Chapter 8

PARTIALLY OBSERVABLE MARKOV DECISION PROCESSES WITH CONTINUOUS OBSERVATIONS FOR DIALOGUE MANAGEMENT

Jason D. Williams
AT&T Labs - Research
Florham Park, NJ, USA
jdw@research.att.com

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

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

Abstract This work shows how a spoken dialogue system can be represented as a Partially Observable Markov Decision Process (POMDP) with composite observations consisting of discrete elements representing dialogue acts and continuous components representing confidence scores. Using a testbed simulated dialogue management problem and recently developed optimisation techniques, we demonstrate that this continuous POMDP can outperform traditional approaches in which confidence score is tracked discretely. Further, we present a method for automatically improving handcrafted dialogue managers by incorporating POMDP belief state monitoring, including confidence score information. Experiments on the testbed system show significant improvements for several example handcrafted dialogue managers across a range of operating conditions.

Keywords: Spoken dialogue systems; Partially observable Markov decision processes; Dialogue management; Decision theory.

1. Introduction

Dialogue management is a difficult problem for several reasons. First, speech recognition errors are common, corrupting the evidence available to the machine about a user's intentions. Second, users may change their intentions at any point - as a result, the machine must decide whether conflicting evidence has been introduced by a speech recognition error, or by a new user intention. Finally, the machine must make tradeoffs between the "cost" of gathering additional information (increasing its certainty of the user's goal, but prolonging the conversation) and the "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 achieve the user's goal despite having imperfect information about that goal. For all these reasons, dialogue management can be cast as planning under uncertainty.

In this context, making use of available information about speech recognition accuracy ought to improve the performance of a dialogue manager. One key piece of information typically provided by the automatic speech recognition process is a confidence score, which provides a real-valued estimate of the probability that the recognition hypothesis is correct. In a traditional spoken dialogue system, a confidence score is used to decide whether to accept or reject a speech recognition hypothesis: if a hypothesis has a high confidence score, it is accepted; otherwise it is rejected. More nuanced approaches create confidence buckets which sub-categorise the accept category into N "buckets" such as low, medium and high. Confidence bucket information can then be incorporated into the dialogue state, and the dialogue manager can subsequently use this information when choosing actions, for example when deciding whether or not to confirm an element of the dialogue state.

This process is illustrated in the first two columns of Figure 1, which shows a conversation with a spoken dialogue system in the pizza-ordering domain. The first column indicates the words spoken by the user and the machine; the bracketed text shows the (possibly erroneous) results from the speech recognition process, followed by the confidence score. The second column shows how a typical spoken dialogue system might track dialogue state. In the last turn, a speech recognition error is made, and it is unclear how this evidence should be incorporated into the form in column 2 - should the new information replace the old information, or should it be ignored?

In this chapter we consider a different model for dialogue management: a partially observable Markov decision process (POMDP, pronounced "pom-dp"). Rather than tracking one explicit dialogue state, a POMDP maintains a

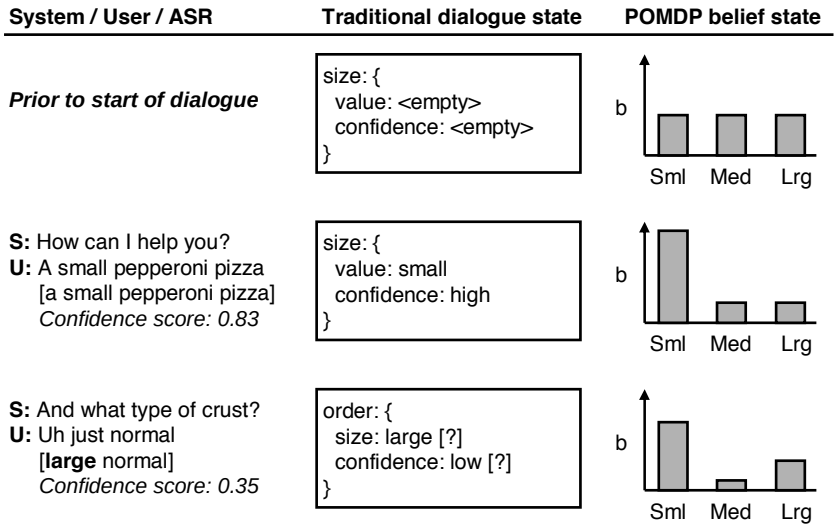

Figure 1. Example conversation with a spoken dialogue system in the pizza-ordering domain. The first column shows the words spoken by the user and the machine. The text in brackets shows the results from the speech recognition process, and the following line shows the resulting confidence score provided by the speech recognition engine. The second column shows how a typical dialogue manager might track a dialogue state, including a "confidence bucket" for the "size" field. The third column shows how the "POMDP belief state" would track the same conversation. Note how the traditional method struggles to account for the conflicting evidence in the last turn, whereas in the POMDP, the confidence score simply scales the magnitude of the update.

probability distribution over all possible dialogue states, called a belief state. As the dialogue progresses, the belief state is updated. This belief state update provides a principled method for interpreting confidence score: intuitively, the confidence score simply scales the magnitude of the update. This process is illustrated in the third column of Figure 1 - note how the (first) higher-confidence recognition causes a large movement of belief mass, whereas the (second) lower-confidence recognition causes a smaller movement of belief mass.

The goal of this chapter is to explain this process in detail and show it represents significant gains over a traditional "confidence bucket" approach through two central contributions. First, we show how a confidence score can be accounted for exactly in a POMDP-based dialogue manager by treating confidence score as a continuous observation. Using a test-bed simulated dialogue management problem, we show that recent optimisation techniques produce policies which outperform traditional MDP-based approaches across a range of operating conditions.

Second, we show how a hand-crafted dialogue manager can be improved automatically by treating it as a POMDP policy. We then show how a confidence

score metric can be easily included in this improvement process. We illustrate the method by creating three hand-crafted controllers for the test-bed dialogue manager, and show that our technique improves the performance of each controller significantly across a variety of operating conditions. This chapter is organised as follows. Section 2 briefly reviews background on POMDPs. Section 3 casts the dialogue management problem as a POMDP, showing how to incorporate a confidence score, and reviewing previous work. Section 4 outlines our test-bed dialogue management simulation, and compares policies produced by our method to a baseline on the test-bed problem which uses the traditional "confidence-bucket" approach. Section 5 shows how a handcrafted policy can be improved using confidence score, and provides an illustration, again using the test-bed problem. Section 6 briefly concludes.

2. Overview of POMDPs

Formally, a POMDP is defined as a tuple $\{S, A_m, T, R, O, Z\}$, where S is a set of states, A_m is a set of actions that an agent may take[1], T defines a transition probability $p(s'|s, a_m)$, R defines the expected (immediate, real-valued) reward $r(s, a_m)$, O is a set of observations, and Z defines an observation probability, $p(o'|s', a_m)$. In this chapter, we will consider POMDPs with discrete S and continuous O. The POMDP operates as follows. At each time-step, the machine is in some unobserved state s. The machine selects an action a_m, receives a reward r, and transitions to (unobserved) state s' , where s' depends only on s and a_m. The machine receives an observation O' which is dependent on s' and a_m. Although the observation gives the system some *evidence* about the current state s, s is not known exactly, so we maintain a distribution over states called a "belief state", b. We write $b(s)$ to indicate the probability of being in a particular state s. At each time-step, we update b as follows:

$$\begin{aligned} b'(s') &= p(s'|o', a_m, b) \\ &= \frac{p(o'|s', a_m, b)p(s'|a_m, b)}{p(o'|a_m, b)} \\ &= \frac{p(o'|s', a_m)\sum_{s\in S} p(s'|a_m, b, s)p(s|a_m, b)}{p(o'|a_m, b)} \\ &= \frac{p(o'|s', a_m)\sum_{s\in S} p(s'|a_m, s)b(s)}{p(o'|a_m, b)} \end{aligned} \tag{8.1}$$

The numerator consists of the observation function, transition matrix, and current belief state. The denominator is independent of s', and can be regarded

[1]In the literature, the system action set is often written as an un-subscripted A. In this work, we will model both machine and user actions, and have chosen to write the machine action set as A_m for clarity.

as a normalisation factor; hence:

$$b'(s') = k \cdot p(o'|s', a_m) \sum_{s \in S} p(s'|a_m, s) b(s) \tag{8.2}$$

We refer to maintaining the value of b at each time-step as "belief monitoring". The immediate reward is computed as the expected reward over belief states:

$$\rho(b, a_m) = \sum_{s \in S} b(s) r(s, a_m) \tag{8.3}$$

A POMDP policy specifies which action should be taken given a belief state[2]. The goal of policy learning is then to find a policy which maximises the cumulative, infinite-horizon, discounted reward called the return:

$$\sum_{t=0}^{\infty} \lambda^t \rho(b_t, a_{m_t}) = \sum_{t=0}^{\infty} \lambda^t \sum_{t=0}^{\infty} r(s, a_{m_t}) \tag{8.4}$$

where b_t indicates the distribution over all states at time t, $b_t(s)$ indicates the probability of being in state s at time-step t, and λ is a geometric discount factor, $0 < \lambda < 1$. Because belief space is real-valued, an optimal infinite-horizon policy may consist of an arbitrary partitioning of S-dimensional space in which each partition maps to an action. In fact, the size of the policy space grows exponentially with the size of the (discrete) observation set and doubly exponentially with the distance (in time-steps) from the horizon (Kaelbling et al., 1998). A continuous observation space compounds this further. Nevertheless, real-world problems often possess small policies of high quality.

In this work, we make use of approximate solution methods. The first, a point-based value iteration algorithm called Perseus (Spaan and Vlassis, 2004), operates on problems with discrete observation sets and is capable of rapidly finding good yet compact policies (when they exist). Perseus heuristically selects a small set of representative belief points, and then iteratively applies value updates to just those points, instead of all of the belief space, achieving a significant speed-up. Perseus has been tested on a range of problems, and found to outperform a variety of other methods, including grid-based methods (Spaan and Vlassis, 2004).

Perseus (like all value-iteration optimisation algorithms) produces a value function represented as a set of N vectors each of dimensionality $|S|$. We write $v_n(s)$ to indicate the s_{th} component of the n_{th} vector. Each vector represents the value, at all points in the belief space, of executing some "policy tree" which starts with an action associated with that vector. We write $\hat{\pi}(n) \in A$ to

[2]We will assume the planning horizon for a policy is infinite unless otherwise stated.

indicate the action associated with the n_{th} vector. If we assume that the policy trees have an infinite horizon, then we can express the optimal policy at all time-steps as:

$$\pi(b) = \hat{\pi}(\underset{n}{\operatorname{argmax}} \sum_{s=1}^{|S|} v_n(s)b(s)) \tag{8.5}$$

In simple terms, a value function provides both a partitioning of the belief space (where each region corresponds to an action which is optimal in that region), as well as the expected return of taking that action. In this chapter we will also make use of an extension to Perseus proposed by Hoey and Poupart (2005) which operates on POMDPs with continuous or very large discrete observation sets. This method exploits the fact that different observations may lead to identical courses of action to discretise continuous observations without any loss of information. In the context of dialogue management with a continuous confidence score, it implicitly and adaptively finds optimal lossless buckets of confidence that are equivalent to using the original continuous confidence score[3].

3. Casting Dialogue Management as a POMDP

Williams et al. (2005) cast a spoken dialogue system as a factored POMDP, and this model will be used as the general framework for the techniques presented here. In this model, the POMDP state variable $s \in S$ is separated 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 history of the dialogue, $s_d \in S_d$. 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 user's goal, s_u, gives the current goal or intention of the user. Examples of a complete user goal include a complete travel itinerary, a desired appointment to make in a calendar, or a product the user would like to purchase. The user's goal persists over the course of the dialogue, and in general it will remain static although it is possible for it to change (for example, if the machine indicates that there are no direct flights, the user's goal might change to include indirect flights).

The user's action, a_u, gives the user's most recent actual action. Examples of user actions include specifying a place the user would like to travel to, responding to a yes/no question, or a "null" response indicating the user took no action. User actions may convey a portion of the user's goal (such as re-

[3]The actual implementation used in this chapter approximates some integrals by Monte Carlo sampling, which means that the confidence buckets are not exactly lossless.

questing a flight "to London"), or may serve a communicative role (such as answering a yes/no question).

The history of the dialogue s_d indicates any relevant dialogue history information. For example, s_d might indicate that a particular slot has not yet been stated, has been stated but not grounded, or has been grounded. s_d enables a policy to make decisions about the appropriateness of behaviours in a dialogue - for example, if there are ungrounded items, a dialogue designer might wish to penalise asking an open question (vs. grounding an item).

Note that we do not include a state component for *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.

The POMDP action $a_m \in A_m$ is the action the machine takes in the dialogue. For example, machine actions might include greeting the user, asking the user where he or she wants to go "to", or confirming that the user wants to leave "from" a specific place.

To factor the model, we decompose the POMDP transition function as follows:

$$\begin{aligned} p(s'|s, a_m) &= p(s'_u, s'_d, a'_u|s_u, s_d, a_u, a_m) \\ &= p(s'_u|s_u, s_d, a_u, a_m) \cdot \\ &\quad p(a'_u|s'_u, s_u, s_d, a_u, a_m) \cdot \\ &\quad p(s'_d|a'_u, s'_u, s_u, s_d, a_u, a_m) \end{aligned} \tag{8.6}$$

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:

$$p(s'_u|s_u, s_d, a_u, a_m) = p(s'_u|s_u, a_m) \tag{8.7}$$

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 his/her (current) goal and the preceding machine action:

$$p(a'_u|s'_u, s_u, s_d, a_u, a_m) = p(a'_u|s'_u, a_m) \tag{8.8}$$

The third term - which we call the *dialogue history model* - indicates how the user and machine actions affect the dialogue history. The current history of the dialogue depends on the previous history combined with the most recent user and machine actions:

$$p(s'_d|a'_u, s'_u, s_u, s_d, a_u, a_m) = p(s'_d|a'_u, s'_u, a_m) \tag{8.9}$$

In sum, our transition function is given by:

$$p(s'|s, a_m) = p(s'_u|s_u, a_m) \cdot p(a'_u|s'_u, a_m) \cdot p(s'_d|a'_u, s_d, a_m) \tag{8.10}$$

This factored representation reduces the number of parameters required for the transition function, and allows groups of parameters to be estimated separately. For example, we could estimate the *user action model* from a corpus by counting user dialogue acts given a machine dialogue act and a user goal, or use a "generic" distribution and adapt it to a particular problem once data becomes available[4]. We could then separately specify the dialogue history model using a handcrafted function such as "Information State" update rules as in for example (Larsson and Traum, 2000).

The POMDP observation o is decomposed into two elements: the speech recognition hypothesis $\tilde{a}_u \in A_u$ and the confidence score $c \in R$. The observation function is given by:

$$p(o'|s', a_m) = p(\tilde{a}'_u, c'|s'_u, s'_d, a'_u, a_m) \tag{8.11}$$

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, and by the grammar g selected by the dialogue manager:

$$p(\tilde{a}'_u, c'|s'_u, s'_d, a'_u, a_m) = p(\tilde{a}'_u, c'|a'_u, g) \tag{8.12}$$

The observation function can be estimated from a corpus or derived analytically using a phonetic confusion matrix, language model, etc. This distribution expresses the probability density of observing recognition hypothesis $\tilde{a}'_u$ with confidence score c when the user actually took action a_u and recognition grammar g was activated. As such, the observation function can be viewed as a model of the errors introduced by the speech recognition channel.

Together equations 8.10 and 8.12 represent a statistical model of a dialogue. The transition function allows future behaviour to be predicted and the observation function provides the means for inferring a distribution over hidden user states from observations. The factoring is general-purpose in that the user goal component s_u allows the user to have a hidden, persistent state which emits unobserved actions a_u that are corrupted into observations $\tilde{a}_u$ by the speech recognition process. Further, the dialogue history component s_d enables actions to be selected with an awareness of dialogue history. Figure 2 summarizes the factored model, depicted as an influence diagram.

[4]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).

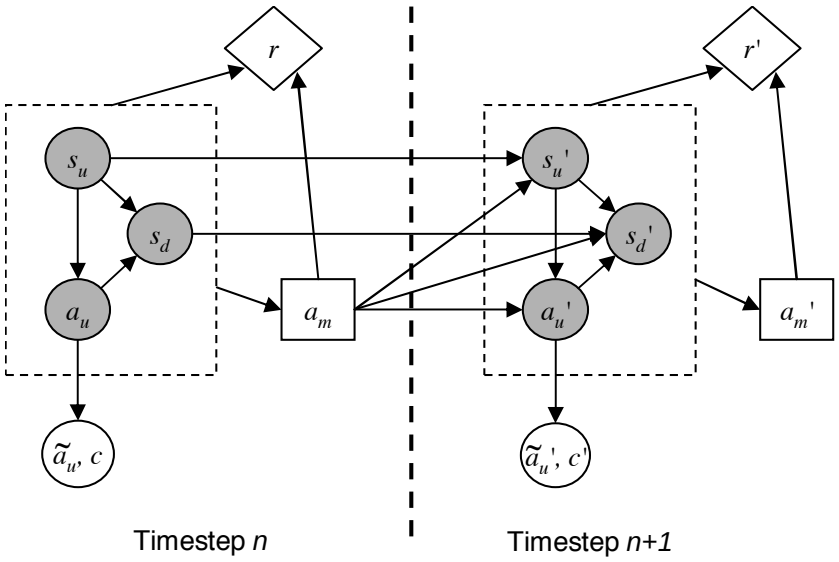

Figure 2. Influence diagram for the factored model. The dotted box indicates the composite state s is comprised of three components, s_u, s_d, and a_u. Shading indicates a component is unobservable. Arcs into circular chance nodes and diamond-shaped utility nodes show influence, whereas arcs into square decision nodes are informational, as in (Jensen, 2001, p140). The arc from the dotted box to a_m indicates that a_m is chosen based on the belief state - i.e., a distribution over s_u, s_d, and a_u.

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 dialogue speed (by using a per-turn penalty), appropriateness (through rewards conditioned on dialogue state), and successful task completion (through rewards conditioned on the user's goal). Weights between these incentives could be estimated through formalisms like PARADISE (Walker et al., 2000), and then adapted to the needs of a particular domain - for example, accuracy in performing a financial transaction is arguably more important than accuracy when obtaining weather information. As described in the previous section, actions are selected based on the belief state to maximise cumulative long-term reward.

Finally, we update the belief state at each time step by substituting equations 8.10 and 8.12 into 8.2 and simplifying:

$$b'(s'_u, s'_d, a'_u) = k \cdot p(\tilde{a}'_u, c'|a'_u, g)p(a'_u|s'_u, a_m) \cdot \sum_{s_u \in S_u} p(s'_u|s_u, a_m) \cdot \quad (8.13)$$

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

The belief monitoring update equation 8.13 exemplifies the key difference between conventional approaches to dialogue management and the POMDP approach. In conventional approaches, a single state vector is maintained which encodes the system's "best guess" about all of the information needed to determine the next system action. For example, a state vector might include a record of all of the informational items supplied by the user, their grounding state, dialogue history, etc. Both the user's input and the subsequent system output are dependent on this state vector, but since there can be errors in this state vector, the system will often make mistakes and must then enter some form of recovery procedure. This is essentially a depth-first greedy search with back-tracking.

In the POMDP approach, all possible states are maintained rather than the single most likely state. Each user input (i.e., the observation) is then interpreted in the context of each possible new state via the observation term $p(\tilde{a}'_u, c'|a'_u, g)$. If the new observation is likely given s', then the subsequent belief in s' will be high and vice versa. The new state s' itself will only be plausible if there is a non-zero likelihood of making a transition from some previous state s to the new state, and since the previous state is unknown, all possible transitions are considered, weighted by the beliefs at the previous turn. In search terms, this is breadth-first search. It has the advantage over depth-first that both inputs and outputs can be determined from a knowledge of all of the alternative interpretations.

In practice the observation function $p(\tilde{a}'_u, c'|a'_u, g)$ will be difficult to estimate directly from data, so we will decompose the distribution by assuming that confidence scores are drawn from just two distributions - one for "correct" recognitions and another for "incorrect" recognitions:

$$p(\tilde{a}'_u, c'|a'_u, g) \approx \begin{cases} p_{correct}(c') \cdot p(\tilde{a}'_u|a'_u, g), & if \quad \tilde{a}'_u = a'_u \\ p_{incorrect}(c') \cdot p(\tilde{a}'_u|a'_u, g) & if \quad \tilde{a}'_u \neq a'_u \end{cases} \quad (8.14)$$

where $p(\tilde{a}'_u|a'_u, g)$ expresses the *confusion matrix* - i.e., probability of observing hypothesis $\tilde{a}'_u$ given that the user took action a'_u and grammar g was active; and $p_{correct}(c')$ and $p_{incorrect}(c')$ express the probability density function of the confidence scores associated with correct and incorrect recognitions. To perform policy improvement on this POMDP we have two options. First, we can use an optimisation method which accounts for the continuous observations, such as that by Hoey and Poupart (2005). This method creates a policy which takes the expected additional information in the confidence score into account, and we call this the *continuous-POMDP* solution. Alternatively, there is still benefit to using the confidence score information for belief state monitoring (as in 8.13) even if it was not used during policy optimisation. Thus a second option for performing policy improvement is to marginalise the confi-

dence score, i.e.:

$$p(\tilde{a}'_u|a'_u, g) = \int_{c'} p(\tilde{a}'_u, c'|a'_u, g) \tag{8.15}$$

and to then optimise the resulting POMDP using a technique such as *Perseus*. At runtime, the full observation function $p(\tilde{a}'_u, c|a'_u, g)$ is used for belief state monitoring. We call this the *discrete-POMDP* solution.

Stated alternatively, the *continuous-POMDP* technique uses infinitely many confidence buckets during planning and belief monitoring, whereas the *discrete-POMDP* technique uses no confidence information during planning, but infinitely many confidence buckets during belief monitoring. By contrast, 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[5].

In the literature, casting dialogue management as planning under uncertainty has been attempted using both (fully observable) Markov decision processes (MDPs) and POMDPs. The application of MDPs was first explored by Levin and Pieraccini (1997). Levin et al. (2000) provide a formal treatment of how an MDP may be applied to dialogue management, and Singh et al. (2002) show application to real systems. However, MDPs assume the current state of the environment (i.e., the conversation) is known exactly, and thus they do not naturally capture the uncertainty introduced by the speech recognition channel.

Partially observable MDPs (POMDPs) extend MDPs by providing a principled account of noisy observations. Roy et al. (2000) compare an MDP and a POMDP version of the same spoken dialogue system, and find that the POMDP version gains more reward per unit time than the MDP version. Further, the authors show a trend that as speech recognition accuracy degrades, the margin by which the POMDP outperforms the MDP increases. Zhang et al. (2001) extend this work in several ways. First, the authors add "hidden" system states to account for various types of dialogue trouble, such as different sources of speech recognition errors. Second, the authors use Bayesian networks to combine observations from a variety of sources (including confidence score). The authors again show that the POMDP-based methods outperform MDP-based methods. In all previous work (using both MDPs and POMDPs), confidence score has been incorporated by dividing the confidence score metric into discrete confidence "buckets". For example, in the MDP literature, Singh et al. (2002) track the confidence bucket for each field as "high, medium, or low" confidence. The authors do not address how to determine an "optimal"

[5]In theory, one could create an MDP with continuous components in its state space, and use these components to track confidence score. While this avoids "binning" the confidence score, it does not aggregate evidence over time: in order to do this in an MDP, state components for "most recent confidence score", "2nd more recent confidence score", etc. would be required, causing rapid growth in the state space. By contrast, a POMDP frames a sequence of confidence scores as observations and naturally accumulates evidence over time through belief monitoring.

number of confidence buckets, nor how to determine the "optimal" thresholds of the confidence score metric that divide each bucket. In the POMDP literature, Zhang et al. (2001) use Bayesian networks to combine information from many continuous and discrete sources, including confidence score, to compute probabilities for two metrics called "Channel Status" and "Signal Status". Thresholds are then applied to these probabilities to form discrete, binary observations for the POMDP. Again, it is not clear how to set these thresholds to maximise POMDP return. Looking outside the (PO)MDP framework, Paek and Horvitz (2003) suggest using an influence diagram to model user and dialogue state, and selecting actions based on "Maximum Expected [immediate] Utility". This proposal can be viewed as a POMDP with continuous observations that greedily selects actions - i.e., which selects actions based only on immediate reward. By choosing appropriate utilities, the authors show how local grounding actions can be automatically selected in a principled manner. In this work, we are interested in POMDPs as they enable planning over any horizon.

4. Comparison with Traditional Approach

To assess the benefits of the POMDP approach versus traditional "confidence bucket" approaches, we created a test-bed dialogue management problem in the travel domain. This test-bed problem enables direct comparisons between dialogue managers produced by casting the problem as a POMDP with continuous observations, and dialogue managers produced by adding "confidence buckets" and casting the problem as an MDP. In both the POMDP and MDP, dialogue managers are produced automatically. Assuming that these represent optimal solutions, then this comparison gives a quantitative indication of the value of the POMDP approach.

4.1 POMDP Test-Bed Dialogue Management Problem

In the test-bed dialogue management problem, the user is trying to buy a ticket to travel from one city to another city. The machine asks the user a series of questions, and then "submits" a ticket purchase request, ending the dialogue. The machine may also choose to "fail". In the test-bed problem, there are three cities, $\{a, b, c\}$. The machine has 16 actions available, including *greet*, *ask-from*/*ask-to*, *conf-to-x*/*conf-from-x*, *submit-x-y*, and *fail*, where $x, y \in \{a, b, c\}$. As above, the POMDP state 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 $s_u = (x, y) : x, y \in \{a, b, c\}, x \neq y$. The dialogue state s_d contains three components. Two of these indicate (from the user's perspective) whether the from place and to place have not been specified (n), are unconfirmed (u),

or are confirmed (c). A third component z specifies whether the current turn is the first turn (1) or not (0). There are a total of 18 values of s_d, given by:

$$s_d = (x_d, y_d, z); \quad x_d, y_d \in \{n, u, c\}; \quad z \in 1, 0 \tag{8.16}$$

The user's action $a_u \in A_u$ and the observation $\tilde{a}_u \in A_u$ are drawn from the set *x*, *from-x*, *to-x*, *from-x-to-y*, *yes*, *no*, and *null*, where $x, y \in \{a, b, c\}, x \neq y$. These state components yield a total of 1944 states, to which we add one additional, absorbing end state. When the machine takes the *fail* action or a *submit-x-y* action, control transitions to this end state, and the dialogue ends. The initial (prior) probability of the user's goal is distributed uniformly over the 6 user goals. In the test-bed problem the user has a fixed goal for the duration of the dialogue, and we define the *user goal model* accordingly.

We define the *user action model* $p(a'_u|s'_u, a_m)$ to include a variable set of responses - for example: the user may respond to *ask-to/ask-from* with x, *to-x/from-x*, or *from-x-to-y*; the user may respond to *greet* with *to-y*, *from-x*, or *from-x-to-y*; the user may respond to *confirm-to-x/confirm-from-x* with *yes/no*, x, or *to/from-x*; and at any point the user might not respond (i.e., respond with $null$). The probabilities in the user action model were chosen such that the user usually provides cooperative but varied responses, and sometimes doesn't respond at all. The probabilities were handcrafted, selected based on experience performing usability testing with slot-filling dialogue systems.

We define the *dialogue model* $p(s'_d|a'_u, s_d, a_m)$ to deterministically implement the notions of dialogue state above - i.e., a field 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. For example, at the beginning of the dialogue, the dialogue state s_d is $(n, n, 1)$. If the user were to say "I'd like to go to b" in his/her first utterance, the resulting dialogue state would be $(n, u, 0)$. If the system were to reply "To b - is that right?", and the user replied "Yes, from a to b", then the resulting dialogue state would be $(u, c, 0)$. We define the confusion matrix $p(\tilde{a}'_u|a'_u, g)$ to encode the probability of making a speech recognition error to be p_{err}. Further, we assume that one recognition grammar is always used:

$$\begin{aligned} p(\tilde{a}'_u, c'|a'_u, g) &= p(\tilde{a}'_u, c'|a'_u) \\ &= \begin{cases} p_{correct}(c') \cdot (1 - p_{err}) & if \quad \tilde{a}_u = a_u \\ p_{incorrect}(c') \frac{p_{err}}{|A_u|-1} & if \quad \tilde{a}_u \neq a_u \end{cases} \end{aligned} \tag{8.17}$$

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

Past work has found the distribution of confidence scores to be exponential (Pietquin, 2004), and here we define the confidence score probability density functions $p_{correct}(c')$ and $p_{incorrect}(c')$ to be exponential probability density functions normalised to the region [0,1], i.e.:

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

$$p_{incorrect}(c) = \begin{cases} \frac{a_{incorrect} e^{(1-c) \cdot a_{incorrect}}}{e^{a_{incorrect}} - 1}, & a_{incorrect} \neq 0 \\ 1, & a_{incorrect} = 0 \end{cases} \quad (8.18)$$

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 $a_x > 0$. To illustrate the meaning of $a_{correct}$ and $a_{incorrect}$, a small classification task was created in which a confidence score is used as a decision variable to classify $\tilde{a}_u$ as either *correct* or *incorrect*. Various concept error rates (values of p_{err}) and a_x were considered and for each pair of values, the confidence threshold which minimised classification error rate was used. Table 1 shows the results. When $a_x = 0$, all hypotheses are classified as *correct* and the classification error rate is the same as p_{err}. As a_x is increased, the classification error rate decreases. Intuitively, Table 1 shows the minimum possible classification error rate achievable with a given a_x, and comparing this with the prior error rate p_{err} gives an indication of the *informativeness* of a_x.

Table 1. Minimum classification error rate possible for various concept error rates (P_{err}) and levels of confidence score informativeness (a_x).

	Concept error rate(P_{err})		
a_x	0.10	0.30	0.50
0	10%	30%	50%
1	10%	30%	38%
2	9%	23%	27%
3	9%	16%	18%
4	6%	11%	12%
5	4%	7%	8%
∞	0%	0%	0%

The reward measure for the test-bed dialogue problem includes components for both task completion and dialogue "appropriateness", including: a reward of -3 for confirming a field before it has been referenced by the user; a reward of -5 for taking the *fail* action; a reward of +10 or -10 for taking the *submit-x-y* action when the user's goal is *(x,y)* or not, respectively; and a reward of

-1 otherwise. The reward measure reflects the intuition that behaving inappropriately or even abandoning a hopeless conversation early are both less severe than getting the user's goal wrong. The per-turn penalty of -1 expresses the intuition that, all else being equal, short dialogues are better than long dialogues. The reward measure also assigned -100 for taking the *greet* action when not in the first turn of the dialogue. This portion of the reward function effectively expresses a design decision: the greet action may only be taken in the first turn. A discount of $\gamma = 0.95$ was used for all experiments.

Both the *Perseus* and the *Hoey-Poupart* algorithms required parameters for the number of belief points and number of iterations. Through experimentation, we found that 500 belief points and 30 iterations attained asymptotic performance for all values of P_{err}. In addition, the *Hoey-Poupart* algorithm required a parameter specifying the number of observations to sample at each belief point. Through experimentation, we found that 300 samples produced acceptable results and reasonable running times.

4.2 MDP Baseline

To test whether the method for incorporating confidence score outperforms current methods, an MDP was constructed to assess performance of a model which does not track multiple dialogue states, and which does not make use of an explicit user model. The MDP was patterned on systems in the literature (Pietquin, 2004). The MDP state contains components for each field which reflect whether, from the standpoint of the machine, (a) a value has not been observed, (b) a value has been observed but not confirmed, or (c) a value has been confirmed. The MDP state also tracks which confidence bucket was observed for each field, as well as for the confirmation. Finally, two additional states - *dialogue-start* and *dialogue-end* - are included in the MDP state space.

The "confidence bucket" is determined by dividing the confidence score into M buckets. Ideally the confidence score bucket sizes would be selected so that they maximise average return. However, it is not obvious how to perform this selection - indeed, this is one of the weaknesses of the "confidence bucket" method. Instead, a variety of techniques for setting the 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 among the techniques explored[6]. That is, we define

[6]The other techniques included dividing the *range of confidence* scores equally (e.g., for two buckets, using a threshold of 0.5), and dividing the *range of error rates* equally (e.g., for two buckets, setting a threshold such that p(observation is correct | confidence score) = 0.5).

$$cThresh_0 = 0 < cThresh_1 < \cdots < cThresh_{M-1} < cThresh_M = 1 \tag{8.19}$$

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

$$\int_{cThresh_{m-1}}^{cThresh_m} p(c)dc = \int_{cThresh_m}^{cThresh_m+1} p(c)dc, \quad m \in 1, 2, \cdots, M-1 \tag{8.20}$$

where $p(c)$ is the prior probability of a confidence score. We find this prior for our test-bed problem as follows. We first find the distribution $p(c|a_u)$ as:

$$p(c|a_u) = \sum_{h \in A} p(h, c|a_u) \tag{8.21}$$

$$= p_{correct}(c|a_u)(1 - p_{err}) + p_{incorrect}(c|a_u)(p_{err}) \tag{8.22}$$

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:

$$p(c) = p_{correct}(c)(1 - p_{err}) + p_{incorrect}(c)(p_{err}) \tag{8.23}$$

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

Because the confidence bucket for each field (including its value and its confirmation) is tracked in the MDP state, the size of the MDP state space grows with the number of confidence buckets. For $M = 2$ confidence buckets, the resulting MDP called *MDP-2* has 51 states[7].

Given the current MDP state, the MDP policy selects an MDP action, and the MDP state estimator then maps the MDP action back to a POMDP action. Because the MDP learns through experience with a simulated environment, an on-line learning technique, (Watkins, 1989) Q-learning, was used to train the MDP baseline. A variety of learning parameters were explored, and the best-performing parameter set was selected: initial Q values set to 0, exploration parameter $\varepsilon = 0.2$, and the learning rate α set to $1/k$ (where k is the number of visits to the $Q(s, a)$ being updated). *MDP-2* was trained with approximately 125,000 dialogue turns. To evaluate the resulting MDP policy, 10,000 dialogues were simulated using the learned policy.

4.3 Results

Figure 3 shows the average returns for the *continuous-POMDP*, *discrete-POMDP*, and *MDP-2* solutions vs. p_{err} ranging from 0.00 to 0.65 for $a_{correct} =$

[7]For reference, $M = 1$ produces an MDP with 11 states, and $M = 3$ produces an MDP with 171 states.

$a_{incorrect} = a = 1$. (At each data point, an error rate p_{err} was set, and errors and confidence scores were generated synthetically according to equations 8.17 and 8.18.). Figure 3 also shows curves for *noconf-POMDP* a POMDP which ignores confidence score information and *MDP*, an MDP which ignores confidence score information (i.e., an MDP with just one confidence score bucket). 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 solutions attain larger returns than the MDP method at all values of p_{err} [8]. From this plot it can be seen that the addition of confidence score information improves both the POMDP and MDP solutions. This plot shows that, at $a = 1$, the addition of confidence score information has a large improvement in performance for the MDP, and a modest but significant improvement on the POMDP.

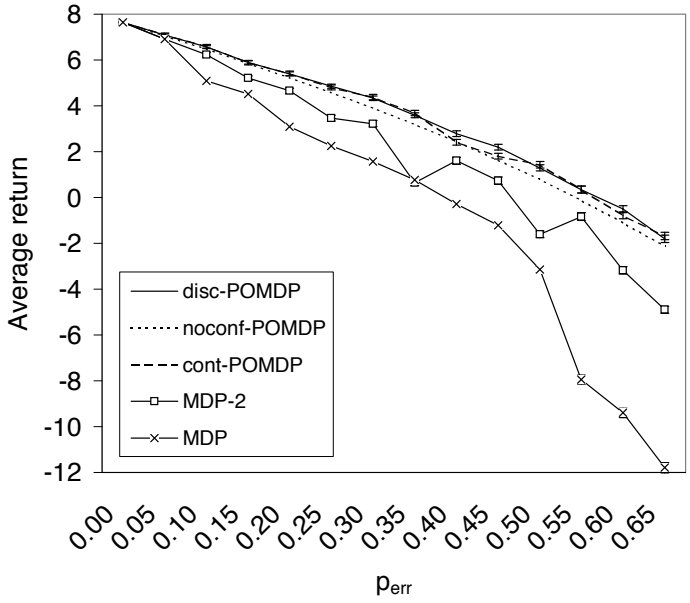

Figure 3. ***Average return*** for continuous-POMDP, discrete-POMDP, noconf-POMDP, MDP-2 and MDP methods for $a = 1$.

As the informativeness of the confidence score increases, it would be expected that the performance of both the MDP and POMDP would continue to improve. This is confirmed in Figures 4, 5, and 6 which show average returns for the *discrete-POMDP* and *continuous-POMDP* methods and *MDP-2* method vs. a for $p_{err} = 0.3$, 0.4, and 0.5, respectively. The error bars show

[8]The *MDP-3* system was also created but we were unable to obtain better performance from it than we did from the *MDP-2* system.

the 95% confidence interval for return assuming a normal distribution. In these figures, we again define $a_{correct} = a_{incorrect} = a$. 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. These figures also provide a quantitative illustration of the benefit of belief monitoring vs. the benefit of the confidence score information. For example, in Figure 6 (in which $p_{err} = 0.5$), at $a = 0$, the POMDP achieves an average of 0.7 units of reward/dialogue whereas the MDP achieves an average of -3.1 units of reward/dialogue. In other words, ignoring confidence score altogether, the belief monitoring provided by the POMDP results in an increase from -3.1 to 0.7. The addition of a very informative confidence score (i.e., $a = 5$) to the POMDP results in an increase from 0.7 units of reward/dialogue to 3.2 units of reward/dialogue.

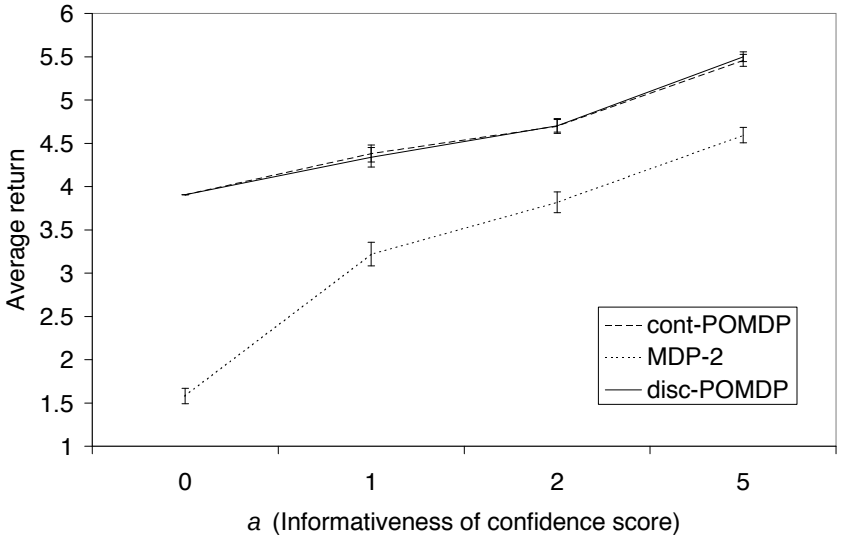

Figure 4. Average return vs. *a (informativeness of confidence score)* at $p_{err} = 0.30$ for continuous-POMDP, discrete-POMDP, and MDP-2 methods.

In Figures 3 through 6, the discrete-POMDP and continuous-POMDP methods performed similarly[9]. In this task, use of the confidence score *during planning* does not improve performance of the POMDP. This could be due to the relatively short horizon in the test-bed problem, as most of the dialogues

[9]Additional experiments were performed (not shown here) which performed POMDP optimisation with 2, 4, and 8 "buckets" and continuous belief monitoring during evaluation, and these produced very similar results.

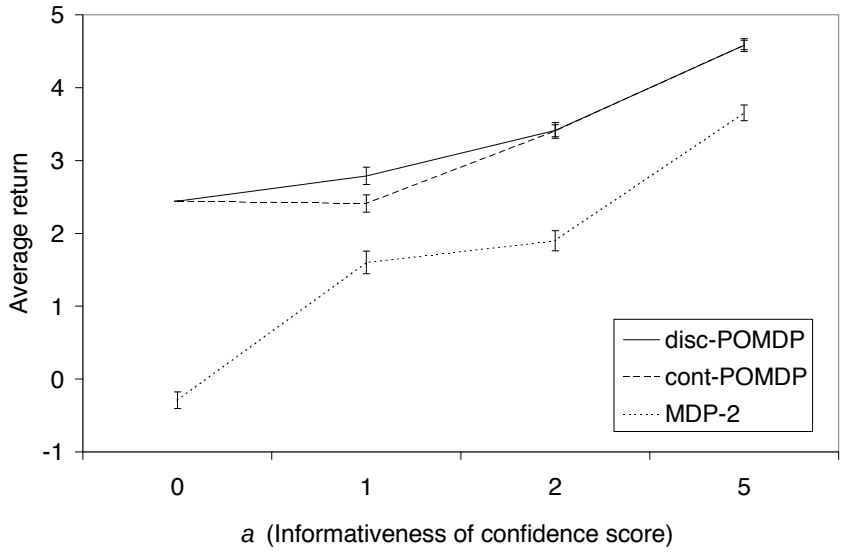

Figure 5. *Average return* vs. *a (informativeness of confidence score)* at $p_{err} = 0.40$ for continuous-POMDP, discrete-POMDP, and MDP-2 methods.

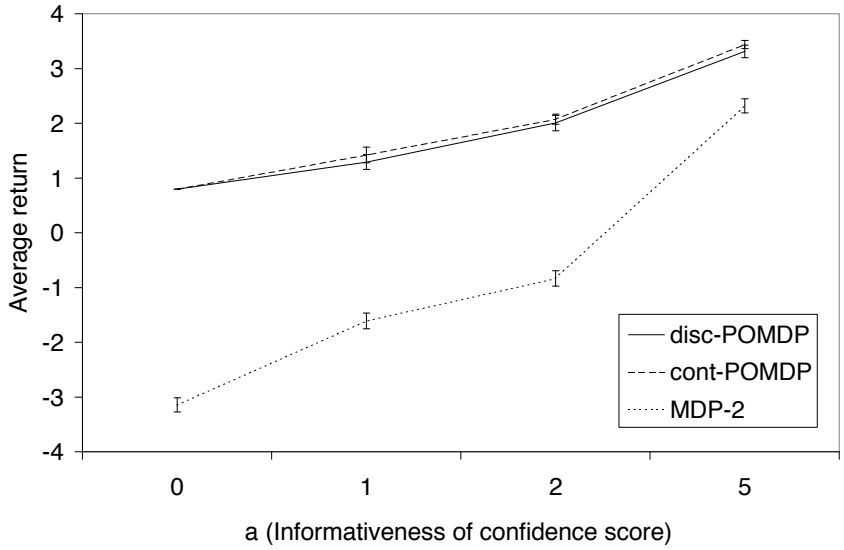

Figure 6. *Average return* vs. *a (informativeness of confidence score)* at $p_{err} = 0.50$ for continuous-POMDP, discrete-POMDP, and MDP-2 methods.

spanned only a handful of turns. We intend to explore this issue with larger dialogue management problems in future work.

5. Improving Handcrafted Policies

In the previous section, a designer specified a reward function, and actions were selected to maximise reward using automated planning. In traditional approaches to dialogue management, the designer specifies actions directly, a process often called *handcrafting*[10].

Automated planning is appealing because adding confidence score information to the dialogue state space increases its size dramatically, complicating the work of a human designer. This section presents an alternative approach in which a human designer produces a handcrafted dialogue manager which *does not include confidence score information*. Rather, the spoken dialogue system is viewed as a POMDP, belief monitoring (which takes confidence score into account) is performed, and the handcrafted controller is executed in conjunction with the belief state. Concretely, the handcrafted policy is evaluated by constructing its value function, and is then executed in the style of the *discrete-POMDP* above.

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 value iteration - i.e., a value function, represented as a set of N vectors each of dimensionality $|S|$. We write $v_n(s)$ to indicate the sth component of the nth vector.

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 dialogue management policies (Pieracinni and Huerta, 2005). More complex handcrafted policies - for example, those created with rules - can usually be compiled into a (possibly very large) policy graph. That said, a policy graph does not make the expected return associated with each controller node explicit, but as pointed out by Hansen (1998), we can find the expected return associated with each controller node by solving this system of linear equations in v:

$$v_n(s) = r(s, \hat{\pi}(n)) + \gamma \sum_{s' \in S} \sum_{o \in O} p(s'|s, \hat{\pi}(n)) p(o|s', \hat{\pi}(n)) v_{l(n,o)}(s') \quad (8.24)$$

[10]In both POMDP and traditional approaches, the designer creates a dialogue model; the focus here is how actions are selected given a dialogue model.

Solving this set of linear equations yields a set of vectors - one vector $v(s)$ for each controller node, $v_n(s)$. To find the expected value of starting the controller in node n and belief state b we compute:

$$\sum_{s=1}^{|S|} v_n(s)b(s) \tag{8.25}$$

To improve the performance of the controller, we use $v_n(s)$ at *run-time*, as follows. At the beginning of the dialogue, we find the node with the highest expected return for b_0 and execute its action. Throughout the dialogue, we perform belief state monitoring - i.e., we maintain the current belief state at each time-step as given in equation 8.13. At each time-step, rather than following 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. 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. Note that, in this style of execution, transitions may occur which are not arcs in the handcrafted policy.

This style of execution is distinct from *policy iteration*, in which the nodes and links of the controller are changed and the controller is re-evaluated (using e.g., equation 8.24) to iteratively improve the controller's expected return. We do not explore policy iteration in this chapter; however, we note that a handcrafted controller could be used to bootstrap a policy iteration process. Since a finite state controller is more intuitive for a (human) designer to understand, we intend to explore policy iteration in future work.

Three handcrafted policies were created for the test-bed dialogue management problem, 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, it re-tries the same action. If it receives an observation which is inconsistent or nonsensical, it re-tries the same action. If it fills both fields without receiving any inconsistent information, it takes the corresponding *submit-x-y* action. A logical diagram showing *HC1* is shown in Figure 7[11].

HC2 is identical to HC1 except that if the machine receives an observation which is inconsistent or nonsensical, it immediately takes the *fail* action. Once it fills both fields, it takes the corresponding *submit-x-y* action.

$HC3$ 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

[11] 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.

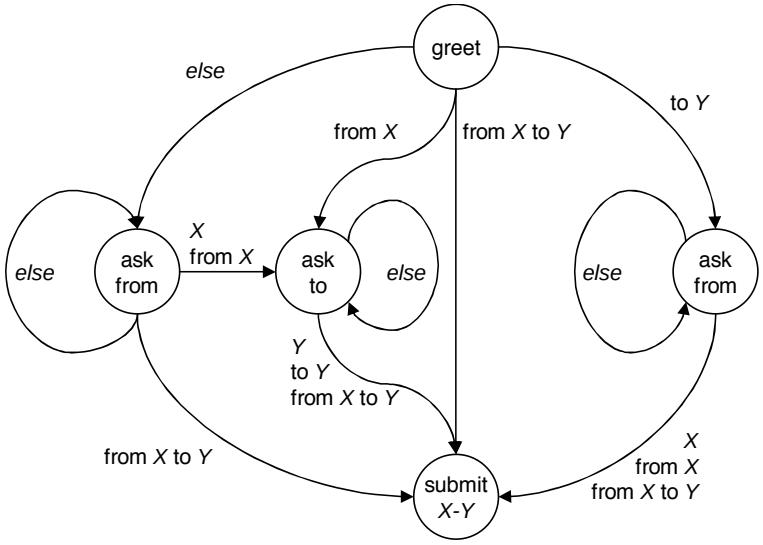

Figure 7. HC1 handcrafted controller.

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.

We first studied the operation of the greedy improvement method without access to confidence score information. We executed 10,000 dialogues for each handcrafted policy at values of p_{err} ranging from 0.05 to 0.65. Figure 8 gives results for $HC1$. To make the gain of the greedy improvement method explicit, Figure 8 shows the difference between the proposed method and the expected value of executing the handcrafted policy directly. For reference, Figure 8 also includes the difference between the handcrafted policies executed normally and the POMDP policy, which we take to be a practical upper bound for the test-bed problem. Error bars show the 95% confidence interval for the true expected return assuming normal distribution. We note that in almost all cases, the greedy improvement method results in a significant improvement. In many cases, the improved handcraft controller is close to the POMDP policy - our assumed practical upper bound. Results for $HC2$ and $HC3$ are shown in Figures 9 and 10.

We next studied the operation of the greedy improvement method when confidence score information is present. Figures 11, 12, and 13 show average returns for the *discrete-POMDP* and improved handcraft methods vs. a for $p_{err} = 0.3$, 0.4, and 0.5, respectively. a is defined as in Section 4.2 - i.e., $a = a_{correct} = a_{incorrect}$. Error bars are negligible and are not shown. For

each of the three handcrafted controllers in each of the three values of p_{err}, increasing a consistently increases average return.

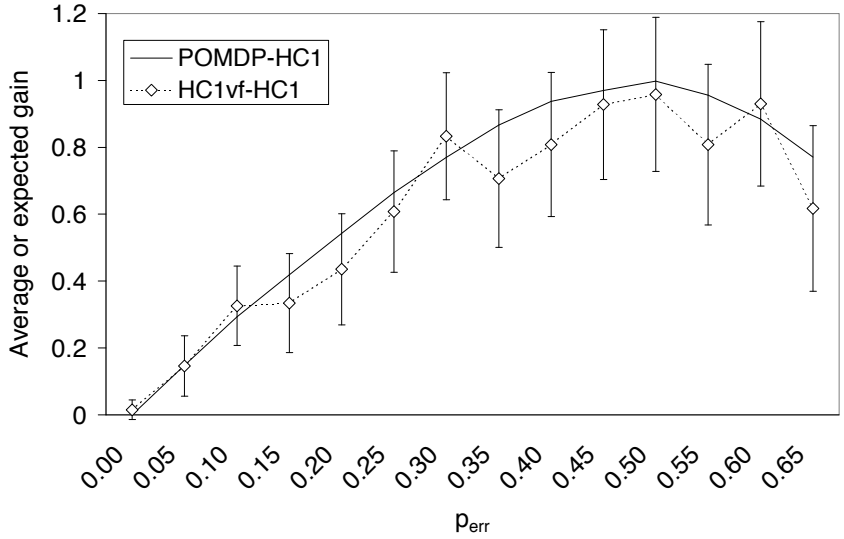

Figure 8. Gain in average/expected return for HC1 executed using belief state monitoring vs. p_{err} for $a = 0$. (The POMDP policy, which we take to be our practical upper bound, is shown for reference in Figures 8 through 10.)

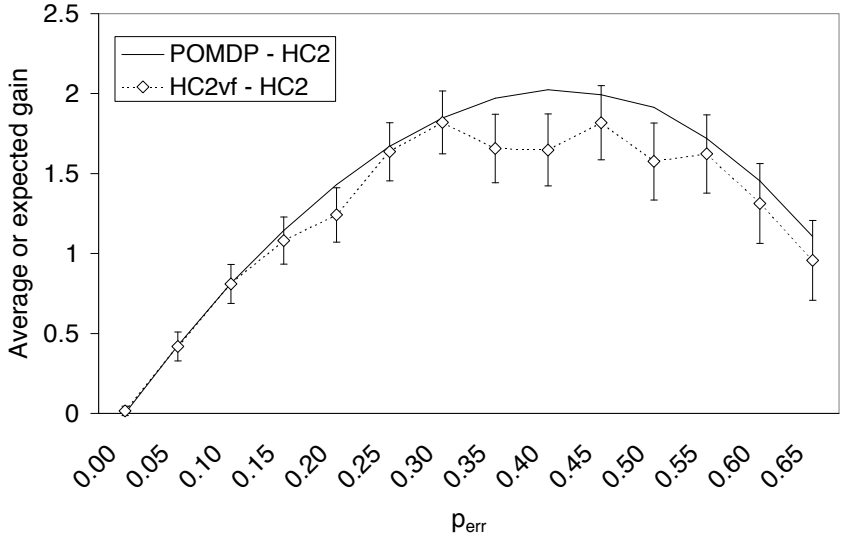

Figure 9. Gain in average/expected return for HC2 executed using belief state monitoring vs. p_{err} for $a = 0$.

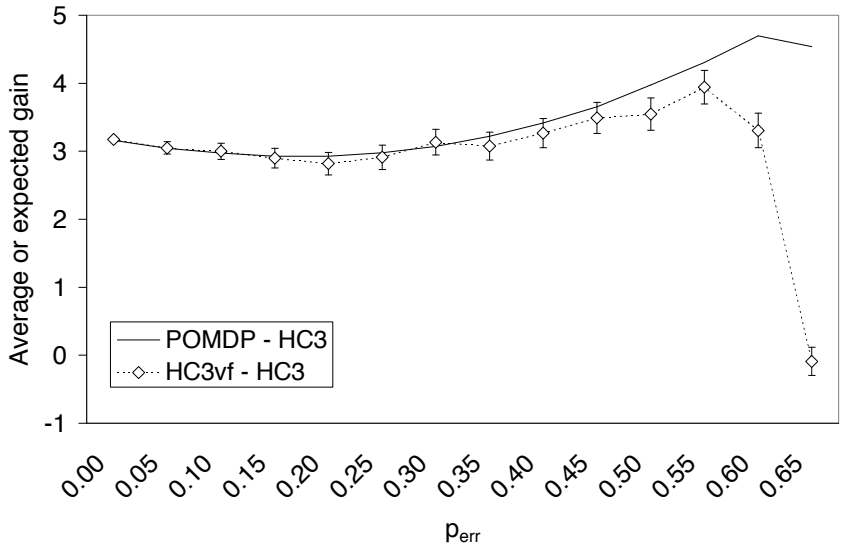

Figure 10. Gain in average/expected return for HC3 executed using belief state monitoring vs. p_{err} for $a = 0$.

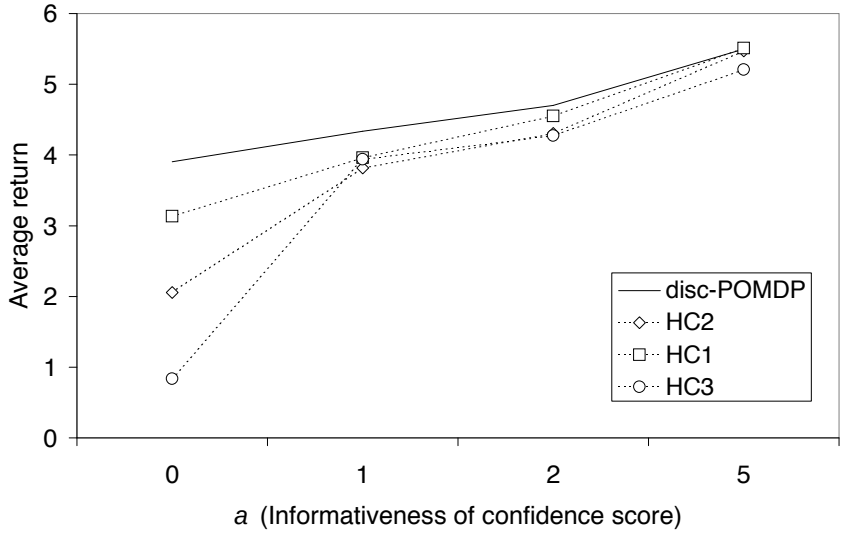

Figure 11. *Average return* vs. a (informativeness of confidence score) for $p_{err} = 0.30$ for discrete-POMDP and handcrafted policies executed with belief state monitoring.

6. Conclusions

This chapter has shown how a confidence score can be directly incorporated into the dialogue model represented as a Partially Observable Markov

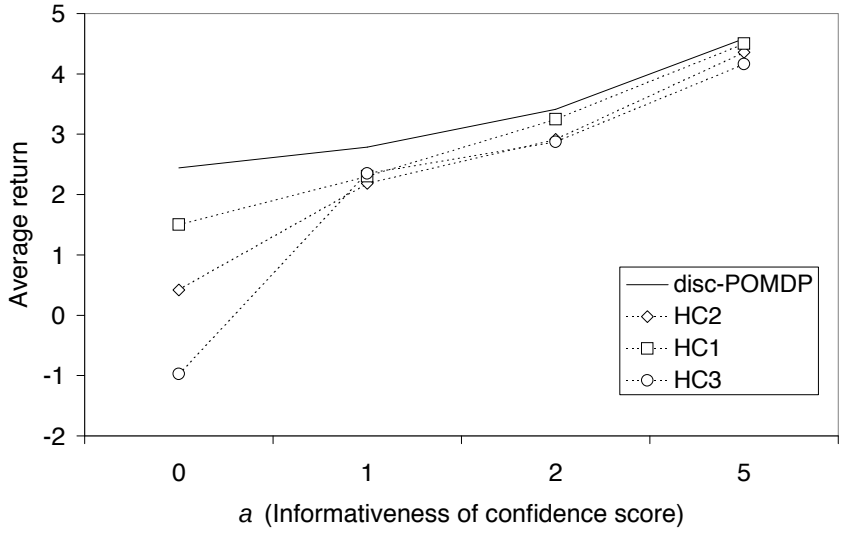

Figure 12. *Average return* vs. a (informativeness of confidence score) for $p_{err} = 0.40$ for discrete-POMDP and handcrafted policies executed with belief state monitoring.

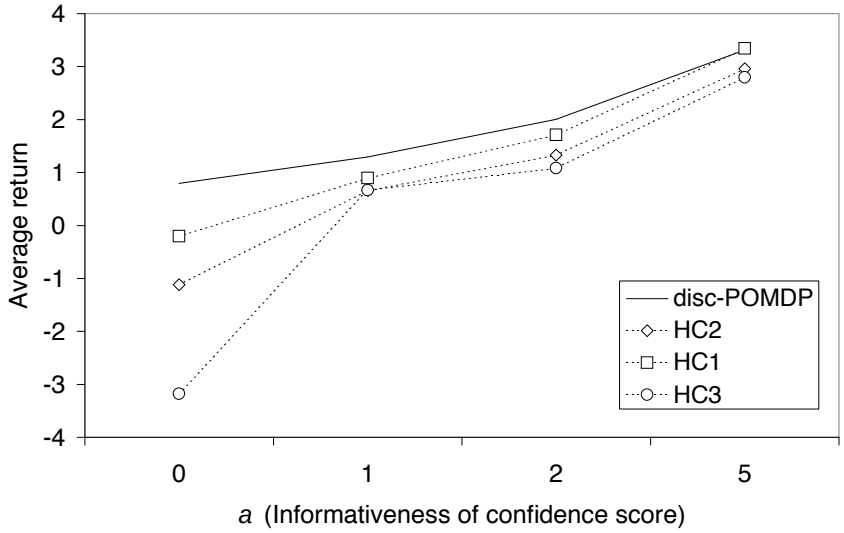

Figure 13. *Average return* vs. a (informativeness of confidence score) for $p_{err} = 0.50$ for discrete-POMDP and handcrafted policies executed with belief state monitoring.

Decision Process (POMDP) used for dialogue management. Unlike traditional approaches which maintain a single dialogue state at each time-step, in effect a POMDP considers all possible dialogue states, and maintains a probability distribution over these called a *belief state*. This representation allows a con-

fidence score to be tracked in the dialogue state in a principled fashion, and optimising the POMDP produces a dialogue manager which exploits this representation when selecting actions. In evaluation, the POMDP significantly outperforms a baseline MDP, which tracks only one hypothesis for the dialogue state.

This chapter has also presented a second approach to policy production in which a handcrafted controller which does not account for confidence score information can be improved to automatically account for confidence score information.

The problems considered here were unrealistically small for real-world deployment, and recent work has shown how to scale POMDPs to slot-filling problems of a realistic size (Williams and Young, 2005). Also, this chapter has considered only the top recognition hypothesis and confidence score. A natural extension would to be consider more complex hypothesis representations such as *N-Best* lists or word lattices, and more recognition features such as prosodic information, parse scores, or acoustic metrics.

Acknowledgements This work was supported in part by the European Union Framework 6 TALK Project (507802).

References

Hansen, E. A. (1998). Solving POMDPs by Searching in Policy Space. In *Proceedings of Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 211–219, Madison, Wisconsin, USA.

Hoey, J. and Poupart, P. (2005). Solving POMDPs with Continuous or Large Discrete Observation Spaces. In *Proceedings of the Joint International Conference on Artificial Intelligence (IJCAI)*, pages 1332–1338, Edinburgh, UK.

Jensen, F. (2001). *Bayesian Networks and Decision Graphs*. Springer Verlag.

Kaelbling, L., Littman, M., and Cassandra, A. (1998). Planning and Acting in Partially Observable Stochastic Domains. *Artificial Intelligence*, 101(1-2):99–134.

Larsson, S. and Traum, D. (2000). Information State and Dialogue Management in the Trindi Dialogue Move Engine Toolkit. *Natural Language Engineering*, 5(3-4):323–340.

Levin, E. and Pieraccini, R. (1997). A Stochastic Model of Computer-Human Interaction for Learning Dialogue Strategies. In *Proceedings of European Conference on Speech Communication and Technology (EUROSPEECH)*, pages 1883–1886, Rhodes, Greece.

Levin, E., Pieraccini, R., and Eckert, W. (2000). A Stochastic Model of Human-Machine Interaction for Learning Dialogue Strategies. *IEEE Transactions on Speech and Audio Processing*, 8(1):11–23.

Paek, T. and Horvitz, E. (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*, pages 95–100, Switzerland.

Pieracinni, R. and Huerta, J. (2005). Where do we Go from Here? Research and Commercial Spoken Dialog Systems (invited talk). In *Proceedings of SIGdial Workshop on Discourse and Dialogue*, pages 1–10, Lisbon, Portugal.

Pietquin, O. (2004). *A Framework for Unsupervised Learning of Dialogue Strategies*. PhD thesis, Faculty of Engineering, Mons, Belgium.

Roy, N., Pineau, J., and Thrun, S. (2000). Spoken Dialogue Management Using Probabilistic Reasoning. In *Proceedings of Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 93 – 100, Michigan, USA.

Singh, S., Litman, D., Kearns, M., and Walker, M. (2002). Optimizing Dialogue Management with Reinforcement Leaning: Experiments with the NJFun System. *Artificial Intelligence*, 16:105–133.

Spaan, M. and Vlassis, N. (2004). Perseus: Randomized Point-Based Value Iteration for POMDPs. Technical report, Informatics Institute, University of Amsterdam, The Netherlands.

Stuttle, M., Williams, J., and Young, S. (2004). A Framework for Dialogue Data Collection with a Simulated ASR Channel. In *Proceedings of International Conference on Spoken Language Processing (ICSLP)*, pages 241–244, Jeju Island, Korea.

Walker, M., Kamm, C., and Litman, D. (2000). Towards Developing General Models of Usability with PARADISE. *Natural Language Engineering*, 6:363–377.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. PhD thesis, Cambridge University, UK.

Williams, J. D., Poupart, P., and Young, S. (2005). Factored Partially Observable Markov Decision Processes for Dialogue Management. In *Proceeedings of 4th Workshop on Knowledge and Reasoning in Practical Dialog Systems, International Joint Conference on Artificial Intelligence (IJCAI)*, pages 76–82, Edinburgh, UK.

Williams, J. D. and Young, S. (2005). Scaling up POMDPs for Dialog Management: The "Summary POMDP" Method. In *Proceedings of IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 177–182, San Juan, Puerto Rico, USA.

Zhang, B., Cai, Q., Mao, J., Chang, E., and Guo, B. (2001). Spoken Dialogue Management as Planning and Acting under Uncertainty. In *Proceedings of European Conference on Speech Communication and Technology (EUROSPEECH)*, volume 2, pages 2169–2172, Aalborg, Denmark.