especially near the beginning of the learning process, iterating to convergence after each dialog may have limited benefits: if we initially had low confidence in our user model estimates, the parameter distributions will take some time to start peaking near the true values. Solving for the intermediate dialog policy completely could be a waste of effort. We compare three approaches to determine the number of backups to complete after each update:

1. Replan to completion. After each completed dialog, we perform enough backups for the new value function to converge.[2] This should lead to the best expected performance given the uncertainty in the user model. However, the current model statistics may come from a few unrepresentative dialogs so that the computed policy is wrong. Here, more learning must be done before performance can improve and careful planning may be wasted effort. Computing a POMDP policy to convergence is also a slow process, leading to long latencies in dialog manager responses.
2. Replan k times. Perform k backups, regardless of how much the parameter statistics change. This approach may prevent latencies in the dialog manager performance, but does not give the dialog manager time to compute better plans once we have 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 our user model. If an update reduces the overall variance, we backup $\lfloor k * dvar \rfloor$ times (proportional to variance reduction). The intuition is that we wish to expend the most computational effort when the user model becomes more certain. For simulation purposes, we also capped the number of backups per update at 50.

4. RESULTS

We tested our approach in an artificial simulation and with a robotic wheelchair (see Table 1 for a summary of the key parameters). In both cases, the user could choose to go to one of five locations. Thus the seven dialog states were dialog-start, dialog-done, and a state for each of the five locations. The twelve actions consisted of four types: general query ("Where do you want to go?"),

[2]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.

confirming a location (e.g., "Do you want to go to the cafe?"), going to a location (e.g., "Going to the cafe."), or doing nothing. We assumed that some static reward was associated each state-action pair (although realistically, a user may be frustrated if the robot repeated the same query). In the simulations, there were observations that corresponded to confirmation, task completion, and junk, as well as a unique observation for each state. The wheelchair test model was more complex: it included additional observations that were initially not associated with any location.

The human-robot interaction proceeded as follows: (1) The user spoke to the robot. (2) The robot chose an action based on the utterance. (3) The robot prompted the user to enter a reward (the user had a set of discrete reward values from which to select). If the user did not enter "Task Done," the robot stored the reward value and waited for further user interaction (the robot only used utterances to make decisions during a dialog, since explicit reward feedback would be given only during an initial training period). If the user entered "Task Done," the stored history of (observation, action, reward)-tuples were used to update the dialog model and policy.

Table 1: Model Parameters for Simulation and User Tests.

Parameter	Simulation	User Test
States	7	7
Actions	12	12
Observations	8	19

4.1 Simulation

The simulations allowed us to test our learning approach against a known ground truth optimal policy. In the 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 initial guesses and true values). We specified self-transition and true observation probabilities, and the probabilities of all the other options were uniformly distributed. Finally, we attributed our guesses to two pseudo-observations per event per state-action pair to indicate relatively low confidence in the initial parameter estimates.

As a measure of correctness of our approach, the first set of tests included an "oracle" that revealed the true user state history after each interaction. This assumption (only tenable in simulation) allowed us to guarantee that data about observation and transition probabilities would be applied to the correct Dirichlet parameter, rather than estimating the state sequence as described at the end of Section 2.2. As seen in Figure 4, all approaches achieved similar results, but replanning proportionally to the variance reduction

Table 2: 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.

	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

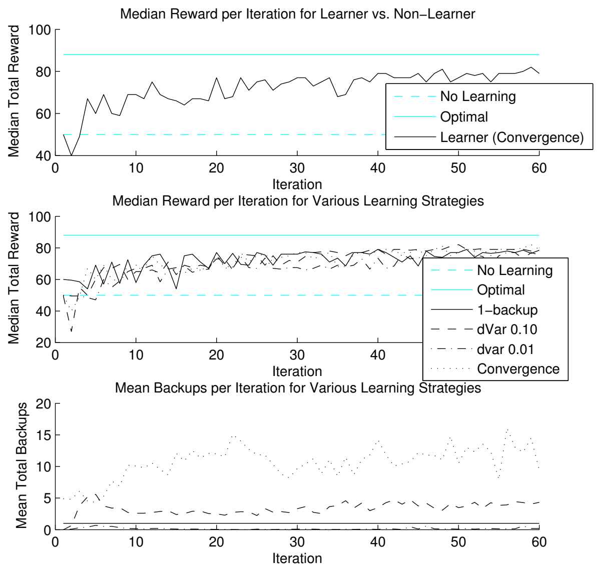

Figure 4: Performance and computation graphs. The learner outperforms the non-learner (top graph), and all of the replanning approaches have roughly the same increase in performance (middle graph), but replanning proportionally to the confidence of the model achieves that performance much less computation (and therefore faster response) than replanning to convergence (bottom graph).

Table 3: 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.

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

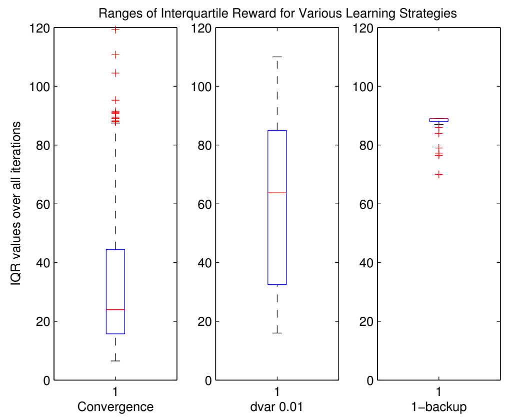

Figure 5: Interquartile Ranges (IQR) of the rewards. All of the replanning approaches have roughly the same median performance, but additional planning results in a more stable solution. Note that an IQR of 0 corresponds to zero variation around the median solution.

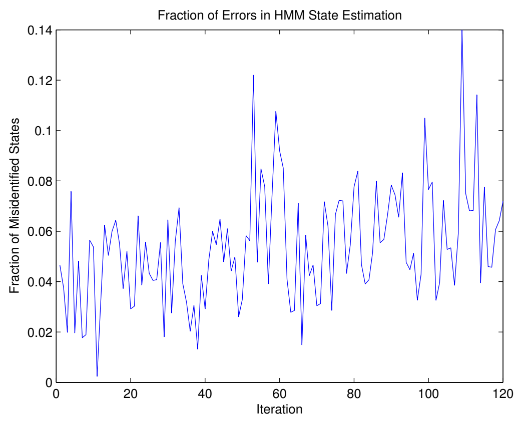

Figure 6: Fraction of states misidentified in a particular interaction. Most errors arose when the simulated user changed his mind part way through the sequence; in these situations the first part of the sequence was usually classified incorrectly (lumped with the second part of the sequence).

achieved that result with almost as little computation as replanning once per update (see Table 3; 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 variance reduction approach focuses replanning near the beginning of the trial when parameters first become certain. Each point represents 100 trials (except 'convergence'—we could complete only 37 trials).

Although the k_{dvar}=0.01 approach completed fewer total backups than backing up once per iteration, the larger number of backups near the beginning of the learning process helped stabilize the solution. Figure 5 shows boxplots of the interquartile ranges (IQRs) of the solutions. An IQR of 0 would mean that the policy always resulted in the median value solution. As expected, iterating to convergence has the smallest IQR range. What is most interesting, however, is that the k_{dvar}=0.01 solution—which performs fewer backups on average than the 1-backup approach—still achieves a lower variation in the performance of the resulting policy.

Next we relaxed the "oracle" requirement and tested our approach with HMM-based history estimation. The HMM performed slightly worse than the oracle due to estimation errors. The errors usually occurred when users changed their intentions midway through a dialog. Because our model stated that user intentions were unlikely to change (with probability 5%), the Viterbi algorithm usually assigned a single to state to the sequence when the true sequence consisted of two states. Instead of attributing the first part of the observation history to the original state and the second part to the final state, all observations were attributed to the final state. Thus, the dialog manager believed that the observation inputs were more noisy than they actually were and therefore learned a slightly more conservative policy. Although the HMM-based simulation achieved almost the same performance as the oracle, we also note (Figure 6) that the fraction of misidentified states grew with time.

4.2 Wheelchair Implementation

In addition to the simulations, we demonstrated dialog management learning on the prototype voice-controlled wheelchair of figure 1. These anecdotal users tests partially used the CMU Sphinx-2 voice recognition system [9].[3] To allow the system to run in real-time, only one backup was performed after each parameter update — the goal of these experiments was not to demonstrate the speed of different planning techniques but to show the improvement in dialog management with model parameter learning. While it is not possible to claim optimality in a study with a real user, we do show that our approach produces results that not only reasonable but are also better than a policy derived from our initial guesses. To expedite user tests, no motion commands were executed, although the wheelchair was capable of driving to the various locations.

To analyze the user input, the voice recognition software (or simulated textual input) first produced a list of phrases given an utterance. Next, the occurrences of (previously specified) keywords were counted in the voice recognition output, without regard to ordering or grammar. Finally, the dialog manager received a vector of normalized counts as the probability of having seen each keyword.[4]

The keyword approach made the wheelchair dialog manager's observation management different than our simulations. The simulation contained only 8 observations: one corresponding to each state, confirmations, and noise. The wheelchair received a vector of probabilities for 19 keywords. The model initially had one unique observation per state (for example 'parking' was highly likely in the state 'I want to go to the parking lot.'). All other keywords—such as 'elevator' or 'tower'—were left as equally likely in all states.

The dialogs contained three principal kinds of errors:

- Speech Recognition Errors. Sphinx often confused similar sounding words; for example, the software tended to mistake the work 'desk' with 'deck.' In the absence of this knowledge, however, we had assumed that the we would observe the word 'desk' more often if the user was inquiring about the information desk and 'deck' more often if the user was inquiring about the parking deck. We also made difficult to recognize words more likely to be dropped (for example, 'parking' and 'information' were harder words for our software to recognize). Note that the speech recognition errors the system encountered were filtered to eliminate interactions where the recognition failed to produce any recognized utterances, so these results do not precisely capture all speech recognition errors of our current system.
- Mapping New Keywords. General keywords, such as 'tower,' or 'elevator,' were not initially mapped to any particular state. Some unmapped keywords were more likely in one particular state (such as 'parking deck' or 'the deck' for 'parking lot'; 'the elevator' for 'the Gates elevator'), while others occurred in several states ('I'd like to go to the information desk a lot' uses the word 'lot' outside the context of 'parking lot').
- Misuse of Keywords. Especially in a spatial context, users could refer to other locations when describing the desired state. For example, they might say 'the elevator by the information desk' or 'the Gates-side booth.' Our simple bag-of-words approach to speech analysis precluded complex language understanding; the dialog manager had to associate higher noise with keywords that occurred in many states.

Our preliminary user tests suggest trends that show that our approach improved the quality of the dialog manager. We compared a one-backup per iteration approach to a static dialog manager using initial parameter estimates provided by a human modeler. In both

[3]Sphinx-2 often failed to interpret utterances. For our tests, users spoke into the microphone until the software produced a set of meaninful interpretations. These phrases, which included common errors, were then inputted as text to produce the results. While not a rigorous experiment, this approach allowed us to demonstrate the ability of our dialog manager to learn.

[4]The standard approach to POMDPs assumes a single observation instead of a distribution over observations. However, we can easily incorporate an observation distribution by introducing an expectation over observations in the belief update (equation 1).

Table 4: Initial and True Parameter Values for User Trial.

	Initial Value	True Value
P(correct obs if ask-which)	0.6	see graphs
P(correct obs if confirm)	0.8	1.0
R(complete task)	100	100
R(ask-which action)	1	-30
R(correct confirm)	10	-1
R(incorrect confirm)	-10	-10
R(incorrect destination)	-200	-500
R(incorrect no-action)	-1000	-100

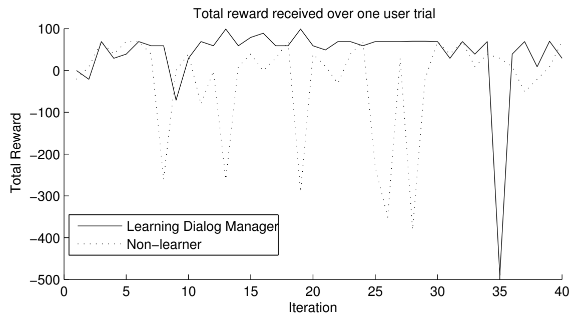

Figure 7: The learner (solid) generally outperforms the non-learner (dashed), rarely making serious mistakes.

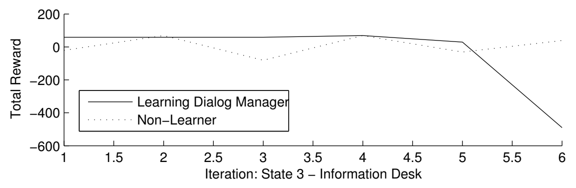

Figure 8: This plot includes only Information Desk requests. Observations in this state were 31% original keywords, 46% new words, and 23% misused keywords. Unmapped words more strongly associated with other states made it tough for the learner to improve the dialog policy.

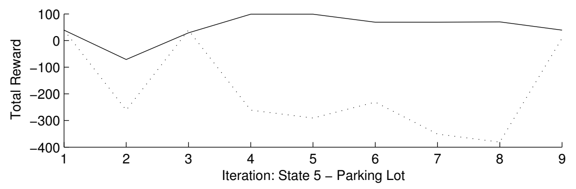

Figure 9: This plot includes only Parking Lot requests (and is representative of the other states). Observations in this state were 9% original keywords, 73% new words, and 18% misused keywords. The learner soon realizes that the user often refers to the parking lot with the unmapped work 'deck.' Since the user is forgiving to confirmation questions, the dialog policy applies them aggressively to disambiguate misused keywords.

Table 5: Mean overall and per state results for a second user test consisting of 36 dialogs. The learning dialog manager improves over the initial, hand-crafted policy in most states. The exception (State 3) is due to a single outlier.

	Overall	S1	S2	S3	S4	S5
Non-learner	-16.9	4.6	10.7	39.0	49.2	-187.7
Learner	37.2	54.6	50.4	-35.8	67.3	49.1

scenarios, the user rewarded the system after each interaction using a graphical interface on the wheelchair. Table 4 shows the guessed and actual rewards provided by the user. Pre-observation counts were five pseudo-observations per event per state-action pair. The dialog manager first believed that there was a 60% chance of hearing the keyword for each state and the remaining probability was spread uniformly among the remaining observations.

Each user requested locations in the same order for both tests. Since users did not change their intention over the course of the interaction, the HMM approach exactly determined the users' state history (no additional oracle was required). Figures 7, 8, and 9 show the results of a test interaction. The dip in reward near the beginning of the test (Figure 7) indicates the first time that the dialog manager encountered misused keywords; misuse errors were the hardest for the dialog manager to learn.

The learning dialog manager outperformed the non-learner for most states (see Table 5). Much of the benefit derived from learning the mappings and distributions of new words. For example, the building contained three elevator locations, but the user usually referred to only one elevator as simply 'the elevator' and to the others by 'location-elevator.' Thus, the dialog manager could (and did) shorten interactions (and thus increase reward) by asking confirmation questions immediately after hearing the word 'elevator.' The improvement in Figure 9 (corresponding to the parking lot) is a dramatic example of how the dialog manager was able to learn new words (see Table 6 for an example interactions).

The dialog manager had the most difficulty in states where many of the keywords (mapped or originally unmapped) corresponded to words more common in other states, shown in Figure 8 (We saw similar behavior in a second user trial where the user did not misuse any of the originally mapped keywords: most states had better performance, but the use of many vague keywords still stifled the performance for that state). Since the dialog manager learned the observation distributions well in both cases, we believe that additional backups may have improved performance; one backup was not enough for the large policy changes required for that state.

Finally, we tested the robotic wheelchair with actual data taken from a voice recognition system. Our voice recognition system was extremely noisy, so to expedite repeated tests we took user samples and discarded those that were clearly junk (containing no keywords). Table 7 summarizes results from a second user test. For the most part, the user did not use vocabulary that taxed the original dialog manager, but we saw a large improvement in the one state where the user tended to use an alternate keyword to refer to a location. The remaining states showed relatively small changes.

5. RELATED WORK

The Medusa algorithm [3] comes closest to solving our problem. Instead of performing backups on a single POMDP, Medusa takes a particle filter approach and maintains a set of weighted POMDPs from an initial distribution. Each POMDP's solution is never updated, but POMDPs are periodically resampled. Actions are chosen stochastically: each POMDP applies its weight toward the action that its policy recommends. Our implementation of MEDUSA was too slow to make a fair comparison to our approach.

Medusa's sampling approach is based on the observations of [1], who suggests keeping a sampling of MDPs and updating their value functions as new data is received. Our approach may be viewed as an extension to POMDPs that avoids solving and updating multiple POMDPs. We also begin to quantify the number of backups that should be performed after a parameter update.

Finally, [5] takes a non-Bayesian approach to solving MDPs with uncertain transition matrices. Instead they apply a minimax ap-

Table 6: Parts of dialogs between the user and the wheelchair. The dialog specification has failed to specify how to handle the words 'deck' and 'elevator.' In the first dialog, the non-learner is so confused when it first hears the word 'parking' that it takes a second utterance for it to select an appropriate action. In the second dialog, the learner realizes that while elevator refers to multiple locations, the user most often uses in the context of the Gates Tower.

Without learning
USER: Take me to the deck between the two towers.
The user says "deck" to refer to the parking lot. The system does not know this yet.
ROBOT: Where did you want to go?
USER: Yes, I want to go to central elevator that goes to the deck
ROBOT: Where did you want to go?
USER: The middle deck, between Gates and Dreyfoos
ROBOT: Do you want to go to the Gates Tower?
USER: No.
ROBOT: Going to the Dreyfoos Tower.
USER: No! I want to go to the deck!
The system still cannot associate "deck" with any destination.
ROBOT: Doing nothing.
USER: Take me to the deck.
ROBOT: Where did you want to go?
USER: The parking lot.
ROBOT: Where did you want to go?
USER: I said the deck.
ROBOT: Where did you want to go?
USER: Yes, the parking deck.
ROBOT: Going to the parking lot.
After learning:
USER: Take me to the deck between the two towers.
ROBOT: Where did you want to go?
USER: Yes, I want to go to central elevator that goes to the deck
ROBOT: Do you want to go to parking lot?
USER: Yes.
ROBOT: Going to parking lot.

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.

Table 7: Mean overall and per state results for a single end-to-end user test consisting of 36 dialogs. In general the learning dialog manager showed slight improvements, especially in mapping unmapped keywords (in this case, State 3).

	Overall	S1	S2	S3	S4	S5
Non-learner	45.0	69.0	59.0	-11.0	59.0	49.0
Learner	55.9	72.4	66.3	32.5	52.6	55.9

proach to computing the policy. Learning is not incorporated into the approach, but extending their algorithm to POMDPs and incorporating model learning may be another alternative to planning with POMDP model uncertainty.

6. DISCUSSION AND CONCLUSION

Our heuristics show that real-time POMDP learning is promising for dialog management, and in the future we hope to place better bounds on the replanning required. 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.

Our dialog policy maximized the expected reward over the uncertain user model, but the few outliers in our user tests suggest that expected reward may not be the right criterion, especially in the healthcare domain. However, if we apply a more conservative dialog policy while learning, we must manage the exploration-exploitation trade-off such that the optimal policy is eventually found. We plan to address this question in our future work.

7. REFERENCES

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

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

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

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

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

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

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

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

[9] M. Ravishankar. *Efficient Algorithms for Speech Recognition*. PhD thesis, Carnegie Mellon, 1996.

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

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

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