Factored Partially Observable Markov Decision Processes for Dialogue Management

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

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

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

Abstract

This work shows how a dialogue model can be represented as a factored Partially Observable Markov Decision Process (POMDP). The factored representation has several benefits, such as enabling more nuanced reward functions to be specified. Although our dialogue model is significantly larger than past work using POMDPs, experiments on a small testbed problem demonstrate that recent optimisation techniques scale well and produce policies which outperform a traditional fully-observable Markov Decision Process. This work then shows how a dialogue manager produced with a POMDP optimisation technique may be directly compared to a handcrafted dialogue manager. Experiments on the testbed problem show that automatically generated dialogue managers outperform several handcrafted dialogue managers, and that automatically generated dialogue managers for the testbed problem successfully adapt to changes in speech recognition accuracy.

1 Introduction

Creating (and improving) a dialogue manager by hand is typically an expensive and time-consuming undertaking. Instead of expressing which actions a machine should take in each dialogue situation, ideally a dialogue designer would simply express the desired *outcomes* of a dialogue. This specification would then be combined with a user model using a planning and optimisation algorithm to produce a dialogue manager. Markov Decision Processes (MDPs) provide a principled framework for this type of approach. The application of MDPs to the dialogue management problem is first explored by Levin and Pieraccini [1997]. Levin *et al*. [2000] provide a formal treatment of how a MDP may be applied to dialogue management, and Singh *et al*. [2002] show application to real systems.

MDPs assume the current state of the environment (i.e., the conversation) is known exactly and do not naturally or precisely model "noisy" evidence from the speech recogniser. This limitation has prompted several dialogue management researchers to explore POMDPs, which naturally express uncertainty in the current state. 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 (e.g., parse score, acoustic confidence score, etc.)

Looking outside the (PO)MDP framework, Paek and Horvitz [2003] suggest using a dynamic 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 that greedily selects actions – i.e., which selects actions based only on immediate reward.[1] 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.

In previous work which has applied POMDPs to dialogue management, three important issues are not addressed. First, it is unclear in these models how to estimate the system dynamics in practice. For example, Zhang *et al*. [2001] indicate that the system dynamics are "handcrafted, depending a lot on the experience of the developer." Second, neither model includes a notion of "dialogue state," and as a result, the reward functions in these models cannot capture the notion of "appropriateness" of an action – for example, the relative appropriateness of confirming vs. querying a slot value.[2] Finally, although handcrafted dialogue managers are often used as a baseline comparison in dialogue system literature, the authors do not attempt a comparison with a handcrafted dialogue manager.

This paper makes two contributions. First, we propose a factored architecture for describing a POMDP-based dialogue manager. Unlike past work applying POMDPs

[1] We can express this formally as a POMDP with discount $\gamma = 0$. See section 2 for background on POMDPs.

[2] Zhang *et al.* [2001] included unobservable states for possible causes of dialog trouble – for example, "channel errors." By contrast, in this work, we're interested in the conventional sense of "dialogue state" *as viewed by the user* – for example, which items have been confirmed.

(and MDPs) to dialogue management, our factored representation adds a component for the state of the dialogue *from the perspective of the user*, enabling dialogue designers to add reward measures for the "appropriateness" of system actions. The factored representation also creates separate distributions for the user model and the speech recognition model, which facilitates estimating or adapting the system dynamics from dialogue data. Although the scope of our model results in a much larger model than past POMDP work on the dialogue management problem, we show (using a simple testbed problem) that the recently developed *Perseus* algorithm [Spaan and Vlassis, 2004] scales sufficiently to optimize our model and finds a policy which outperforms an MDP baseline.

Second, we show how to make direct comparisons between a hand-crafted and an automatically generated policy. We demonstrate this technique by introducing three hand-crafted dialogue managers for the testbed problem, and find that a dialogue manager created with an automated technique outperforms all of them.

The paper is organised as follows. Section 2 briefly reviews background on POMDPs. Section 3 presents the factored architecture. Section 4 shows an example testbed system using this architecture. Section 5 compares the testbed system to an MDP baseline, and assesses robustness in the face of changing speech recognition accuracy. Section 6 shows how a handcrafted policy can be compared to an automatically-generated policy, and makes this comparison for the testbed problem. Section 7 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,[3] T defines a transition probability $p(s' \mid s, a_m)$, R defines the expected (immediate, real-valued) reward $r(s, a_m)$, O is a set of observations, and Z defines an observation probability, $p(o' \mid s', a_m)$.

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 dependant 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_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:

$$\rho(b_t, a_{m_t}) = \sum_{s \in S} b_t(s) r(s, a_{m_t}) . \qquad (1)$$

The goal of the machine is to maximise the cumulative, infinite-horizon, discounted reward called the *return*:

$$\sum_{t=0}^{\infty} \gamma^t \rho(b_t, a_{m_t}) = \sum_{t=0}^{\infty} \gamma^t \sum_{s \in S} b_t(s) r(s, a_{m_t}) . \qquad (2)$$

where γ is a geometric discount factor, $0 \le \gamma \le 1$. At each time step, the next belief state $b'(s')$ can be computed exactly as shown in Eq. 12 below.

Because belief space is real-valued, an optimal infinite-horizon policy may consist of an arbitrary partitioning of S-dimensional space. In fact, the size of the policy space grows exponentially with the size of the observation set and doubly exponentially with the distance (in time-steps) from the horizon [Kaelbling *et al*., 1998]. Nevertheless, real-world problems often possess small policies of high quality.

In this work, we make use of a recent *approximate* method called *Perseus*. *Perseus* [Spaan and Vlassis, 2004] 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 belief space, thereby 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].

3 Factored architecture

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

First, 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 state 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 travel itinerary, a request for information about a calendar, or a product the user would like to purchase.

The user's action, a_u, gives the user's most recent user's *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.

The state of the dialogue s_d indicates any relevant dialogue state information from the perspective of the user. 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.[4]

[3] 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.

[4] Future work will explore how a speech recognition confidence score can be incorporated in a principled way.

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 they want to go "to", or confirming that the user wants to leave "from" a specific place. The POMDP observation *o* is drawn from the same set as a_u, i.e., $o \in A_u$. Note that at each time step the POMDP receives a single observation, but maintains a distribution over all possible user actions.

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

$$\begin{aligned} p(s' \mid s, a_m) &= p(s'_u, s'_d, a'_u \mid s_u, s_d, a_u, a_m) \quad (3) \\ &= p(s'_u \mid s_u, s_d, a_u, a_m) \cdot \\ &\quad p(a'_u \mid s'_u, s_u, s_d, a_u, a_m) \cdot \\ &\quad p(s'_d \mid a'_u, s'_u, s_u, s_d, a_u, a_m). \quad (4) \end{aligned}$$

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:

$$p(s'_u \mid s_u, s_d, a_u, a_m) = p(s'_u \mid s_u, a_m) . \quad (5)$$

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:

$$p(a'_u \mid s'_u, s_u, s_d, a_u, a_m) = p(a'_u \mid s'_u, a_m) . \quad (6)$$

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

$$p(s'_d \mid a'_u, s'_u, s_u, s_d, a_u, a_m) = p(s'_d \mid a'_u, s_d, a_m) . \quad (7)$$

In sum, our transition function is given by:

$$\begin{aligned} p(s' \mid s, a_m) &= p(s'_u \mid s_u, a_m) \cdot \\ &\quad p(a'_u \mid s'_u, a_m) \cdot \quad (8) \\ &\quad p(s'_d \mid a'_u, s_d, a_m). \end{aligned}$$

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.[5] We could then separately specify the dialogue model using a handcrafted function such as "Information State" update rules as in for example [Larsson and Traum, 2000].

The observation function is given by:

$$p(o' \mid s', a_m) = p(o' \mid s'_u, s'_d, a'_u, a_m) . \quad (9)$$

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:[6]

$$p(o' \mid s'_u, s'_d, a'_u, a_m) = p(o' \mid a'_u) = p(o \mid a_u) . \quad (10)$$

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). Figure 1 shows an influence diagram of our proposal.

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.

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

$$\begin{aligned} b'(s') &= p(s' \mid o', a_m, b) \\ &= \frac{p(o' \mid s', a_m, b)\, p(s' \mid a_m, b)}{p(o' \mid a_m, b)} \\ &= \frac{p(o' \mid s', a_m) \sum_{s \in S} p(s' \mid a_m, b, s)\, p(s \mid a_m, b)}{p(o' \mid a_m, b)} \\ &= \frac{p(o' \mid s', a_m) \sum_{s \in S} p(s' \mid a_m, s) b(s)}{p(o' \mid a_m, b)} \quad (11) \end{aligned}$$

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 normalisation factor; hence:

$$b'(s') = k \cdot p(o' \mid s', a_m) \sum_{s \in S} p(s' \mid a_m, s) b(s) . \quad (12)$$

Substituting equation (8) and (10) into (12) and simplifying, we can write:

[5] 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].

[6] 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.

$$b'(s_u', s_d', a_u') = k \cdot p(o' \mid a_u') p(a_u' \mid s_u', a_m) \cdot \sum_{s_u \in S_u} p(s_u' \mid s_u, a_m) \cdot \sum_{s_d \in S_d} p(s_d' \mid a_u', s_d, a_m) \cdot \sum_{a_u \in A_u} b(s_u, s_d, a_u). \quad (13)$$

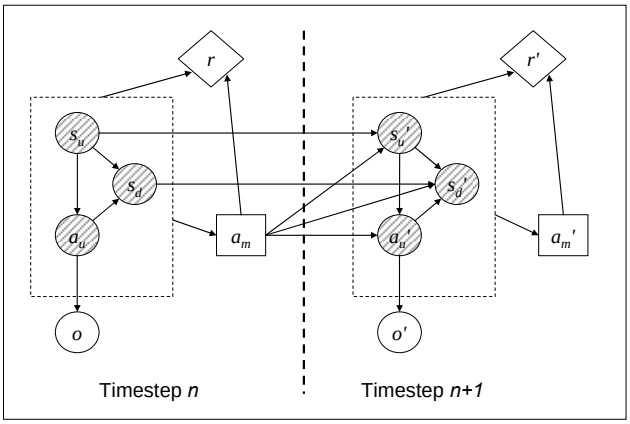

Figure 1: Influence diagram for the factored model. The dotted box indicates the composite state *s* is comprised of three components, s_u, s_d, and a_u. Shading indicates a component is unobservable. Arcs into circular chance nodes and diamond-shaped utility nodes show influence, whereas arcs into square decision nodes are informational (see Jensen [2001], p140).

4 Testbed spoken dialogue system

To test the ideas in our proposal, we created a simulated dialogue management problem in the travel domain in which 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" the ticket purchase request, ending the dialogue. The machine may also choose to "fail". In the testbed 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 $x, y \in \{a,b,c\}, x \neq y$. 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 $s_u \in (x,y), x, y \in \{a,b,c\}, x \neq y$. 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 dialogue states, given by:

$$s_d \in (x_d, y_d, z); \quad x_d, y_d \in \{n,u,c\}, z \in \{1,0\} \quad (14)$$

The user's action $a_u \in A_u$ and the observation $o \in A_u$ 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 testbed 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* 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 the authors' experience performing usability testing with slot-filling dialogue systems.[7] In future work, we intend to estimate a user model based on dialogue data.

We define the *dialogue model* 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*.

We define the observation function to encode the probability of making a speech recognition error to be p_{err}, and define the observation function as:

$$p(o \mid a_u) = \begin{cases} 1 - p_{err} & \text{if } o = a_u \\ \dfrac{p_{err}}{|A_u| - 1} & \text{if } o \neq a_u \end{cases} \quad (15)$$

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

The reward measure 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.

The *Perseus* algorithm requires two parameters: 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}.

[7] Because of space limitations, the detail of this distribution isn't shown here.

5 Testbed evaluation

5.1 Comparison with an MDP Baseline

To test whether an automated solution to the POMDP is both feasible and worthwhile, we created an MDP-based dialogue manager baseline, patterned on systems in the literature (e.g., [Pietquin, 2004]). The MDP is trained and evaluated through interaction with a model of the environment, which is formed of the POMDP transition, observation, and reward functions. This model of the environment takes an action from the MDP as input, and emits an observation and a reward to the MDP as output.

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. Two additional states – *dialogue-start* and *dialogue-end* – which were also in the POMDP state space, are included in the MDP state space for a total of 11 MDP states.

An MDP state estimator maps from POMDP observation to MDP state, and from MDP action to POMDP action. For example, 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, which updates the environment model. The MDP state estimator tracks the most recent value observed for a slot, enabling it to map from an MDP action like *confirm-from* to a POMDP action like *confirm-from-a* or an MDP action like *submit* to *submit-from-a-to-b*. This behaviour of the MDP state estimator is identical to that used in the MDP/spoken dialogue system literature (e.g., [Pietquin, 2004] and [Levin et al., 2000]).

Because the MDP learns through experience with a simulated environment, we selected an on-line learning technique, Watkins Q-learning, 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.). To evaluate the resulting MDP policy, 10,000 dialogs were simulated using the learned policy.

Figure 2 shows expected return for the POMDP solution, and the average return for the MDP solutions vs. p_{err} ranging from 0.00 to 0.65. The (negligible) 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 solution attains the largest return of the solutions at all values of p_{err}. Further, the performance gain of the POMDP solution over the other solutions increases as p_{err} increases. From this result we conclude that the POMDP solution copes with higher speech recognition error rates better than the MDP approach, consistent with [Roy *et al*., 2000].

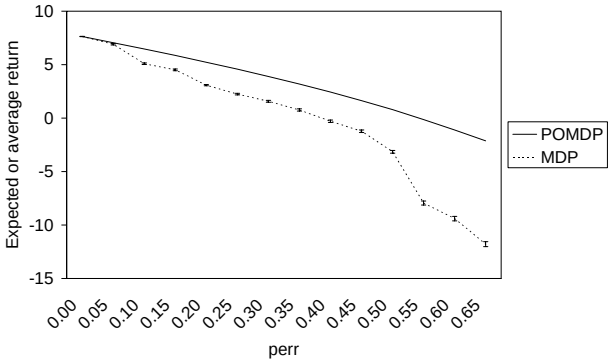

Figure 2: Expected or average return of POMDP policies and MDP baseline. Error bars show 95% confidence interval.

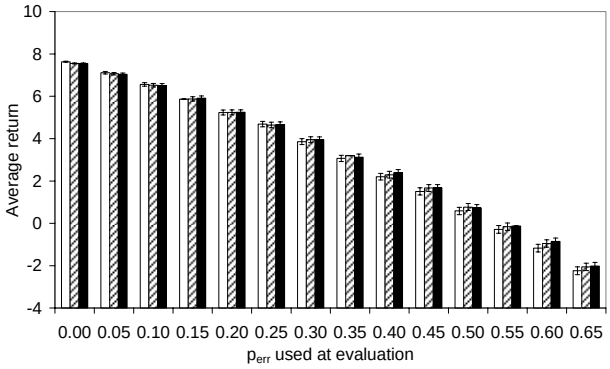

Figure 3: Performance of POMDP policies vs. p_{err}. White bars show a policy trained using p_{err}=0.15, checked bars p_{err}=0.35, and shaded bars p_{err}=0.55. Error bars show 95% confidence interval.

5.2 Robustness to changes in error rate

In practice, the error rate of a spoken dialogue system varies from user to user. Thus we were interested to see how a POMDP solution adapts to a value of p_{err} for which it was not designed. Figure 3 shows average return for three dialogue managers when executed using a different value for p_{err}. Error bars show 95% confidence interval for true average return sampled over 10,000 dialogues. From this we see that the POMDP solutions are not "brittle" – i.e., they do not fail catastrophically as p_{err} deviates from that used in training.

6 Comparison with a handcrafted policy

6.1 Method to evaluate a handcrafted policy

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 $\upsilon_n(s)$ to indicate the *sth* component of the *nth* 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 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 all timesteps as:

$$\pi(b) = \hat{\pi}\left(\arg\max_{n} \sum_{s=1}^{|S|} \upsilon_n(s)\, b(s) \right) \quad (16)$$

Thus the value-function method provides both a partitioning of belief space into regions corresponding to optimal actions as well as the expected return of taking that action. 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. 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 by Hansen [1998], we can find the expected return associated with each controller node by solving this system of linear equations in υ:

$$\upsilon_n(s) = r(s, \hat{\pi}(n)) + \gamma \sum_{s' \in S} \sum_{o \in O} p(s' \mid s, \hat{\pi}(n))\, p(o \mid s', \hat{\pi}(n))\, \upsilon_{l(n,o)}(s') \quad (17)$$

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:

$$\sum_{s=1}^{|S|} \upsilon_n(s) b(s) \quad (18)$$

6.2 Example handcrafted policies and results

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, it re-tries the same action. If it receives an observation which is inconsistent or nonsensical, it re-tries the same action. Once it fills both fields, it takes the corresponding *submit-x-y* action. A logical diagram showing *HC1* is shown in Figure 4.[8]

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 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.

Figure 5 shows the expected return for the handcrafted policies and the optimised POMDP solution. The POMDP solution outperforms all of the handcrafted policies for all values of p_{err}.

We inspected the POMDP solution in order to characterise how it differs from the handcrafted solutions. Conceptually, the POMDP policy differs from the handcrafted policies in that it tracks conflicting evidence rather than discarding it. For example, whereas the POMDP policy can interpret the "best 2 of 3" observations for a given slot, the handcrafted policies can maintain only 1 hypothesis for each slot.

As an illustration, consider an environment with no uncertainty – i.e., no speech recognition errors. In this environment, there is no benefit to maintaining multiple hypotheses for a user goal, and thus we would expect a POMDP to perform identically to a policy which does not track multiple hypotheses for a user goal. Figure 5 demonstrates this point: where $p_{err} = 0$, HC1 and HC2 perform identically to the POMDP policy.[9]

It is interesting to note that *HC3*, which confirms all inputs, performs least well for all values of p_{err}. For the reward function we have provided in the testbed system, requiring 2 consistent recognition results (the response to *ask* and the response to *confirm*) gives rise to longer dialogs which outweigh the benefit of the increase in certainty

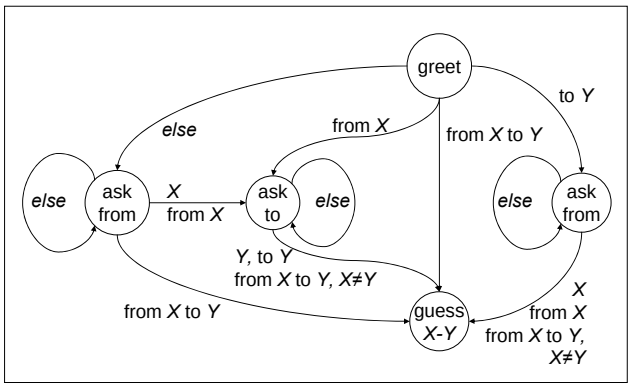

Figure 4: *HC1* handcrafted controller

[8] 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.

[9] HC3 performs worse because it confirms each element, lengthening the dialogue and thus reducing return.

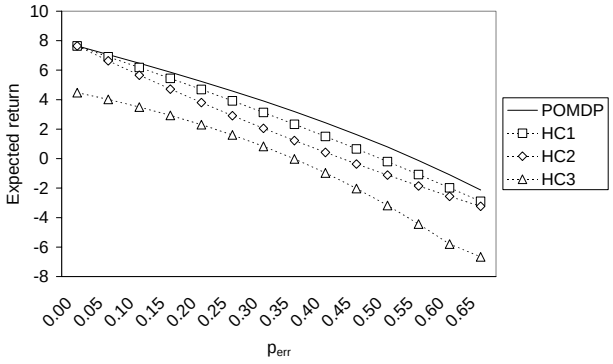

Figure 5: Expected return vs. p_{err} for POMDP policy and 3 handcrafted policies.

7 Conclusion

We have proposed a factored architecture for describing POMDPs applied to spoken dialogue management. The factored representation is useful for two reasons – first, it facilitates estimating or specifying the system dynamics by reducing the number of parameters, and enabling different aspects of the system dynamics to be specified independently. Second, it enables incorporation of an explicit dialogue model from the user's standpoint, which allows a dialogue designer to add rewards for "appropriate" dialogue behaviour. Further, we have 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. Our model is much larger than past POMDP dialogue managers; however, using our testbed problem, we have shown that a recent POMDP optimisation technique finds policies which outperform both an MDP baseline and three handcrafted controllers over all operating conditions. Further, the POMDP solution appears to adapt to changes in speech recognition error rate well.

A crucial theoretical issue is how to scale this model to handle larger problems since the state, action, and observation sets grow exponentially with the number of concepts in the problem. Although we have not used the factored representation to assist the optimisation process in this work, it may be possible to exploit the factoring to make the optimisation algorithms more efficient.

8 Acknowledgement

The work reported in this paper was supported by the EU FP6 Talk Project.

References

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

[Jensen, 2001]. Finn V. Jensen. *Bayesian Networks and Decision Graphs.* New York: Springer Verlang, 2001.

[Kaelbling et al., 1998] Leslie Pack Kaelbling, Michael L. Littman and Anthony R. Cassandra. Planning and Acting in Partially Observable Stochastic Domains. *Artificial Intelligence*, Vol. 101, 1998.

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

[Levin et al., 2000] Esther Levin, Roberto Pieraccini, and Wieland Eckert. A Stochastic Model of Human-Machine Interaction for Learning Dialogue Strategies. *IEEE Transactions on Speech and Audio Processing*, Volume 8, No. 1, 11-23, 2000.

[Levin and Pieraccini, 1997] Esther Levin and Roberto Pieraccini. A Stochastic Model of Computer-Human Interaction For Learning Dialogue Strategies. *Eurospeech*, Rhodes, Greece, 1997.

[Paek and Horvitz, 2000] Tim Paek and Eric Horvitz. Conversation as Action Under Uncertainty. In *Proc. Uncertainty in Artificial Intelligence (UAI)*, Stanford, CA, June 2000.

[Pietquin, 2004] Olivier Pietquin. *A Framework for Unsupervised Learning of Dialogue Strategies.* Ph D thesis, Faculty of Engineering, Mons, Belgium, 2004.

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

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

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

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

[Walker et al., 2000] Marilyn A. Walker, Candace Kamm, and Diane Litman. Towards Developing General Models of Usability with PARADISE. *Natural Language Engineering*, Vol. 6, No. 3, 2000.

[Zhang et al., 2001] Zhang Bo, Cai Qingsheng, Mao Jianfeng, and Guo Baining. Planning and Acting under Uncertainty: A New Model for Spoken Dialogue System. *Proceedings of the 17th Annual Conference on Uncertainty in Artificial Intelligence (UAI-01).* San Francisco, USA, 2001.