Using Factored Partially Observable Markov Decision Processes with Continuous Observations for Dialog Management

J. D. Williams, P. Poupart, S. Young

24 March 2005

University of Cambridge, Dept. of Engineering

Technical Report: CUED/F-INFENG/TR.520

Table of Contents

Using Factored Partially Observable Markov Decision Processes with Continuous Observations for Dialog Management

Jason D. Williams
Cambridge University
Engineering Department
Cambridge, UK
jdw30@cam.ac.uk

Pascal Poupart
University of Waterloo
School of Computer Science
Ontario, Canada
ppoupart@cs.uwaterloo.ca

Steve Young
Cambridge University
Engineering Department
Cambridge, UK
sjy@eng.cam.ac.uk

Abstract

Within a spoken dialog system, the dialog model is responsible for tracking the current state of the conversation, and the dialog manager is responsible for action selection and planning. This work shows how a dialog model can be represented as a factored Partially Observable Markov Decision Process (POMDP), and how automated techniques can be used to produce a dialog manager. It then shows how direct comparisons can be made between automated and handcrafted dialog managers, and also how handcrafted dialog managers in this context can be easily improved. Finally, the work describes how a speech recognition confidence score can be incorporated into the model. To test our proposals, an example problem is presented, and a variant of Point-Based Value Iteration is used for policy improvement. Empirical comparisons show that the POMDP-based model outperforms fully-observable Markov Decision Processes and hand-crafted dialog managers.

1 Introduction

Spoken Dialog Systems help users achieve some goal through spoken language. Within a Spoken Dialog System, a Dialog Manager (DM) interprets evidence from the conversation and decides what *system action* to take to reliably and efficiently satisfy a user's goal. Actions might include asking a question, confirming a user's goal, querying a database, or stating information.

The dialog management task is complex for several reasons. First, the system observes the user's actions via automated speech recognition and language parsing: imperfect technologies which corrupt the evidence available to the system. Second, each user action (even if it could be observed accurately) provides incomplete information about a user's goal, so the system must assemble evidence over time to infer a user's goal. Because the user might change their goal at any point, inconsistent evidence could either be due to a channel (speech recognition) error or due to a changed user goal. Thus,

deciding how to interpret conflicting evidence is a challenge. Finally, the system must make trade-offs between the "cost" of gathering additional information (increasing its certainty of the user's goal, but prolonging the conversation) and "cost" of committing to an incorrect user goal. That is, the system must perform *planning* to decide what sequence of actions to take to best to achieve the user's goal despite having imperfect information about that goal.

For all of these reasons, the dialog management problem can be regarded as planning under uncertainty. Researchers have previously applied both (fully-observable) Markov Decision Processes (MDPs) and Partially Observable Markov Decision Processes (POMDPs) to the dialog management problem.

The application of MDPs to the dialog management problem is first explored in (Levin and Pieraccini, 1997). (Levin et al., 1998) and (Levin et al., 2000) provide a formal treatment of how an MDP may be applied to the dialogue management task. (Walker et al., 1998) and (Singh et al, 2002) show application to practical systems.

Building an MDP model typically requires extensive training data, which is expensive to collect. To address this issue, researchers have explored training with user models (Scheffler and Young, 2002), methods for generalizing actions to new situations (Denecke et al., 2004)). , and "bootstrapping" a solution using supervised learning (Williams and Young, 2003). More thorough investigations of state space and reward functions have also been undertaken (Pietquin, 2004).

Fully-Observable Markov Decision Processes take a principled approach to planning in which a dialog strategy can be automatically determined from a designer's specification of desired outcomes, or from users' feedback. Further, the quality of the "plan" can be automatically improved over time with reinforcement learning techniques that interact with the environment.

However, because MDPs assume the current state of the environment (i.e., the dialog) is known exactly, they do not naturally or precisely model "noisy" evidence. Thus one might hypothesize that MDP-based techniques would under-perform POMDP-based techniques in the Dialog Management domain. (Roy et al., 2000) tests this hypothesis by evaluating an MDP and a POMDP version of the same spoken dialog system.[1] Two evaluations are conducted – one using a user model, and another with a small handful of usability subjects. In each case, the POMDP version of the spoken dialog system gains more reward per unit time than the MDP version. Further, the authors show a trend that as speech recognition errors increase, the margin by which the POMDP outperforms the MDP increases.

While (Roy et al., 2000) show promise for the POMDP technique, many issues are left unaddressed. First, the state space of 13 states is handcrafted, and there is a tight coupling between the user's goal, the user's action, and the state of the conversation. It is not clear how to separate these components in a principled way, and thus it is not clear how to apply the ideas in this work to other dialog systems. Second, the dynamics of the problem are handcrafted, and it is not clear how the transition or observation function

[1] The POMDP version is solved as an "Augmented MDP" which summarizes the multi-dimensional belief state as the single most likely state and the (real-valued) entropy of the belief state.

should be estimated. Finally, the reward function used in (Roy et al., 2000) contains rewards for only "right-answer", "wrong-answer" and "neither," yet a more nuanced reward function, taking into account "appropriate" dialog behavior – seems appealing. However, a more nuanced reward measure would require that the state space includes a "dialog state" – i.e., information such as whether a piece of information has been requested already, has been confirmed, etc. The state space in (Roy et al., 2000) does not include an explicit dialog state.

(Zhang et al., 2001) extends the work in (Roy et al., 2000) in several ways. First, the authors add five "hidden" system states to account for various types of dialog trouble, such as different sources of speech recognition errors, or the lack of a response from the user. These "hidden" system states are combined with states representing the user's goal to form a factored state space. Second, the authors use 2-stage temporal Bayesian Networks to combine observations from a variety of sources (e.g., parse score, acoustic confidence score, etc.) Finally, the authors compare a range of solution techniques, including MDP, the so-called Q-MDP approximation, and grid-based approximation, which they find is most successful when evaluated using a model.

In comparison to (Roy et al., 2000), (Zhang et al., 2001) demonstrated the utility of adding "hidden" system states to model dialog trouble, and the ability of grid-based solution techniques to perform policy improvement on a small POMDP which captures basic elements of the dialog management problem. However, several issues are still left unaddressed. First, the "user model" (i.e., how the user is likely to respond in a given situation) and the "speech recognition model" (i.e., what kinds of errors the speech recognizer is likely to make) are conflated into the POMDP observation function. The authors indicate that the system dynamics are "handcrafted, depending a lot on the experience of the developer." It is not clear how to form the POMDP observation function, nor estimate its parameters in practice. Second, although the reward function has costs for various "repair" actions such as asking the user to repeat, confirming, or troubleshooting, these actions are given fixed penalties. The reward measures are not used to guide "appropriate" behavior for the current dialog state: in fact, a persistent dialog state is not included in the state space.

This paper makes several contributions in the area of Dialog Management through novel insight into the Dialog Management problem combined with recent work in the POMDP community. This paper introduces:

- *A factored architecture* for describing POMDPs applied to spoken dialog management. Unlike past POMDP-based (or MDP-based) dialog management work, our factored representation adds a component for the state of the dialog *from the standpoint of the user*, enabling reward measures for the "appropriateness" of system actions. It also creates separate distributions for the user model and the speech recognition model, which reduces the number of components which need to be calculated, and allows groups of parameters to be estimated separately.[2] These

[2] In the literature, factored models have also been explored as a technique to address large state spaces. In this work we don't make use of the factored architecture when performing optimization. Here we propose the factored architecture as a technique to more easily incorporate real data into the model. We plan to explore using the factored architecture to assist with policy optimization in future work.

components are also quite general and easy to adapt for dialog managers applied to various tasks.

- A method to *convert a hand-crafted policy into a POMDP policy* and makes direct comparisons with automated solutions. Hand-crafted policies are often used as a baseline for dialog managers created with automated technique, yet past work has not shown a method for performing this comparison. We also show a simple method to *improve the performance of a hand-crafted policy* by incorporating elements of basic POMDP solution techniques. This is particularly interesting given that domain experts can often specify simple hand-crafted dialog managers, which can then be automatically refined.
- A principled method to *integrate a speech recognition "confidence score"* into this architecture. A "confidence score" is a metric of reliability provided by the speech recognizer. Past work has relied on discretizing the confidence score; however, it is not clear how this discretization should be performed. In this paper we show how a continuous confidence score can be naturally incorporated into the belief state update.

We show proof-of-concept in terms of both tractability and effectiveness for each of these concepts through a testbed DM problem. With our factored model, we demonstrate how to represent a more expressive dialogue model than previously applied to POMDPs, and show how recent algorithms developed in the POMDP community applied to this problem produce dialog managers which outperform both MDP and handcrafted baselines.

The paper is organized as follows. Section 2 presents background on spoken dialog systems, POMDPs, and relevant solution techniques. Section 3 presents the factored architecture. Section 4 shows an example testbed system using this architecture, and compares it to a baseline MDP system. Section 5 describes a method to compare these results with a hand-crafted controller, and how a handcrafted controller can be easily improved by applying insights from the POMDP literature. Section 6 describes a method for integrating confidence score into the architecture and shows results. Section 7 concludes.

2 Background

2.1 Spoken dialog systems

In this work, we define a spoken dialog system to be a machine which helps a user to achieve some goal through spoken language. A typical spoken dialog system consists of the components shown in .

The user enters the conversation with the intention of fulfilling a goal, s_u. To achieve this, the user takes a communicative action, a_u, where a_u might be represented by slot-value pairs, dialog acts, speech acts, or a number of other formalisms. The user's action a_u is encoded into a sequence of words which is rendered as a speech signal, $\hat{a}_u$, and provided to the machine as input. $\hat{a}_u$ is then processed by a speech recognition and language understanding component, which produces an observation *o* in an attempt to

reconstruct a_u. The recognition process is error-prone, and in general we cannot regard o as faithfully representing a_u. The speech recognition and language understanding component may also provide additional information such as a confidence score c which attempts to provide a clue about the reliability of o. Alternative hypotheses for a_u might also be provided, perhaps with their own confidence scores.

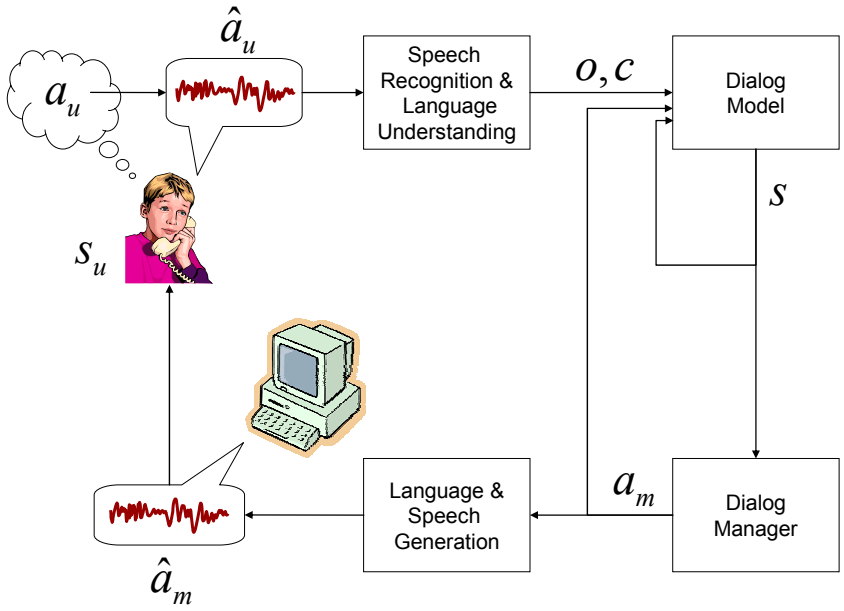

Figure 1: Overview of a typical spoken dialog system. See text for description.

o and c (if c is available) are then passed to the dialog model component. The dialog model component attempts to track the state of the dialog, s, based in part on o. s might include components for the values of the slots which must be provided by the user, whether each slot has been confirmed, the confidence recorded for each slot, etc.

Based on s, the dialog manager decides what machine action, a_m, to take next.[3] This action may serve to communicate with the user through a question or statement or be a "processing" action such as consulting a database.[4]

The machine action a_m is also required to capture and update the current state of the dialog and is also provided to the dialog model. a_m, o and the current dialog state s form the complete input to the dialog model, which produces the next dialog state s' as its output.

[3] There may be more information in the dialog state s than is required by the dialog manager. For example, it may be important to track the value a user has provided for a slot (such as *from=London*), but that value might not be important for action selection (for example, *ask(to)*, or *confirm(from)*). For this reason, spoken dialog systems sometimes include a *state estimator*, which maps the complete state s to an *estimated state*, which forms the input to the dialog manager.

[4] For clarity of exposition, "processing" actions are not shown in the diagram.

Communicative system actions a_m are passed to the language and speech generation components, which render a_m as speech $\hat{a}_m$ to the user. The language and speech generation component typically first generates a word string, and then converts this word string into the speech signal $\hat{a}_m$ using pre-recorded utterances or text-to-speech technology.

Creation of the dialog management (DM) component has historically been approached as a design problem, and many frameworks have been proposed to enable human experts to efficiently craft a dialog manager. Typical design decisions address the uncertainty introduced by the error-prone speech and language understanding components – e.g., when to confirm, re-ask, or accept a slot. Dialog managers are also responsible for making progress toward satisfying the user's goal; thus, they must often make trade-offs between accuracy and efficiency.

Designing successful dialog managers by hand is difficult in part because the state space of s is often very large, and selecting actions appropriate for each state is a tedious process. Further, the effects of the uncertainty introduced by the recognition and understanding component is not always well understood, and user behavior patterns can be difficult to predict at design time, requiring time-consuming development iterations.

To address these issues, researchers have begun applying decision-theoretic techniques, the focus of this paper, to the dialog modelling and dialog management problems.

2.2 Partially Observable Markov Decision Processes

To define a Partially-Observable Markov Decision Processes (POMDP) it is easiest to start with a Markov Decision Process (MDP). An MDP is defined by a tuple $\{S, A_m, T, R\}$ where S is a set of states, A_m is a set of actions that an agent may take, T defines a transition probability $p(s' \mid s, a_m)$ (sometimes called the transition matrix), and R defines the expected (immediate, real-valued) reward $r(s, a_m)$.[5] At each time-step, the machine is in a particular state s. The machine selects an action a_m, receives a reward r, and transitions to a state s', where s' depends only on s and a_m. The goal of the agent is to maximise the cumulative, infinite-horizon, discounted reward R (sometimes called the *return)* over an infinite horizon:

$$R = \sum_{t=0}^{\infty} \lambda^t r(s_t, a_{m_t}) \tag{1}$$

where λ is a geometric discount factor, $0 \le \lambda \le 1$.

A *policy*, $\pi(s) \in A_m$, is a mapping from states to actions; an *optimal policy,* $\pi^*(s) \in A_m$, is one which maximises cumulative reward over time. It can be shown that an MDP

[5] The reward function may also be a function of the next state *s'*. Also, some definitions of an MDP include a discount factor λ, and the initial state, s_0. In the literature, the system action is often written as an un-subscripted *a*. In this work, we will model both system and user actions, and here have chosen to write the system action explicitly as a_m for clarity.

evaluated over an infinite horizon in which $\{S, A_m, T, R, \lambda\}$ are fixed has a deterministic policy $\pi^*(s)$ which is independent of time step *t*. An influence diagram depiction of an MDP is shown in .

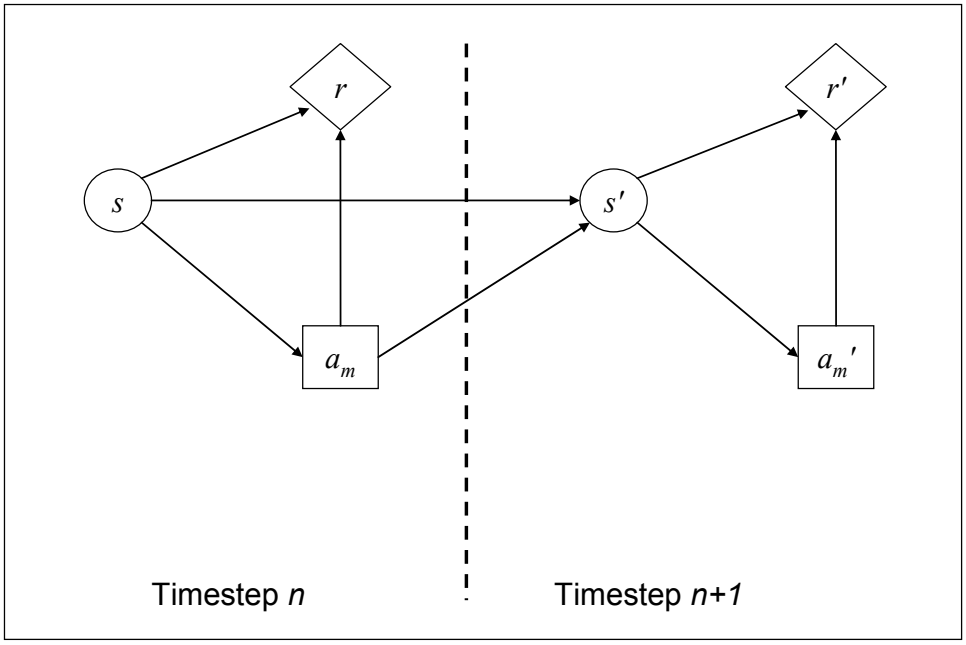

Figure 2: Influence diagram representation of an MDP. *s* and *a* are discrete. Unfilled circles indicate random, observed variables; diamonds indicate real-valued utilities and squares show decision nodes. Arrows indicate direction of influence.

POMDPs extend Markov Decision Processes by removing the requirement that the current state is known precisely. Instead, the machine makes *observations* which give incomplete information about the true current state, and the machine maintains a distribution over MDP states which is referred to as a belief state.

Formally, a POMDP is defined as a tuple $\{S, A, T, R, O, Z\}$, where $\{S, A, T, R\}$ define an MDP, O is a set of observations, and Z defines an observation function given by $p(o' \mid s', a_m)$.[6]

At a given time step, we no longer know the precise state *s* – rather, we maintain the belief state, *b*. We write b_t to indicate the distribution over all states at time *t,* and $b_t(s)$ to indicate the probability of being in a particular state *s* at time *t*.

The immediate reward is computed as the expected reward over belief states, $\rho(b_t, a_{m_t})$:

$$\rho(b_t, a_{m_t}) = \sum_{s \in S} b_t(s) r(s, a_{m_t}), \tag{2}$$

and the *return* is again the discounted sum of immediate rewards at each time step:

[6] This can be read as "The probability of making an observation given that action *a* was taken *and* the underlying MDP transitioned to state *s'*." Also, some definitions of a POMDP include the discount factor, λ, and the initial belief state, b_0.

$$R = \sum_{t=0}^{\infty} \lambda^t \rho(b_t, a_{m_t}) = \sum_{t=0}^{\infty} \lambda^t \sum_{s \in S} b_t(s) r(s, a_{m_t}) . \quad (3)$$

At each time step, the next belief state $b'(s')$ can be computed exactly as shown in Equation (12).

We note here that the set of observations may be discrete & finite, discrete & infinite, continuously-valued, or a mixture of these types. In this work we will make use of several observation types. An influence diagram depiction of a POMDP is shown in .

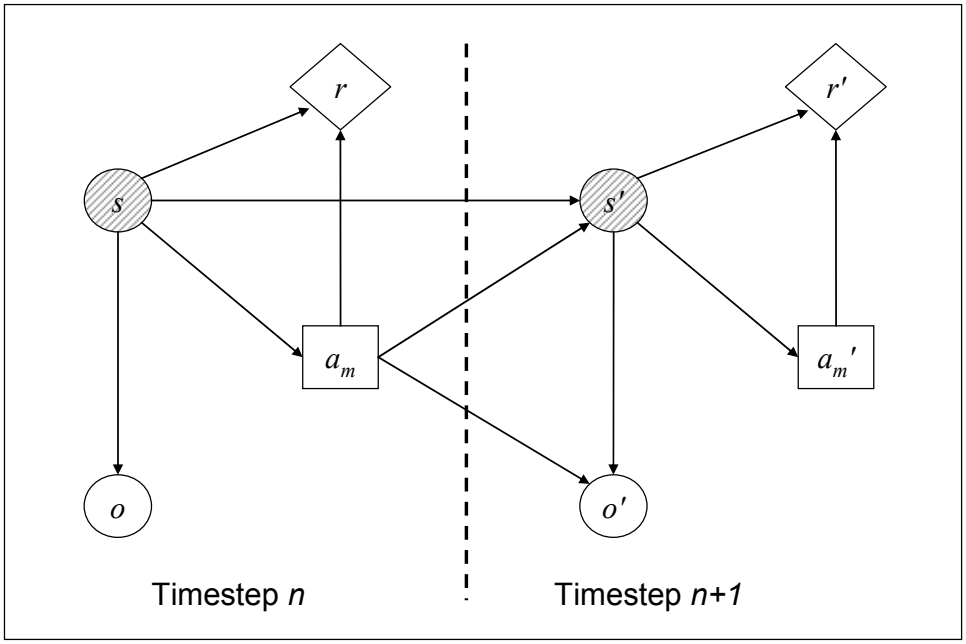

Figure 3: Influence diagram representation of a POMDP. Symbols are the same as used in ; shaded circles indicate unobserved random variables. *o* can be discrete & finite, discrete & infinite, continuously valued, or a mixture of these types. Note that a_s depends on the belief state *b(s)* – the distribution over all states – and not the true current (unobservable) state.

2.3 Optimization

Although POMDPs and MDPs consist of a defined tuple, in practice we may or may not know the *values* of the functions *T, R,* and for POMDPs, *Z*. If these values are known, we can apply analytic methods to improve a policy. If these values are not known and not easily estimated, we can observe actual interactions with a real or simulated environment to improve a policy with reinforcement learning.

Policy improvement with a simulated environment is particularly appealing when the environment can be readily modelled but not easily mapped to the MDP parameters – for example, when the environment maintains some internal, hidden state. For example (Scheffler and Young, 2002) and (Pietquin, 2004) optimized policies for MDP-based dialog managers using interactions with a simulated user and speech recognizer. The user model in each case has a consistent goal (i.e., an internal state) which is not directly observable by the MDP.

One of the strengths of POMDPs is that they can represent a hidden internal state directly – i.e., a POMDP can *contain* a probabilistic user model and maintain a distribution over true user goals. Hence, in this work, we will assume that a probabilistic user model & speech recognition model can be constructed and embedded into a POMDP, directly yielding values for *T* and *Z*. Thus, this work focuses on POMDP policy improvement using analytic methods rather than through interaction.

Policy improvement for POMDPs is notoriously difficult. First, because belief space is real-valued, an optimal infinite-horizon policy may consist of an arbitrary partitioning of *S*-dimensional space.[7] Moreover, the size of the policy space grows exponentially with the size of the observation set and *doubly* exponentially with the distance (in timesteps) from the horizon. Searching for an (exact) optimal policy using value iteration is also plagued by the so-called "curse of dimensionality" – the dimensionality of the belief space grows exponentially with the number of state variables (Kaelbling et al., 1998).

Nevertheless, real-world problems tend to exhibit a significant amount of structure that can be exploited to speed up computation. State abstraction (Boutilier and Poole, 1996) and compression techniques (Poupart and Boutilier, 2002), (Poupart and Boutilier, 2004), (Roy and Gordon, 2002) can be used to mitigate the curse of dimensionality. Similarly, point-based value iteration (Pineau et al., 2003), (Spaan and Vlassis, 2004) and bounded policy iteration algorithms (Poupart and Boutilier, 2003), (Poupart and Boutilier, 2004) can be used to mitigate the complexity of policy spaces.

In this work, we use a randomized value iteration algorithm called *Perseus* (Spaan and Vlassis, 2004). *Perseus* heuristically selects a small set of representative belief points, and iteratively applies value updates to those points. Whereas an iteration (also called a "back-up") of policy improvement performed on real-valued belief space runs in exponential time, an iteration of policy improvement on a finite set of points runs in quadratic time. In *Perseus*, belief point selection can be done in a number of ways. We select belief points by sampling *trajectories* through belief space. That is, we record visited belief points in random trajectories obtained by choosing actions at random. The recorded belief points form a subset of the reachable belief states

Point-based value iteration algorithms (including *Perseus*) have been shown to outperform grid-based methods on a variety of problems (Pineau et al., 2003), (Spaan and Vlassis, 2004). Whereas grid-based methods maintain only a value at each belief point, *Perseus* maintains both a value and a gradient at each belief point. This property enables *Perseus* to generalize to unexplored belief points better than grid-based approximations.

In this work we also extend *Perseus* to perform policy improvement on problems with observations composed of discrete and continuous components. In *Perseus*, point-based backups are done by considering all observations. As described in (Hoey and Poupart, 2005), we handle continuous observations by sampling a subset of them. As the number of sampled observations increases, the quality of the solution improves.

[7] Technically an $|S|$-dimensional simplex.

3 A factored POMDP architecture for Spoken Dialog Systems

3.1 Method

Our proposal is to formulate the Dialog Manager of a Spoken Dialog System as a POMDP as follows.

First, we break the POMDP state variable $s \in S$ into three components: (1) the user's goal, $s_u \in S_u$; (2) the user's action, $a_u \in A_u$; and (3) the state of the dialog, $s_d \in S_d$. The POMDP state s is given by the tuple $\{s_u, a_u, s_d\}$. We note that, from the machine's perspective, all of these components are unobservable.

The state variables are described as follows:

- First, the user's goal, s_u, gives the current goal or intention of the user. Examples of a complete user goal include a travel itinerary, a request for information about a calendar, or a product the caller would like to purchase. At each time step, we wish to form a distribution over the set of possible user goals, S_u. For that reason, we expect that the propositional content of a user's goal (e.g., the places the user wants to travel to, or the name of the product a caller wants to buy) need to be included in the set S_u.
- Second, the user's action, a_u, gives the most recent user's *actual* action. Examples of user actions include specifying a place the user would like to travel to, providing a date to query the user's calendar, and indicating that the caller would like to make a purchase. Other actions include responding to yes/no questions or a "null" response indicating the caller made no response.
- Finally, the state of the dialog s_d indicates any relevant dialogue state information from the perspective of the user. For example, s_d might indicate which slots have been grounded, ungrounded, or not raised. s_d enables the POMDP to make decisions about the appropriateness of behaviours in a dialog – for example, it would be odd to confirm a piece of information which had not previously been mentioned.

Note that we do not include a component for *speech recognition confidence* associated with a particular user goal. The concept of confidence is naturally captured by the distribution of probability mass assigned to a particular user goal in the belief state.[8]

The POMDP action $a_m \in A_m$ is the action the machine takes in the dialog. For example, machine actions might include greeting the user, asking the user where they want to go "to", or confirming that the user wants to leave from a specific place. As above, we expect that this action will include propositional content.

[8] In Section 6 we show a method for incorporating ASR confidence directly into the belief state.

The POMDP observation o is drawn from the same set as a_u, i.e., $o \in A_u$. That is, the observations available to the POMDP are equal to the user actions the POMDP can represent in its state space. Note that at each time step the POMDP receives a single observation, but maintains a distribution over all possible user actions.

We decompose the POMDP transition function as follows:

$$p(s' \mid s, a_m) = p(s'_u, s'_d, a'_u \mid s_u, s_d, a_u, a_m) \tag{4}$$

$$= p(s'_u \mid s_u, s_d, a_u, a_m) p(a'_u \mid s'_u, s_u, s_d, a_u, a_m) p(s'_d \mid a'_u, s'_u, s_u, s_d, a_u, a_m) . \tag{5}$$

We then assume conditional independence as follows. The first term – which we call the *user goal model* – indicates how the user's goal changes (or does not change) at each time step. We assume the user's goal at a time step depends only on the previous goal and the machine's action:

$$p(s'_u \mid s_u, s_d, a_u, a_m) = p(s'_u \mid s_u, a_m) \tag{6}$$

The second term – which we call the *user action model* – indicates what actions the user is likely to take at each time step. We assume the user's action depends on their (current) goal and the preceding machine action:

$$p(a'_u \mid s'_u, s_u, s_d, a_u, a_m) = p(a'_u \mid s'_u, a_m) . \tag{7}$$

The third term – which we call the *dialog model* – indicates how the user and machine's actions affect the state of the conversation. We assume the current state of the dialog depends on the previous state of the dialog, the user's action, and the machine's action:

$$p(s'_d \mid a'_u, s'_u, s_u, s_d, a_u, a_m) = p(s'_d \mid a'_u, s_d, a_m) . \tag{8}$$

In sum, our transition function is given by:

$$p(s' \mid s, a_m) = p(s'_u \mid s_u, a_m) p(a'_u \mid s'_u, a_m) p(s'_d \mid a'_u, s_d, a_m) . \tag{9}$$

The two user models could be estimated from a corpus of labelled interactions. For example, we could estimate conditional distributions over user dialog acts given a machine dialog act and a user goal. To appropriately cover all of the conditions, the corpus would need to include variability in the strategy employed by the machine – for example, using a Wizard-of-Oz framework with a simulated ASR channel (Stuttle et al., 2004). The dialog model could either be estimated from data, handcrafted, or replaced by a deterministic function representing a dialog controller's information state update rules as in for example (Larsson and Traum, 2000).

The observation function is given by:

$$p(o' \mid s', a_m) = p(o' \mid s'_u, s'_d, a'_u, a_m) . \tag{10}$$

The observation function accounts for the corruption introduced by the speech recognition engine, so we assume the observation depends only on the action taken by the user:[9]

$$p(o' \mid s'_u, s'_d, a'_u, a_m) = p(o' \mid a'_u) = p(o \mid a_u). \tag{11}$$

The observation function can be estimated from a corpus or derived analytically using a phonetic confusion matrix, language model, etc. The observation can be discrete (i.e., a recognition hypothesis), or a mixture of discrete and continuous (i.e., a recognition hypothesis and a confidence score).

The reward function is not specified explicitly in this proposal since it depends on the design objectives of the target system. We note that the reward measure could contain incentives for dialog speed (by using a per-turn penalty) and successful task completion (through rewards conditioned on the user's goal). Weights between these could be learned through systems like PARADISE (Walker et al., 2000). Also, capturing successful task completion is straightforward since the state space explicitly contains the user's goal.

The reward function may also penalize inappropriate actions. We note that the reward function is a natural place to specify action appropriateness through rewards conditioned on the dialog state, s_d. For example, the reward can specify a penalty for confirming an item which has not been discussed yet (this is discussed further in section 3.2).

shows an influence diagram depiction of our proposal.

Finally, given the definitions above, we update the belief state at each time step by:

$$\begin{aligned} b'(s') &= p(s' \mid o', a_m, b) \\ &= \frac{p(o' \mid s', a_m, b)\, p(s' \mid a_m, b)}{p(o' \mid a_m, b)} \\ &= \frac{p(o' \mid s', a_m) \sum_{s \in S} p(s' \mid a_m, b, s) P(s \mid a_m, b)}{p(o' \mid a_m, b)} \\ &= \frac{p(o' \mid s', a_m) \sum_{s \in S} p(s' \mid a_m, s) b(s)}{p(o' \mid a_m, b)}. \end{aligned} \tag{12}$$

The numerator consists of the observation function, transition matrix, and current belief state. The denominator is independent of s', and can be regarded as a normalization factor, hence:

$$b'(s') = k \cdot p(o' \mid s', a_m) \sum_{s \in S} p(s' \mid a_m, s) b(s). \tag{13}$$

Substituting equation (9) and (11) into (13) and simplifying, we can write:

[9] This implicitly assumes that the same recognition grammar is always used. The model could be readily extended to enable a system "action" which activates a particular grammar

$$b'(s'_u, s'_d, a'_u) = k \cdot p(o' | a'_u) p(a'_u | s'_u, a_m) \sum_{s_u \in S_u} p(s'_u | s_u, a_m) \sum_{s_d \in S_d} p(s'_d | a'_u, s_d, a_m) \sum_{a_u \in A_u} b(s_u, s_d, a_u) \quad (14)$$

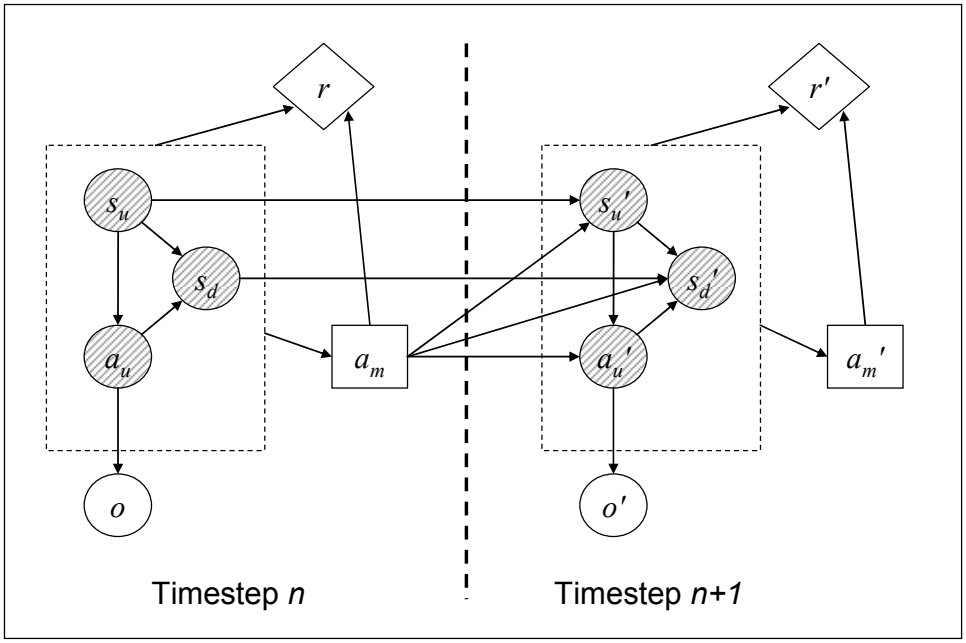

Figure 4: Influence diagram representation of proposed POMDP for Dialog Management. Symbols are as defined in and . The dotted box indicates the composite state *s* is comprised of three components, s_u, s_d, and a_u – see text for a complete definition of variables. As above, note that a_s depends on the composite belief state *b(s)* – the distribution over all states – and not the true current (unobservable) state.

3.2 Discussion

This method differs from MDP approaches in three respects.

First, in an effort to reduce the size of the state space, MDPs typically do not include propositional content in the state and action set provided to the dialog manager. Since the POMDP is maintaining a well-formed distribution over user goals (and using this information to inform action selection), our proposal relies on including propositional content in the state space and action set.

Second, in MDP schemes, the state of the dialog (i.e., grounding information) is typically included in the state variable to help measure the certainty associated with the hypothesized user goal. In the POMDP framework, certainty in a particular user goal is naturally reflected by the distribution of probability mass in the belief state.

Finally, in an MDP, inappropriate actions can be limited by specifying a set of "valid" actions for each state. In a POMDP, the belief state includes most or all states, so it is not practical to exclude inappropriate machine actions on a per-state basis as in an MDP. "Appropriateness" is instead conveyed through the reward measure.

Many machine learning approaches to dialog management have incorporated a simulated user, and used interaction-based (on-line) learning for policy improvement – for example, (Scheffler and Young, 2002) and (Pietquin, 2004). In contrast, this method *embeds* an

explicit model of the user into the model, along with an explicit transition and reward function to enable the use of model-based (off-line) learning. One advantage to this approach is that algorithms for model-based learning typically have well-defined bounds on convergence. While interaction-based (on-line) learning algorithms can be shown to converge in the limit, convergence bounds are not easy to determine and in practice it can be difficult to know when to "stop" learning.[10]

As stated, the proposed model assumes "flat" listings of state space components, observations, and actions. However, most spoken dialog systems make use of hierarchical structures such as *flight(to(London),from(Boston))*. A flat listing is clearly capable of expressing any finite list; however, because our proposal includes no method to exploit the redundancy and structure of more complex representation structures, it scales very poorly. The flat listing approach implies that the size of the state space, action set, and observation set are exponential with respect to the number of concepts in the model and this is a further issue to be addressed in future work.

4 Testbed spoken dialog system

To test the ideas in our proposal, we created a testbed dialog management POMDP in the travel domain. In the testbed problem, the user is trying to buy a ticket to travel from one city to another city. The machine asks a series of questions, and then "submits" the ticket purchase request. The machine may also choose to "fail". In the testbed problem, there are three cities, *{a,b,c}*.

The machine has the following 16 actions available:

- *greet* – greet the caller and ask "How can I help you?"
- *ask-from/ask-to* – ask where the caller wants to go from/to
- *conf-to-x/conf-from-x* – confirm that the caller wants to go to/from city *x* where $x \in \{a,b,c\}$.
- *submit-x-y* – place an order for a ticket from city *x* to city *y*, $x,y \in \{a,b,c\}, x \neq y$
- *fail* – give up the current dialogue

As above, the state space is given by the tuple $\{s_u, a_u, s_d\}$. The user's goal $s_u \in S_u$ specifies the user's desired itinerary. There are a total of 6 user goals, given by:

$$s_u \in (x,y); \quad x,y \in \{a,b,c\}, x \neq y. \tag{15}$$

The dialog state contains three components. Two of these indicate whether each item is either not specified *(n),* unconfirmed *(u)*, or confirmed *(c)*. We define "not specified" as meaning that a user has not referenced this piece of information, "unconfirmed" as meaning "referenced once" by the user, and "confirmed" as meaning "referenced more than once" by the user. A third component, *z,* specifies whether the current turn is the first turn (1) or not (0). There are a total of 18 dialog states, given by:

[10] On the other had, on-line learning allows systems to adapt during use. This can be an important advantage for real systems and we intended to pursue this in further work.

$$s_d \in (x_d, y_d, z); \quad x_d, y_d \in \{n,u,c\}, z \in \{1,0\} \tag{16}$$

The user's action $a_u \in A_u$ is drawn from the following set of 18 actions:

- *x* – The user expressed "city *x*" where $x \in \{a,b,c\}$
- *from-x* – The user expressed "I want to leave *from* city x" where $x \in \{a,b,c\}$
- *to-x* – The user expressed "I want to go *to* city *x*" where $x \in \{a,b,c\}$
- *from-x-to-y* – The user expressed "I want to go *from* city *x to* city *y*" where $x, y \in \{a,b,c\}, x \neq y$
- *yes, no* – The user expressed yes or no
- *null* – The user did not respond

These state components yield a total of 1944 states. Finally, we added one additional, absorbing state called *end-state*. When (and only when) the machine takes the *fail* action or a *submit-x-y* action, control transitions to *end-state*.

Observations are drawn from the set of 18 user actions; i.e., $o \in A_u$.

In the testbed problem the user has a fixed goal. The initial (prior) probability of the user's goal is distributed uniformly:

$$p(s_{u0}) = \frac{1}{|S_u|}. \tag{17}$$

We define the *user goal model* as:

$$p(s'_u \mid s_u, a_m) = p(s'_u \mid s_u) = \begin{cases} 1 \; if \; s'_u = s_u \\ 0 \; otherwise \end{cases}. \tag{18}$$

We define the *user action model* to include a variable set of responses:[11]

- The user responds to *ask-to/from* with *x, to/from-x,* or *from-x-to-y.*
- The user responds to *greet* with *to-y, from-x,* or *from-x-to-y.*
- The user responds to *confirm-to/from-x* with *yes/no, x,* or *to/from-x.*
- At any point the user might not respond (i.e., respond with *null*).

We define the *dialog model* distribution to deterministically implement a notion of "grounding" from the user's perspective – i.e., a field (either *to* or *from*) which has not been referenced by the user takes the value *n*; a field which has been referenced by the user exactly once takes the value *u*; and a field which has been referenced by the user more than once takes the value *c*.

[11] Because of space limitations, this distribution isn't shown here.

In this section we consider a discrete observation function – i.e., the observation function provides (only) a single discrete recognition hypothesis. We define the probability of making a speech recognition error to be p_{err}, and define the observation function as:

$$p(o \mid a_u) = \begin{cases} 1 - p_{err} & if\ o = a_u \\ \dfrac{p_{err}}{|A_u| - 1} & if\ o \neq a_u \end{cases} \tag{19}$$

Below we will vary p_{err} to explore the effects of speech recognition errors.

The reward measure includes the following components:

- If the machine uses the action *greet* when it is not the first turn of the dialog, the reward is -100.
- If the machine confirms a field before it has been referenced by the user, the reward is -3.
- If the machine selects the *fail* action, the reward is -5.
- If the machine selects a *submit* action and the values submitted match the user's goal, the reward is +10. Otherwise the reward is -10.
- The reward for any action taken in the absorbing *end* state is 0.
- The reward for any other action is -1.

A discount of $\gamma = 0.95$ was used for all experiments.

4.1 MDP Baseline

To evaluate our approach, we compare the POMDP method with an MDP-based dialog manager, patterned on systems in the literature (e.g., (Pietquin, 2004)). The MDP baseline consists of three elements: Portions of the POMDP, the MDP state estimator, and the MDP itself. We use portions of the POMDP as a simulation of the environment. The POMDP state is maintained, and an observation *o* is produced at each timestep. The POMDP observation is provided as input to an MDP State Estimator. Note that neither the belief state nor the (unobserved) POMDP state contributes to action selection in the MDP baseline.

The MDP State Estimator maintains its own state, $s_{SE} \in S_{SE}$. s_{SE} is factored into two components which give the *observed* user's goal and the *observed* state of the dialog.[12] In addition, S_{SE} includes a special *end* state, for a total of 39 states. The MDP State Estimator is updated using a deterministic function:

$$s'_{SE} = f_{S_{SE}}(s_{SE}, o', a_{MDP}) \tag{20}$$

[12] The number of state estimator states is reduced somewhat by eliminating unreachable state – for example, if no value has been observed for the *to* slot, then the *to* slot can only be *not-known* – it cannot be *unconfirmed* or *confirmed.*

$f_{S_{SE}}$ encodes dialogue heuristics, such as how to interpret evidence which is inconsistent with what has been observed so far. As is common in the literature, this function was handcrafted. For example, if the MDP State Estimator receives inconsistent evidence for a particular field, the field is reset to "*observed once"*. If the caller is asked to confirm a field and the observed response is *"no"*, then we set that field to *unknown*.

The actual MDP state, s_{MDP}, is a deterministic function of s_{SE}:

$$s_{MDP} = f_{S_{MDP}}(s_{SE}) \tag{21}$$

The MDP state, $s_{MDP} \in S_{MDP}$ includes only the *observed* state of the dialog. S_{MDP} also includes a special *end* state, for a total of 11 states.

The MDP actions are given by $a_{MDP} \in A_{MDP}$. A_{MDP} includes 7 actions:

- *greet* – greet the caller and ask "How can I help you?"
- *ask-from/ask-to* – ask where the caller wants to go from/to
- *conf-to/conf-from* – confirm the values maintained by the state estimator in the *from* or *to* field.
- *submit* – submits the values maintained by the state estimator in the *from* and *to* fields. If these values are not present, the state estimator will guess.
- *fail* – give up the current dialogue

Because the MDP consists of only the observed state of the dialog (and not the observed user goal), it cannot take actions drawn from A_m. Another deterministic function maps a_{MDP} to a_m:

$$a_m = f_{A_{MDP}}(a_{MDP}, s_{SE}) \tag{22}$$

As with $f_{S_{SE}}$, $f_{A_{MDP}}$ encodes dialog heuristics. For example, if the MDP takes the action to *submit* but no value for the *from* slot has yet been observed, $f_{A_{MDP}}$ must make a decision about which value for the *from* slot to use. This function was also handcrafted. An influence diagram showing the MDP baseline is given in .

Because the MDP learns through experience with a simulated environment which maintains its own hidden state, we cannot easily estimate the transition function nor reward function of the MDP. Thus we selected an on-line reinforcement learning technique, Watkins Q-learning, to train the MDP baseline. A variety of parameters were explored, and the best-performing parameter set was selected as follows:

- Initial *Q* values were set to 0
- Exploration was performed with the epsilon-greedy method, and $\varepsilon = 0.2$
- The learning rate α was set to *1/k*, where *k* is the number of visits to the *Q(s,a)* being updated

To evaluate the resulting MDP policy, 10,000 dialogs were simulated using the learned policy.

For comparison, we also evaluated a simple POMDP optimization technique called the Q-MDP approximation which assumes that the uncertainty in the current state will disappear one time step in the future. We first solve the underlying MDP using value iteration for its optimal *action-value function,* $Q^*_{MDP}(s,a)$, and then select actions according to:

$$\pi^*_{QMDP}(b) = \arg\max_a \sum_{s \in S} b(s) Q^*_{MDP}(s,a) \tag{23}$$

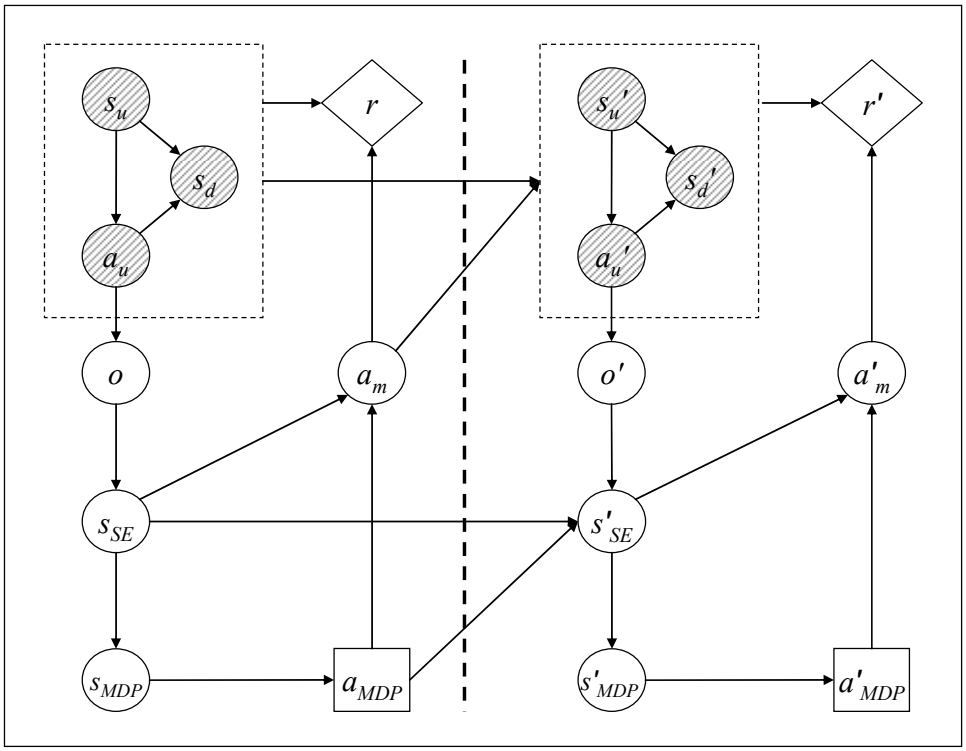

Figure 5: Influence diagram showing MDP baseline, which is modelled on systems from the literature. The MDP State Estimator s_{SE} maintains its own state, which includes the *observed* propositional content and confirmation status for each recognized slot. s_{SE} is updated using a hand-crafted function which takes the observation and system action as inputs. The MDP state s_{MDP} is a function of s_{SE} which factors out the semantic content, leaving only confirmation status.

4.2 Results & discussion

Values of p_{err} were explored ranging from 0.00 to 0.65 at intervals of 0.05.

In early trials, it was found that the POMDP policy was quite sensitive to changes in the reward function. For example, in one early trial, the reward function provided incentives for successful and efficient completion, but no rewards were specified which related to action appropriateness. Surprisingly, the POMDP policy would confirm a field *before* asking for it. Upon investigation, we found this behavior was optimal given the relatively higher accuracy of confirmation questions and the small domain, which was enabling the policy to use elimination to identify a field. This finding is similar in nature to early trials

in (Pietquins, 2004). The reward function was adjusted to the reward function presented above, which penalizes confirming a field before a user has provided a value for it.[13]

shows the expected return after each iteration ("back-up") of the solution algorithm for $p_{err} = 0.30$ and various number of belief points. We found that 500 belief points and 30 iterations attained asymptotic performance for all values of p_{error}. The progress of the value functions for these solutions is shown in .

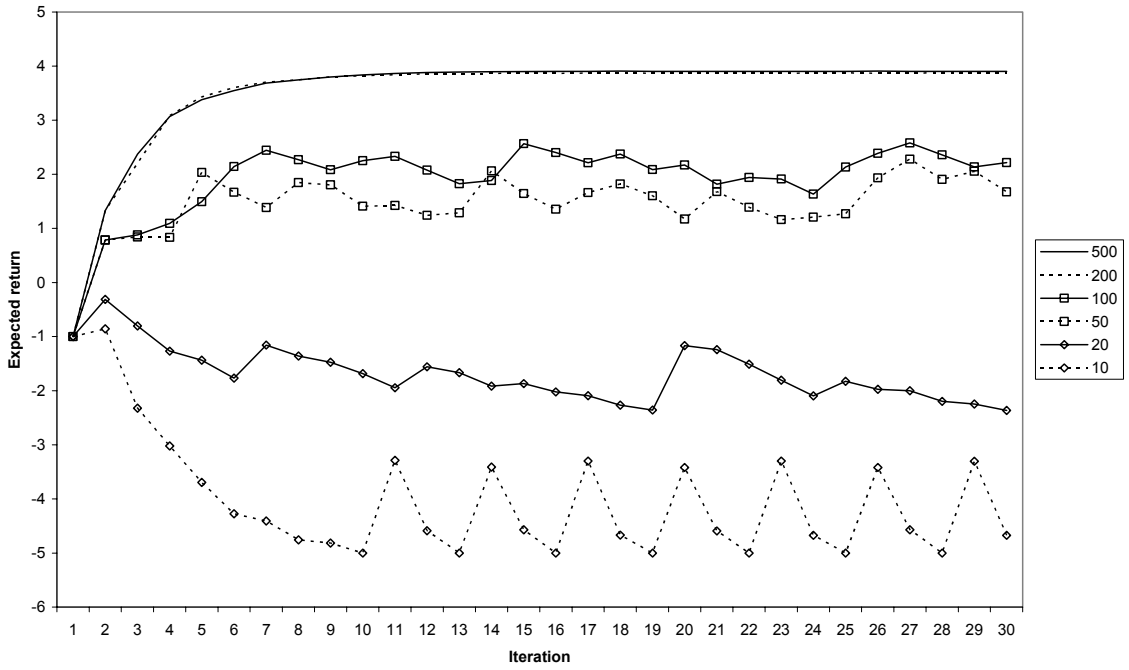

Figure 6: Expected return vs. optimization iteration for p_{err} = 0.30, no confidence score information, and various numbers of belief points.

shows expected return for the POMDP policy, and the average return for the MDP and Q-MDP solutions vs. p_{err}. The error bars show the 95% confidence interval for return assuming a normal distribution. Note that return decreases consistently as p_{err} increases for all solution methods, but the POMDP policy attains the largest return of the solutions at all values of p_{err}. Further, the performance gain of the POMDP policy solution over the other solutions increases as p_{err} increases. From this result we conclude that the POMDP method copes with higher speech recognition error rates better than the MDP or Q-MDP approaches.

In practice, the error rate of a spoken dialog system varies considerably from user to user. Thus we were interested to see how a POMDP policy performs at a value of p_{err} for which it was not designed. shows average return for three POMDP policies when executed using a different value for p_{err}. From this we see that the POMDP policies are

[13] One consequence of this decision is that POMDP policies will sometimes ask for a field's value several times rather than taking a confirmation action to build certainty in a user's belief. For an example, see .

not "brittle" – i.e., they do not fail catastrophically as the actual value of p_{err} deviates from that used in training.

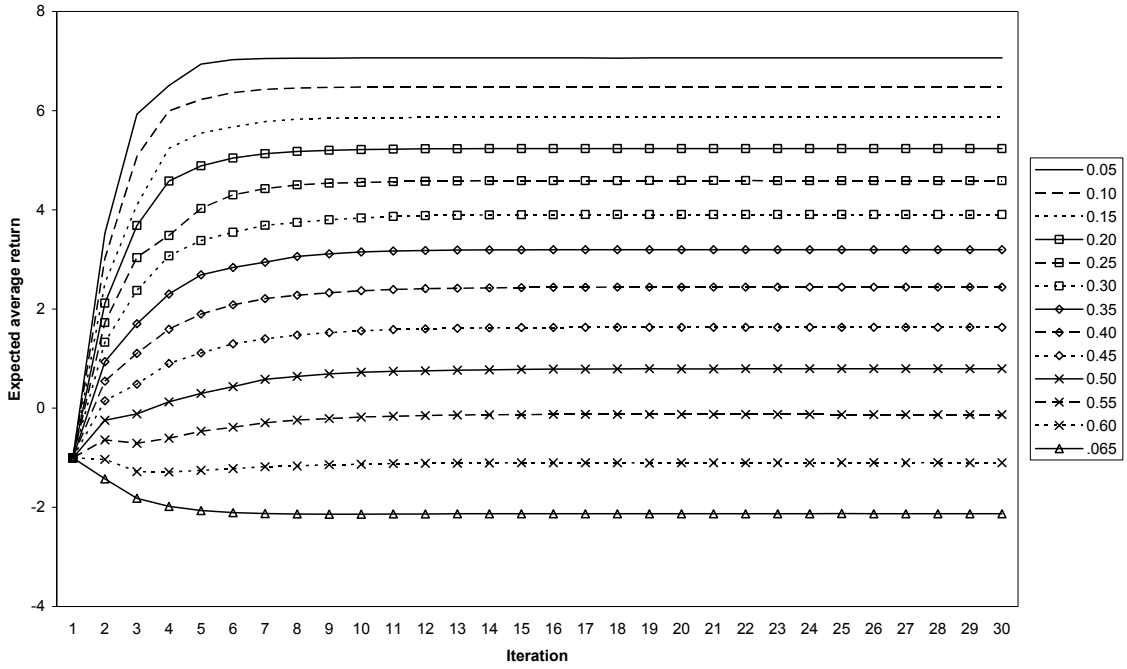

Figure 7: Expected return vs. optimization iteration for p_{err} = 0.05 ... 0.65. Solutions computed using 500 belief points and no confidence score information.

Finally, shows an example conversation with two different machines. Note how both examples are able to cope with a speech recognition error, and that the conversation at the higher value of p_{err} uses an additional conversational step to increase its certainty in the user's goal.

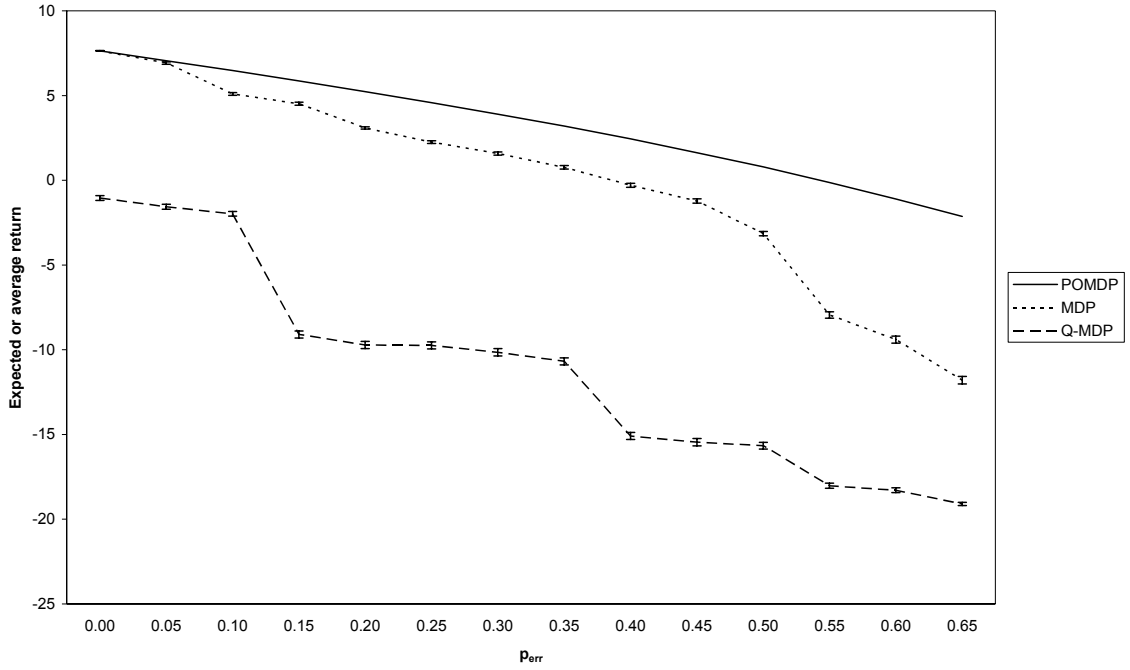

Figure 8: Expected or Average return of POMDP policies with 500 belief points, MDP, and Q-MDP approximations vs. p_{err}. No confidence score information was present. POMDP dataset shows (exact) expected return; others show average return over 10,000 simulated dialogs with error bars showing 95% confidence interval for exact expected return.

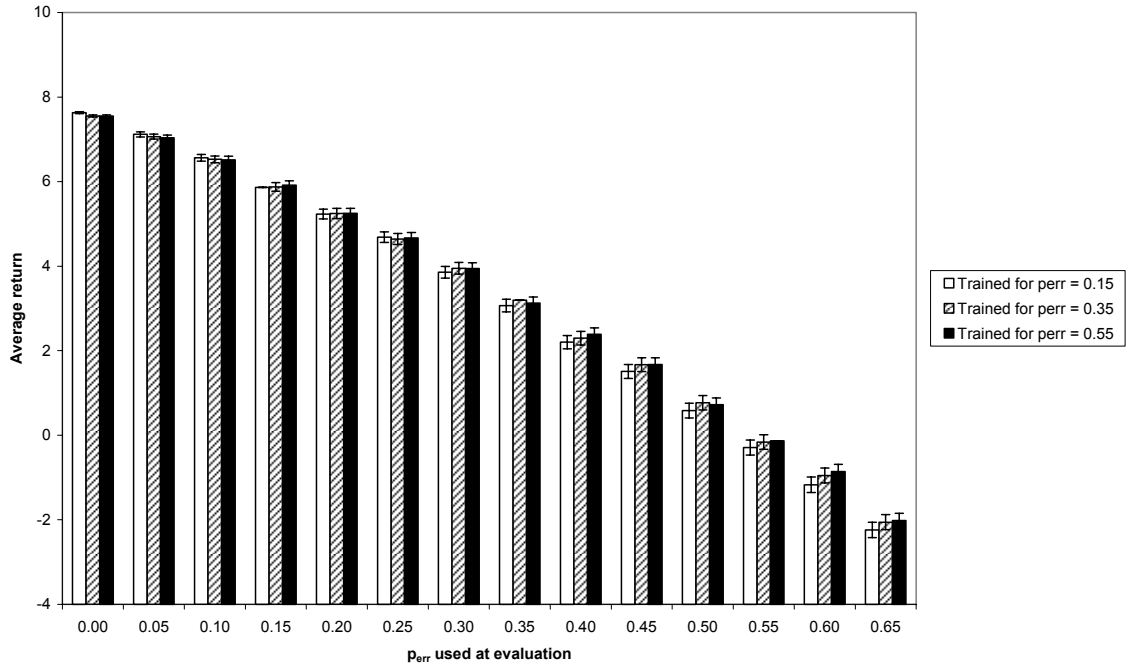

Figure 9: Performance of POMDP policies trained at one speech recognition error rate ("Trained for p_{err}") and evaluated at another ("p_{err} used at evaluation"). 500 belief points used for all policies. No confidence score information was present. Averages are over 10,000 simulated dialogs; error bars show 95% confidence interval for exact expected return.

p_{err} = 0.10	p_{err} = 0.30
Machine: How can I help you?	**Machine**: How can I help you?
User: *I'd like to go to B* (**Misrecognized as** *I'd like to go from B*)	***User***: *I'd like to go to B* (**Misrecognized as** *I'd like to go from B*)
Machine: Where would you like to go to?	**Machine**: Where would you like to go to?
User: *To B*	***User***: *To B*
Machine: And where are you leaving from?	**Machine**: And where are you leaving from?
User: *From A*	***User***: *From A*
Machine: Ok, I'm printing a ticket from A to B.	**Machine**: Where are you leaving from again?
	User: *From A*
	Machine: Ok, I'm printing a ticket from A to B.

Table 1: Example conversations with two different POMDP policies. Speech recognition is accurate except where noted. No confidence score information is present. Actual machine actions are at the abstract level; the verbiage shown here is illustrative and assumes basic generation rules have been applied. Note the extra turn for the policy developed for the p_{err} = 0.30 environment (i.e., the environment with lower speech recognition accuracy).

5 Evaluating & improving a handcrafted policy

5.1 Evaluating handcrafted policies

There are several reasons why we might want to create a handcrafted POMDP policy for dialog management: for example, we may want to compare the average return for a handcrafted policy with an automated solution, or we might want to use a handcrafted policy to bootstrap a solution technique. This section describes how a handcrafted policy can be implemented and evaluated in the POMDP environment.

Intuitively, a policy specifies what action to take in a given situation. In the previous section, we relied on the representation of a POMDP policy produced by our choice of solution method – i.e., a value function, represented as a set of N vectors each of dimensionality $|S|$ in belief space:

$$\upsilon_n(s), 1 \le n \le N, 1 \le s \le |S| \tag{24}$$

Each vector represents the value, at all points in belief space, of executing some "policy tree" which starts with an action $\hat{\pi}(n) \in A$ to indicate the action associated with the *nth* vector. If we assume that the policy trees have an infinite horizon, then we can express the optimal policy at any timestep as:

$$\pi(b) = \hat{\pi}\left(\arg\max_n \sum_{s=1}^{|S|} \upsilon_n(s)\, b(s) \right) \tag{25}$$

Thus the value-function method provides both a partitioning of belief space into regions corresponding to actions as well as the expected return of taking that action. Although an

optimal infinite-horizon POMDP policy may have an infinite number of such regions, we can usually find reasonable approximations with finite numbers of regions.

A second way of representing a POMDP policy is as a "policy graph" – a finite state controller consisting of *N* nodes and some number of directed arcs. Each controller node is assigned a POMDP action, and we will again write $\hat{\pi}(n)$ to indicate the action associated with the *nth* node. Each arc is labelled with a POMDP observation, such that all controller nodes have exactly one outward arc for each observation. $l(n,o)$ denotes the successor node for node *n* and observation *o*.

A policy graph is a general and common way of representing handcrafted dialog management policies. More complex handcrafted policies – for example, those created with rules – can usually be compiled into a (possibly very large) policy graph.

A policy graph does not make the expected return associated with each controller node explicit. However, as pointed out in (Hansen, 1998), we can find the expected return associated with each controller node by solving this system of linear equations in υ:

$$\upsilon_n(s) = r(s,\hat{\pi}(n)) + \gamma \sum_{s' \in S} \sum_{o \in O} p(s' \mid s,\hat{\pi}(n)) p(o \mid s',\hat{\pi}(n)) \upsilon_{l(n,o)}(s') \tag{26}$$

Solving this set of linear equations yields a set of vectors – one vector for each controller node. To find the expected value of starting the controller in node *n* and belief state *b* we compute:

$$\sum_{s=1}^{|S|} \upsilon_n(s) b(s) \tag{27}$$

To find the optimal node *n* to begin execution in given a belief state *b* we compute:

$$\arg\max_{n \in N} \sum_{s=1}^{|S|} \upsilon_n(s) b(s) \tag{28}$$

Thus, by finding the value-function vectors associated with a controller, we can make direct comparisons between the expected return of a handcrafted controller and a vector-based solution produced by many optimization techniques, including *Perseus*. Also, we can use the vectors computed from a handcrafted controller as a "seed" value function for automated improvement.

5.2 Example handcrafted policies

Three handcrafted policies were created, called *HC1*, *HC2*, and *HC3*.

All of the handcrafted policies first take the action *greet*.

HC1 takes the *ask-from* and *ask-to* actions to fill the *from* and *to* fields, performing no confirmation. If the user does not respond, or if it receives a nonsensical response, it re-tries the same action. Once it finds values for both fields, it takes the corresponding *submit-x-y* action. A logical diagram showing *HC1* is given in .[14]

[14] A logical diagram is shown for clarity: the actual controller uses the real values *a, b,* and *c,* instead of the variables *X* and *Y,* resulting in a controller with 15 states.

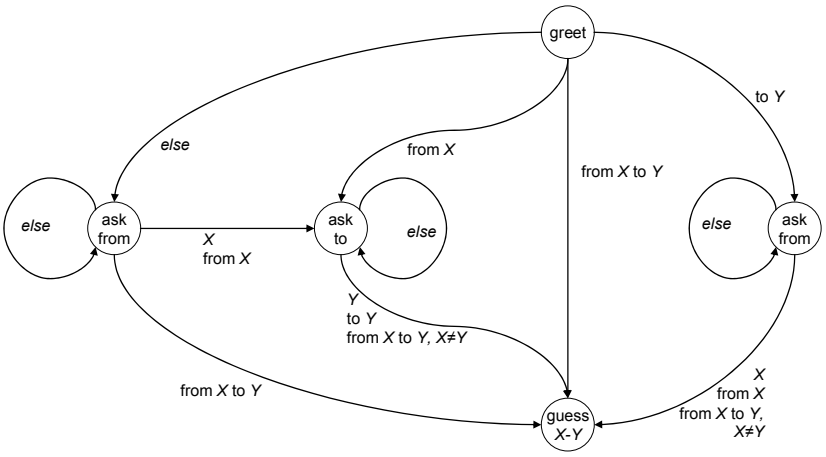

Figure 10: "*HC1*" handcrafted controller. When the controller receives inconsistent or nonsensical information, it takes the *fail* action.

HC2 is identical to *HC1* except that if it receives inconsistent or nonsensical information, it immediately takes the *fail* action. A logical diagram showing *HC2* is given in .

HC3 employs a similar strategy to *HC1* but extends *HC1* by confirming each field as it is collected. If the user responds with "*no*" to a confirmation, it re-asks the field. If the user provides inconsistent information, it treats the new information as "correct" and confirms the new information. If the user does not respond, or if the machine receives any nonsensical input, it re-tries the same action. Once it has successfully filled and confirmed both fields, it takes the corresponding *submit-x-y* action.[15]

Empirical assessment of the average return produced by the handcrafted policies showed agreement with the analytic solutions within statistical significance. shows the expected return for the handcrafted policies and the POMDP policy. The POMDP method outperforms all of the handcrafted policies for all values of p_{error}. It is interesting to note that *HC3,* which confirms all inputs, performs least well for all values of p_{error}.

5.3 Improving handcrafted policies

As noted above, it is relatively straightforward to compute the value function for a policy represented as a finite state controller. Recall this value function expresses, for a given belief state & controller node, the expected return of starting controller execution in that node.

[15] The diagram for *HC3* is not included as it is highly connected and difficult to show clearly.

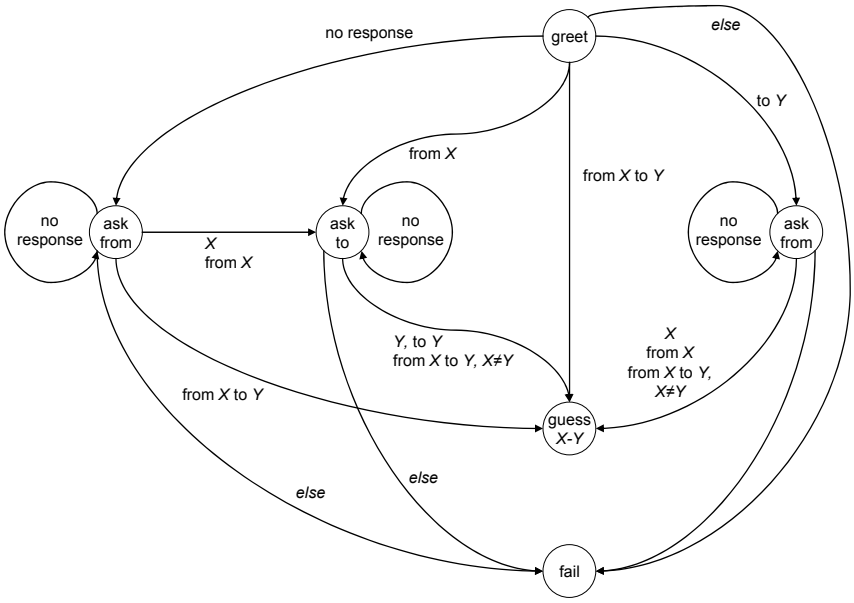

Figure 11: "*HC2*" handcrafted controller. When the controller receives inconsistent or nonsensical information, it retries the same action.

A method for improving a handcrafted policy represented as a finite state controller is as follows. First, we compute the value function of the finite state controller as described above. At the beginning of the dialog, we find the node with the highest expected return for b_0 and execute its action. Throughout the dialog, we perform belief state monitoring – i.e., we maintain the current belief state at each timestep. Then, at each timestep, rather than follow the policy specified by the finite state controller, we *re-evaluate* which node has the highest expected return for the current *b.* We then take the action specified by *that* node. Note that, in this style of execution, node transitions may occur which are not arcs in the handcrafted policy. Because the node-value function and belief state are exact, this style of execution is guaranteed to perform at least as well as the original handcrafted controller.

To test this method, we executed 10,000 dialogs for each handcrafted policy at each value of p_{err}. Results for *HC1*, *HC2*, and *HC3* are shown in , , and respectively. These plots show the *difference* between the proposed method and the expected value of executing the handcrafted policy directly to make the gain of the proposed method explicit. For reference, these plots also include the difference between the handcrafted policies executed normally and the POMDP solution, which we take to be a practical upper bound for the testbed problem. Error bars show the 95% confidence interval for the true expected return assuming normal distribution. We note that in almost all cases, the proposed method results in a significant improvement. In many cases, the proposed method is close to the POMDP solution – our assumed practical upper bound.

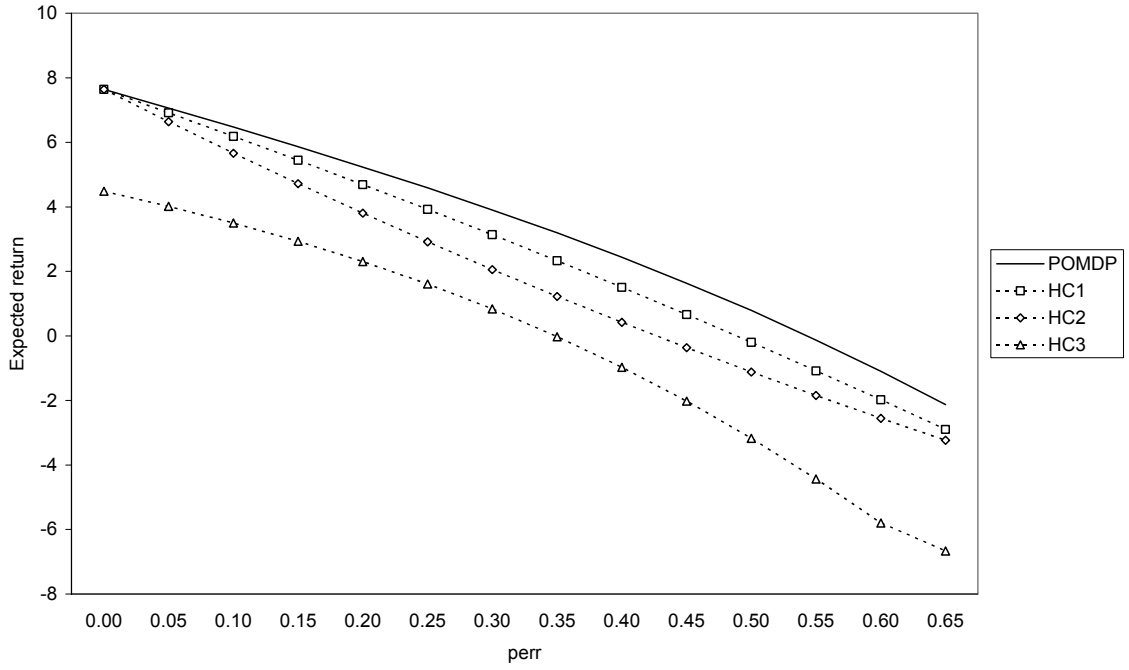

Figure 12: Expected return vs. p_{err} for POMDP policies and the 3 handcrafted policies. POMDP solution was created with 500 belief points and no confidence score information.

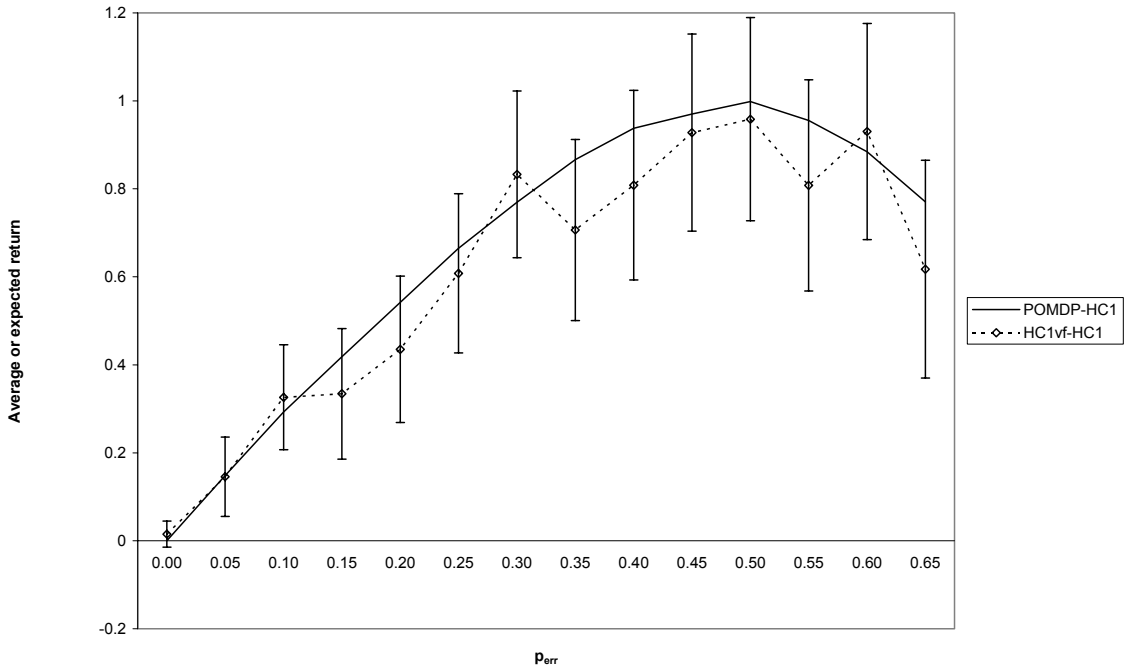

Figure 13: Gain in average/expected return for HC1 executed using belief state monitoring vs. p_{err}. The POMDP policy, which we take to be our practical upper bound, is shown for reference. Error bars show 95% confidence interval for true expected return over 10,000 dialogs.

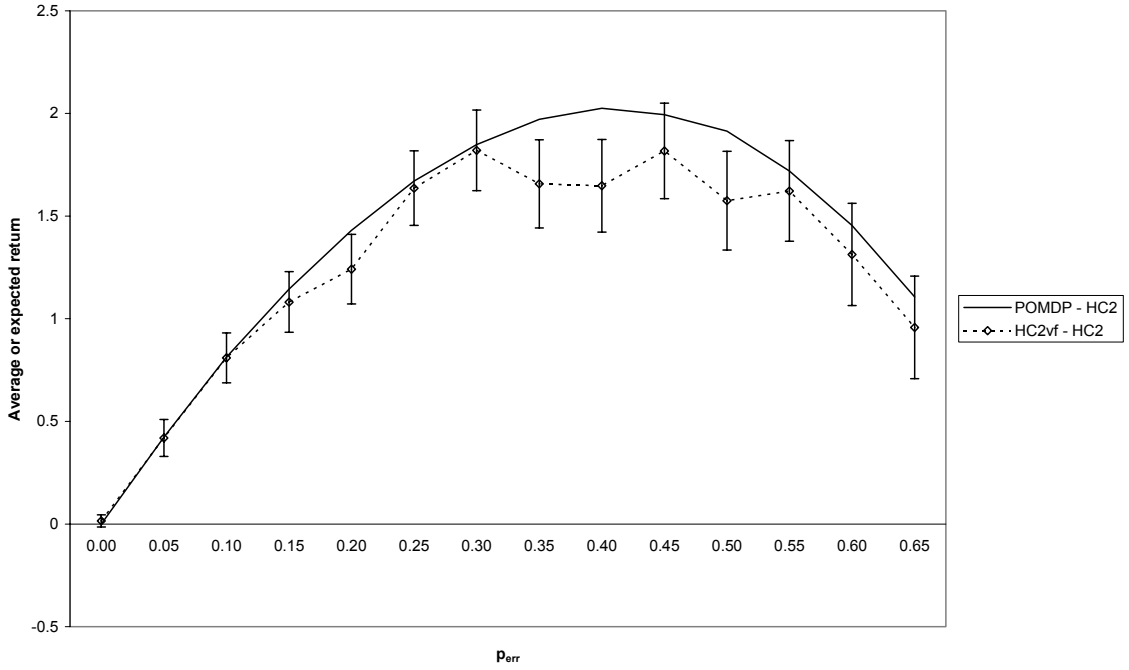

Figure 14: Gain in average/expected return for HC2 executed using belief state monitoring vs. p_{err}. The POMDP policy, which we take to be our practical upper bound, is shown for reference. Error bars show 95% confidence interval for true expected return over 10,000 dialogs.

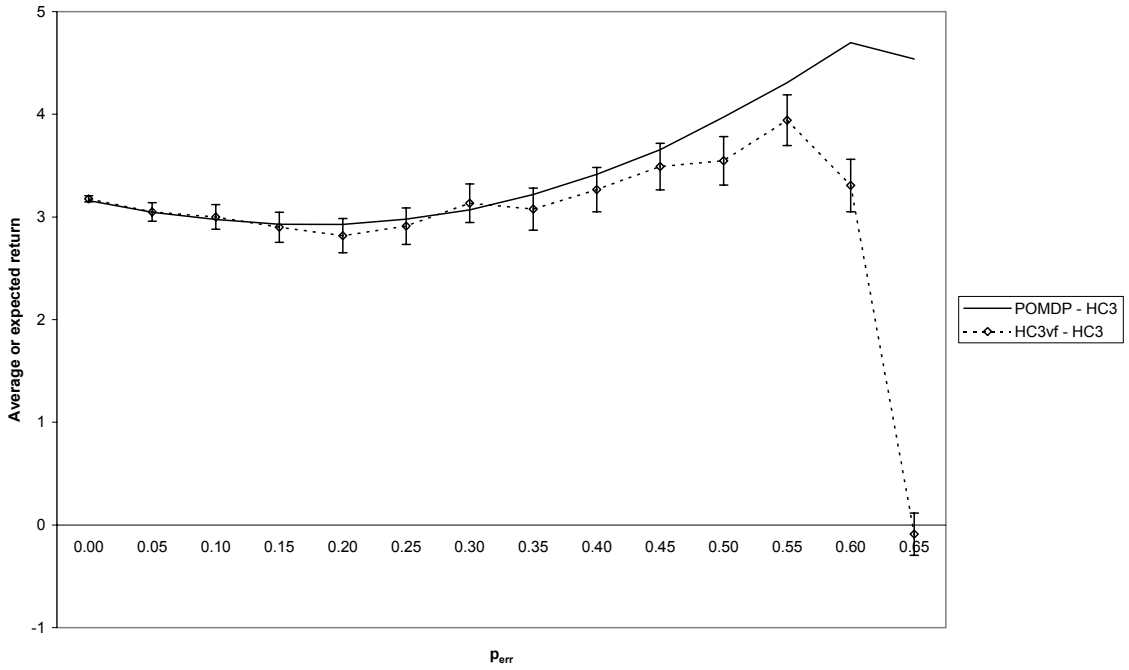

Figure 15: Gain in average/expected return for HC3 executed using belief state monitoring vs. p_{err}. The POMDP policy, which we take to be our practical upper bound, is shown for reference. Error bars show 95% confidence interval for true expected return over 10,000 dialogs.

6 Incorporating confidence score into the POMDP framework

6.1 Background

To this point we have not accounted for *confidence scores* in our model. A *confidence score c* is a real-valued random variable which accompanies the (discrete) *recognition result o*. The confidence score *c* is intended to provide a measure of the reliability of the recognition result *o*.

When building spoken dialog systems, designers typically divide confidence scores into *buckets*. For example, at design time, a system designer specifies a confidence threshold, *cThresh*. At run time, if an observed confidence score *c* is above *cThresh* then the machine accepts the recognition result as accurate, and if the confidence score *c* is below *cThresh* the machine rejects it as inaccurate. More buckets can be created so that a recognition results *o* can have, for example, high, medium, or low confidence. Designers typically choose to employ more time-consuming but deliberate confirmation strategies for low-confidence recognition results and faster but less reliable confirmation strategies for high-confidence recognition results. This approach has been used in most of the dialog management research employing MDPs. However, in practice it is not clear how to set the threshold(s) *cThresh,* and changing the thresholds requires re-optimizing the MDP policy.

Recently, researchers have suggested Decision Theoretic grounding models which incorporate confidence measures directly into distributions over user goals (Paek and Horvitz, 2003). However, to our knowledge, no algorithm has been suggested or tested for incorporating both the recognition result and confidence score into the POMDP framework.

6.2 Method

We assume the framework described above for a POMDP-based dialog manager. To this framework, we add a continuous element to the observation function, $c \in \Re$, yielding a new observation function, $p(o,c \mid a_u)$.

The belief state update function in Equation (14) becomes:

$$b'(s'_u, s'_d, a'_u) = k \cdot p(o', c' \mid a'_u) p(a'_u \mid s'_u, a_m) \sum_{s_u \in S_u} p(s'_u \mid s_u, a_m) \sum_{s_d \in S_d} p(s'_d \mid a'_u, s_d, a_m) \sum_{a_u \in A_u} b(s_u, s_d, a_u) \quad (29)$$

There will in general be insufficient data to estimate $p(o,c \mid a_u)$ directly. To form $p(o,c \mid a_u)$, a confusion matrix $p(o \mid a_u)$ can be combined with two confidence score distributions – one for correct recognitions and one for incorrect recognitions:

$$p(o,c \mid a_u) = \begin{cases} p_{correct}(c) \cdot p(o \mid a_u) & if\ o = a_u \\ p_{incorrect}(c) \cdot p(o \mid a_u) & if\ o \neq a_u \end{cases} \quad (30)$$

This is the approach we take in the example problem, below. More complex estimates of confidence score distribution which take into account the confusability between specific values of *o* are also possible. For examples, see (Pietquin, 2004).

To perform policy improvement on this POMDP we have two options. First, we can perform policy improvement using a solution method which accounts for the continuous observations, such as the modified *Perseus* method described in Section 2.3. We note that this method creates a policy which takes the expected additional information in the confidence score into account. We call this the *continuous-POMDP* solution.

Second, we note that there is benefit to using the confidence score information for belief state monitoring even if it was not used during policy optimization. In other words, we would expect an improvement in average return by using confidence scores to improve our estimate of the current belief state at runtime (as in Equation (29)) *even if* the policy followed has been created with an observation function which ignores the confidence score; i.e.,

$$p(o \mid a_u) = \int_c p(o, c \mid a_u) . \tag{31}$$

We call this the *discrete-POMDP* solution.

Stated alternatively, the *continuous-POMDP* technique uses infinitely many confidence buckets during planning and belief monitoring. In contrast, the *discrete-POMDP* technique uses no confidence information during planning, but infinitely many confidence buckets during belief monitoring. Finally, MDP methods (in the literature, and our baseline, presented below) use a handful of confidence buckets for planning, but do not perform any belief monitoring.

6.3 Evaluation

To test the method, we extended the observation function of the testbed problem as follows:

$$p(o, c \mid a_u) = \begin{cases} p_{correct}(c) \cdot (1 - p_{err}) & \text{if } o = a_u \\ p_{incorrect}(c) \cdot \dfrac{p_{err}}{|A_u| - 1} & \text{if } o \neq a_u \end{cases} \tag{32}$$

We define c on the interval [0,1] and define the probability densities $p_{correct}(c)$ and $p_{incorrect}(c)$ as the exponential probability density functions normalized to the region [0,1]; i.e., :

$$p_{correct}(c) = \begin{cases} \dfrac{a_{correct}\, e^{c \cdot a_{correct}}}{e^{a_{correct}} - 1}, & a_{correct} \neq 0 \\ 1, & a_{correct} = 0 \end{cases}$$
$$p_{incorrect}(c) = \begin{cases} \dfrac{a_{incorrect}\, e^{(1-c) a_{incorrect}}}{e^{a_{incorrect}} - 1}, & a_{incorrect} \neq 0 \\ 1, & a_{incorrect} = 0 \end{cases} \quad (33)$$

where $a_{correct}$ and $a_{incorrect}$ are constants defined on $(-\infty, \infty)$. We note that as a_x approaches positive or negative infinity, $p_x(c)$ becomes deterministic and conveys complete information; when $a_x = 0$, $p_x(c)$ is a uniform density and conveys no information. Since we expect the confidence value for correct recognition hypotheses to tend to 1, and for incorrect recognition hypotheses to tend to 0, we would expect $a_x > 0$.

We compared our proposal with the MDP technique to evaluate it empirically. As is customary with MDP-based approaches, we augmented the MDP approach described above to include *M* confidence buckets. Ideally the confidence score bucket sizes would be selected so that they maximize average return. However, it is not obvious how to perform this selection. Instead, a variety of techniques for setting confidence score threshold were explored. It was found that dividing the probability mass of the confidence score *c* evenly between buckets produced the largest average returns. That is, we define

$$cThresh_0 = 0 < cThresh_1 < \cdots < cThresh_{M-1} < cThresh_M = 1 \quad (34)$$

and then find the values of $cThresh_m$ such that:

$$\int_{cThresh_{m-1}}^{cThresh_m} p(c)\, dc = \int_{cThresh_m}^{cThresh_{m+1}} p(c)\, dc, \quad m \in 1,2,...,M-1 \quad (35)$$

where *p(c)* is the prior probability of a confidence score. We find this prior for our testbed problem as follows. We first find the distribution $p(c \mid a_u)$ as:

$$p(c \mid a_u) = \sum_{o \in A} p(c, o \mid a_u) \quad (36)$$

$$= p_{correct}(c \mid a_u)(1 - p_{err}) + p_{incorrect}(c \mid a_u)(p_{err}). \quad (37)$$

In the MDP context, we assume the confidence score buckets are formed without access to a prior $p(a_u)$. From this assumption, we find:

$$p(c) = p_{correct}(c)(1 - p_{err}) + p_{incorrect}(c)(p_{err}) \quad (38)$$

from which the values of $cThresh_m$ can be derived.

We also compared our proposal to the (discrete) Q-MDP approach, using the belief state update function given in equation (29). We also compared our proposal to handcrafted solutions executed with belief state monitoring, as described in Section 5.3.

6.4 Results & discussions

shows average returns for the continuous, discrete, and MDP methods for $a_{correct} = a_{incorrect} = a = 1$.[16] For the MDP method, we use 2 confidence buckets, and approximately 125,000 dialog turns. Note the POMDP methods outperform the MDP method, and the discrete-POMDP and continuous-POMDP solutions performed similarly.

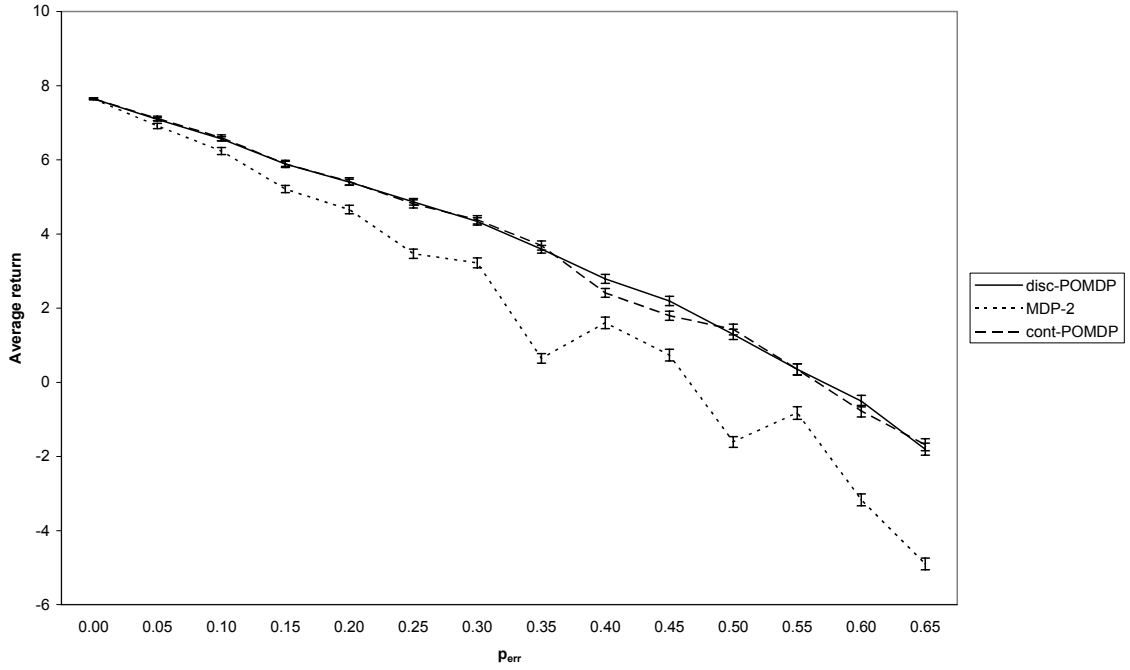

Figure 16: Average return for *continuous-POMDP*, *discrete-POMDP*, and MDP-2 methods for *a* = 1. MDP-2 has 2 confidence buckets and was trained for 125,000 dialog turns. Averages shown are over 10,000 simulated dialogs with the resulting policies; error bars show 95% confidence interval for exact expected return.

Figures 17, 18, and 19 show average returns for the POMDP and MDP methods vs. *a* for p_{err} = 0.3, 0.4, and 0.5, respectively. In these figures, we again define $a_{correct} = a_{incorrect} = a$. For the MDP method, we use 2 confidence buckets. The POMDP methods outperform the baseline MDP method consistently. Note that increasing *a* increases average return for all methods, and that the greatest improvements are for p_{err} = 0.5 – i.e., the information in the confidence score has more impact as speech recognition accuracy degrades. Again, the discrete and continuous POMDP solutions performed similarly.

[16] The Q-MDP method performed significantly worse than all other methods and is not shown.

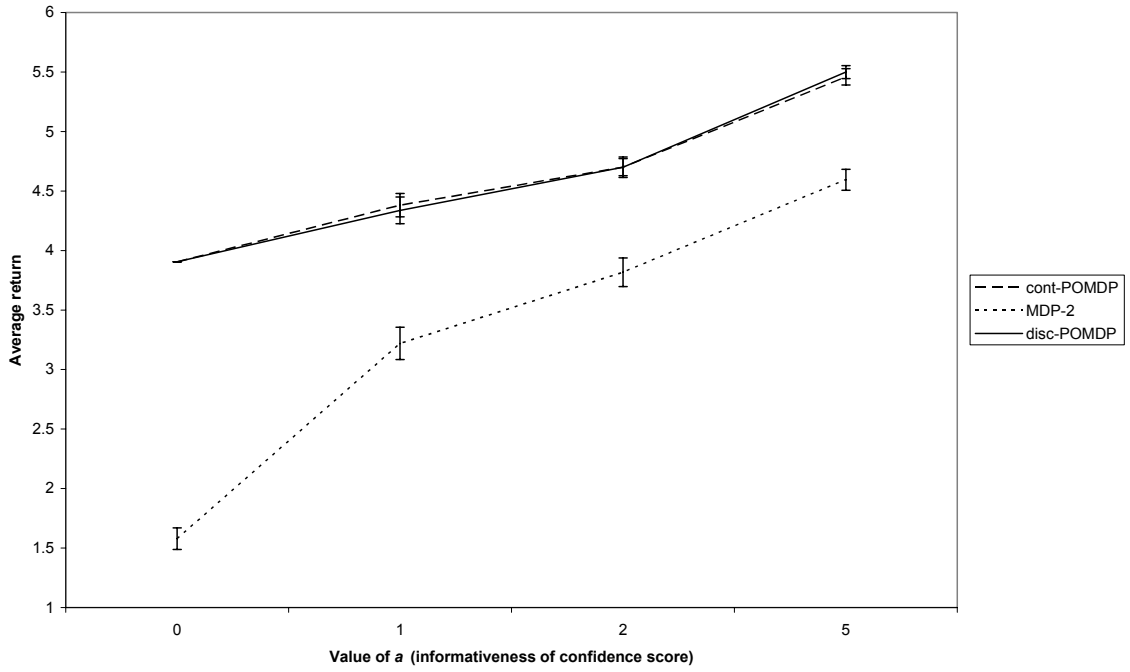

Figure 17: Average return vs. *a* (informativeness of confidence score) at p_{err} = 0.30 for *continuous-POMDP*, *discrete-POMDP*, and MDP methods. MDP-2 has 2 confidence buckets and was trained for 10,000 dialogs. Averages shown are over 10,000 simulated dialogs; error bars show 95% confidence interval for exact expected return

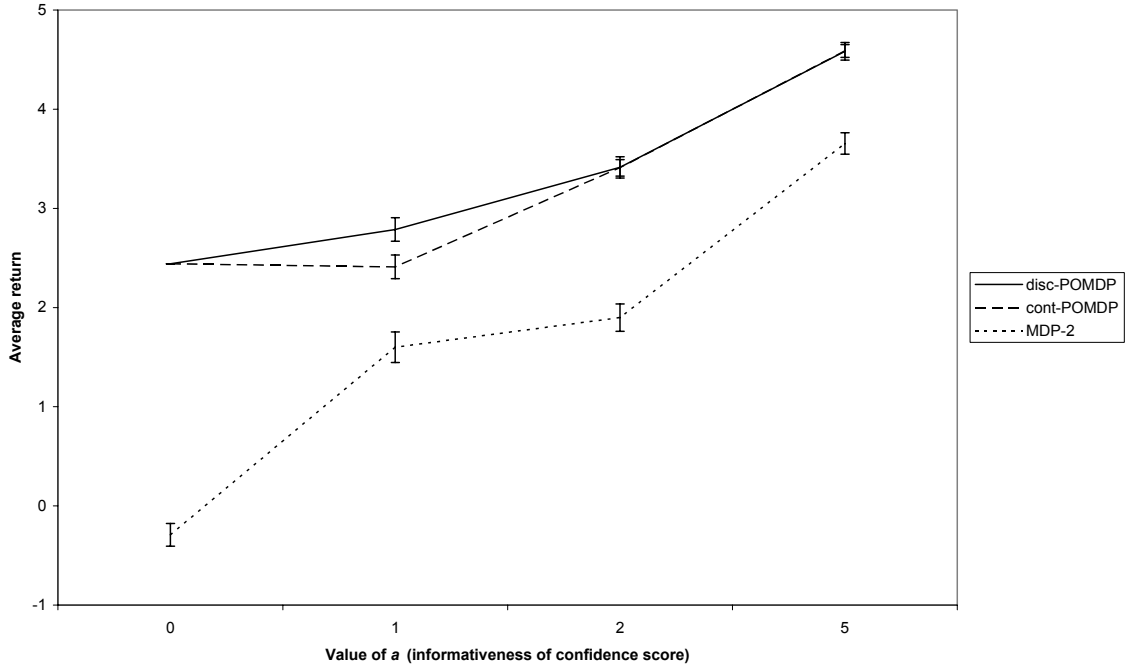

Figure 18: Average return vs. *a* (informativeness of confidence score) for p_{err} = 0.40 for *continuous-POMDP*, *discrete-POMDP,* and MDP methods. MDP-2 has 2 confidence buckets and was trained for 10,000 dialogs. Averages shown are over 10,000 simulated dialogs; error bars show 95% confidence interval for exact expected return

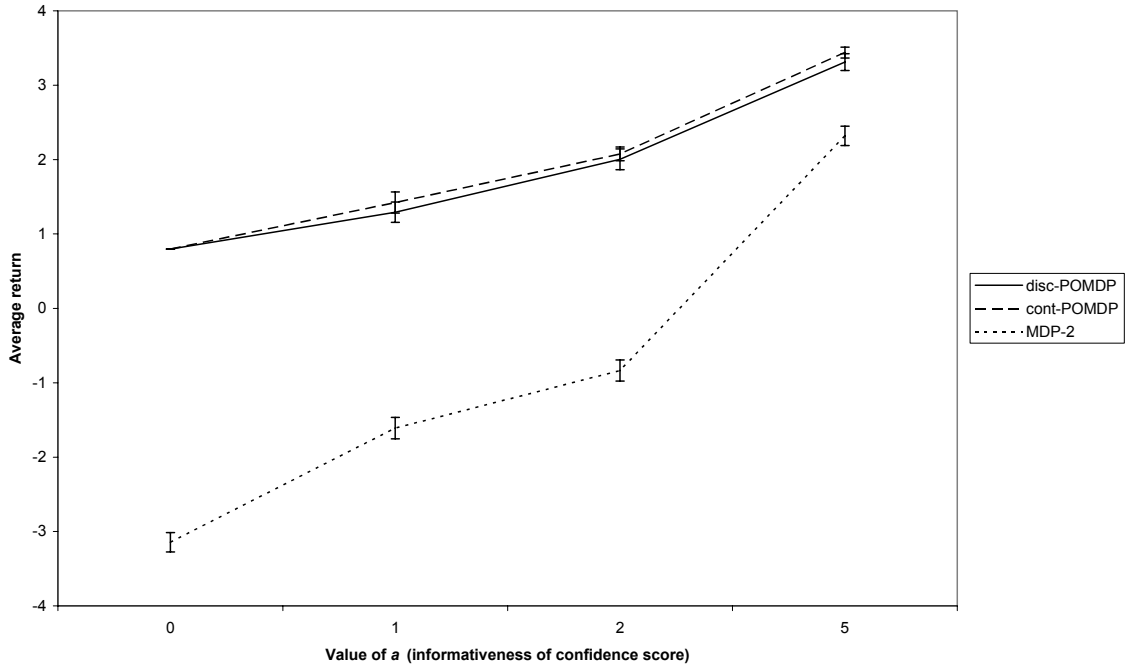

Figure 19: Average return vs. *a* (informativeness of confidence score) for p_{err} = 0.50 for *continuous-POMDP*, *discrete-POMDP,* and MDP methods. MDP-2 has 2 confidence buckets and was trained for 10,000 dialogs. Averages shown are over 10,000 simulated dialogs; error bars show 95% confidence interval for exact expected return

Figures 20, 21, and 22 show average returns for the discrete-POMDP and handcraft methods vs. *a* for p_{err} = 0.3, 0.4, and 0.5, respectively. *a* is defined as above. Increasing *a* increases average return for the handcrafted controllers executed with belief state monitoring. For highly informative confidence scores (*a=5)*, the handcrafted policies executed with belief state monitoring perform similarly to the POMDP policy.

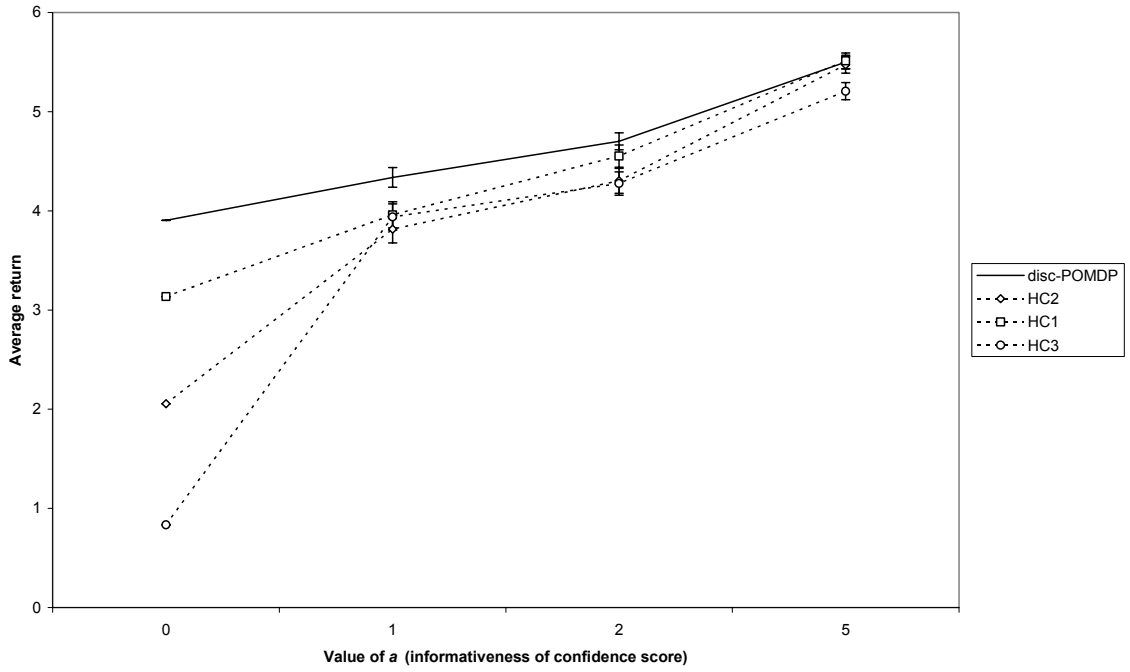

Figure 20: Average return vs. *a* (informativeness of confidence score) for p_{err} = 0.30 for *discrete-POMDP* and handcrafted policies. Policies for handcrafted controllers were executed as described in the text. Averages shown are over 10,000 simulated dialogs; error bars show 95% confidence interval for exact expected return.

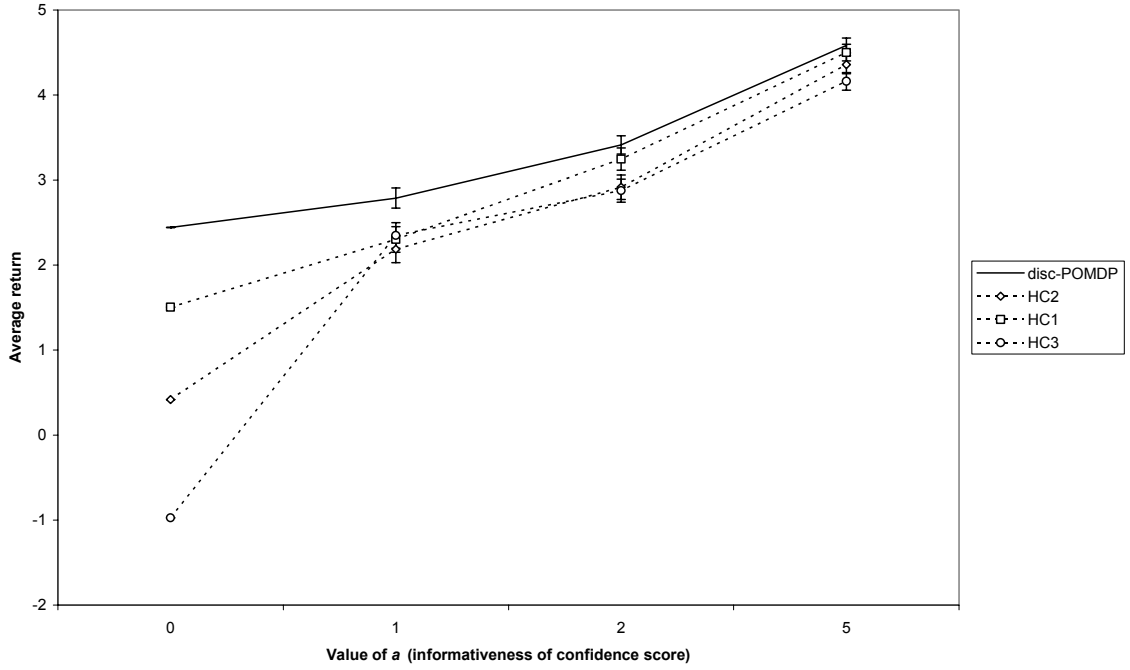

Figure 21: Average return vs. *a* (informativeness of confidence score) for p_{err} = 0.40 for *discrete-POMDP* and handcrafted policies. Policies for handcrafted controllers were executed as described in the text. Averages shown are over 10,000 simulated dialogs; error bars show 95% confidence interval for exact expected return.

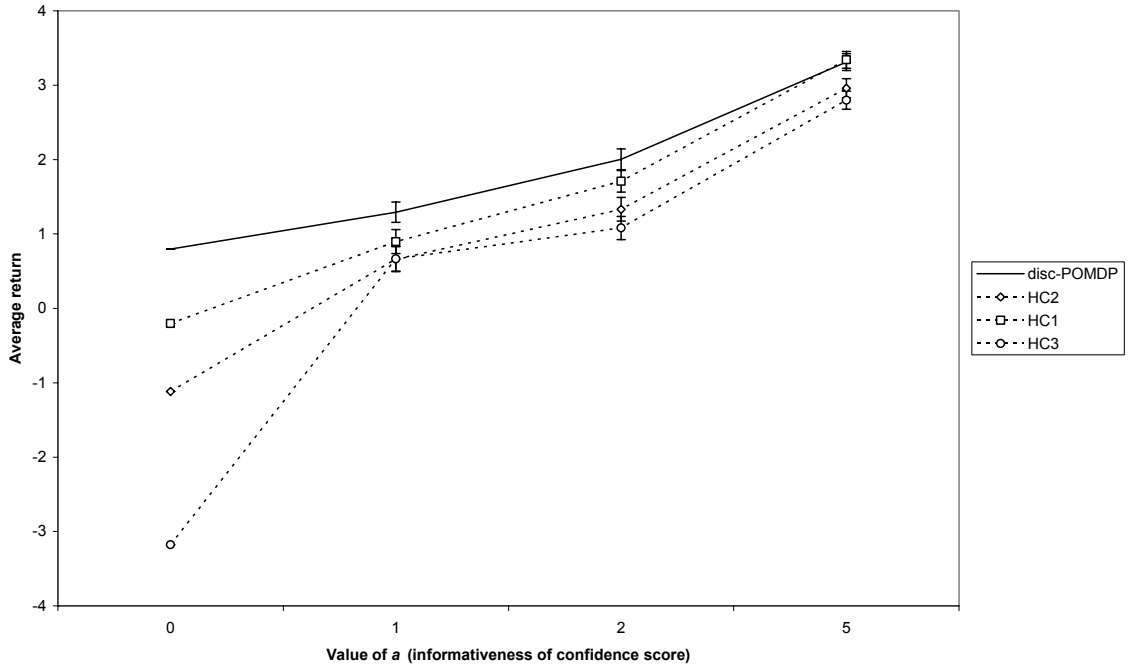

Figure 22: Average return vs. *a* (informativeness of confidence score) for p_{err} = 0.50 for *discrete-POMDP* and handcrafted policies. Policies for handcrafted controllers were executed as described in the text. Averages shown are over 10,000 simulated dialogs; error bars show 95% confidence interval for exact expected return.

7 Conclusions & future work

We have proposed a factored architecture for describing POMDPs applied to spoken dialog management, illustrated with a testbed spoken dialog system. By performing policy improvement with *Perseus*, we have shown that automated solutions are both tractable for small problems, and outperform baseline MDP methods. The factored architecture has two benefits. First, it enables incorporation of an explicit dialog model, which allows a dialog designer to add rewards for "appropriate" dialog behaviour from the standpoint of the user. Second, it facilitates estimating/specifying the system dynamics – i.e., the user models, recognition model, and dialog model – from dialog data.

We have also shown how to convert a handcrafted policy represented as a finite-state controller into a value function, providing a principled way for handcrafted policies to be compared directly with policies produced with automated solutions. Using the testbed problem, we have shown that POMDP policies produced with an optimization algorithm outperform three typical handcrafted solutions. We have also shown how these value functions can be used to improve handcrafted policies.

Finally, we have shown a method for incorporating confidence score directly into the belief state by using a continuous observation function. We've shown that this method outperforms a baseline MDP-based *confidence bucket* approach, again using a testbed problem.

There are a several interesting extensions worthy of further research. From the standpoint of system developers, policies represented as a partitioning of belief space are not easy to interpret: it would be interesting to produce a policy represented as a finite state controller – e.g., (Poupart and Boutilier, 2004). In addition, N-best recognition lists could be incorporated into the observation.

A crucial theoretical problem is how to scale the methods presented here to handle larger problems since the state, action, and observation sets grow exponentially with the number of concepts in the problem. A method of exploiting redundancy (Boutilier and Poole, 1995) or otherwise compressing the problem state space – using methods like those in (Poupart and Boutilier, 2002), (Poupart and Boutilier, 2004), or (Roy and Gordon, 2002) – is needed to apply the method to domains with 100s or 1000s of concepts. The factored nature of the architecture may be of some help here.

Despite these issues, we believe this work supports our claim that POMDPs are a theoretically elegant framework for dialog management. We believe that POMDPs are ready for practical evaluation in limited, real systems, and plan research into practicability in future work.

7 References

Boutilier, Craig and David Poole. (1996) Computing Optimal Policies for Partially Observable Decision Processes using Compact Representations, *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, Portland, OR, pp.1168--1175. ..... 12, 30

Denecke, Matthias, Kohji Dohsaka, and Mikio Nakano, (2004). Learning Dialogue Policies using State Aggregation in Reinforcement Learning. *Proceedings of International Conference on Spoken Language Processing (ICSLP)*, Jeju Korea. 6

Hansen, Eric A. (1998). Solving POMDPs by searching in policy space. In *Uncertainty in Artificial Intelligence*, Madison, Wisconsin. 23

Hoey, Jesse and Pascal Poupart (2005). Solving POMDPs with continuous or large observation spaces, To appear in *Proceedings of the Joint International Conference on Artificial Intelligence (IJCAI)*, Edinburgh, Scotland. 13

Kaelbling, Leslie Pack Michael L. Littman and Anthony R. Cassandra. (1998). Planning and Acting in Partially Observable Stochastic Domains. *Artificial Intelligence*, Vol. 101. 12

Larsson, Staffan and David Traum. (2000). Information state and dialogue management in the trindi dialogue move engine toolkit. *Natural Language Engineering*, 5(3–4):323–340. 15

Levin, Ester, Roberto Pieraccini, and Wieland Eckert. (1998). Using Markov Decision Processes For Learning Dialogue Strategies. *International Conference on Acoustics, Speech and Signal Processing*, Seattle, USA. 6

Levin, Ester, Roberto Pieraccini, and Wieland Eckert. (2000). A Stochastic Model of Human-Machine Interaction for Learning Dialogue Strategies. *IEEE Transactions on Speech and Audio Processing,* Volume 8, No. 1, 11-23. 6

Levin, Esther. and Roberto Pieraccini. (1997). A Stochasitc Model of Computer-Human Interaction For Leaning Dialogue Strategies. *Eurospeech*, Rhodes, Greece. 6

Paek, Tim and Eric Horvitz (2003). On the Utility of Decision-Theoretic Hidden Subdialog. In *Proceedings of International Speech Communication Association (ISCA) Workshop on Error Handling in Spoken Dialogue Systems*, Chateaux d'Oex, Switzerland. 27

Pietquin, Olivier. (2004) *A Framework for Unsupervised Learning of Dialogue Strategies*. Ph D thesis, Faculty of Engineering, Mons, Belgium. 7, 12, 17, 20, 22, 27

Pineau, Joelle, Geoff Gordon, and Sebastian Thrun. (2003). Point-based value iteration: An anytime algorithm for POMDPs. In *International Joint Conference on Artificial Intelligence (IJCAI)*. 13

Poupart, Pascal and Craig Boutilier. (2002). Value-directed Compression of POMDPs, *Advances in Neural Information Processing Systems 15 (NIPS)*, Vancouver, Canada. 13, 30

Poupart, Pascal and Craig Boutilier. (2003) Bounded Finite State Controllers, *Advances in Neural Information Processing Systems 16 (NIPS)*, Vancouver, Canada. 13

Poupart, Pascal. and Craig Boutilier. (2004). VDCBPI: an Approximate Scalable Algorithm for Large Scale POMDPs. To appear in *Advances in Neural Information Processing Systems 17 (NIPS)*, Vancouver, Canada, 2004. ..... 13, 30

Roy, Nicholas, Geoffrey Gordon. (2002). Exponential Family PCA for Belief Compression in POMDPs, *Advances in Neural Information Processing Systems 15 (NIPS)*, Vancouver, Canada. 13, 31

Roy, Nicholas, Joelle Pineau and Sebastian Thrun. (2000). Spoken Dialogue Management Using Probabilistic Reasoning. *Annual meeting of the the Association for Computational Linguistics (ACL-2000).* 7, 8

Scheffler, Konrad. and Steve Young. (2002). Automatic learning of dialogue strategy using dialogue simulation and reinforcement learning. *Human Language Technology Conference (HLT-2002)*, San Diego, USA. 6, 12, 17

Singh, Satinder, Diane Litman, Michael Kearns and Marilyn Walker. (2002). Optimizing Dialogue Management with Reinforcement Leaning: Experiments with the NJFun System. *Journal of Artificial Intelligence*, Vol. 16, 105-133. 6

Spaan, Matthijs T. J. and Nikos Vlassis. *Perseus*: randomized point-based value iteration for POMDPs. Technical Report IAS-UVA-04-02, Informatics Institute, University of Amsterdam, 2004. 13

Stuttle, Matthew, Jason D. Williams, and Steve Young. (2004). A Framework for Wizard-of-Oz Experiments with a Simulated ASR-Channel. *International Conferences on Spoken Language Processing (ICSLP-2004)*, Jeju, South Korea..... 15

Walker, Marilyn A., Candace Kamm, and Diane Litman. (2000). Towards Developing General Models of Usability with PARADISE. *Natural Language Engineering*, Vol. 6, No. 3. 16

Walker, Marilyn A., Jeanne C. Fromer, and Shrikanth Narayanan. (1998). Learning Optimal Dialogue Strategies: A Case Study of a Spoken Dialogue Agent for Email. *Annual meeting of the Association for Computation Linguistics and Conference on Computational Linguists (ACL/COLING-98)*. 6

Williams, Jason D. and Steve Young. (2003). Using Wizard-of-Oz simulations to bootstrap Reinforcement-Learning-based dialog management systems. *Proceedings of the 4th SIGDIAL Workshop on Discourse and Dialogue*, Sapporo, Japan. 6

Zhang, B., Q. Cai, J. Mao, E. Chang, and B. Guo. (2001). Spoken Dialogue Management as Planning and Acting under Uncertainty. *Proceedings of Eurospeech, Aalborg, Denmark*. 7, 8