The Hidden Information State Model: a practical framework for POMDP-based spoken dialogue management

Steve Young * , Milica Gašić, Simon Keizer, François Mairesse, Jost Schatzmann, Blaise Thomson, Kai Yu

Cambridge University Engineering Department, Trumpington Street, Cambridge, CB2 1PZ, UK

Abstract

This paper explains how Partially Observable Markov Decision Processes (POMDPs) can provide a principled mathematical framework for modelling the inherent uncertainty in spoken dialogue systems. It briefly summarises the basic mathematics and explains why exact optimisation is intractable. It then describes in some detail a form of approximation called the *Hidden Information State model* which does scale and which can be used to build practical systems. A prototype HIS system for the tourist information domain is evaluated and compared with a baseline MDP system using both user simulations and a live user trial. The results give strong support to the central contention that the POMDP-based framework is both a tractable and powerful approach to building more robust spoken dialogue systems.

Key words: Statistical dialogue systems, POMDP, Hidden Information State model

1 Introduction

Spoken dialogue systems allow a human user to interact with a machine using voice as the primary communication medium. The structure of a conventional spoken dialogue system (SDS) is shown in Fig 1(a). It contains three major components: speech understanding, speech generation and dialogue management. The speech understanding component typically consists of a speech

* Corresponding author.
Email address: sjy@eng.cam.ac.uk (Steve Young).

recogniser and semantic decoder, and its function is to map user utterances into some abstract representation of the user's intended speech act a_u. The speech generation component consists of a natural language generator and a speech synthesiser and it performs the inverse operation of mapping the machine's response a_m back into speech.

The core of the dialogue manager is a data structure which represents the system's view of the world in the form of a machine state s_m. This machine state typically encodes an estimate of three distinct sources of information: the user's input act $\tilde{a}_u$, an estimate of the intended user goal $\tilde{s}_u$ [1] and some record of the dialogue history $\tilde{s}_d$ [2]. Most conventional dialogue managers rely on hand-crafted deterministic rules for interpreting each (noisy) user dialogue act $\tilde{a}_u$ and updating the state. Based on each new state estimate, a dialogue policy is used to select an appropriate machine response in the form of a dialogue act a_m. This dialogue cycle continues until either the user's goal is satisfied or the dialogue fails.

The designers of such systems have to deal with a number of problems. Since the user's state s_u is unknown and the decoded inputs $\tilde{a}_u$ are prone to errors [3], there is a significant chance that $\tilde{s}_m$ will be incorrect. Hence, the dialogue manager must include quite complex error recovery procedures. Recognition confidence scores can reduce the incidence of misunderstandings but these require thresholds to be set which are themselves notoriously difficult to optimise. Modern recognisers can produce alternative recognition hypotheses but it is not clear in practice how these can be used effectively. Finally, the impact of decisions taken by the dialogue manager do not necessarily have an immediate effect, hence dialogue optimisation requires forward planning and this is extremely difficult in a deterministic framework.

As has been argued previously, taking a statistical approach to spoken dialogue system design provides the opportunity for solving many of the above problems in a flexible and principled way (Young, 2002). Early attempts at using a statistical approach modelled the dialogue system as a Markov decision process (MDP) (Levin et al., 1998, 2000; Young, 2000). MDPs provide a good statistical framework since they allow forward planning and hence dialogue policy optimisation through reinforcement learning (Sutton and Barto, 1998). However, MDPs assume that the entire state is observable. Hence, they cannot account for either the uncertainty in the user state $\tilde{s}_u$ and dialogue history $\tilde{s}_d$, or the uncertainty in the decoded user's dialogue act $\tilde{a}_u$.

1 Examples of user goals are "finding flight information between London and New York", "finding a Chinese restaurant near the centre of town", "ordering three Pepperoni pizza's", etc.
2 Since both a_u and s_u are noisy, the record of dialogue history is also noisy, hence the tilde on s_d.
3 Word error rate (WER) is typically in the 10% to 30% range.

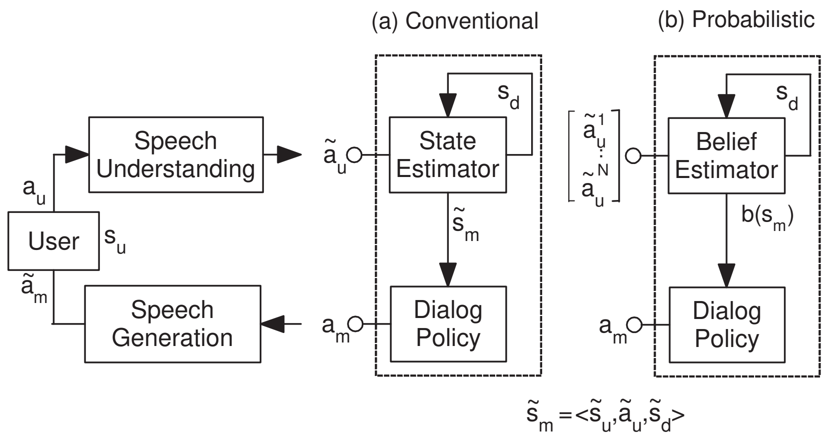

Fig. 1. Structure of a spoken dialogue system: a_u and a_m denote user and machine dialogue acts, s_u is the user goal and s_d is the dialogue history. The tilde indicates an estimate. Part (a) shows a conventional dialogue manager which maintains a single state estimate; (b) shows a dialogue manager which maintains a distribution over all states and accepts an N-best list of alternative user inputs.

Fig 1(b) shows an alternative model for the dialogue management component in which the uncertainty in the user's dialogue act and the uncertainty in the machine state are shown explicitly. In this new model, the state estimator maintains a distribution across all states rather than a point-estimate of the most likely state. The dialogue manager therefore tracks all possible dialogue paths rather than just the most likely path. The ensuing dialogue decision is then based on the distribution over all dialogue states rather than just a specific state. This allows competing hypotheses to be considered in determining the machine's next move and simplifies error recovery since the dialogue manager can simply shift its attention to an alternative hypothesis rather than trying to repair the existing one.

If the decoded user input act is regarded as an observation, then the dialogue model shown in Fig 1(b) is a Partially Observable MDP (POMDP) (Kaelbling et al., 1998). The distribution over dialogue states is called the *belief state* b and dialogue policies are based on b rather than the estimated state. The key advantage of the POMDP formalism is that it provides a complete and principled framework for modelling the inherent uncertainty in a spoken dialogue system. Thus, it naturally accommodates the implicit uncertainty in the estimate of the user's goal and the explicit uncertainty in the N-best list of decoded user acts. Associated with each dialogue state and machine action is a reward. The choice of reward function is a dialogue design issue, but it will typically provide positive rewards for satisfying the user's goal, and negative rewards for failure and wasting time. As with regular MDPs, dialogue optimisation is equivalent to finding a decision policy which maximises the total reward.

The use of POMDPs for any practical system is, however, far from straightforward. Firstly, in common with MDPs, dialogue states are complex and hence the full state space of a practical SDS would be intractably large. Secondly, since a belief distribution b over a discrete state s of cardinality $n + 1$ lies in a real-valued n-dimensional simplex, a POMDP is equivalent to an MDP with a continuous state space $b \in \Re^n$. Thus, a POMDP policy is a mapping from regions in n-dimensional belief space to actions. Not surprisingly these are extremely difficult to construct and whilst exact solution algorithms do exist, they do not scale to problems with more than a few states/actions.

There are two broad approaches to achieving a practical and tractable implementation of a POMDP-based dialogue system. Firstly, the state can be factored into a number of simple discrete components. It then becomes feasible to represent probability distributions over each individual factor. The most obvious examples of these are so-called *slot filling* applications where the complete dialogue state is reduced to the state of a small number of slots that require to be filled (Williams and Young, 2007a,b). For more complex applications, the assumption of independence between slots can be relaxed somewhat by using dynamic Bayesian Networks (Thomson et al., 2008b,a). Provided that each slot or network node has only a few dependencies, tractable systems can be built and belief estimates maintained with acceptable accuracy using approximate inference (Bishop, 2006).

A second approach to approximating a POMDP-based dialogue system is to retain a full and rich state representation but only maintain probability estimates over the most likely states. Conceptually, this approach can be viewed as maintaining a set of dialogue managers executing in parallel where each dialogue manager follows a distinct path. At each dialogue turn, the probability of each dialogue manager representing the true state of the dialogue is computed and the system response is then based on the probability distribution across all dialogue managers. This viewpoint is interesting because it provides a migration path for current dialogue system architectures to evolve into POMDP-based architectures (Henderson and Lemon, 2008).

This paper describes a specific implementation of the second approach called the Hidden Information State (HIS) model. The HIS system uses a full state representation in which similar states are grouped into partitions and a single belief is maintained for each partition. The system typically maintains a distribution of upto several hundred partitions corresponding to many thousands of dialogue states. The HIS system has been described in outline in a number of conference papers (Young, 2006; Young et al., 2007; Gašić et al., 2008). The aim of this paper is to provide a single coherent and more detailed description of how the HIS system works, and an assessment of its performance characteristics. The paper is structured as follows. Section 2 reviews the theory of POMDP-based dialogue management in general and then gives the specific theory underlying the HIS system. Section 3 explains how the various probability models in the HIS system are implemented and section 4 deals with policy representation and optimisation. Section 5 describes how HIS systems are trained using a user simulator and section 6 presents experimental results. The paper ends in section 7 with our conclusions.

2 POMDPs for Dialogue Management

2.1 POMDP Basics

Formally, a POMDP is defined as a tuple $\{S_m, A_m, T, R, O, Z, \lambda, b_0\}$ where S_m is a set of machine states; A_m is a set of actions that the machine may take; T defines a transition probability $P(s'_m|s_m, a_m)$; R defines the expected immediate reward $r(s_m, a_m)$; O is a set of observations; Z defines an observation probability $P(o'|s'_m, a_m)$; λ is a geometric discount factor $0 \leq \lambda \leq 1$; and b_0 is an initial belief state. [4]

A POMDP operates as follows. At each time-step, the machine is in some unobserved state $s_m \in S_m$. Since s_m is not known exactly, a distribution over states is maintained called a belief state such that the probability of being in state s_m given belief state b is $b(s_m)$ [5]. Based on the current belief state b, the machine selects an action $a_m \in A_m$, receives a reward $r(s_m, a_m)$, and transitions to a new (unobserved) state s'_m, where s'_m depends only on s_m and a_m. The machine then receives an observation $o' \in O$ which is dependent on s'_m and a_m. Finally, the belief distribution b is updated based on o' and a_m as follows

$$\begin{aligned}
b'(s'_m) &= P(s'_m|o', a_m, b) \\
&= \frac{P(o'|s'_m, a_m, b)P(s'_m|a_m, b)}{P(o'|a_m, b)} \\
&= \frac{P(o'|s'_m, a_m)\sum_{s_m \in S_m} P(s'_m|a_m, b, s_m)P(s_m|a_m, b)}{P(o'|a_m, b)} \\
&= k \cdot P(o'|s'_m, a_m) \sum_{s_m \in S_m} P(s'_m|a_m, s_m)b(s_m) \qquad (1)
\end{aligned}$$

where $k = 1/P(o'|a_m, b)$ is a normalisation constant (Kaelbling et al., 1998). Maintaining this belief state as the dialogue evolves is called *belief monitoring*.

[4] Here and elsewhere, primes are used to denote the state of a variable at time $t+1$ given that the unprimed version is at time t.

[5] In other words, a belief state b is a vector whose component values give the probabilities of being in each machine state.

At each time step t, the machine receives a reward $r(b_t, a_{m,t})$ based on the current belief state b_t and the selected action $a_{m,t}$. The cumulative, infinite horizon, discounted reward is called the *return* and it is given by

$$R = \sum_{t=0}^{\infty} \lambda^t r(b_t, a_{m,t}) = \sum_{t=0}^{\infty} \lambda^t \sum_{s_m \in S_m} b_t(s_m) r(s_m, a_{m,t}). \tag{2}$$

Each action $a_{m,t}$ is determined by a policy $\pi(b_t)$ and building a POMDP system involves finding the policy π^* which maximises the return. Unlike the case of MDPs, the policy is a function of a continuous multi-dimensional variable and hence its representation is not straightforward. However, it can be shown that for finite horizon problems the value function of the optimal policy is piecewise linear and convex in belief space (Sondik, 1971). Hence, it can be represented by a set of *policy vectors* where each vector v_i is associated with an action $a(i) \in A_m$ and $v_i(s)$ equals the expected value of taking action $a(i)$ in state s. Given a complete set of policy vectors, the optimal value function and corresponding policy is

$$V^{\pi^*}(b) = \max_i \{v_i.b\} \tag{3}$$

and

$$\pi^*(b) = a(\operatorname*{argmax}_i \{v_i.b\}). \tag{4}$$

This representation is illustrated in Fig 2(a) for the case of $|S_m| = 2$ and a value function requiring just 3 distinct linear segments. The value function itself is the upper heavy line. In this case, b is a 2-D vector such that $b_1 = 1 - b_2$, hence it can be denoted by a single point on the horizontal axis. The linear segments divide belief space into 3 regions and the optimal action to take in each region is the action associated with the uppermost vector in that region. So for example, if $b < x$ in Fig. 2a, then action $a(1)$ would be chosen, if $x < b < y$ then action $a(2)$ would be chosen, and so on.

The optimal exact value function can be found by working backwards from the terminal state in a process called *value iteration*. At each iteration t, policy vectors are generated for all possible action/observation pairs and their corresponding values are computed in terms of the policy vectors at step $t-1$. As t increases, the estimated value function converges to the optimal value function from which the optimal policy can be derived. Many spurious policy vectors are generated during this process, and these can be pruned to limit the combinatorial explosion in the total number of vectors (Kaelbling et al., 1998; Littman, 1994). Unfortunately, this pruning is itself computationally expensive and in practice, exact optimisation is not tractable. However, approximate

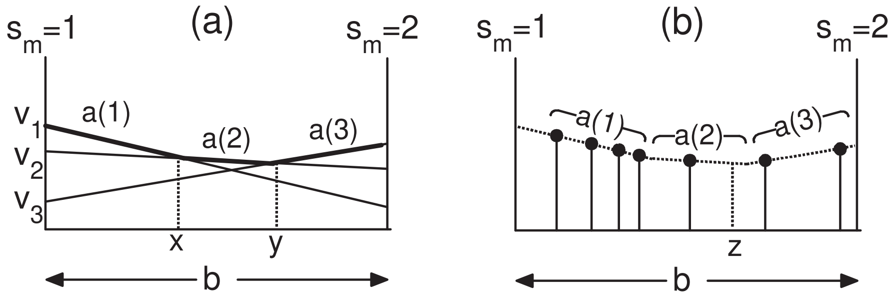

Fig. 2. POMDP Value function representation: (a) shows exact an representation; (b) shows a grid-based representation.

solutions can still provide useful policies. The simplest approach is to discretise belief space and then use standard MDP optimisation methods (Sutton and Barto, 1998). Since belief space is potentially very large, grid points are concentrated on those regions which are likely to be visited (Brafman, 1997; Bonet, 2002). This is illustrated in Fig 2(b). Each belief point represents the value function at that point and it will have associated with it the corresponding optimal action to take. When an action is required for an arbitrary belief point b, the nearest belief point is found and its action is used. However, this can lead to errors and hence the distribution of grid points in belief space is very important. For example, in Fig. 2(b), if $b = z$ then $a(3)$ would be selected although from Fig. 2(a) it can be seen that the optimal action was actually $a(2)$.

Grid-based methods are often criticised because they do not scale well to large state spaces and hence methods which support interpolation between points are often preferred (Pineau et al., 2003). However, the HIS model described below avoids the scaling problem by mapping the full belief space into a much reduced summary space where grid-based approximations appear to work reasonably well.

2.2 The SDS POMDP

As discussed in the introduction, when using a POMDP to model a spoken dialogue system, it is natural to factor the machine state into three components $s_m = \langle s_u, a_u, s_d \rangle$ (Williams et al., 2005)[6]. The belief state b is then a distribution over these three components.

The transition function for an SDS-POMDP follows directly by substituting the factored state into the regular POMDP transition function and making

[6] Note that alternative POMDP formulations can also be used for SDS eg (Roy et al., 2000; Zhang et al., 2001).

some reasonable independence assumptions, i.e.

$$P(s'_m|s_m, a_m) = P(s'_u, a'_u, s'_d|s_u, a_u, s_d, a_m) \\ \approx P(s'_u|s_u, a_m)P(a'_u|s'_u, a_m)P(s'_d|s'_u, a'_u, s_d, a_m). \quad (5)$$

This is the *transition model.* Making similar reasonable independence assumptions regarding the observation function gives

$$P(o'|s'_m, a_m) = P(o'|s'_u, a'_u, s'_d, a_m) \approx P(o'|a'_u). \quad (6)$$

This is the *observation model.*

The above factoring simplifies the belief update equation since substituting (5) and (6) into (1) gives

$$b'(s'_u, a'_u, s'_d) = k \cdot \underbrace{P(o'|a'_u)}_{\substack{\text{observation}\\ \text{model}}} \underbrace{P(a'_u|s'_u, a_m)}_{\substack{\text{user action}\\ \text{model}}} \sum_{s_u} \underbrace{P(s'_u|s_u, a_m)}_{\substack{\text{user goal}\\ \text{model}}} \\ \cdot \sum_{s_d} \underbrace{P(s'_d|s'_u, a'_u, s_d, a_m)}_{\substack{\text{dialogue history}\\ \text{model}}} b(s_u, s_d). \quad (7)$$

As shown by the labelling in (7), the probability distribution for a'_u is called the *user action model.* It allows the observation probability that is conditioned on a'_u to be scaled by the probability that the user would speak a'_u given the goal s'_u and the last system prompt a_m. The observation o is typically an N-best list of hypothesised user acts, each with an associated probability, i.e.

$$o = [\langle \tilde{a}^1_u, p_1\rangle, \langle \tilde{a}^2_u, p_2\rangle, \ldots, \langle \tilde{a}^N_u, p_N\rangle] \quad (8)$$

such that $p_n = P(\tilde{a}^n_u|o)$ for $n = 1 \ldots N$. Thus, the combination of the observation model with the user action model allows *a posterior* reranking of an N-best recognition output to be made based on the dialogue system's current beliefs. The *user goal model* determines the probability of the user goal switching from s_u to s'_u following the system prompt a_m. Finally, the *dialogue history model* enables information relating to the dialogue history to be maintained such as a grounding state. Fig 3 shows the SDS-POMDP in the form of a Bayesian network. The form of the statistical models used in the HIS system to represent the components of (7) are described in detail in section 3.

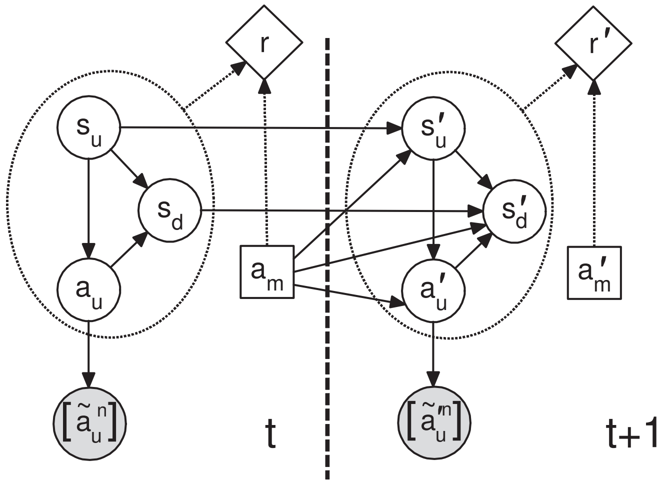

Fig. 3. SDS-POMDP dialogue framework as a Bayesian network. Solid arrows denote conditional dependencies, open circles denote hidden variables and shaded circles denote observations. The machine action a_m is a function of the belief distribution $b(s_u, a_u, s_d)$.

2.3 The HIS POMDP

The key idea underlying the HIS model is that at any point in a dialogue, most of the possible user goal states have identical beliefs simply because there has been no evidence offered by the user to distinguish them. For example, if the system believes that the user might have asked for a Chinese restaurant then it must record `food=Chinese` as a possible goal state, but there is no point in maintaining individual goal states for all of the other possible food types such as `food=Italian`, `food=French`, etc. which have not been mentioned since they are all equally likely. Significant computation can therefore be saved by grouping these states into equivalence classes. The HIS model therefore assumes that at any time t, the space of all user goals S_u can be divided into a number of equivalence classes $p \in P$ where the members of each class are tied together and are indistinguishable. These equivalence classes are called *partitions*. Initially, all states $s_u \in S_u$ are in a single partition p^0. As the dialogue progresses, this root partition is repeatedly split into smaller partitions. This splitting is binary i.e. $p \to \{p', p - p'\}$ with probability $P(p'|p)$.[7]

Since multiple splits can occur at each time step, this binary split assumption places no restriction on the possible refinement of partitions from one turn to the next. Given that user goal space is partitioned in this way, beliefs can be computed based on partitions of S_u rather than on the individual states of S_u. Initially the belief state is just $b^0(p^0) = 1$. Whenever a user state partition p

[7] The notation $p - p'$ should be read as denoting the partition containing all states in p except for those in p'.

is split, its belief mass is reallocated as

$$b(p') = P(p'|p)b(p) \text{ and } b(p - p') = (1 - P(p'|p))b(p). \tag{9}$$

Note that this splitting of belief mass is simply a reallocation of existing mass, it is not a belief update, rather it is *belief refinement*.

A further simplification made in the HIS model is to assume that user goals change rarely and when they do, they are relatively easy to detect. Hence, instead of assuming that the user goal can change every turn, the HIS model assumes that user goal changes will be explicitly identified by the dialogue policy decision process and signalled by explicit system responses (this is explained further in section 4.3 below). Normal turn-by-turn belief updating is therefore based on the assumption that

$$P(s'_u|s_u) = \delta(s'_u, s_u). \tag{10}$$

where $\delta(s'_u, s_u) = 1$ if $s'_u = s_u$ and 0 otherwise.

Substituting (9) and (10) into (7) gives the belief update equation for the HIS model

$$\begin{aligned} & b'(p', a'_u, s'_d) = \\ & k \cdot \underbrace{P(o'|a'_u)}_{\substack{\text{observation} \\ \text{model}}} \underbrace{P(a'_u|p', a_m)}_{\substack{\text{user action} \\ \text{model}}} \sum_{s_d} \underbrace{P(s'_d|p', a'_u, s_d, a_m)}_{\substack{\text{dialogue history} \\ \text{model}}} \underbrace{P(p'|p)b(p, s_d)}_{\substack{\text{belief} \\ \text{refinement}}} \end{aligned} \tag{11}$$

where p is the parent of p' and where s_d is the dialogue history shared by all states in partition p.

3 Implementation of the HIS Probability Models

The HIS system represents a complete dialogue state as a triple $\langle p, a_u, s_d \rangle$ representing a partition of equivalent user goal states, the last user dialogue act and a common dialogue history. This triple and its associated belief represents a hypothesis regarding the true (hidden) state of the dialogue system and the HIS system maintains a list of the M most likely hypotheses as an approximation to the full distribution. Each complete dialogue cycle consists of splitting any necessary partitions and updating their beliefs according to equation (11). In order to understand this process in more detail, it is first necessary to explain how goal states, partitions and dialogue acts are represented in the HIS system. The implementation of the three core probability models is then described. The section ends with a summary of how belief monitoring in the HIS system works in practice.

3.1 Dialogue Acts

Dialogue acts in the HIS system take the form *actt*($a_1 = v_1, a_2 = v_2, ..$) where *actt* denotes the type of dialogue act and the arguments are *act items* consisting of attribute-value pairs [8]. Attributes refer to nodes in the user goal state tree described below and values are the atomic values that can be assigned to those nodes. In some cases, the value can be omitted, for example, where the intention is to query the value of an attribute. The same dialogue act representation is used for both user inputs and the dialogue manager outputs. The most common acts are listed in Table 1 and a simple dialogue illustrating their use is shown in Table 2. A full description of the dialogue act set used by the HIS system is given in Young (2007).

Note that in the HIS system every utterance translates into a single dialogue act. When the speech understanding system is uncertain, the input to the dialogue manager is typically a list of alternative dialogue acts. For example, the utterance "I want an Italian place near the cinema" spoken in a noisy background might yield

```
inform(type=restaurant,food=Italian, near=cinema) {0.6}
inform(type=restaurant,food=Indian, near=cinema) {0.3}
inform(type=bar,near=cinema) {0.1}
```

where the number in braces is the probability of each dialogue act hypothesis. This list corresponds to the form of observation defined in (8).

3.2 User Goal State Partitioning

The HIS system is designed primarily for information retrieval tasks. However, compared to simple slot filling systems, it supports a much richer set of user goal representations based on tree-like structures built from classes, subtypes, and atomic values where a class represents a collection of related values and a subtype denotes a specific variant of a class. The space of all possible user goals is described by a set of simple ontological rules which define the various subtypes of each class and the atomic values which can be assigned to terminal classes.

[8] Attributes are referred to as *slots* in some dialogue systems.

Act	Sys	Usr	Description
hello(a=x,b=y,...)	$\surd$	$\surd$	open a dialogue and give info a=x, b=y, ...
inform(a=x,b=y,...)	$\surd$	$\surd$	give information a=x, b=y, ...
request(a, b=x,...)	$\surd$	$\surd$	request value for a given b=x ...
reqalts(a=x,..)	$\times$	$\surd$	request alternative with a=x,...
confirm(a=x,b=y,..)	$\surd$	$\surd$	explicitly confirm a=x,b=y,..
confreq(a=x,.., d)	$\surd$	$\times$	implicitly confirm a=x,.. and request value of d
select(a=x,a=y)	$\surd$	$\times$	select either a=x or a=y
affirm(a=x, b=y,...)	$\surd$	$\surd$	affirm and give further info a=x, b=y, ...
negate(a=x)	$\times$	$\surd$	negate and give corrected value a=x
deny(a=x)	$\times$	$\surd$	deny that a=x
bye()	$\surd$	$\surd$	close a dialogue

Table 1
The principal dialogue acts used by the HIS System. The Sys and Usr columns indicate which are valid acts for the system outputs and user inputs, respectively.

The way this works is best illustrated by example. Table 3 shows a fragment of the rule set used by a simple application in the Tourist Information domain.[9] Rules of the form $x \to y(p, q, r, \ldots)$ define how a class x can be expanded as a subtype y with class members $\langle p, q, r, \ldots\rangle$. Rules of the form $x = (a|b|c| \ldots)$ define which atomic values $\langle a, b, c, ..\rangle$ can be assigned to the terminal class x. As an example, the tree structure shown in Fig. 4 would represent the goal of a user who was looking for a cheap Chinese restaurant in the centre of town.

For a given set of rules, there are many possible trees that can be derived and as noted previously, it is not practical to instantiate all of them and maintain a belief for each. Furthermore, without evidence from the user, the majority of possible goals would all have the same very low probability.

To avoid this, it may be observed that a partially instantiated tree can be used to efficiently represent the set of all possible user goals derivable from that tree. A partially instantiated tree therefore represents a partition of user goal space. Splitting a partition then equates to selecting one of the uninstantiated leaf nodes and *splitting* it so that one copy is left unchanged and the other copy is expanded according to one or more ontology rules. Furthermore, if a prior probability mass is associated with each tree node, then splitting a

[9] It should be noted that apart from the database itself, there is no other application dependent data or code in a HIS dialogue manager.

	Utterance	Dialogue Act
U:	Hi, I am looking for somewhere to eat.	hello(task=find,type=restaurant)
S:	You are looking for a restaurant. What type of food do you like?	confreq(type=restaurant,food)
U:	I'd like an Italian somewhere near the museum.	inform(food=Italian,near=museum)
S:	Roma is a nice Italian restaurant near the museum.	inform(name="Roma",type=restaurant, food=Italian,near=museum)
U:	Is it reasonably priced?	confirm(pricerange=moderate)
S:	Yes, Roma is in the moderate price range.	affirm(name="Roma", pricerange=moderate)
U:	What is the phone number?	request(phone)
S:	The number of Roma is 385456.	inform(name="Roma",phone="385456")
U:	Ok, thank you goodbye.	bye()

Table 2
An example dialogue and its representation at the dialogue act level.

task	$\rightarrow$	find(entity)	0.4
entity	$\rightarrow$	venue(name,type,area)	0.8
type	$\rightarrow$	bar(drinks,music)	0.4
type	$\rightarrow$	restaurant(food,pricerange)	0.3
area	=	(central|east|west| ...)	
food	=	(Italian|Chinese| ...)	

Table 3
Example Ontology Rules for a simple Tourist Information domain.

node carrying mass P using a rule with prior probability p simply requires that the mass of the remaining uninstantiated node is reduced to probability $P - p$ and the split off node has mass p. In addition, the belief associated with the original partition is divided between the two new partitions in the same proportion. Provided that no rule is used to split a node more than once, this mechanism ensures that all partitions are unique, the sum of the prior probabilities over all partitions is always unity and splitting does not change the total belief over all partitions.

For the case of non-terminal nodes, the partition split probability $P(p'|p)$ is specified as a prior in the node expansion rules (indicated by a $\rightarrow$). This prior

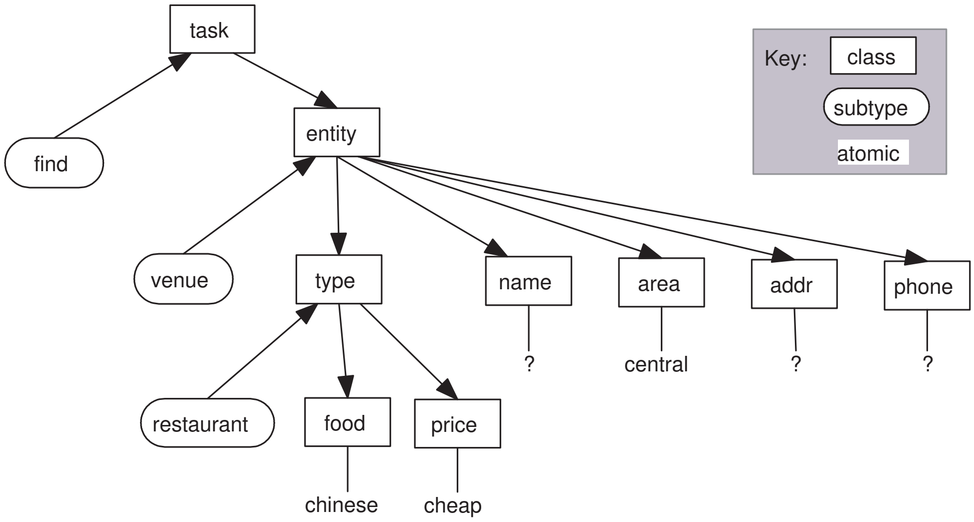

Fig. 4. Tree structure representing the user goal: "find a cheap Chinese restaurant in the centre of town".

can be estimated by counting occurrences of each class type in a training corpus. However, for the case where an atomic value a is assigned to a terminal node x, using a simple prior for $P(p'|p) = P(a|x)$ would severely underestimate the probability since in practice it will be heavily conditioned by the values of the other terminal nodes in the goal tree. Hence, in this case, $P(p'|p)$ is estimated as $ne(x, a, s_u)/ne(x, s_u)$ where the numerator is the number of database entities consistent with the current goal hypothesis s_u when $x = a$ and the denominator is the number of database entities consistent with s_u when x is unspecified.

Thus, in the HIS model, partitions of user goal space are represented by a forest of trees where each tree represents a single partition. At the start of a dialogue, there is just one partition represented by a single root node with belief mass unity. Each incoming user act is matched against each partition in turn and if there is a match, nothing needs to be done. However, if there is no match, the ontology rules are consulted and the system attempts to create a match by expanding the tree. This expansion will result in partitions being split and their belief mass redistributed accordingly. This is illustrated in Fig. 5. Following the initial system prompt, the user requests something but due to poor speech recognition the understanding component generates two possible hypotheses. These firstly cause the `task` node to be split to create a `find` task. Since there is no match for the `type=restaurant` item, an `entity` node is created with subtype `venue` and then its `type` node is split to create a subtype `restaurant`. The second user act hypothesis is then matched against the set of partitions, and the `type` node is split again to create the `bar` subtype. The way in which the prior probabilities are applied is indicated by the numbers on the top of the nodes.

The figure also illustrates the way that the belief is redistributed by the splitting process. It is important to stress that this belief refinement is quite distinct from belief monitoring. The probability of each hypothesised user act is irrelevant to the splitting process. The whole process is designed to be conceptually equivalent to a system where all possible trees are fully expanded from the outset and belief monitoring is applied to all possible partitions.

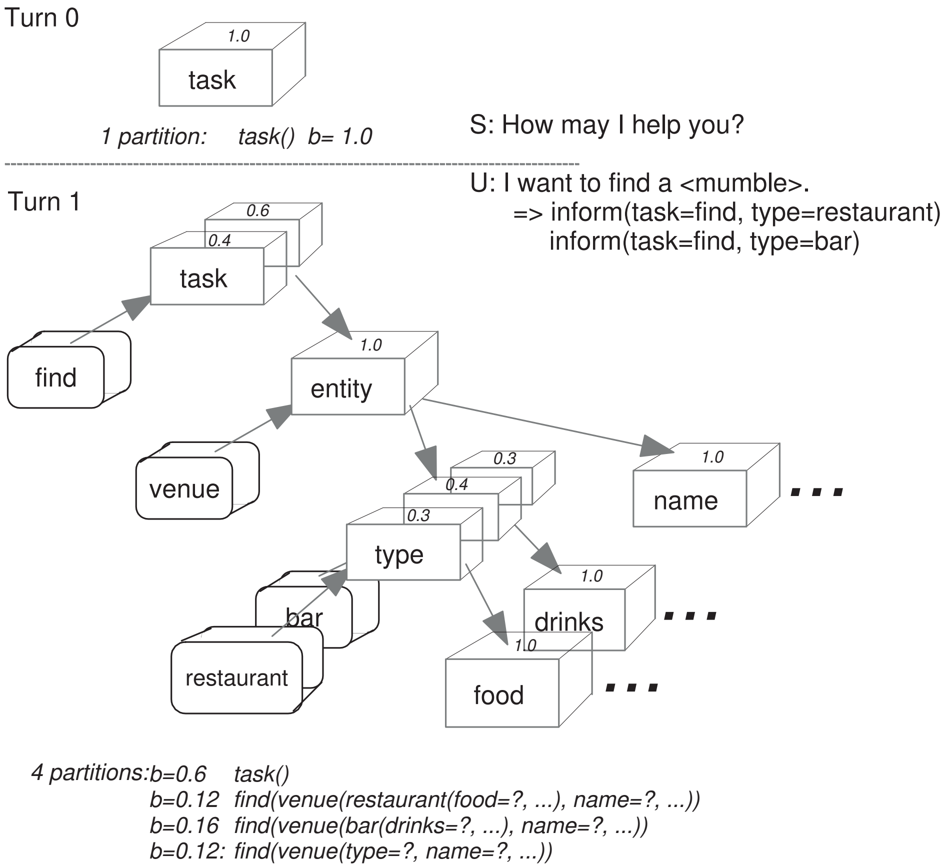

Fig. 5. Illustration of Partition Splitting

3.3 The Observation Model

The observation model probabilities $P(o|a_u)$ are derived directly from the N-best list of hypotheses generated by the speech understanding component by assuming that

$$P(o|a_u = \tilde{a}_u^i) = k^o p_i \tag{12}$$

where $\tilde{a}_u^i$ and p_i are defined in (8) i.e. the posterior probability of the N-best list element corresponding to a_u. In practice, the constant k^o can be ignored since it is subsumed by the constant k in the belief update equation (11).

In practice, the quality of the N-best list of user acts is crucial to obtaining robust performance (Thomson et al., 2008c). In the current system, the speech recogniser generates a word lattice which is then converted to a confusion network (Evermann and Woodland, 2000). A ranked list of word level hypotheses is then generated from this confusion network in which the probability of each hypothesis is given by the product of each constituent word posterior including any null arcs. Each word level hypothesis is then parsed to produce a user dialogue act and any resulting duplicates are merged by summing their probabilities. Currently, the speech understanding component is deterministic and does not modify the word level probabilities. [10]

3.4 The User Action Model

The HIS user action model is a hybrid model, consisting of a dialogue act type *bigram model* and an *item matching model*

$$P(a'_u|p', a_m) = \underbrace{P(\mathcal{T}(a'_u)|\mathcal{T}(a_m))}_{\text{bigram model}} \cdot \underbrace{P(\mathcal{M}(a'_u)|p', a_m)}_{\text{item matching model}} \tag{13}$$

where $\mathcal{T}(\cdot)$ denotes the *type* of the dialogue act and $\mathcal{M}(\cdot)$ denotes whether or not the dialogue act *matches* the given user goal partition p' and the last system act a_m.

The bigram model reflects the dialogue phenomenon of *adjacency pairs* (Schegloff and Sacks, 1973). For example, a question is typically followed by an answer, an apology by an apology-downplayer, and a confirmation ("You want Chinese food?") is typically followed by an affirmation ("Yes please."), or negation ("No, I want Indian."). The bigram model is trained from data using maximum likelihood with Witten–Bell smoothing.

The item matching model is deterministic, assigning either a *full match probability* or a *no match probability*, depending on the outcome of matching the user act with the given user goal partition. For example, the user is not likely to ask about Indian food when the user goal actually indicates that he wants a Chinese restaurant. Therefore, the item arguments of an `inform` act should match the partition. On the other hand, a negation is not likely if the content of the last system act matches the partition. The matching probabilities themselves are optimised empirically.

[10] The Phoenix decoder is currently used for semantic parsing (Ward, 1991).

The design of the matching model is formalised in terms of dialogue act *preconditions* (Cohen and Perrault, 1979). The preconditions of an action specify the conditions that the assumed dialogue state has to satisfy in order for an agent to perform that action. For example, a user wanting to find a Chinese restaurant is motivated to perform the action `inform(type=restaurant, food=Chinese)` (assuming cooperativity in the sense that the system will try to satisfy the user's goal once it has been informed about it). Each precondition is defined in terms of an agent (typically the user U), a propositional attitude (typically WANTS), and a propositional argument (typically an attribute-value pair). For example, `inform(food=Chinese)` has the precondition 'U WANTS (food=Chinese)', whereas a `negate()` after `confirm(food=Indian)` has the precondition 'U **not** WANTS (food=Indian)'. Table 4 presents some examples of HIS system user dialogue acts and their preconditions in terms of the propositional attitudes of the user, and what matching operations against the given user goal partition are required for these preconditions to be satisfied. Note that since the preconditions do not depend on specific attribute names or their values, the item match model specification is domain-independent.

User act	**Preconditions**	**Items to match**
`inform(a=x,b=y)`	U WANTS a=x, b=y	a=x, b=y
`request(a,b=x)`	U WANTS b=x U WANTS U KNOWS_VAL a	b=x a
`reqalts(a=x)`	U WANTS a=x	a=x
`confirm(a=x)`	U WANTS U KNOWS_IF a=x	a=x
`affirm()`	[sys: `confirm(a=x)`] U WANTS a=x	 a=x
`affirm(b=y)`	[sys: `confirm(a=x)`] U WANTS a=x U WANTS b=y	 a=x b=y
`negate()`	[sys: `confirm(a=x)`] **not**(U WANTS a=x)	 **not**(a=x)
`negate(b=y)`	[sys: `confirm(a=x)`] U BEL S BEL U WANTS a=x **not**(U WANTS a=x) U WANTS b=y	 **not**(a=x) b=y

Table 4
Preconditions and item match conditions required by the item match model for some typical user acts. Where relevant, the last system act a_m is shown as [sys: act]. The relation KNOWS_VAL means "knows the value of"; BEL means "believes"; and **not**(a=x) means that the item (a=x) may not match the partition.

Each user goal partition represents one possible interpretation of the goal that is in the user's mind and which is motivating the current query. As the dialogue progresses, the attributes and values which comprise the goal will be mentioned by both the user and the system in various contexts. The purpose of the dialogue history model is to track the status of these attributes and values using a grounding-model(Traum, 1999). Each terminal node in the associated partition is assigned a grounding state as shown in Table 5.

These states are updated according to a simple set of transition rules as the dialogue progresses. For example, if the user requests "a bar in the centre of town", the grounding state of the nodes representing `area=central` will be *UInf*. If the system then queries that the desired area is indeed "central" and the user confirms it, then the grounding state will be updated to *Grnd*.

State	Description
Init	initial state
UReq	item requested by user with expectation of an immediate answer
UInf	item supplied by user during formation of a query
SInf	item supplied by system
SQry	item queried for confirmation by system
Deny	item denied
Grnd	item grounded

Table 5
User Goal Node Grounding States

It is important to emphasise that the grounding states of nodes in user goal trees are not deterministic. Any node may have multiple possible states depending on the possible dialogue histories that led to the current state. The combination of a specific user goal partition, last user act and specific dialogue history constitute a *hypothesis* in the HIS system. For example, Figure 6 illustrates the way that hypotheses are formed in more detail. In this example, the system had previously output `inform(music=Jazz)` and the user's response was either `request(food)` or `deny(music=Jazz)`. Previously there was a single dialogue history hypothesised for the given fragment of partition p' with both nodes in the *Init* state. After completing the turn, there are two distinct dialogue history states corresponding to the two different interpretations of the user input.

The actual probability $P(s'_d|p', a'_u, s_d, a_m)$ returned by the dialogue history

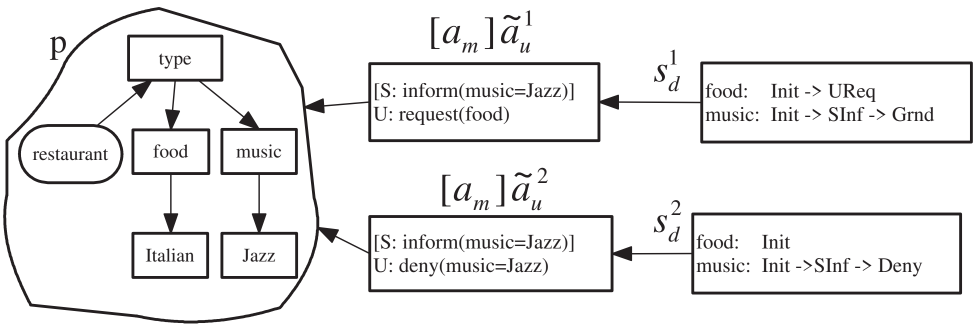

Fig. 6. Example hypothesis formation where the same partition has differing grounding states depending on the interpretation of the previous user act.

model is deterministic. If after updating the history from s_d to s'_d, a resulting hypothesis $\langle p', a'_u, s'_d\rangle$ is inconsistent, for example the user has denied a goal in p', then $P(s'_d|p', a'_u, s_d, a_m) \approx 0$, otherwise $P(s'_d|p', a'_u, s_d, a_m) \approx 1$.

3.6 Summary of Belief Updating in the HIS System

Before moving to the topic of policy representation and optimisation, it may be helpful to summarise the process of belief updating in the HIS system. Referring to Fig. 7, the inputs to the system consist of an observation from the user and the previous system act. The observation from the user typically consists of an N-best list of user acts, each tagged with their relative probability ((6) and section 3.1). The user goal is represented by a set of branching tree structures which represent partitions of the user goal space and they are grown downwards controlled by application-dependent ontology rules (section 3.2). Initially, there is a single tree node representing a single partition with belief unity. As the trees are grown, the partitions are repeatedly split allowing the belief assignment to be refined.

The tree growing process is driven entirely by the dialogue acts exchanged between the system and the user. Every turn, the previous system act and each input user act is matched against every partition in the branching tree structure. If a match can be found then it is recorded. Otherwise the ontology rules are scanned to see if the tree representing that partition can be extended to enable the act to match. Once the matching and partition splitting is complete, all the partitions are rescanned and for each partition, all hypothesised user acts which have items matching that partition are attached to it.

Each combination of a user goal partition and an input user act $\langle p, a_u\rangle$ forms a partial hypothesis and the observation probability and user act model probability can be calculated as in (12) and (13).

The grounding status of each tree node is recorded in the dialogue history state

s_d. Since the grounding status of a tree node can be uncertain, any $\langle p, a_u \rangle$ pair can have multiple grounding states attached to it. However, unlike the user act component of the state which is memoryless, the dialogue history evolves as the dialogue progresses. Thus, at the beginning of each dialogue cycle, the various dialogue state instances are stored directly with the partitions. Once the input user acts have been attached to the partitions, the dialogue history states are updated to represent the new information in the dialogue acts. At this point, the dialogue history state probabilities (section 3.5) are computed. At the end of the turn, identical dialogue history states attached to the same partition are merged ready for the next cycle.

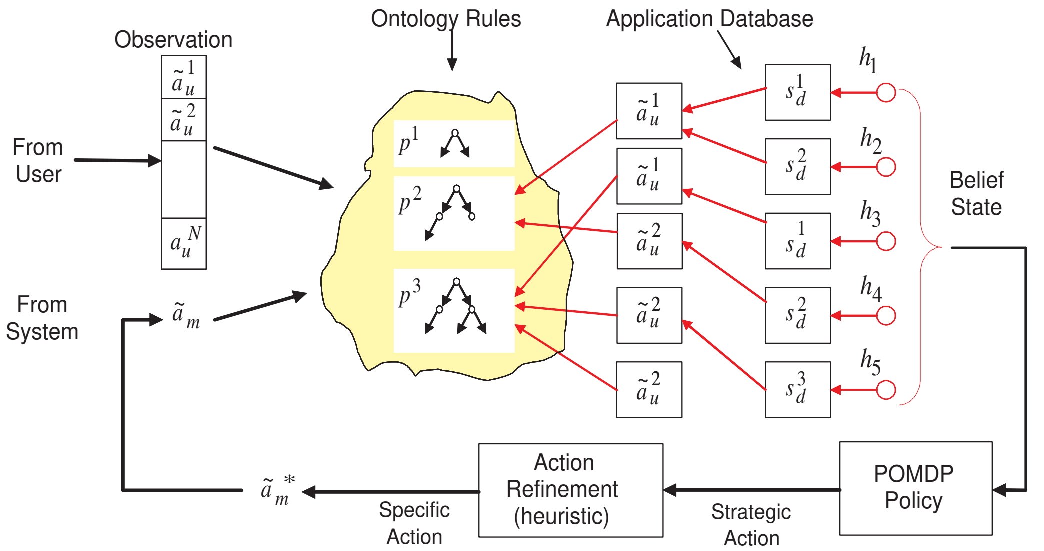

Fig. 7. Overview of the HIS System operation

Every distinct triple $\langle p, a_u, s_d \rangle$ remaining at the end of the above process represents a single dialogue hypothesis h_k. The belief in each h_k is computed using 11 and the complete set of values $b(h_k)$ represents the current estimate of the POMDP belief state. This belief state is input to the POMDP policy which determines the next system output. The way that policies are represented and the implementation of the decision process is described next.

4 Policy Representation and Optimisation

As mentioned in section 2.1, the HIS system represents policies by a set of grid points in *summary belief space* and an associated set of summary actions. Beliefs in *master space* are mapped first into summary space and then mapped into a summary action via a dialogue policy. The resulting summary action is then mapped back into master space and output to the user. This mapping is necessary because accurate belief monitoring requires that the full

H-Status	Meaning
initial	initial state of a hypothesis
supported	at least one grounded node in associated partition
offered	entity consistent with this hypothesis has been offered to user
accepted	offered entity has been accepted
rejected	at least one node in associated partition is denied
notfound	no solution to user goal defined by partition is possible

Table 6
Hypothesis status values used in summary belief space

P-Status	Meaning
initial	initial state of a partition
generic	partition is consistent with at least one node instantiated
hugegroup	set of database entities matching partition is underspecified
smallgroup	set of database entities matching partition is fully specified
unique	partition is consistent with a single unique matching entity
unknown	no entities in database are consistent with this partition

Table 7
Partition status values used in summary belief space

propositional content of user goals and dialogue acts be maintained, whereas policy optimisation requires a more compact space which can be covered by a reasonable number of grid points. This section explains this mapping process in more detail and describes the policy optimisation algorithm.

4.1 Summary Space and Dialogue Policies

In the current HIS system, each summary belief point is a vector consisting of the probabilities of the top two hypotheses in master space; two discrete status variables, h-status and p-status, summarising the state of the top hypothesis and its associated partition (see Tables 6 and 7); and the type of the last user act.

The set of possible machine dialogue acts is also compressed in summary space. This is achieved by removing all act items leaving only a reduced set of dialogue act types. When mapping back into master space, the necessary items (i.e. attribute-value pairs) are inferred by inspecting the most likely dialogue hypotheses. The full list of summary actions is given in Table 8.

Summary Act	Meaning
greet	greet the user
request	request information
confreq	implicitly confirm and request further information
confirm	explicitly confirm some information
offer	offer an entity to the user
inform	provide further information
split	ask user to distinguish between two options
findalt	find an alternative solution to users goal
querymore	ask user if more information is required
bye	say goodbye

Table 8
List of summary acts

Given the above, a dialogue policy can be represented as a fixed set of belief points in summary space (i.e. a grid) along with the action to take at each point. In order to use such a policy, a distance metric in belief space is required to find the closest grid point to a given arbitrary belief state. In the prototype HIS system, this distance metric is [11]

$$|\hat{b}_i - \hat{b}_j| = \sum_{k=1}^{2} \alpha_k \cdot \sqrt{(\hat{b}_i(k) - \hat{b}_j(k))^2} + \sum_{k=3}^{5} \alpha_k \cdot (1 - \delta(\hat{b}_i(k), \hat{b}_j(k))) \qquad (14)$$

where the α's are weights, the index k ranges over the 2 continuous and 3 discrete components of $\hat{b}$ and $\delta(x, y)$ is 1 iff $x = y$ and 0 otherwise.

4.2 Master-Summary Space Mapping

The process of mapping between master and summary space is illustrated in more detail in Fig. 8. On the left of this figure is master space consisting of a set of dialogue hypotheses where, as discussed previously, each hypothesis consists of a user goal partition p, a dialogue history state s_d and the last user act a_u. Note also that each hypothesis has associated with it a notional set of database entries consisting of all entities in the database which are consistent with the hypothesis's partition. On the right of Fig. 8 is summary space represented by a single vector or belief point.

The policy is shown as an irregular grid of these belief points and the figure

[11] But see section 6

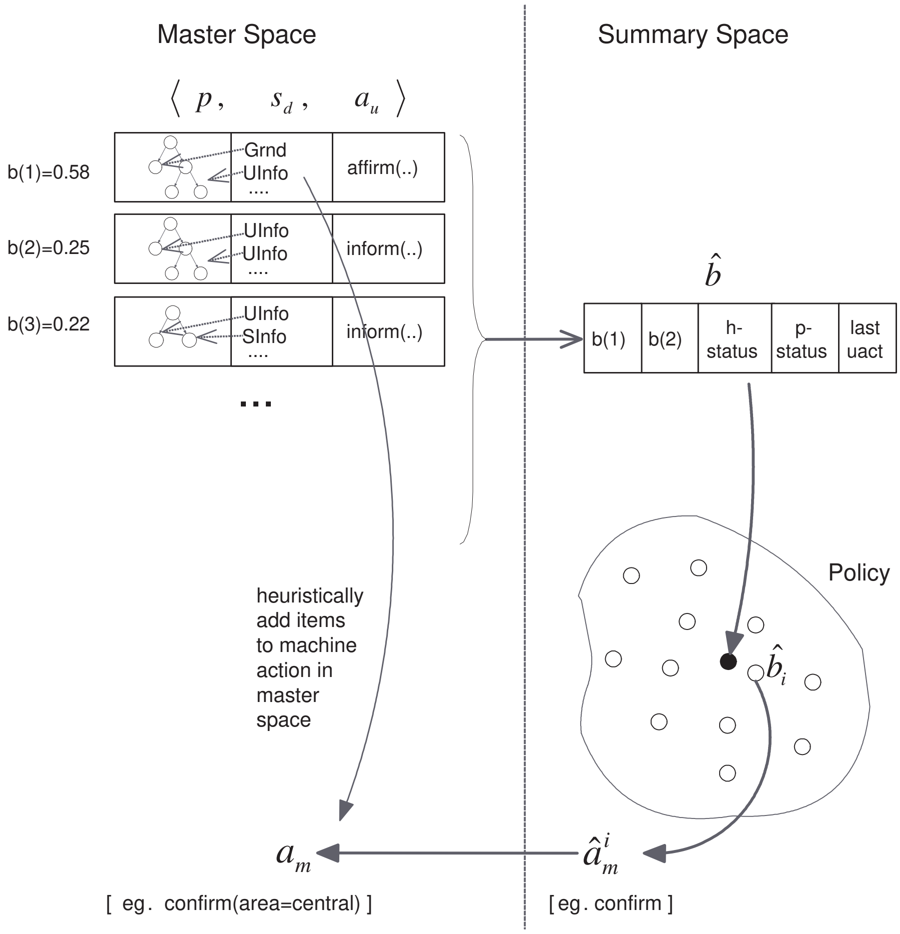

Fig. 8. Master-Summary State Mapping

shows how a system response is generated by mapping the current belief state b into a summary belief state $\hat{b}$, then finding the nearest stored point in the policy $\hat{b}_i$ which in turn yields a summary action $\hat{a}^i_m$. This is then mapped back into master space by a heuristic which assumes that the selected summary action refers to the top hypothesis and therefore constructs the full machine action a_m taking note of the grounding state of all the nodes in the associated partition (Williams and Young, 2007b). [12]

4.3 Switching User Goals

As mentioned in section 2.3, the HIS POMDP model assumes that the user's goal does not change. In practice, of course the user's goal does occasionally change but this is usually signalled by the user explicitly requesting an alternative or as a consequence of the system not being able to find an entity in

[12] In the case of the *split* summary action, the response is constructed from the top two hypotheses.

the database that satisfies the user's current goal. This is handled by associating a change of goal with a specific summary state action called *findalt* (see Table 8). When a *findalt* action is executed, the effect is equivalent to an *offer* action, except that the most likely hypothesis in master belief space is replaced by an alternative hypothesis lower down the N-best list.

4.4 A Hand-crafted Policy

Before discussing policy optimisation, it should be noted that at the level of summary space, it is relatively easy to hand-craft a policy and such a policy serves as a useful baseline for experimental performance evaluations. The strategy used by the HIS hand-crafted policy is to first check the most likely hypothesis. If it contains sufficient grounded keys to match between one and three database entities, then the *offer* action is selected. If any part of the hypothesis is inconsistent or the user has explicitly asked for another suggestion, then the *findalt* action is selected. If the user has asked for information about an offered entity then *inform* is selected. Finally, if an ungrounded component of the top hypothesis can identified then depending on the belief, one of the *confreq* or *confirm* actions is selected; otherwise a *request* action is generated to expand one of the partition's leaf nodes.

4.5 Policy Optimisation

As explained in section 2.1, representing a POMDP policy by a grid of exemplar belief points yields an MDP optimisation problem for which many tractable solutions exist. Here a simple Monte Carlo Control algorithm is used (Sutton and Barto, 1998). Associated with each belief point is a function $Q(\hat{b}, \hat{a}_m)$ which records the expected reward of taking summary action $\hat{a}_m$ when in belief state $\hat{b}$. Q is estimated by repeatedly executing dialogues and recording the sequence of $\langle \hat{b}_t, \hat{a}_{m,t} \rangle$ belief point-action pairs. At the end of each dialogue, each $Q(\hat{b}_t, \hat{a}_{m,t})$ estimate is updated with the actual discounted reward. Dialogues are conducted using the current policy π but to allow previously unvisited regions of the state-action space to be explored, actions are occasionally taken at random with some small probability ϵ [13].

Once the Q values have been estimated, the optimal policy can easily be found by setting

$$\pi(\hat{b}) = arg \max_{\hat{a}_m} Q(\hat{b}, \hat{a}_m), \qquad \forall \hat{b} \in \mathcal{B} \tag{15}$$

[13] This is referred to as an ϵ-greedy policy in the literature.

In order to minimise the errors caused by quantising belief space, belief points should only be placed in regions of the state space which are likely to be visited during actual dialogues. Belief points are therefore generated on demand during the policy optimisation process. Initially, just a single belief point is allocated. Then everytime a belief point is encountered which is sufficiently far from any existing point in the policy grid, it is added to the grid as a new point.

The complete policy optimisation algorithm is shown in Fig. 9.

5 Training the HIS System

This section explains how the HIS system policy is optimised in practice using a user simulator.

5.1 Reward Function

The choice of reward function can have a significant effect on system performance and in principle quite elaborate reward schemes could be devised to encourage specific desired behavioural properties. For the experiments described here, however, a very simple reward function is used where the reward for each dialogue is obtained by subtracting 1 point for each turn and adding 20 points in case of a successful dialogue. Since a typical dialogue will require around 5 or 6 turns to complete, this implies that around 15 represents an upper bound on the achievable average return.

5.2 User Simulation

Policy optimisation using the Monte Carlo algorithm requires many thousands of dialogues to obtain robust estimates for the Q values. Hence in practice, a user simulator is used to generate responses to system actions. This has two main components: a *User Goal* and a *User Agenda*(Schatzmann et al., 2007a). At the start of each dialogue, the goal is randomly initialised with requests such as `name`, `addr`, `phone` and constraints such as `type=restaurant`, `food=Chinese`, etc. The agenda stores the dialogue acts needed to elicit this information in a stack-like structure which enables it to temporarily store actions when another action of higher priority needs to be issued first. This enables the simulator to refer to previous dialogue turns at a later point. To generate a wide spread of realistic dialogues, the simulator reacts wherever

Let $Q(\hat{b}, \hat{a}_m)$ = expected reward on taking action $\hat{a}_m$ from belief point $\hat{b}$
Let $N(\hat{b}, \hat{a}_m)$ = number of times action $\hat{a}_m$ is taken from belief point $\hat{b}$
Let $\mathcal{B}$ be a set of grid-points in belief space
Let $\pi : \hat{b} \rightarrow \hat{a}_m; \forall \hat{b} \in \mathcal{B}$ be a policy
repeat
    $t \leftarrow 0$
    $\hat{a}_{m,0} \leftarrow$ initial greet action
    $b = b_0$ [= all states in single partition - see section 2.3]

    Generate dialogue using ϵ-greedy policy
    repeat
        $t \leftarrow t + 1$
        Get user turn $a_{u,t}$ and update belief state b
        $\hat{b}_t \leftarrow$ SummaryState(b)
        $\hat{a}_{m,t} \leftarrow \begin{cases} \text{RandomAction} & \text{with probability } \epsilon \\ \pi(\text{Nearest}(\hat{b}, \mathcal{B})) & \text{otherwise} \end{cases}$
        record $\langle \hat{b}_t, \hat{a}_{m,t} \rangle$, $T \leftarrow t$
    until dialogue terminates with reward R from user simulator

    Scan dialogue and update $\mathcal{B}$, Q and N
    for $t = T$ **downto** 1 **do**
        if $\exists b_k \in \mathcal{B}$, $|\hat{b}_t - \hat{b}_k| < \delta$ **then** $\leftarrow$ update nearest pt in $\mathcal{B}$
            $Q(\hat{b}_k, \hat{a}_{m,t}) \leftarrow \frac{Q(\hat{b}_k, \hat{a}_{m,t}) * N(\hat{b}_k, \hat{a}_{m,t}) + R}{N(\hat{b}_k, \hat{a}_{m,t}) + 1}$
            $N(\hat{b}_k, \hat{a}_{m,t}) \leftarrow N(\hat{b}_k, \hat{a}_{m,t}) + 1$
        else $\leftarrow$ create new grid point
            add $\hat{b}_t$ to $\mathcal{B}$
            $Q(\hat{b}_t, \hat{a}_{m,t}) \leftarrow R$, $N(\hat{b}_t, \hat{a}_{m,t}) \leftarrow 1$
        end if
        $R \leftarrow \gamma R$ $\leftarrow$ discount the reward
    end for

    Update the policy
    for all $\hat{b}_k$ with updated $Q(\hat{b}_k, \hat{a}_m)$ for any $\hat{a}_m$ **do**
        $\pi(\hat{b}_k) = \arg\max_{\hat{a}_m} Q(\hat{b}_k, \hat{a}_m)$
    end for
until converged

Fig. 9. Monte Carlo Policy Optimisation Algorithm

possible with varying levels of patience and arbitrariness. In addition, the simulator will relax its constraints when its initial goal cannot be satisfied. This allows the dialogue manager to learn negotiation-type dialogues where only an approximate solution to the user's goal exists. Speech understanding errors are simulated at the dialogue act level using confusion matrices trained on labelled dialogue data (Schatzmann et al., 2007b).

5.3 Policy Optimisation

When training a system to operate robustly in noisy conditions, a variety of strategies are possible. For example, the system can be trained only on noise-free interactions, it can be trained on increasing levels of noise or it can be trained on a high noise level from the outset. A related issue concerns the generation of grid points and the number of training iterations to perform. For example, allowing a very large number of points leads to poor performance due to over-fitting of the training data. Conversely, having too few points leads to poor performance due to a lack of discrimination in decision making.

Based on some initial experimentation, the following training schedule is used for HIS policy optimisation. Training starts in a noise free environment using a small number of grid points and it continues until the performance of the policy asymptotes. The resulting policy is then taken as an initial policy for the next stage where the noise level is increased, the number of grid points is expanded and the number of iterations is increased. This process is repeated until the highest noise level is reached. This approach was motivated by the observation that a key factor in effective reinforcement learning is the balance between exploration and exploitation. Since the HIS system policy optimisation uses dynamically allocated grid points, maintaining this balance is crucial. In this case, the noise introduced by the simulator is used as an implicit mechanism for increasing the exploration. Each time exploration is increased, the areas of state-space that will be visited will also increase and hence the number of available grid points must also be increased. At the same time, the number of iterations must be increased to ensure that all points are visited a sufficient number of times. In practice a total of 750 to 1000 grid points have been found to be sufficient and the total number of simulated dialogues needed for training is around 100,000.

A second issue when training in noisy conditions is whether to train on just the 1-best output from the simulator or train on the N-best outputs. Intuitively, a larger N-best list provides a better approximation of the observation. However, a limiting factor here is that the computation required for N-best training is significantly increased since the rate of partition generation in the HIS model increases exponentially with N.

In preliminary tests, it was found that when training and testing with 1-best outputs, there was little difference between policies trained entirely in no noise and policies trained on increasing noise as described above. However, policies trained on 2-best using the incremental strategy when tested using 2-

best outputs did exhibit increased robustness to noise, whereas policy trained in no noise did not show any improvement when tested on 2-best. Hence, incremental training with 2-best outputs was adopted for all policy training (Gašić et al., 2008).

6 Evaluation

This section describes the evaluation of a HIS-based dialogue system designed for the Tourist Information domain. The system allows users to request information about venues in a ficticious town called Jasonville. The user may provide information about nine attributes in their attempts to find a suitable venue. These are: name of the venue, type of venue, area, price range, nearness to a particular location, type of drinks, food type (for restaurants), number of stars (for hotels and restaurants) and music (for restaurants and bars). Once a suitable venue is found the user may ask about four further attributes: address, telephone number, a comment on the venue and the price (for hotels and restaurants). There are 47 different venues in total.

The primary evaluation is based on comparing the HIS POMPD-based dialogue manager with an MDP-based manager. The latter was developed in 2007 as part of another project in a parallel development to the HIS system described here. It has been used in various trials Schatzmann (2008) and since considerable effort has been put into optimising this system, it serves as a good baseline for comparison.

For each system there are the following variants:

MDP07-HDC the baseline 2007 MDP system with a hand-crafted policy

MDP07-TRA the baseline 2007 MDP system with a trained policy

HIS07-HDC the prototype 2007 HIS system with a hand-crafted policy

HIS07-TRA the prototype 2007 HIS system with a trained policy

HIS08-TRA a refined 2008 HIS system with a trained policy

The MDP07-HDC system is in effect a conventional dialogue manager utilising a finite state network with hand-crafted decisions. The MDP07-TRA system uses a state-action table trained using reinforcement learning via interactions with the user simulator. The HIS07 system represents the first complete HIS-based spoken dialogue system which was capable of conducting realistic and complete dialogues in the tourist information domain including negotiating with the user when the users initially query was under or over specified. The HIS07 system has all of the components described above except that it uses a simplified form of user action model.

As described in more detail below, the HIS07 system was trialled in February 2008 and hence its performance as measured by the user simulator can be verified with real user trial data. Subsequent to the trial, a number of improvements have been made to the HIS system. In particular, the user action model has been refined following the schema described in section 3.4 and the grid-based training has been improved by replacing the Euclidean metric in equation 14 by a quantiser in which the probabilities of the top two hypotheses are placed in one of three bins corresponding to probabilities $(1.0, 0.0)$ i.e. the top hypothesis is certain; $(0.5, 0.5)$ i.e. the top two hypotheses are equally likely; and $(0.0, 0.0)$ i.e. all hypotheses are uniformly unlikely. This new HIS08 system is substantially more robust than the HIS07 system and even though no user trial results are currently available, its inclusion in the simulated results provides a useful indication of progress.

6.1 Evaluation via User Simulation

The limitations of testing a statistical dialogue system on the same user simulator that was used to train its policy have been well-documented (Schatzmann et al., 2005). Nevertheless, evaluation via user simulation can provide a useful basis for comparing systems and characterising their behaviour.

All of the results given in this section were obtained by using the user simulator described in section 5.2 to conduct dialogues over a range of simulated error rates. Depending on the error rate, the simulator provides upto 2-best user dialogue acts at each turn. In the actual trial described below, the average number of semantically distinct dialogue acts derived from the 10-best asr output was 1.4. Hence, the simulator set-up is comparable to the trial system.

Each graph point represents 3000 simulated dialogues. In all cases, the same reward function is used for evaluation as was used in training i.e. 20 points are awarded for a successful dialogue and 1 point is deducted for every turn taken. In addition to recording the rewards achieved, success rates are also recorded. A dialogue is considered to be successful, if the correct venue has been offered and the additional information has been given, as specified in the final user goal.

6.1.1 Performance on simulated data

The average success and reward rates as a function of error rate for the five dialogue systems are shown in Figs. 10 and 11. The general trend is for the trained systems to perform better than their hand-crafted counterparts. The trained HIS systems clearly outperform the MDP system and the increased robustness of the HIS08 system compared to the HIS07 system is striking. This is thought to be mostly due to the improved accuracy of the user action model which is key to the POMDP system's ability to discriminate between user act hypotheses based on context.

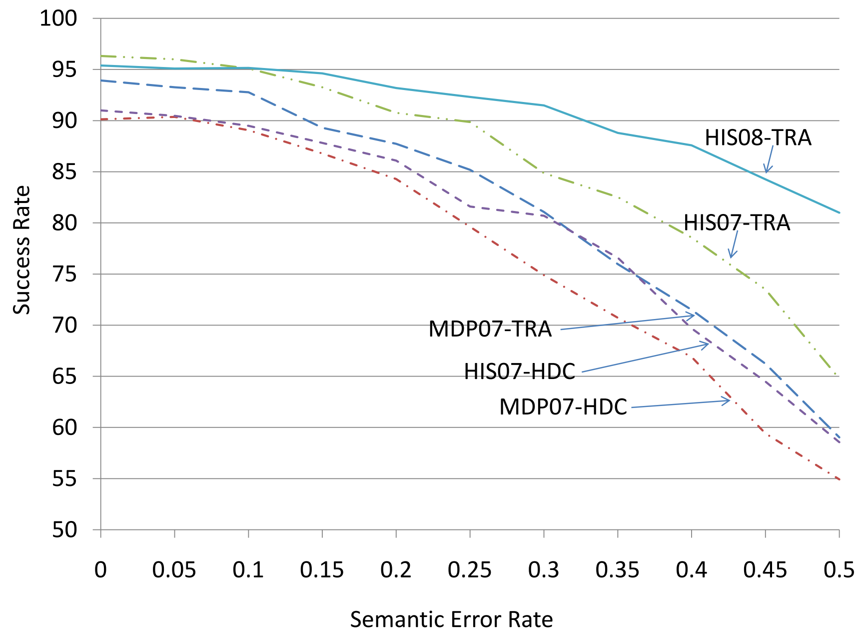

Fig. 10. Success rates at varying error rates for five systems tested on a user simulator.

6.1.2 Effect of the User Action Model

Given the crucial role of the user action model (UAM) in computing beliefs, it is of interest to determine how much it contributes to robustness in practice (Keizer et al., 2008).

Figure 12 shows simulation results comparing the success rates for the HIS08-TRA system with various configurations of the user action model. The solid uppermost line (UAM) shows the performance with the full user action model as described in section 3.4. This is identical to the HIS08-TRA plot in Fig. 10. Also shown is the performance with only the 1-best user act input to the system (UAM 1-best), the user action model with the item match component disabled (UAM Bigram only) and the complete user action model disabled (UAM disabled).

The results clearly show a dramatic improvement in success rate when using the UAM. They also clearly demonstrate the increase in robustness to errors provided by the 2-best input user act list compared to just the 1-best list. Fi-

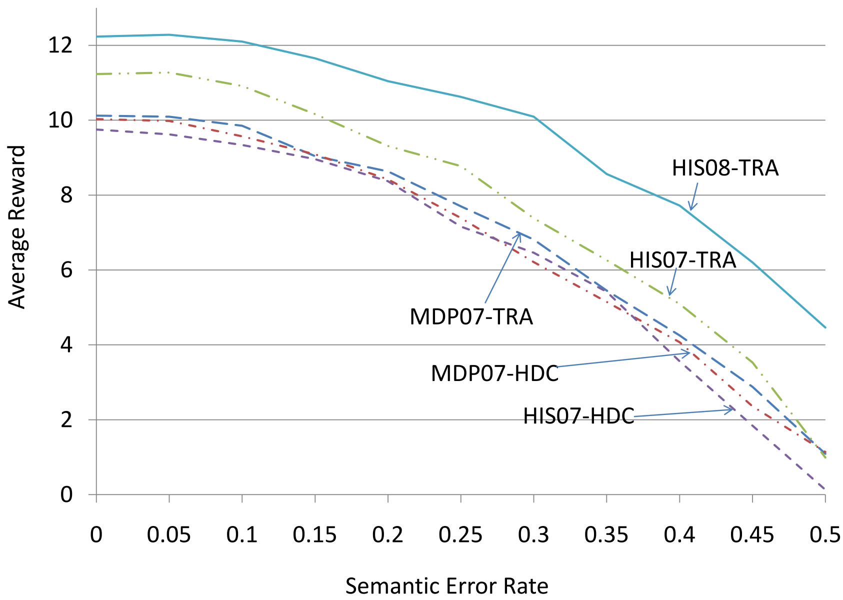

Fig. 11. Average reward rates at varying error rates for five systems tested on a user simulator.

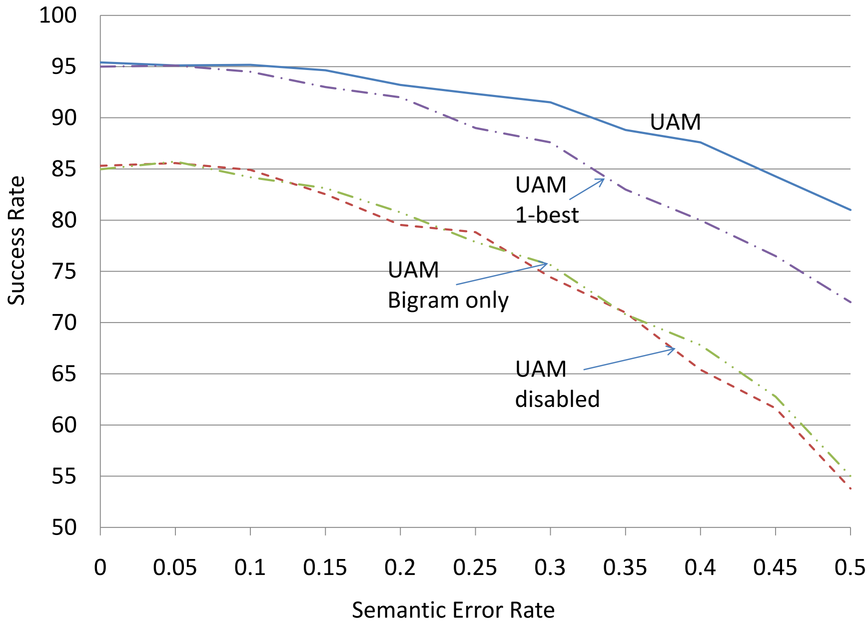

Fig. 12. Success rates for the HIS08-TRA system, comparing different configurations of the UAM.

nally, comparing the UAM disabled case with the Bigram only case, it appears that the bigram component of the UAM makes only a small contribution to performance compared to the item match model.

6.2 Evaluation via a User Trial

Whilst user simulation studies can be useful for understanding the broad characteristics of a spoken dialogue system, actual performance can only be reliably measured by conducting system trials with real users. This section describes one such trial conducted in February 2008 in which the HIS07 system was compared with the MDP07 system. In both cases, performance with the trained policy and the hand-crafted policy was tested.

6.2.1 System Configuration and Trial Setup

The dialogue system used for the trial consisted of an ATK-based speech recogniser (Young, 2005), a Phoenix-based semantic parser (Ward, 1991), a template-based response generator and the FLite diphone-based speech synthesiser (Black and Lenzo, 2005). The semantic parser uses simple phrasal grammar rules to extract the dialogue act type and a list of attribute/value pairs from each utterance.

In a POMDP-based dialogue system, accurate belief-updating is very sensitive to the confidence scores assigned to each user dialogue act. Ideally these should provide a measure of the probability of the decoded act given the observation. In the HIS system, the recogniser output is in the form of a confusion network (Mangu et al., 2000) which is used to compute the inference evidence for each hypothesis. The latter is defined as the sum of the log-likelihoods of each arc in the confusion network and when exponentiated and renormalised this gives a simple estimate of the probability of each hypothesised utterance. Each utterance in the 10-best list is passed to the semantic parser. Equivalent dialogue acts output by the parser are then grouped together and the dialogue act for each group is then assigned the sum of the sentence-level probabilities as its confidence score.

For the trial itself, 36 subjects were recruited (all British native speakers, 18 male, 18 female). Each subject was asked to imagine himself to be a tourist in a fictitious town called Jasonville and try to find particular hotels, bars, or restaurants in that town. Each subject was asked to complete a set of predefined tasks where each task involved finding the name of a venue satisfying a set of constraints such as food type is Chinese, price-range is cheap, etc., and then getting the value of one or more additional attributes of that venue such as the address or the phone number.

For each task, subjects were given a scenario to read and were then asked to solve the task via a dialogue with the system. The tasks set could either have one solution, several solutions, or no solution at all in the database. In cases where a subject found that there was no matching venue for the given task, he/she was allowed to try and find an alternative venue by relaxing one or more of the constraints.

In addition, subjects had to perform each task at one of three possible noise levels. These levels correspond to signal/noise ratios (SNRs) of 35.3 dB (low noise), 10.2 dB (medium noise), or 3.3 dB (high noise). The noise was artificially generated and mixed with the microphone signal, in addition it was fed into the subject's headphones so that they were aware of the noisy conditions. The initial intention was to compute results at each noise level but in the event the ensuing error rates varied significantly within each noise level and statistically significant results could only be achieved by averaging across all noise levels.

A supervisor was present at all times to indicate to the subject which task description to follow, and to start the right system with the appropriate noise-level. Each subject performed an equal number of tasks for each system (3 tasks), noise level (6 tasks) and solution type (6 tasks for each of the types 0, 1, or multiple solutions). Also, each subject performed one task for all combinations of system and noise level. Overall, each combination of system, noise level, and solution type was used in an equal number of dialogues.

6.2.2 Trial Results

Table 9 provides some general statistics of the corpus resulting from the user trial. The semantic error rate is based on substitutions, insertions and deletions errors on semantic items. When tested after the trial on the transcribed user utterances, the semantic error rate was 4.1% whereas the semantic error rate in the actual trial was 25%. This implies that 84% of the error rate was due to speech recognition errors and the parser was otherwise reasonably robust.

Number of dialogues	432
Number of dialogue turns	3972
Number of words	18239
Words per utterance	4.58
Word Error Rate	32.9%
Semantic Error Rate (SER) on ASR output	25.2%
SER on corrected transcriptions	4.1%

Table 9
General corpus statistics for the user trial.

Tables 10 and 11 present success rates and average rewards, comparing the two HIS07 dialogue managers with the two MDP07 baseline systems. For the success rates, also the standard deviation (std.dev) is given, assuming a binomial distribution. The success rate is the percentage of successfully completed dialogues. A task is considered to be fully completed when the user is able to find the venue he/she is looking for and has also obtained all of the additional information asked for; if the task has no solution and the system indicates to the user no venue could be found, this also counts as full completion. A task is considered to be partially completed when only the correct venue has been given. The results on partial completion are given in Table 10, and the results on full completion in Table 11. The reward function is similar to that used in training, i.e. a reward of 20 points is given for full completion and 1 point is subtracted for the number of turns up until a successful recommendation (i.e. partial completion).

Partial Task Completion statistics			
System	Success (std.dev)	#turns	Reward
MDP07-HDC	68.52 (4.83)	4.80	8.91
MDP07-TRA	70.37 (4.75)	4.75	9.32
HIS07-HDC	74.07 (4.55)	7.04	7.78
HIS07-TRA	84.26 (3.78)	4.63	12.22

Table 10
Success rates and average reward on partial completion of each task.

Full Task Completion statistics			
System	Success (std.dev)	#turns	Reward
MDP07-HDC	64.81 (4.96)	5.86	7.10
MDP07-TRA	65.74 (4.93)	6.18	6.97
HIS07-HDC	63.89 (4.99)	8.57	4.20
HIS07-TRA	78.70 (4.25)	6.36	9.38

Table 11
Success rates and performance results on full completion.

The results show that the trained HIS07 system significantly out-performs both the MDP07 systems, whilst the hand-crafted HIS07 system is roughly similar. It is interesting to compare the Full Task Completion results with the simulation results shown in Figs. 10 and 11 at the error rate of 25.2%.. The rankings suggested by the simulations are very similar to those achieved in the trial. The success rates only differ in the 3rd and 4th ranks being inverted and the reward rates differ only in the 2nd and 3rd ranks being inverted. In both cases, there was no significant difference between the transposed ranks.

The absolute performance was also reasonably correlated. The HIS07-TRA system average reward at 25% was predicted by the simulator to be 8.77 and in the trial it was 9.38. The MDP07-TRA reward was predicted to be 7.7

and in the trial it was 6.97. The simulations tended to over-estimate success rates. For example, the HIS07-TRA system achieved an 89.9% success rate on the simulator whereas in the trial it was 78.9%. However, this might be a consequence of the difficulty of accurately measuring success in the trial in cases where the users had to negotiate a suitable venue when their pre-defined constraints were either under- or over-specified. In any event, it is clear that overall the simulation results did give useful predictions of performance in the real user trial.

6.2.3 Subjective Results

In the user trial, the subjects were also asked for a subjective judgement of the systems. After completing each task, the subjects were asked whether they had found the information they were looking for (yes/no). They were also asked to give a score on a scale from 1 to 5 (best) on how natural/intuitive they thought the dialogue was. Table 12 shows the results for the 4 systems used. The performance of the HIS systems is similar to the MDP systems, with a slightly higher success rate for the trained one and a slightly lower score for the handcrafted one. However, these subjective results do not reflect the same clear performance advantage of the HIS system compared to the objective results. A possible reason for this is that because users were attempting artificial tasks, they were less interested in finding the correct information than they were on simply completing the dialogue.

System	Succ. Rate (std.dev)	Score
MDP07-HDC	78 (4.30)	3.52
MDP07-TRA	78 (4.30)	3.42
HIS07-HDC	71 (4.72)	3.05
HIS07-TRA	83 (3.90)	3.41

Table 12
Subjective performance results from the user trial.

7 Conclusions

This paper has described a POMDP-based dialogue manager called the Hidden Information State (HIS) system. The HIS system was built in order to demonstrate that the POMDP framework can be scaled to real world problems, and that because it naturally integrates N-best recognition hypotheses and similar sources of uncertainty such a system can provide increased robustness to noise compared to a conventional MDP or finite state system.

As described in the paper, the required scaling has been achieved by two principal mechanisms. Firstly, the state space is partitioned into equivalence classes which are refined on demand as the dialogue progresses. Secondly, policy implimentation and optimisation is performed in a reduced summary state-space where a simple grid-based learning is effective. One interpretation of the HIS architecture is that it allows multiple dialogue hypotheses to be maintained in parallel and the system response at each turn to be determined from the state of all the hypotheses. As a consequence, the evidence from alternative user dialogue act hypotheses can be incorporated and integrated over time to provide robust behaviour in noisy conditions.

The performance of a prototype HIS system has been evaluated by interaction with a simulated user over a range of noisy levels, and through a live user trial held in noisy conditions. The simulation results show that the trained HIS system out-performs the MDP system especially at higher noise levels. Furthermore, a similar performance gain is obtained in the user trial. The simulation results clearly indicated the crucial role of the user action model in filtering the noisy user inputs and the ability of the system to exploit alternative recognition hypotheses. In addition, there was a reasonable correlation between the user trial results and the simulation results showing that for development purposes there is much to be gained from the availability of a user simulator.

The HIS system was built primarily to demonstrate that the POMDP approach is computationally feasible and has the potential for significant improvements in robustness compared to today's hand-crafted finite-state systems. The HIS design is still evolving and there is considerable work still to do. In particular, the current partition splitting mechanism can result in excessive fragmentation of the state space in long dialogues (> 20 turns) and in high noise conditions. In addition, the summary state space mapping and grid-based optimisation is currently rather crude and there is much scope for refinement. Also, as noted in the introduction, there are other approaches to scaling POMDP's for practical implementation (eg Williams (2007); Thomson et al. (2008b); Bui et al. (2008)). Each of these differ in the way that they approximate the state space and the way that they perform policy optimisation. More work is needed to refine and implement these approaches in real-world systems before any meaningful comparison can be made. Hence, there is currently insufficient evidence to determine which approach, if any, is superior.

Finally, it should be acknowledged that the philosophy of this statistical approach to dialogue system design has been questioned by some industry-focussed practitioners on the grounds that it is inconsistent with the need to provide guarantees to service providers regarding specific system behaviours (Paek and Pieraccini, 2008). This is certainly a legitimate concern. However, there are ways to constrain system behaviour (see for example, Williams (2008)). More importantly, if statistical approaches can deliver significant reductions in development and maintenance costs, along with significant improvements in robustness and ease of use, then the industrial perspective will surely change. In the meantime, we claim that the design and experimental results presented in this paper give strong support to the central contention that the POMDP-based framework is both a tractable and powerful approach to building spoken dialogue systems.

8 Acknowledgement

The work described in this paper was partly funded by the UK EPSRC under grant agreement EP/F013930/1 and by the EU FP7 Programme under grant agreement 216594 (CLASSiC project: `www.classic-project.org`).

References