D4.4: POMDP-based System

Steve Young, Jost Schatzmann, Blaise Thomson,
Karl Weilhammer, Hui Ye

Distribution: Public

TALK
Talk and Look: Tools for Ambient Linguistic Knowledge
IST-507802 Deliverable 4.4

6th December, 2006

Project funded by the European Community
under the Sixth Framework Programme for
Research and Technological Development

The deliverable identification sheet is to be found on the reverse of this page.

Project ref. no.	IST-507802
Project acronym	TALK
Project full title	Talk and Look: Tools for Ambient Linguistic Knowledge
Instrument	STREP
Thematic Priority	Information Society Technologies
Start date / duration	01 January 2004 / 36 Months

Security	Public
Contractual date of delivery	M36 = December 2006
Actual date of delivery	6th December, 2006
Deliverable number	4.4
Deliverable title	D4.4: POMDP-based System
Type	Report and Prototype
Status & version	Final 1.0
Number of pages	34 (excluding front matter)
Contributing WP	4
WP/Task responsible	UCAM
Other contributors	Oliver Lemon, Jamie Henderson, Roi Georgila
Author(s)	Steve Young, Jost Schatzmann, Blaise Thomson, Karl Weilhammer, Hui Ye
EC Project Officer	Evangelia Markidou
Keywords	statistical dialogue modelling; partially observable Markov decision processes (POMDPs)

The partners in TALK are:

Saarland University	USAAR
University of Edinburgh HCRC	UEDIN
University of Gothenburg	UGOT
University of Cambridge	UCAM
University of Seville	USE
Deutches Forschungszentrum fur Künstliche Intelligenz	DFKI
Linguamatics	LING
BMW Forschung und Technik GmbH	BMW
Robert Bosch GmbH	BOSCH

For copies of reports, updates on project activities and other TALK-related information, contact:

The TALK Project Co-ordinator
Prof. Manfred Pinkal
Computerlinguistik
Fachrichtung 4.7 Allgemeine Linguistik
Postfach 15 11 50
66041 Saarbrücken, Germany
pinkal@coli.uni-sb.de
Phone +49 (681) 302-4343 - Fax +49 (681) 302-4351

Copies of reports and other material can also be accessed via the project's administration homepage, `http://www.talk-project.org`

Contents

Executive summary

Partially observable Markov decision processes (POMDPs) provide a principled mathematical framework for modelling the uncertainty inherent in spoken dialogue systems. However, conventional POMDPs scale poorly with the size of state and observation space. This report describes a variation of the classic POMDP called the Hidden Information State (HIS) model. The HIS system is based on two key ideas.

Firstly, a belief distribution over an extremely large state space can be represented efficiently by grouping states together into partitions. Initially, all states are deemed to be in a single partition with belief unity. As the dialog progresses, the partitions are split and belief is redistributed amongst the splits. Eventually, some partitions become very low in cardinality, and in the limit singletons. Action selection is then dominated by these low cardinality partitions. The overall result is that full-scale POMDP belief monitoring is achieved without ever explicitly calculating the beliefs of (the majority of) irrelevant states. Furthermore, partitions are represented efficiently using tree structures and these tree structures also provide a very natural representation for real-world knowledge.

Secondly, although accurate belief monitoring must consider the full state space, adequate planning can be achieved in a more compact "summary space". The HIS model provides a mechanism for mapping between these two spaces and therefore allows POMDP-based policy optimisation and action selection to be made tractable by performing it in in the reduced summary space.

This report describes the HIS model and the way that it is used to build the prototype POMDP-based system. This work is an extension of the Bayes-net based system presented in D4.3 [1], some of the technical details presented in this report are therefore common to the earlier report but are repeated here in order to make this report self-contained.

The outcome of this work is a prototype system for the in-car tourist information domain which we believe is the first ever full-scale implementation of a dialogue manager using partially-observable Markov Decision Processes. The key benefits of the POMDP formulation which are demonstrated by this prototype system are

- fully-data driven dialog policy learning using a user simulator
- explicit representation of uncertainty by maintaining a large number of dialogue hypotheses in parallel
- ability to handle N-best output hypotheses from the recogniser/semantic decoder
- ability to recover from errors without scripted repair dialogues
- seamless integration with the recogniser providing barge-in, response timeouts and automatic filler ("um", "er", etc) detection

In a preliminary evaluation held in November consisting of 160 dialogues from 40 different speakers, the system demonstrated acceptable performance across a range of word error rates and user styles. Following, refinements made in the light of that evaluation, the system has been further improved and it is now judged to provide state-of-the-art performance. Furthermore, this is only a first prototype and there are many additional improvements that could be made including refinement of the probability models and finding a better summary state representation.

Most importantly of all, this prototype of the HIS system demonstrates that the POMDP framework is tractable and it can support real-world applications. This offers the genuine opportunity to develop a new class of dialogue system which can significantly out-perform the current generation of hand-crafted systems.

Chapter 1

Introduction

The structure of a conventional dialogue system is shown in Fig. 1.1 both in terms of a block diagram showing the data flow, and an influence diagram showing the dependencies from one time slot (i.e. turn) to the next.

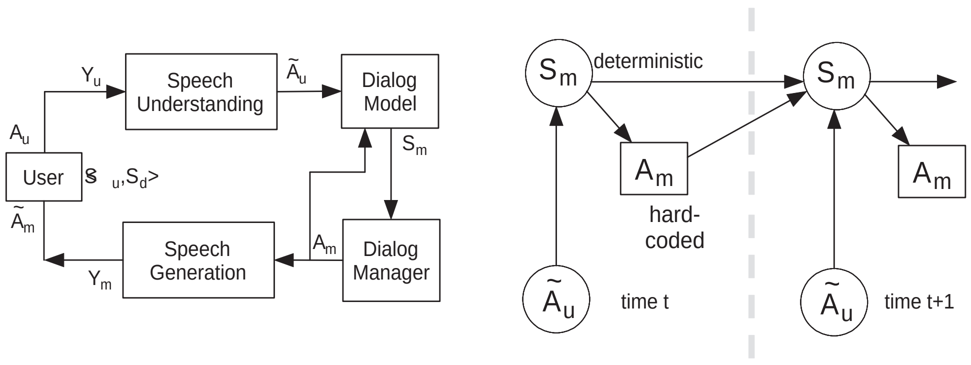


Figure 1.1: A traditional Spoken Dialogue System along with its corresponding influence diagram

The processing involved in a single dialogue turn proceeds as follows. A dialogue manager generates a prompt to the user in the form of a machine dialogue act A_m. This is converted to an acoustic signal Y_m and subsequently interpreted by the user as $\tilde{A}_m$. The user has a state which encodes both a goal to achieve S_u and the dialogue history S_d. On receiving, $\tilde{A}_m$ the user updates this state and generates a user dialogue act A_u. This is converted to an acoustic signal Y_u and interpreted by the system's speech understanding component to give $\tilde{A}_u$. The dialogue system maintains its own view of the world in state variable S_m. The estimate $\tilde{A}_u$ is used to update this estimate of the machine's state and based on this updated estimate, the dialogue manager generates a new machine dialogue act A_m.

Although greatly simplified, this description of the dialogue turn cycle applies to nearly all existing systems. In particular, the information state update approach (ISU) to dialogue system design can be viewed as a direct implementation of this model [2].

However, although it is simple and intuitive, this traditional deterministic dialogue model has a number of severe weaknesses. Firstly, and crucially, in real systems, the estimate of the user's dialogue act $\tilde{A}_u$ will be extremely noisy. Hence, the system state S_m must be updated based on a "best guess" of A_u and since this best guess will often be wrong, the system state can be easily corrupted by erroneous information. This will typically lead to misunderstanding and confusion, requiring a perhaps lengthy recovery dialogue to repair it. The incidence of this problem can be reduced by making use of a confidence measure output by the speech understanding component. This measure provides an estimate of $P(\tilde{A}_u|Y_u)$ which is typically compared with a threshold and based on the result either $\tilde{A}_u$ is accepted as true, or it is queried with the user. Unfortunately, however, confidence measures themselves are unreliable, and there is no clear basis for setting the threshold.

A second problem with the traditional architecture is that speech understanding errors are not the only source of uncertainty: the user's goals and intentions are uncertain and can change over time. Thus, a model of the user's goals and intentions must be integrated into the overall dialogue management process.

A final problem with the traditional architecture is the determinism itself. In order to implement optimal dialogue strategies, a system must predict the future in order to plan for differing eventualities. Since exact prediction is not possible, such plans can only be probabilistic and, as with the use of confidence thresholds, these can only be used in a very crude way by a deterministic decision process. Also, of course, it is very hard to adapt deterministic systems from training data, and in practice, adaptation is limited to manual system tuning following an off-line analysis of system logs. This process is labour intensive and cannot be extended to automatic on-line adaptation.

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 [3, 4]. Early attempts at using a statistical approach modelled the dialogue system as a Markov decision process (MDP) [5], and this approach was also explored in the TALK project [4, 6, 7]. MDPs provide a good statistical framework since they allow forward planning and hence dialogue policy optimisation through reinforcement learning. However, they suffer from a number of problems. Firstly, and crucially, MDPs assume that the machine state is observable. Hence, they cannot account for either the uncertainty in the user state (S_u and S_d in Fig. 1.1), or the uncertainty in the decoded user's dialogue act ($\tilde{A}_u$ in Fig. 1.1). Secondly, and perhaps less obviously, the MDP approach provides a poor interface for integrating heuristics. The main problem is that heuristics typically involve making hard decisions based on the assumed system state. However, in the case of an MDP, the assumed system state might be incorrect. To deal with this, the state must be expanded to include confidence measures so that the heuristics can deal explicitly with the uncertainty. However, this rapidly leads to an excessively large state space and complex heuristics.

A more general alternative to the fully observable MDP is the Partially Observable MDP (POMDP) [8]. A dialogue system based on a POMDP maintains a distribution over all possible states. This distribution is called the *belief state* and dialogue policies are based on this belief state rather than the true underlying state. The key advantage of the POMDP formalism is that it provides a complete and principled framework for modelling the main sources of uncertainty. Furthermore, when cast as a so-called Spoken Dialogue System POMDP (SDS-POMDP) [9, 10], the framework also allows heuristics to be incorporated in a very simple way since the principal components on which the heuristics depend (e.g. S_u and A_u) are by definition assumed to be true. In computational terms, this means that in an MDP heuristics are executed once per turn but have to be programmed to explicitly take account of uncertainty. In a POMDP, heuristics are much simpler to program because the state is assumed correct. They do however have to be executed many times per turn, once for each possible state value.

The use of POMDPs for any practical system is, however, far from straightforward. Since a belief distribution b of a discrete state S of cardinality $n+1$ lies in real-valued n-dimensional simplex, a POMDP can be thought of as an MDP with a continuous state space $b \in \Re^n$. Thus, assuming that the POMDP machine has a finite set of actions to select from, a POMDP policy is a mapping from partitions in n-dimensional belief space to actions. Not surprisingly these are extremely difficult to construct and whilst exact solution algorithms such as the Witness algorithm [11] do exist, they rarely scale to problems with more than a few states/actions. Fortunately, there are a number of ways of finding approximate solutions which are sufficiently accurate to yield useful results. Firstly, the very large "master state space" required to model real-world systems can be mapped into a more compact "summary state space" which although small is sufficiently detailed to allow effective planning[12]. Secondly, approximate solutions such as grid-based methods[13] and point-based value iteration[14] can be used to solve problems with several hundreds of state/actions, and although this is still insufficient for planning in master space, it is adequate for planning in summary space.

Whatever approach is taken to the construction of policies, there is an other fundamental barrier to using POMDPs in spoken dialogue systems. Real systems deal with real-world knowledge which is complex, hierachical, and multi-valued. The potential state-space of even a simple travel booking system is enormous. Furthermore, dialogue acts cannot easily be enumerated as a simple finite set. The types of act (*request*, *inform*, etc) are easily enumerated, but the arguments to such acts (names of places, prices, dates, etc) are not so simple. Whereas research into MDPs was able to side-step this problem on the grounds that only a few global indicators needed to be modelled[15, 16], a central claim of the POMDP approach is that it is truly holistic and, in particular, propositional content should not be ignored. Thus, whilst POMDPs provide a theoretical framework for modelling complete dialogue systems, what is also needed in practice is a framework which can integrate the applicable knowledge representations with the appropriate statistical models.

This report describes the development of the Hidden Information State (HIS) dialogue manager which can support POMDPs with very large hierarchical state spaces. The key idea of the HIS system is that a belief distribution over an extremely large state space can be represented efficiently by grouping states together into partitions[17, 18, 19, 20]. Initially, all states are deemed to be in a single partition with belief unity. As the dialogue progresses, the partitions are split and belief is redistributed amongst the splits. Eventually, some partitions become very low in cardinality, and in the limit singletons. Action selection is then dominated by these low cardinality partitions. The overall result is that full-scale POMDP belief maintenance is achieved without ever explicitly calculating the beliefs of (the majority of) irrelevant states. Furthermore, partitions are represented efficiently using tree structures and these tree structures also provide a very natural representation for real-world knowledge.

1.1 Report structure

The remainder of this report is structured as follows. Chapter 2 briefly reviews the general framework of the Spoken Dialogue System POMDP (SDS-POMDP) and chapter 3 explains how the HIS system fits into this framework. Chapter 4 then describes the implementation of the HIS system in some detail. Chapter 5 presents experimental results from a user trial held in November 2006 and also discusses future work and conclusions.

Chapter 2

The SDS-POMDP

The aim of this chapter is to review the basic POMDP equations and then present a factored form called the SDS-POMDP which is suitable for spoken dialogue systems [9, 10]. This lays the foundation for describing the HIS model in the following chapter.

2.1 POMDP Basics

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

The POMDP operates as follows. At each time-step, the machine is in some unobserved state $s \in S$. Since s is not known exactly, a distribution over states is maintained called a "belief state," b, with initial belief state b_0. Thus, the probability of being in state s given belief state b is $b(s)$. Based on the current belief state b, the machine selects an action $a_m \in A_m$, receives a reward $r(s, a_m)$, and transitions to a new (unobserved) state s', where s' depends only on s and a_m. The machine then receives an observation $o' \in O$ which is dependent on s' and a_m. The belief distribution is then updated based on o' and a_m.

The belief update equations are easily derived using Bayes rule:

$$\begin{aligned}
b'(s') &= P(s'|o', a_m, b) \\
&= \frac{P(o'|s', a_m, b)P(s'|a_m, b)}{P(o'|a_m, b)} \\
&= \frac{P(o'|s', a_m, b)\sum_{s\in S} P(s'|a_m, b, s)P(s|a_m, b)}{P(o'|a_m, b)} \\
&= \frac{P(o'|s', a_m)\sum_{s\in S} P(s'|a_m, s)b(s)}{P(o'|a_m, b)} \\
&= k \cdot P(o'|s', a_m) \sum_{s\in S} P(s'|a_m, s)b(s) \qquad (2.1)
\end{aligned}$$

where k is a normalising constant. In equation 2.1, the summation uses the transition probability to predict each next state s' as an expectation wrt to the belief state over preceding states. The observation term before the summation weights the prediction for each new state s' based on the likelihood that the most recent observation o' could have been generated from s'.

Note that the action taken by the machine at each time step depends on the complete distribution b. This is often initially a flat distribution reflecting ignorance. At each time-step, the belief state distribution b is updated based on the new observation, and typically this will result in the distribution "sharpening" around specific states.

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:

$$\begin{aligned} R &= \sum_{t=0}^{\infty} \lambda^t R(b_t, a_{m,t}) && (2.2) \\ &= \sum_{t=0}^{\infty} \lambda^t \sum_{s \in S} b_t(s) r(s, a_{m,t}). && (2.3) \end{aligned}$$

Each action $a_{m,t}$ is determined by a policy $\pi(b_t)$ and it is the goal of the machine to find the policy π^* which maximises the return. Such a policy is called an optimal policy. Since belief space is a real-valued simplex, the policy can be viewed as a partitioning of belief space into regions, where each region corresponds to the single unique action which should be taken if the current belief state lies in that region.

Finding the optimal policy involves using the transition matrix to predict the reward expected from each state for each possible machine action. This is very similar to the forward-backward algorithm of E-M and for regular fully observed Markov Decision Processes, it is essentially a dynamic programming search over a discrete state space. POMDPs solutions are much more complex, however, because the state space is effectively continuous. As mentioned in the introduction, exact solution algorithms do exist (e.g. see the Witness Algorithm [11, 8]) but they can only handle very small problems. Fortunately, approximate solutions can handle significantly larger problems (e.g. Perseus [14]).

For cases where the model is unknown or there is insufficient data to estimate accurately, on-line learning techniques analogous to Q-learning are also possible. For example, active learning can be used to simultaneously update an approximate model whilst optimising the return[21]. In the work described here, however, a simple grid-based batch-mode Monte Carlo learning scheme is used (see section 4.7)[22].

2.2 The SDS-POMDP: a factored POMDP for spoken dialog systems

Referring back to Fig. 1.1, it can be seen that the state space represented by the dialog model S_m must entail the user goal and dialog state and since these cannot be observed, S_m must correspond to a distribution over those states. In addition, since the last user act is also uncertain, it is convenient to include it also within the unobserved state space. This suggests that the state space of a POMDP for dialog systems should be factored as follows[9]. First, the unobserved state is factored into 3 components:

$$s = (s_u, a_u, s_d). \qquad (2.4)$$

The system state S_m then becomes the belief state b over s_u, a_u and s_d, i.e.

$$s_m = b(s_u, a_u, s_d). \qquad (2.5)$$

The observation o is the estimate of the user dialog act $\tilde{a}_u$. In the general case this will be an N-best list of hypothesised user acts, each with an associated probability, i.e.

$$o = [(\tilde{a}_u^1, p_1), (\tilde{a}_u^2, p_2), \ldots, (\tilde{a}_u^N, p_N)] \tag{2.6}$$

such that

$$P(\tilde{a}_u^n | o) = p_n, \qquad n = 1 \ldots N \tag{2.7}$$

The transition function for an SDS-POMDP follows directly by substituting equation 2.4 into the regular POMDP transition function and making some reasonable independence assumptions, i.e.

$$\begin{aligned} P(s'|s, a_m) &= P(s'_u, a'_u, s'_d | s_u, a_u, s_d, a_m) \\ &= P(s'_u | s_u, a_m) P(a'_u | s'_u, a_m) P(s'_d | s'_u, a'_u, s_d, a_m) \end{aligned} \tag{2.8}$$

Making similar reasonable independence assumptions regarding the observation function gives,

$$\begin{aligned} P(o'|s', a_m) &= P(o'|s'_u, a'_u, s'_d, a_m) \\ &= P(o'|a'_u) \end{aligned} \tag{2.9}$$

This is the *observation model*.

The above factoring simplifies the belief update equation since substituting equation 2.8 and equation 2.9 into equation 2.1 gives

$$b'(s'_u, a'_u, s'_d) = \tag{2.10}$$

$$\begin{aligned} & k \cdot P(o'|a'_u) \sum_{s_u, a_u, s_d} P(s'_u|s_u, a_m) P(a'_u|s'_u, a_m) P(s'_d|s'_u, a'_u, s_d, a_m) b(s_u, a_u, s_d) \\ = \; & k \cdot P(o'|a'_u) P(a'_u|s'_u, a_m) \sum_{s_u} P(s'_u|s_u, a_m) \sum_{s_d} P(s'_d|s'_u, a'_u, s_d, a_m) \sum_{a_u} b(s_u, a_u, s_d) \\ = \; & k \cdot \underbrace{P(o'|a'_u)}_{\substack{\text{observation} \\ \text{model}}} \underbrace{P(a'_u|s'_u, a_m)}_{\substack{\text{user action} \\ \text{model}}} \sum_{s_u} \underbrace{P(s'_u|s_u, a_m)}_{\substack{\text{user goal} \\ \text{model}}} \sum_{s_d} \underbrace{P(s'_d|s'_u, a'_u, s_d, a_m)}_{\substack{\text{dialog} \\ \text{history} \\ \text{model}}} b(s_u, s_d) \end{aligned} \tag{2.11}$$

As shown by the labelling to equation 2.11, 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 *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 *dialog history model* represents the transition matrix for the dialog state component. This term allows information relating to the dialog history to be maintained such as grounding and focus.

2.3 POMDP Policy Optimisation and Summary Space

A POMDP policy of exactly t steps can be described by a *t-step conditional plan* which is a branching tree in which each node is associated with an action and each branch is labelled by an observation (see Fig. 2.1). To execute such a policy, the machine would first take the action at the root node then on

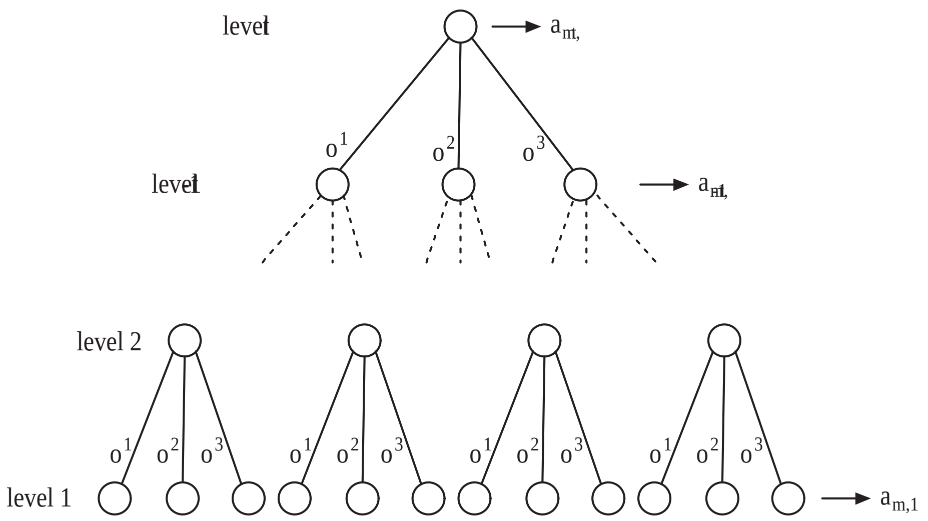


Figure 2.1: A t-Step Conditional Plan (for a problem with 3 possible observations)

receiving the subsequent observation, it would follow the corresponding branch and execute the action at the daughter node. This would be repeated until a leaf node was reached $t-1$ steps later. Let each node be labelled with its level and the observation that leads to it, where the level at the root node is t and at the leaf nodes is 1. Then if the current state is s_m, the expected reward for a given conditional plan is given by its *value function* which can be computed recursively for $\tau = 1 \dots t$ by

$$V_\tau(s_m) = \begin{cases} r(s_m, a_{m,\tau}), & \text{if } \tau = 1 \\ r(s_m, a_{m,\tau}) + \lambda \sum_{s'_m} P(s'_m | s_m, a_{m,\tau}) \sum_{o'} P(o' | s'_m, a_{m,\tau}) V^{o'}_{\tau-1}, & \text{otherwise} \end{cases} \tag{2.12}$$

In practice, the current state is not known, however, the expected value for belief state b is just

$$V(b) = \sum_s b(s) V(s) \tag{2.13}$$

Thus, the expected value function associated with a given conditional plan is a hyperplane in belief space. If all possible t-step conditional plans are enumerated to form the set $\mathcal{N}_t$, the value of the best plan at belief state b is just

$$V^*_{\mathcal{N}_t}(b) = \max_{n \in \mathcal{N}_t} \sum_s b(s) V^n(s) \tag{2.14}$$

This is the optimal value function and the action associated with the root node of the member in $\mathcal{N}_t$ which maximises $V^*_{\mathcal{N}_t}(b)$ belongs to the optimal policy. Since the value function of each plan is just a hyperplane in belief space, the piece-wise linear convex surface formed from the upper surface of the full set of plans defines the optimal value function. This is illustrated graphically in Fig. 2.2 which shows the hyperplanes for 5 different plans in a simple binary state space. The optimal value function is the upper dotted surface. When $s = s_a$ at the left side of belief space, the optimal value function belongs to plan 1. When $s = s_b$ at the far right side of belief space, the optimal value function belongs to plan 5. In the middle of belief

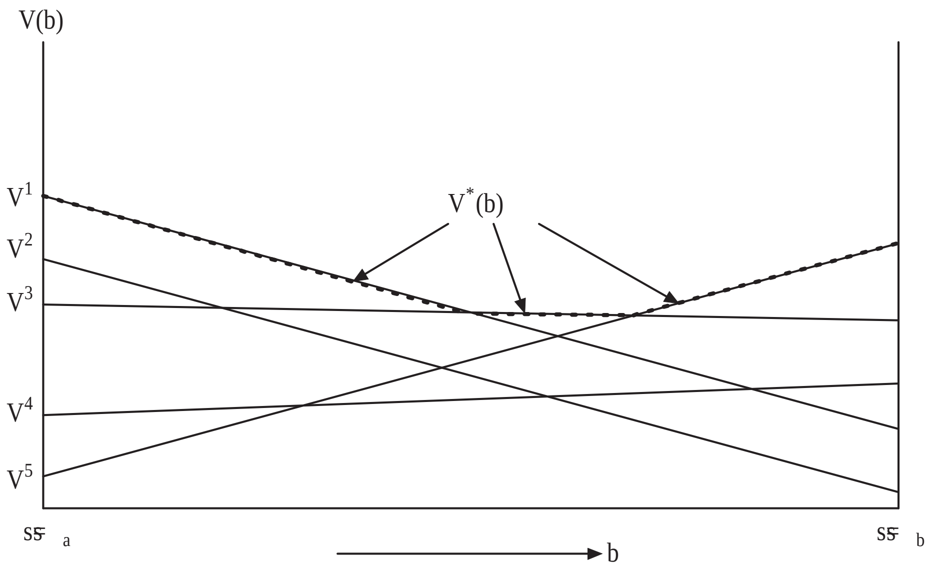


Figure 2.2: Value Function Hyperplanes

space, plan 3 is optimal. Note that plans 2 and 4 do not contribute to the optimal solution and can be discarded.

In summary, an optimal policy for a POMDP is found by generating all plans at step 1 and computing their value functions. Then all plans at step 2 are generated by taking all combinations of the plans at step 1 extended by 1 step, and computing their value functions using (2.12). Any plans which do not contribute to the optimal 2-step solution must be pruned to limit the combinatorial explosion in the number of possible plans. The process is then repeated for $t = 2,3,\ldots.$ until $|V_t^*(b) - V_{t-1}^*(b)|$ is sufficiently small for all b.

The above exact POMDP solution algorithm is in practice intractable since even the most efficient algorithms (see eg [11]) can only cope with a few (i.e. less than 10) states/actions/observations. Hence, in practice, approximate algorithms must be used to find good solutions for real-world problems such as spoken dialogue.

The full state-space of the SDS-POMDP is very large indeed, and the generation of (approximately) optimal policies requires two steps. Firstly, the full state-space, referred to as the *master space*, must be mapped into a simpler *summary space*. Secondly, the exact POMDP solution approach described above must be simplified by discretising belief space. The former reduces the state space to a few hundred states, and the latter makes this size of state space tractable.

Summary space mapping is relatively straightforward. The basic idea is that at each turn, immediately after updating beliefs, the set of master states S is mapped into a much reduced set of summary states $\tilde{S}$ where $\tilde{S}$ consists of the top N states in S_u with the highest belief, plus abstracted versions of S_d and A_u. Since N is typically just 1 or 2, the resulting state space is very compact. The mapping from master to summary space is made invertable by ensuring that actions in summary space implicitly refer to the currently most likely, or next-most likely user goal. Thus, for example, a *confirm* act in summary space for which the top goal was currently “venue(hotel,location=central)” would map back into “confirm(type=hotel, location=central)” in master space. Similarly, if the next to top goal was “venue(bar,location=central”, the summary space action “select” would map into “select(venue=hotel,venue=bar)” in master space.

The mapping to summary space brings the SDS-POMDP within reach of practical solutions algorithms. These typically discretise belief space by selecting a set of belief points which will cover (i.e. lie close to) all of the belief states that the system is likely to encounter. This set is usually constructed by running the system in some form of simulation mode. Either by sampling from its own distributions, or by running it with an external user simulator.

In the simplest grid-based methods, a set of distinct belief points are created and value functions estimated at just those points. This essentially reduces the problem to a classic MDP optimisation problem and all of the solutions developed for MDPs apply. A slightly more elaborate scheme called point-based value iteration (PBVI) discretises belief space, but then calculates a value hyperplane for each belief point[14]. For many problems, PBVI appears to be one of the most efficient algorithms known. However, in all grid-based methods there is a trade-off between the complexity of the value estimation and the number of points. In spoken dialogue applications, even summary spaces are quite large and, in practice, it appears that simple grid-based schemes often perform better than PBVI [23].

The current version of the HIS dialog manager uses a simple grid-based batch mode Monte Carlo reinforcement learning scheme, which relies on an external user simulator to drive the system through a large number of learning cycles. It is described further below.

Chapter 3

The Hidden Information State dialog model

Having reviewed the general form of the SDS-POMDP in the previous chapter, this chapter derives a specific form of SDS-POMDP called the Hidden Information State model.

Although the factoring introduced in the last chapter is helpful, the size of the state spaces needed to represent real-world dialog systems would quickly render a direct SDS-POMDP implementation intractable. The dialog history component is computed heuristically and as will be explained later, this results in a relatively small set of dialog states being tracked from turn to turn. However, the user goal and action state components require reasonably accurate distributions to be maintained and this is not easy since the size of the user goal space is enormous and the set of user actions cannot even be enumerated. The HIS model deals with these two components in different ways.

3.1 User action model

Consider first the user action model. As shown by equation 2.11 of the previous chapter, the user action component of the state space is memoryless, i.e. the value of the previous user action a_u is not required to apply the belief update equation. This means that the distribution for a'_u can be approximated by considering just those user action values which are deemed to have non-zero probabilities in the current turn. These will be precisely those actions which appear in the N-best list of hypotheses from the speech understanding component. To guard against the case of very poor recognition resulting in the correct value of a'_u being dropped from the observation altogether, a *null* action is always included with a floor probability representing all of the user acts not in the N-best list.[1]

3.2 User goal component

To deal with the user goal component, it is necessary to be a little more specific about what is meant by a *user goal*. The initial target of the HIS model is database inquiry applications such as traffic information,

[1]Note that in the context of a POMDP-based spoken dialog system, the terms *user act* and *user action* are synonymous.

tourist information, flight booking, etc. In this context, a user goal is deemed to be a specific entity that the user has in mind. For example, in a tourist information system, the user might be wishing to find a moderately priced restaurant near to the theatre. The user would interact with the system, effectively refining his or her query until an appropriate establishment was found. If the user wished to find an alternative restaurant, or even something different entirely such as the nearest tube station to the restaurant, this would constitute a new goal. In the HIS system, the duration of a dialog is defined as being the interaction needed to satisfy a single goal. Hence by definition, the user goal model simplifies trivially to a delta function, i.e.

$$P(s'_u|s_u) = \delta(s'_u, s_u). \tag{3.1}$$

Substituting equation 3.1 into equation 2.11 gives

$$b'(s'_u, a'_u, s'_d) \quad = \quad k \cdot P(o'|a'_u)P(a'_u|s'_u, a_m)\sum_{s_d} P(s'_d|s'_u, a'_u, s_d, a_m)b(s'_u, s_d) \tag{3.2}$$

To further simplify belief updating, it will be assumed that at any time t, S_u can be divided into a number of equivalence classes where the members of each class are tied together and are indistinguishable. These equivalence classes will be called *partitions* of user goal space. Initially, all states $s_u \in S_u$ are in a single partition p_0. As the dialog progresses, this root partition is repeatedly split into smaller partitions. This splitting is binary

$$p \rightarrow \{p', p - p'\} \qquad \text{with probability} \qquad P(p'|p). \tag{3.3}$$

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

$$b_0(p_0) = 1. \tag{3.4}$$

Whenever a partition p is split, its belief mass is reallocated according to equation 3.3, i.e.

$$b(p') = P(p'|p)b(p) \qquad \text{and} \qquad b(p - p') = (1 - P(p'|p))b(p) \tag{3.5}$$

Note that this splitting of the belief mass is simply a reallocation of existing mass, it is not a belief update. It will be referred to as *belief refinement*.

3.3 Belief updating

The belief update equation for a partitioned state space is easily derived from the non-partitioned case. Let partition p' consist of states $\{s'_u|s'_u \in p'\}$, then summing both sides of equation 3.2 over all $\{s'_u\}$ gives,

$$b'(p', a'_u, s'_d) \quad = \quad k \cdot P(o'|a'_u) \sum_{s'_u \in p'} P(a'_u|s'_u, a_m)\sum_{s_d} P(s'_d|s'_u, a'_u, s_d, a_m)b(s'_u, s_d) \tag{3.6}$$

As a dialog progresses, the user goal partitions are split repeatedly to ensure that everything which has been mentioned so far in the dialog is explicitly represented in the partitions. This being so, it is reasonable to assume that

$$P(a'_u|s'_u, a_m) = P(a'_u|p', a_m) \tag{3.7}$$

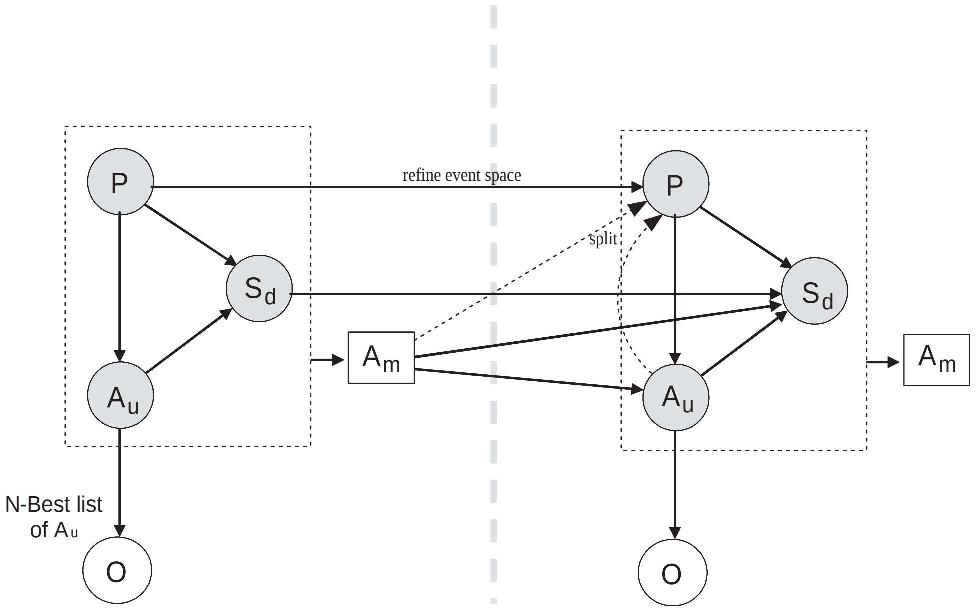


Figure 3.1: Influence diagram for the Hidden Information State dialog model

and

$$P(s'_d|s'_u, a'_u, s_d, a_m) = P(s'_d|p', a'_u, s_d, a_m) \tag{3.8}$$

Hence, using these simplifying assumptions and equation 3.5, equation 3.6 becomes

$$\begin{aligned} b'(p', a'_u, s'_d) &= k \cdot P(o'|a'_u)P(a'_u|p', a_m) \sum_{s_d} P(s'_d|p', a'_u, s_d, a_m) \sum_{s'_u \in p'} b(s'_u, s_d) \\ &= k \cdot P(o'|a'_u)P(a'_u|p', a_m) \sum_{s_d} P(s'_d|p', a'_u, s_d, a_m) b(p', s_d) \\ &= k \cdot \underbrace{P(o'|a'_u)}_{\text{observation model}} \underbrace{P(a'_u|p', a_m)}_{\text{user act model}} \sum_{s_d} \underbrace{P(s'_d|p', a'_u, s_d, a_m)}_{\text{dialog history model}} \underbrace{P(p'|p) b(p, s_d)}_{\text{belief refinement}} \end{aligned} \tag{3.9}$$

where p is the parent of p'. Equation 3.9 is the belief update equation for the HIS model, it is shown in the form of an influence diagram in Fig. 3.1. Note that in this diagram the dotted arrows represent the influence of a_m and a_u on the *refinement* of p' but not on its update i.e. they influence the splitting of p' but not its conditional probability.

As shown by the labelling on equation 3.9, the HIS update equation depends on four probability distributions:

1. *Observation Model* - this is approximated by the N-best probability from the speech understanding component

$$P(o'|a'_u) \approx k' \cdot P(a'_u|o) \tag{3.10}$$

2. *User Action Model* - this is composed of two parts: the bigram probability of the current user act type given the preceding system act type, and a probability denoting the degree to which the current user act is consistent with the given partition p'. Thus,

$$P(a'_u|p',a_m) \approx P(\mathcal{T}(a'_u)|\mathcal{T}(a_m))P(\mathcal{M}(a'_u)|p') \tag{3.11}$$

where $\mathcal{T}(a)$ denotes the *type* of the dialog act a, for example, the type of the act "inform(food=Indian)" is *inform*. There are a total of 12 different dialog act types supported by the HIS model and these are described in detail section 4.3. $\mathcal{M}(a)$ denotes whether or not the dialog act *a matches* the current partition p'. The first component can be estimated from a dialog corpus, the second component is set to 1 if the act matches and zero otherwise.

3. *Dialog History Model* - this is entirely heuristic.

$$\begin{aligned} P(s'_d|p',a'_u,s_d,a_m) &= 1 \quad \text{iff } s'_d \text{ is consistent with } p',a'_u,s_d,a_m & (3.12)\\ &= 0 \quad \text{otherwise} & (3.13) \end{aligned}$$

The way that this is computed in the HIS model is described in section 4.

4. *Belief Refinement* - this depends on the ontology rules used to define the application domain. User goals are built using probabilistic context free rules, with rule probabilities set *a priori*. If the sequence of rules $r_1,r_2,\ldots,r_k$ is used to split partition p into sub-partition p', the belief refinement probability is

$$P(p'|p) = \prod_{i=1}^{k} P(r_i) \tag{3.14}$$

where $P(r)$ is the prior probability of rule r. This process is described in more detail in section 4.2.

Having described the mathematical basis of the HIS model, the remainder of this report describes its specific implementation.

Chapter 4

Implementation of the HIS Model

This chapter describes a specific implementation of the HIS model. It begins with a high level overview of how the model operates. It then describes each of the main components in more detail.

4.1 Overview of HIS Model Operation

Before describing the details of the HIS system, it will be helpful to give a brief overview of the principal data structures and the overall operation. As shown in Fig. 4.1, 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. The user goal is represented by a set of branching tree structures each of which initially consist of just a single node. These tree structures can be grown downwards by applying ontology rules which describe the application domain. For example, there might be a rule which states that a `venue` can be either a `hotel`, a `restaurant` or a `bar`. In each case, the derived venues will have further nodes describing features of that type of venue. Ambiguity is represented by allowing nodes to expand into multiple alternatives. Each distinct tree forms a partition of user goal space as described in section 3. The initial single tree node represents a single partition with belief unity. As the trees are grown, the partitions are repeatedly split allowing the belief assignment to be refined. Eventually, the hope is that a single complete tree will be formed which represents the actual user's goal and that this tree has a high belief.

The tree growing process is driven entirely by the dialog 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. For example, if the act was `request(ensuite)`, and the partition represented the higher level node `venue`, then the venue node would be extended to a hotel node with associated properties, one of which would be `ensuite`. The `request(ensuite)` act would then match. Note however that an ontology rule can be used to extend a specific node just once. This ensures that all partitions are unique and there are no duplicates.

Once the matching and partition splitting is complete, all the partitions are rescanned and where possible each hypothesised input user act is attached to each partition. Similarly the system act is attached to each partition (not shown in the figure). The combination of a partition and an input user act (p, a_u) forms a partial hypothesis and the user act model probability is calculated as in equation 3.11.

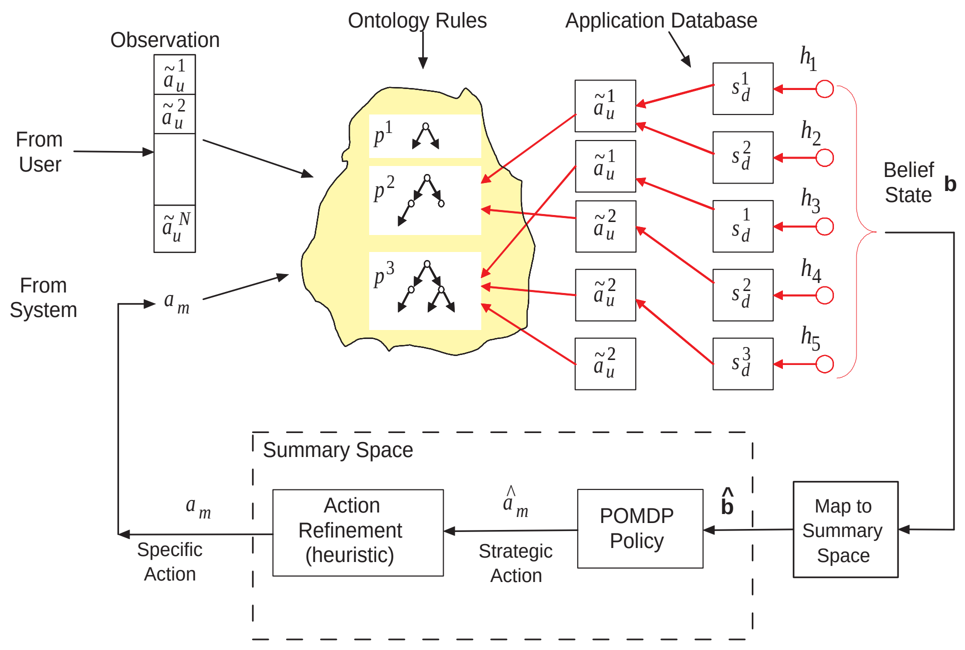
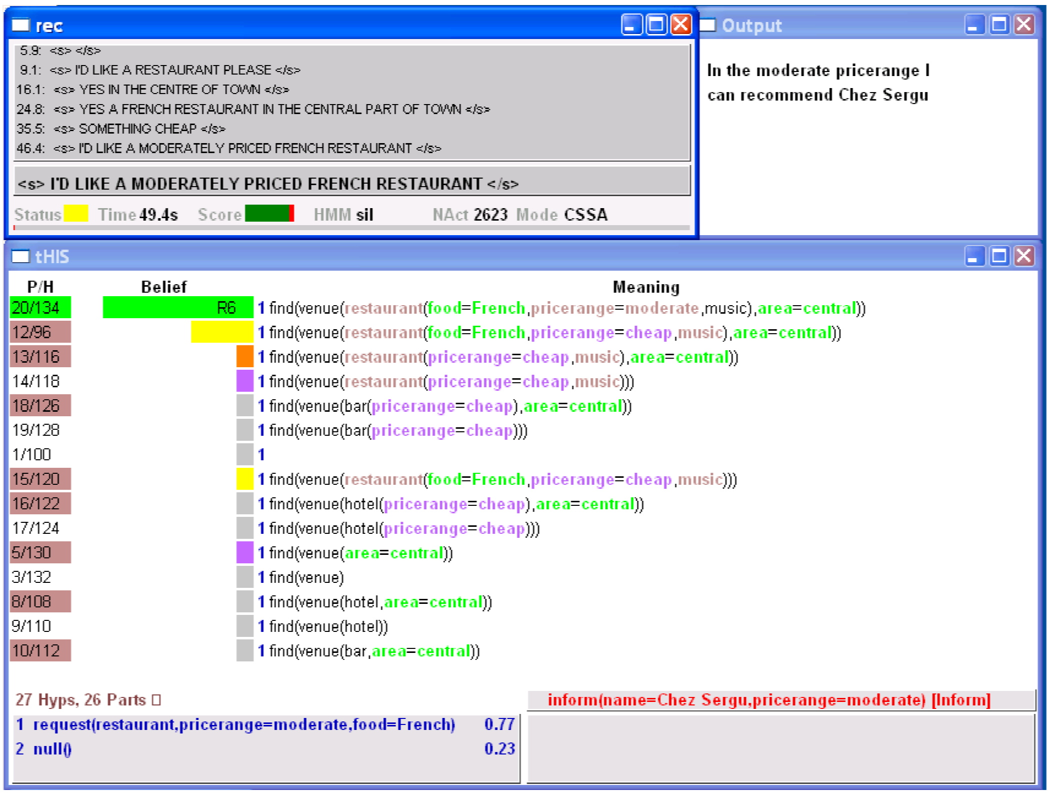


Figure 4.1: Overview of the HIS System operation

As explained above, partitions are grown based entirely on dialog act inputs. If the user (or the system) mentions a node such as `ensuite` this will cause other nodes to be created. The grounding status of each tree node is recorded in a dialog state data structure. Since the grounding status of a tree node can be uncertain, any (p, a_u) pair can have multiple dialog states attached to it. However, unlike the user act component of the state which is memoryless, the dialog component s_d evolves as the dialog progresses. Thus, at the beginning of each dialog cycle, the various dialog state instances are attached directly to the partitions. Once the input user acts have been attached to the partitions, the current dialog states are extended to represent the new information in the dialog acts. At this point, the dialog state probabilities given by equation 3.13 are computed. At the end of the turn, identical dialog states attached to the same partition are merged ready for the next cycle.

Every triple (p, a_u, s_d) represents a single dialog hypothesis h_k. The belief in each h_k is computed using equation 3.9 and the complete set of values $b(h_k)$ represents the current estimate of the POMDP belief state in master space. This belief state is then mapped to summary space and input to the POMDP policy which yields a high level strategic summary space action. This strategic action defines the broad class of response (e.g. request more info, clarify, confirm, etc) and which set of hypotheses it refers to (topmost, top-two, rest). Given this strategic action and the information in the relevant partitions and dialog histories, the action refinement heuristics map the summary space action back into master space where it generates a specific system action.

A screen shot of the HIS system in operation is shown in Fig 4.2. The main central display shows the top few hypotheses. Above this is the recogniser output and below are the semantic decoder inputs from the previous turn (left) and the current turn (right). In the top right, is a text version of the speech output.



```
5.9:  <s> </s>
9.1:  <s> I'D LIKE A RESTAURANT PLEASE </s>
16.1:  <s> YES IN THE CENTRE OF TOWN </s>
24.8:  <s> YES A FRENCH RESTAURANT IN THE CENTRAL PART OF TOWN </s>
35.5:  <s> SOMETHING CHEAP </s>
46.4:  <s> I'D LIKE A MODERATELY PRICED FRENCH RESTAURANT </s>

<s> I'D LIKE A MODERATELY PRICED FRENCH RESTAURANT </s>

Status   Time 49.4s   Score   HMM sil   NAct 2623  Mode CSSA
```

```
In the moderate pricerange I
can recommend Chez Sergu
```

```
P/H        Belief     Meaning
20/134       R6    1 find(venue(restaurant(food=French,pricerange=moderate,music),area=central))
12/96              1 find(venue(restaurant(food=French,pricerange=cheap,music),area=central))
13/116             1 find(venue(restaurant(pricerange=cheap,music),area=central))
14/118             1 find(venue(restaurant(pricerange=cheap,music)))
18/126             1 find(venue(bar(pricerange=cheap),area=central))
19/128             1 find(venue(bar(pricerange=cheap)))
1/100              1
15/120             1 find(venue(restaurant(food=French,pricerange=cheap,music)))
16/122             1 find(venue(hotel(pricerange=cheap),area=central))
17/124             1 find(venue(hotel(pricerange=cheap)))
5/130              1 find(venue(area=central))
3/132              1 find(venue)
8/108              1 find(venue(hotel,area=central))
9/110              1 find(venue(hotel))
10/112             1 find(venue(bar,area=central))

27 Hyps, 26 Parts □                                     inform(name=Chez Sergu,pricerange=moderate) [Inform]
1  request(restaurant,pricerange=moderate,food=French)  0.77
2  null()                                               0.23
```

Figure 4.2: Screenshot of the Prototype HIS System

4.2 User Goal Trees and Ontology Rules

User goals are represented by a branching tree structure whose hierarchy reflects both the natural structure of the data and a natural order in which to introduce the individual concepts into a conversation. User goal trees are constructed from four types of tree node:

1. class nodes - these have non-terminal offspring. Conceptually a class node represents an instance of a type, and the offspring of the node denote the members of that type.
2. lexical nodes - these have only terminal offspring i.e. atoms.
3. subclass nodes - these have no offspring. They act like a tag to the parent node indicating a particular flavour of that class. They are provided mainly for notational convenience, especially in the way that database entitities are defined.
4. atomic nodes - these are the offspring of lexical nodes. They represent actual values such as `Hotel Grand`, `Jazz`, `yes`, `27`, etc.

An example of a fully expanded user goal tree is shown in Fig. 4.3. This example is a simplified representation of a restaurant. The top level node represents an arbitrary entity. It has a subclass `venue` and corresponding subclass members `type`, `name`, and `location`. These members are generic for any kind of venue (e.g. restaurant, bar, hotel, etc). In this case, the type is a restaurant with restaurant-specific class members `food`, `music` and `decor`. The location is specified as a specific address and therefore has a `street` member. It could have been specified by some other means such as `nearto`, `gridref`, etc, and these would be alternate subclasses of location.

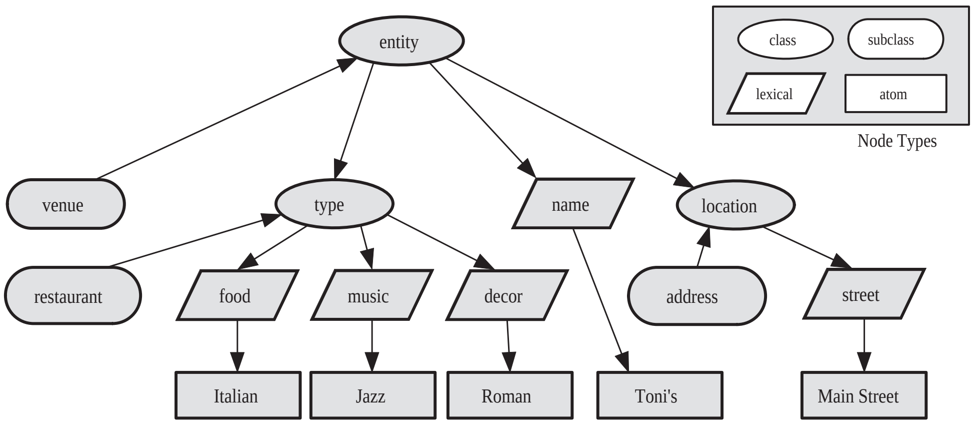


Figure 4.3: Example Fully Expanded User Goal Tree

User goal trees are built using a set of rules which adhere to the syntax set out in Fig. 4.4[1]. As an example, the rules set out in Fig. 4.5 describe the restaurant goal described above. There are two basic forms of rules: class definition rules and lexical definition rules. The basic function of these should be clear from the table, however, some of the details require further explanation.

[1]Atomic names containing non-alphadigit characters must be enclosed in double quotes

```
ruleset   = ruledef";" { ruledef ";" } {dbasefile}
ruledef   = classdef | lexdef
classdef  = classinst "->" [subclass] [classbody] [prob]
classbody = "(" [opt] member { "," [opt] member } ")"
lexdef    = classinst "=" "(" atom[prob] {"|" atom[prob] ")"
prob      = "{" float "}"
opt       = "-" | "+"
classinst = name {"." name}
member    = name
subclass  = name
atom      = name
dbasefile = "+" "filename"
```

Figure 4.4: Syntax of HIS Ontology Rules

Firstly, the members of a class can have an optional “+” or “-” specifier indicating that the node is primarily *selectional* or *informational*, respectively. These markers are optional and only influence the selection of system responses. The plus specifier indicates that a value is normally required for that member in order to identify the requested entity. Conversely, the minus specifier indicates that the member will rarely be specified by the user to identify the entity but does contain information that the user may wish to know about once the entity has been selected. In the example rules, the food type is marked with a “+” since it is frequently specified by users in order to identify a suitable restaurant, whereas the `decor` is marked with a “-” since it is rarely specified by users when searching for an appropriate restaurant. It might, however, be required once a candidate restaurant has been located.

Secondly, note that in the left hand side of class definition rules, a simple name can be qualified using a dotted path notation. This is provided as a convenience to allow generic labels such as `name` to be used in different contexts, and then specific instances identified. In the example, the lexical definition for `name` is qualified by `venue` to distinguish it from other types of name.

Finally, all rules can have a probability assigned to them. Where no probability is given, then equal probability is assumed. These probabilities represent prior knowledge. In the example, the venue type is restaurant with probability 0.35. This would reflect the fact that in practice when users want to locate a venue, 35% of the time they require a restaurant. As explained in section 4.4, these prior probabilities are used to reallocate belief mass when a partition is split.

The ontology rules defined above describe the structure of the data. The data itself must be stored in a second file in the form of entity definitions, where each entity consists of a list of attribute value pairs. An example entity definition is shown in Fig. 4.6. Entity definitions must begin with an `id` attribute and should normally include `name` and `type` attributes. All remaining attribute-value pairs are arbitrary but must be consistent with the rules. For example, all values must appear in at least one lexical definition[2].

The HIS system attempts to interpret attribute value pairs in a flexible way. For example, given the location rule in Fig. 4.5, an address could be specified by any of: `addr("Main Street")`, `location("Main Street")` or `street("Main Street")`. Note, however, that if there was also a rule such as

[2]Numbers are dealt with as a special case

```
entity     -> venue(type,name,location) {0.2};
type       -> restaurant(+food,music,-decor) {0.35}
location   -> addr(street) {0.8};
venue.name =  ("Toni’s","Quick Bite", ....);
food       =  (Italian,Chinese,English, ...);
music      =  (Jazz,Pop,Folk, ...);
decor      =  (Traditional,Roman,ArtDeco,...
street     =  ("Main Street", "Market Square", ...);
```

Figure 4.5: Example of using Ontology Rules

```
id("R23")
name("Toni’s")
type("restaurant")
food("Italian")
addr("Main Street")
near("Cinema")
phone("2095252")
decor("Roman")
```

Figure 4.6: Example Database Entity Definition

```
location -> nearto(street);
```

then the latter two forms would be ambiguous.

4.3 Dialog Acts

As shown in Fig 4.7, a dialog act consists of a type and a list of zero or more `name=value` pairs referred to as `items`. An item name refers to a node in a user goal tree, it can be a simple name or a qualified name where the qualifier is either the name of the parent node or the name of the parent’s subclass, if any. There may be zero or many items in a single act, and the interpretation depends on the act type of which there are 15 in total.

The full set of acts supported by the HIS system is summarised in Table 4.1. The meaning of each act

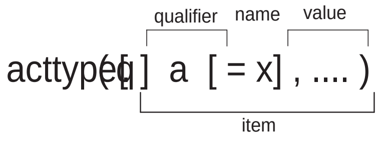


Figure 4.7: Structure of a Dialog Act

should be clear from the table, but the following amplifies a number of important points.

Firstly, the HIS system does not support multiple dialog acts in a single turn. Thus, for example, if

```
U: inform(food=Italian)
U: inform(music=Jazz)
```

is input to the system, it is interpreted as

```
U: inform(food=Italian) {0.5}
U: inform(music=Jazz)   {0.5}
```

i.e. the user said either that the food is Italian *or* that the music is Jazz with equal probability. To convey both pieces of information in a single turn, an `inform` act with two items must be used, i.e.

```
U: inform(food=Italian, music=Jazz)
```

In some cases, items are treated differently depending on their position in the item list. For example,

```
S: confreq(type=restaurant,food)
```

is a request to confirm that the required type is restaurant and then request a value for food. If the response was

```
U: affirm(type=restaurant, food=Italian)
```

this would confirm the type and provide the required food information. The sequence

```
S: confirm(type=restaurant)
U: affirm()
```

is identical to

```
S: confirm(type=restaurant)
U: affirm(type=restaurant)
```

If `negate` is used, however, the first item is taken to be a correction thus the response

```
U: negate(food=Russian)
```

would be interpreted as “No, I want Russian food”. To refute a specific value, `deny` is used, for example

```
U: deny(food=Italian,food=Russian)
```

means “The food is not Italian, it is Russian”.

There is a special qualifier `more`. This can be used only inside a user request act and a system hello() act, for example,

```
U: request(more)
```

Act	System	User	Description
hello()	√	√	start dialog
hello(more)	√	×	prompt for more
bye()	√	√	end dialog
inform(a=x,b=y,...)	√	√	give information a=x, b=y, ...
inform(name=none)	√	√	inform user that no suitable entity found
request(a,b,...)	√	√	request values for a,b, ...
request(more)	×	√	request more information
request(more.a)	×	√	request more information about a
confirm(a=x,b=y,..)	√	×	confirm a=x,b=y,..
confreq(a=x,..,c=z, d)	√	×	confirm a=x,..,c=z and request value of d
select(a=x,b=y)	√	×	select either a=x or b=y
affirm()	×	√	simple yes
affirm(a=x,b=y,...)	×	√	confirm and give further info a=x, b=y, ...
negate()	×	√	simple no
negate(a=x,b=y,...)	×	√	no, a=x and give further info b=y, ...
deny(a=x,b=y)	×	√	no, a!=x and give further info b=y, ...
repeat()	√	√	request to repeat last act
reqalts()	×	√	request alternative goal
reqalts(a=x,..)	×	√	request alt with new information
null()	√	√	null act - does nothing
xxx(a=dontcare)	×	√	in any info item a can be any value

Table 4.1: Supported Dialog Acts

means "Tell me more about the current suggestion" and

```
U: request(more.hotel.name="The Grand")
```

means "Tell me more about "The Grand" hotel". The system act

```
S: hello(more)
```

means "Do you want anything more?".

When an act is processed by the HIS system, its items are matched against the user goal tree. If a value is given, then an item can only match if there is an atomic leaf node with the same value and its parent (or the subclass of its parent) matches the name of the `name=value` pair. If no value is given then the name must match a node in the tree. If the name is qualified, then the qualifier must match the parent (or the subclass of the parent) of the matched node.

User dialog acts are presented to the system as lists of alternatives. Each alternative can have a probability attached to it. All acts without probabilities are assumed equally likely and assigned probabilities so as to make the total sum to one. Every input list must include a null dialog act with a non-zero probability. If no null act is included, the system inserts one.

4.4 Partitions and Partition Splitting

Section 4.2 explained how a single user goal is encoded in a branching tree structure. In fact, the HIS system maintains a forest of partially and fully-expanded trees. Each partially expanded tree represents a partition of equivalent user goal states. Each fully expanded tree is also a partition, but it is a singleton partition i.e., it encodes a single user goal state.

This forest of trees is stored in such a way that no partition is duplicated, and the sum of the probability of all partitions is always unity. As shown in Fig. 4.8(a), at system start up the user goal forest consists of a single node called `task`. This single partition p has belief $b(p) = 1$ and it represents all possible user goals. Since this node is built by default, all application rule sets must start with rules to expand this node. Thus, in practice, the rule set shown in Fig. 4.5 must be augmented by a rule such as:

```
task -> find(entity) {0.3};
```

which expresses the prior knowledge that 30% of the time, a user will wish to find something (e.g. a hotel, a restaurant etc). Fig. 4.8(b) shows what happens when this rule is applied. The `task` node is split into two parallel nodes and the probability mass is divided in proportion to the prior probability of applying the rule. The result is two partitions with beliefs $b = 0.7$ and $b = 0.3$ respectively. Suppose now that the rule for `entity` in Fig. 4.5 is applied, partition 2 is split to form a new partition and the belief mass is divided again. The result is as shown in Fig. 4.8(c). And so the process continues. The result in this case is three partitions which can be described via their leaf nodes as

```
P1: task {0.70}
P2: find(entity){0.24}
P3: find(venue(type,name,location)){0.06}
```

where the belief in each partition is shown in braces and always sums to one. Note that these prior beliefs give relatively high weight to unexpanded nodes because they represent the largest equivalence sets. However, once belief updating occurs, this situation is quickly reversed since the evidence typically supports only the more specific partitions.

The above explains how partitions are split but not when. In fact partition splitting is entirely on demand and it is driven by the items in the input user and system dialog acts. Referring back to Fig. 4.1, the first stage of the dialog cycle is to match the items of all of the input user acts and the previous system act against all of the existing partitions. Note that the act type is not relevant here since the goal is simply to expand the partitions sufficiently to match as many as possible of the input act items. Each item of each act is taken in turn and applied against each existing partition. If the item matches the partition, then the result is recorded and nothing further happens. If however the item does not match, then the ontology rules are scanned and the system tests to see whether the current partition could be extended sufficiently to allow the item to match. If it concludes that a match is possible, then the partition is extended and a match is recorded. For example, if the user goal forest was as shown in Fig. 4.8(b) at the point when the item `(music=Jazz)` was received, then the system would determine that a match could be achieved by first expanding the `entity` node using the first rule in Fig. 4.5. This node is referred to as the *expansion node*. The newly created offspring of the expansion node includes a `type` node and this can be expanded using the second rule in Fig. 4.5. Finally, expanding the lexical node `music` to derive the atomic node `Jazz` would allow the required match. Having determined that it is indeed possible to construct a *matching subtree* which if attached to the expansion node would support an item match, then that matching subtree is created.

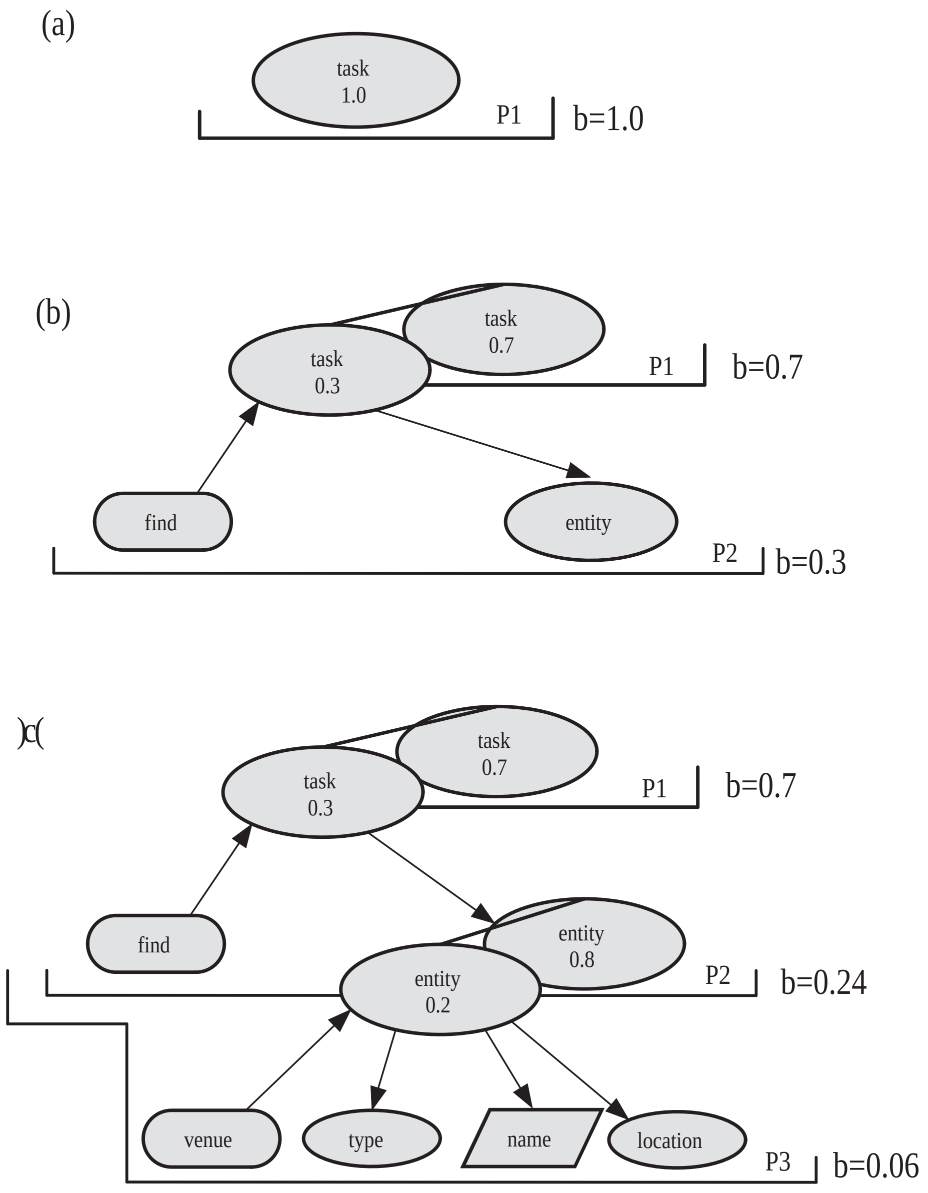


Figure 4.8: Example of Partition Splitting

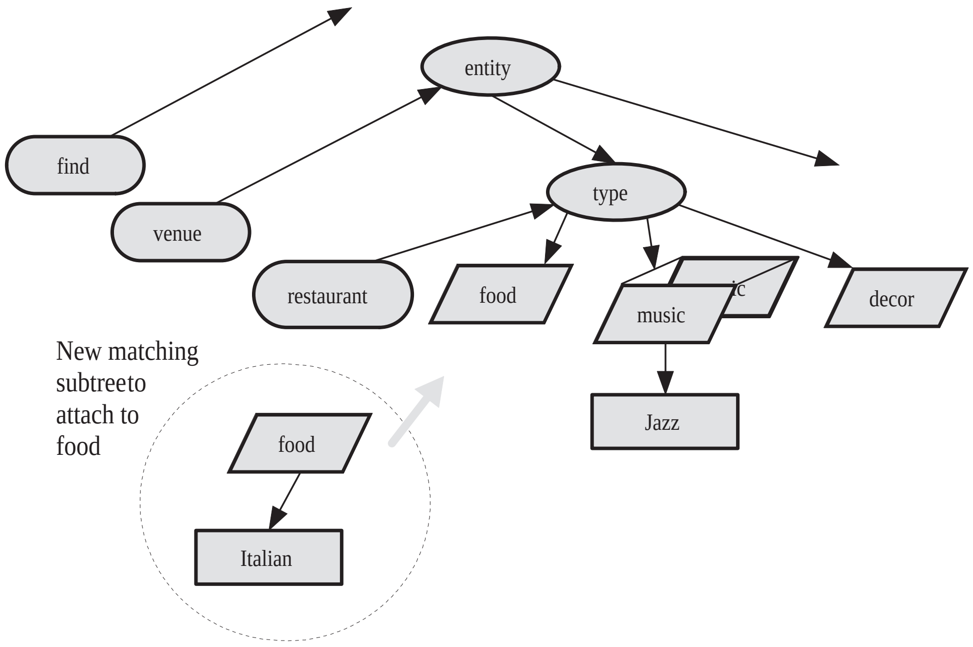


Figure 4.9: Splitting a Partition with a Shared Expansion Node

The detailed implementation of this splitting process needs to consider a number of subtleties. Firstly, in order to ensure that all partitions are unique, a rule must be applied to a node only once. This is implemented by attaching to each expanded node, a reference to the rule used to expand it. It is then simple to check whether or not a rule has been applied before to that node, and if it has, the rule cannot be applied again. Secondly, when node expansion results in multiple levels of rule application, then new subtree nodes will be created with probability less than one. In each such case, a new parallel node must be created to hold the *unused* probability mass. Each new node created in this way creates a new partition. An example of this is shown in Fig. 4.8 where the expansion of partition `P1:task` to give partition `P3:find(venue(type,name,location))` results in an intermediate partition `P2:find(entity)` being created. In the further expansion needed to accommodate the item `(music=Jazz)`, the expansion of the `type` node with probability 0.35 to `restaurant` would leave a parallel `type` node with probability 0.65 and this would form yet another partition.

Finally, as an act item is tested against successive partitions, there may be other partitions which have not yet been examined but which share the same expansion node. Each of these as yet unexamined partitions, must be cloned and the expansion node replaced by the leaf nodes of the matching subtree. For example, in Fig. 4.9, there are two partitions

```
Px: find(venue(restaurant(food,music,decor)))
Py: find(venue(restaurant(food,music(Jazz),decor)))
```

If now the item `food=Italian` is matched against Py, then a new partition

```
Pyy: find(venue(restaurant(food(Italian),music(Jazz),decor)))
```

is created. However, the expansion node `food` is shared with Px, and hence a further partition

State	Description
Init	initial state
UReq	item requested by user
UInf	item informed by user
SInf	item informed by system
SQry	item queried by system
Deny	item denied
Grnd	item grounded

Table 4.2: Node Grounding States

```
Pxx: find(venue(restaurant(food(Italian),music,decor)))
```

must also be created.

4.5 Hypotheses and the Dialog History State

The previous subsections have explained how partitions are grown as a side effect of attempting to match dialog act items. Once all input items have been processed and all possible matches made, the next step is to construct a new set of updated beliefs for the current dialog turn. As indicated by Fig. 4.1, belief update is implemented by building an explicit list of hypotheses where each hypothesis corresponds to one possible combination of p', a'_u and s'_d in the left hand side of equation 3.9. At the start of each turn, each partition p has attached to it a list of possible dialog state records s_d where each combination $\{p, s_d\}$ corresponds to a summation term in equation 3.9. At the end of the turn, the current hypotheses are updated by applying the current system dialog act and the assumed user dialog act to the current dialog history to produce a new updated dialog history.

There are three possible values for $P(s'_d|p', a'_u, s_d, a_m)$

$P(s'_d\|p', a'_u, s_d, a_m)$	Situation
≈ 1	a consistent history update is possible
≈ 0	some item in p' has been denied by the user
P_{null}	no consistent history update is possible

where $P_{\text{null}} >> 0$. The idea here is that if the user act appears to be irrelevant to this hypothesis then the hypothesis belief should not change significantly. However, explicit denial should cause belief to be reduced to close to zero. Note, however, that since a null act is always included in the list of hypothesised user acts and since a null act can never fail, recovery is always possible.

The dialog state records information about the dialog history which is relevant to the decision making process. Each terminal node X in the associated partition has an associated grounding state as shown in Table 4.2. These states are updated according to the transition diagram shown in Fig. 4.10 where the generic user and system acts are derived from the actual user and system acts as given in Table 4.3.

Figure 4.11 illustrates hypothesis updating in more detail. In this example, the system had previously output `inform(music=Jazz)` and the user's response was either `request(food)` or `negate(not.music=Jazz`.

Grnd Action	Dialog Acts
SInf(X)	inform(X)
SICon(X)	confreq(X,..)
SConf(X)	confirm(X), select(X,..), select(..,X)
SOther	all other system actions
UInf(X)	inform(X), affirm(..,X), negate(X), deny(..,X), reqalt(X)
UReq(X)	request(X), request(more.X)
UAff()	affirm()
UAff(X)	affirm(X)
UDeny()	negate(), negate(Y), deny()
UDeny(X)	deny(X)
UOther	all other dialog actions

Table 4.3: Dialog Act Grouping for Grounding Node X

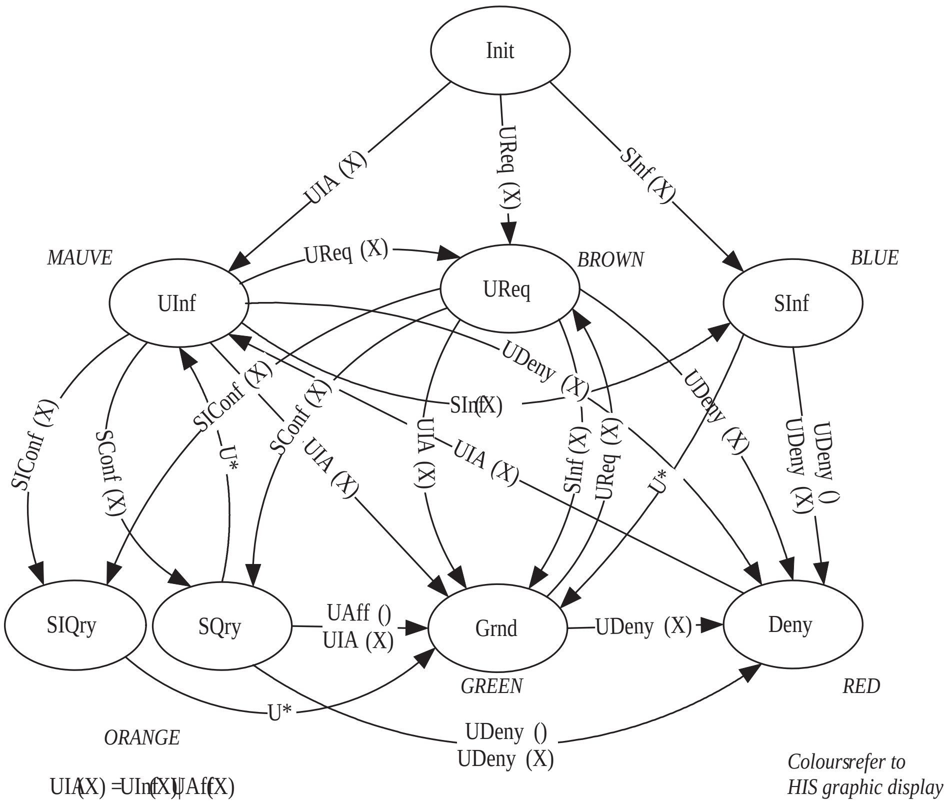


Figure 4.10: Grounding Transition Diagram

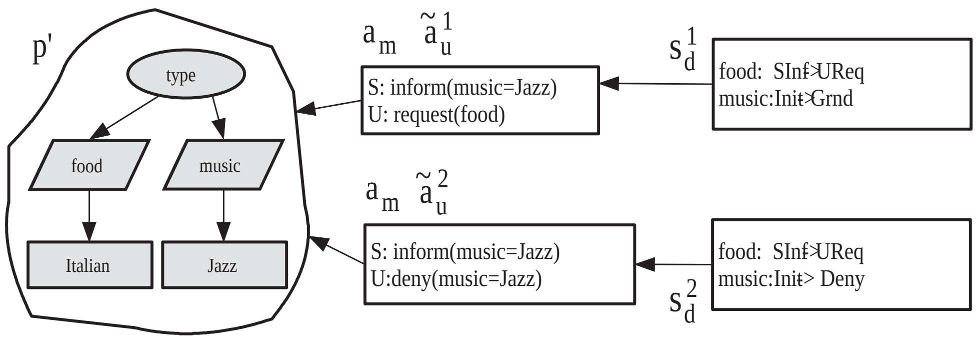


Figure 4.11: Example Hypotheses

Field	Description	Size
P(Top)	Probability of the most likely hypothesis	float
P(Nxt)	Probability of the next most likely hypthesis	float
T12Same	true if top two hyps refer to same partition	0-1
TPStatus	Status of top partition	0-5
THStatus	Status of top hypothesis	0-4
TUserAct	Type of user act in top hypothesis	0-12
LastSA	Last summary act	0-9

Table 4.4: POMDP Summary State

Previously there was a single dialog history hypothesised for the given fragment of partition p' with both nodes in the “Init” state. After completing the turn, there are two distinct dialog states corresponding to the two different intepretations of the user input.

4.6 Action Generation

Action generation follows the *Summary POMDP* scheme outlined in [12], adapted for the HIS model. Each system response a'_m is the result of a two step process:

1. the current system state is mapped into a reduce summary state, and an appropriate summary action is determined by a POMDP policy

2. the summary action is mapped into a real action by applying a number of heuristic rules.

The summary state is shown in Table 4.4 and the summary actions are shown in Table 4.5.

Greet	Greet the user
BoldRQ	Make a bold query request i.e. use `request()`
TentRQ	Make a tentative query request i.e. use `confreq()`
Confirm	Confirm an ungrounded node i.e. use `confirm()`
Offer	Offer a database entity
Inform	Give more information about current offer
Split	Use `select()` to distinguish top 2 hypotheses
FindAlt	Find an alternative database entity
QMore	Query if user wants any more information
Bye	Sign off

Table 4.5: POMDP Summary Actions

4.7 POMDP Training

Policy optimisation in the HIS model utilises a grid-based discretisation of summary belief space and on-line batch ε-greedy policy iteration. Given an existing policy π, dialogs are executed and machine actions generated according to π except that with probability ε a random action is generated. The system maintains a set of belief points $\{\hat{b_i}\}$. At each turn in training, the nearest stored belief point $\hat{b_k}$ to $\hat{b}$ is located using a distance measure. If the distance is greater than some threshold, $\hat{b}$ is added to the set of stored points and $\hat{b_k} = \hat{b}$. The sequence of points $\hat{b_k}$ traversed in each dialog is stored in a list. Associated with each $\hat{b}_i$ is a function $Q(\hat{b}_i, \hat{a}_m)$ whose value is the expected total reward obtained by choosing summary action $\hat{a}_m$ from state $\hat{b}_i$. At the end of each dialog, the total reward is calculated and added to an accumulator for each point in the list, discounted by λ at each step. On completion of a batch of dialogs, the Q values are updated according to the accumulated rewards, and the policy updated by choosing the action which maximises each Q value. The whole process is then repeated until the policy stabilises.

Since even the summary state space is very large, around 10^5 dialogs are required for policy convergence and learning using real users is not practical. Hence, a user simulation tool is used for training. This consists of two main components: a *User Goal* and a *User Agenda*. 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”, “pricerange=cheap”. The agenda is subsequently populated with corresponding dialogue acts such as “hello”, “request(restaurant)”, “inform(food=Chinese)”, etc. During the course of the dialogue, the goal ensures that the user behaves in a consistent, goal- directed manner. The agenda serves as a convenient way of capturing what information the user intends to transmit to the system, with the top act on the agenda representing the next user action a_u.

At every dialogue turn, the user agenda and goal are updated according to the incoming system action a_m using a complex set of handcrafted rules. The rules allow the user to request and provide appropriate and/or alternative pieces of information, to affirm or negate questions, to select between different options and to detect and correct misunderstandings. The agenda makes it possible to temporarily store actions when another action of higher priority needs to be issued first, hence enabling the simulator to refer to previous dialogue turns at a later point.

To introduce some degree of “life-like” randomness, the user reacts with a varying level of patience and arbitrary choices are made during the agenda update process whenever possible. As a result, thousands of

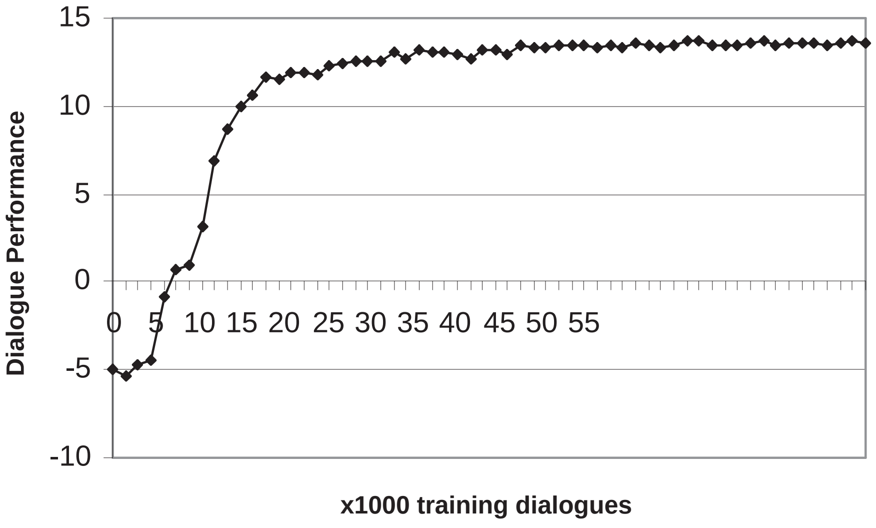


Figure 4.12: Average return during policy learning vs number of training dialogs

dialogues can be generated without seeing the same dialogue twice.

Speech understanding errors are simulated at the dialogue act level. The user action is fed through a *Scrambler* which generates an N-best list of parsed recognition hypotheses with associated confidence scores at a given error rate. The confusion matrices for dialogue act types, attributes, and values and the probability of the correct hypothesis being in the N-best list are read from file. The parameters are currently handcrafted but could easily be trained on real dialogue data at a later stage.

Each policy iteration uses a batch size of 5000 dialogs, the discount factor is 0.95 and *epsilon* is held constant at 0.1. The reward function returns -1 per system turn and $+20$ if the system recommends a venue that matches all the constraints in the user's goal. In all cases, the initial policy is random.

As a typical example of training, Fig. 4.12 shows the average return achieved by the HIS system at differing user act error rates when tested against the user simulator as a function of the number of dialogs used for training. As can be seen learning increases rapidly at first and then asymptotes. At higher error rates, learning is slower and the asymptotic return reduces.

Chapter 5

Evaluation and Conclusions

An evaluation of the prototype HIS system was conducted on the 6/7th November in Cambridge and Edinburgh. [1] The task scenarios involved finding a bar, a restaurant or a hotel in a ficticious town given a set of constraints (e.g. find a cheap, chinese restaurant in the centre of town). The tasks themselves were presented to subjects in two different ways: either as a set of images, or as a situation scenario. [2] [3] Table 5 summarises the outcome of the trial.

	Cambridge	Edinburgh	Combined
#subjects	23	17	40
#dialogues	92	68	160
#turns	676	1240	1916
#words	3294	3373	6667
%WER	21.1	37.3	29.3
#dialogues completed	88	57	145
%completion rate	95.7	83.8	90.6
Avg. turns to completion	3.8	8.1	5.6
Objective metric	91.9	75.6	85.0

Table 5.1: Results of the HIS Dialog System User Trial

There were a total of 40 subjects who performed 4 tasks each. A dialogue was judged to be completed if the system recommended a venue which matched the user's constraints. The tasks were defined so that every task had a solution. As can be seen, the completion rate was significantly higher in Cambridge than Edinburgh. This is mostly due to the higher proportion of non-natives in the Edinburgh trial and the consequent degradation in Word Error Rate (WER) of the recogniser (see 3rd row of the table).

[1]The assistance of the UEDIN team in helping design the tasks, recruit subjects and run the trial is gratefully acknowledged.

[2]The intention was to see how much the presentation of the task influences the choices of words and syntactic forms used by subjects.

[3]Due to insufficient development time, the system tested had a number of technical limitations. In particular, only the 1-best output from the recogniser was available and there were known to be bugs in the dialogue grounding model.

The average number of turns to completion was computed as follows. If the correct venue was offered, the number of turns was taken as the total up to the point of offer. If no correct venue was offered, then the total turns in the dialogue was used. As can be seen, the higher overall WER and the lower completion rates at Edinburgh lead to significantly extended dialogues.

The objective metric $= R - N$ where $R = 100$ if the dialogue was completed successfully and 0 otherwise. N is the number of turns. This form of metric is commonly used in system evaluation and it is similar to the reward function used to train the POMDP.

In summary, the system as tested was very preliminary and there is a great deal of refinement still to do. Nevertheless, the results of this trial suggest that the POMDP-based HIS system is viable for building robust real-world dialogue systems. Across the range of word error rates encountered, an average completion rate above 90% is considered very respectable. Once full N-best handling is incorporated it is expected that the robustness to errors will improve significantly.

5.1 Conclusions

This report has outlined a new framework called the Hidden Information State (HIS) model for designing and implementing spoken dialog systems. The model is based on the SDS-POMDP but it avoids the usual computational issues associated with POMDPs by partitioning the space of user goals into a small number of equivalence classes. Probabilistic context-free ontology rules are used to describe the iterative splitting of partitions to eventually form unique goal states. By computing beliefs on partitions rather than the underlying states, belief monitoring remains tractable even for complex real-world systems. POMDP policy optimisation and response generation uses the Summary POMDP model developed earlier for simple slot-based systems. The system has been evaluated in a preliminary trial with encouraging results. A number of problems which arose during the trial have now been fixed, and the system performance is further improved.

Overall the prototype HIS system implemented in the TALK Project is believed to be the first-ever full-scale POMDP-based dialogue system. We believe that it represents a major step forwards in designing dialogue systems which are inherently robust, which can naturally handle uncertainty in recognition output and which can be trained and adapted automatically on real dialog data.

Bibliography

[1] Steve Young, Jason Williams, Jost Schatzmann, Matt Stuttle, and Karl Weilhammer. D4.3: Bayes Net Prototype - the Hidden Information State Dialogue Manager. Technical report, TALK Project, 2006.

[2] S Larsson and D Traum. Information State and Dialogue Management in the TRINDI Dialogue Move Engine Toolkit. *Natural Language Engineering*, pages 323–340, 2000.

[3] SJ Young. Talking to Machines (Statistically Speaking). In *Int Conf Spoken Language Processing*, Denver, Colorado, 2002.

[4] Oliver Lemon, Kallirroi Georgila, James Henderson, Malte Gabsdil, Ivan Meza-Ruiz, and Steve Young. D4.1: Integration of Learning and Adaptivity with the ISU approach. Technical report, TALK Project, 2005.

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

[6] Oliver Lemon, Kallirroi Georgila, and Matthew Stuttle. D4.2: Showcase exhibiting Reinforcement Learning for dialogue strategies in the in-car domain. Technical report, TALK Project, 2005.

[7] Oliver Lemon, Kallirroi Georgila, James Henderson, and Matthew Stuttle. An ISU dialogue system exhibiting reinforcement learning of dialogue policies: generic slot-filling in the TALK in-car system. In *Proceedings of EACL*, 2006.

[8] LP Kaelbling, ML Littman, and AR Cassandra. Planning and Acting in Partially Observable Stochastic Domains. *Artificial Intelligence*, 101:99–134, 1998.

[9] JD Williams, P Poupart, and SJ Young. Factored Partially Observable Markov Decision Processes for Dialogue Management. In *4th Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, Edinburgh, 2005.

[10] JD Williams, P Poupart, and SJ Young. Partially Observable Markov Decision Processes with Continuous Observations for Dialogue Management. In *6th SIGdial Workshop on DISCOURSE and DIALOGUE*, Lisbon, 2005.

[11] ML Littman. The Witness Algorithm: solving partially observable Markov decision processes. Technical report, Brown University, 1994.

[12] JD Williams and SJ Young. Scaling up POMDPs for Dialogue Management: the Summary POMDP Method. In *IEEE workshop on Automatic Speech Recognition and Understanding (ASRU2005)*, Puerto Rico, 2005.

[13] B Zhang, Q Cai, J Mao, and B Guo. Planning and Acting under Uncertainty: A New Model for Spoken Dialogue System. In *Proc 17th Conf on Uncertainty in AI*, Seattle, 2001.

[14] MTJ Spaan and N Vlassis. Perseus: randomized point-based value iteration for POMDPs. Technical report, Universiteit van Amsterdam, 2004.

[15] S Singh, DJ Litman, M Kearns, and M Walker. Optimizing Dialogue Management with Reinforcement Learning: Experiments with the NJFun System. *J Artificial Intelligence Research*, 16:105–133, 2002.

[16] K Scheffler and SJ Young. Automatic Learning of Dialogue Strategy using Dialogue Simulation and Reinforcement Learning. In *HLT 2002*, San Diego, USA, 2002.

[17] AW Moore and CG Atkeson. The Parti-game Algorithm for Variable Resolution Reinforcement Learning in Multidimensional State-spaces. In SJ Hanson, JD Cowan, and CL Gi, editors, *Advances in Neural Information Processing Systems*. Morgan Kaufmann, 1994.

[18] AK McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, University of Rochester, 1995.

[19] WTB Uther and MM Veluso. Tree Based Discretization for Continuous State Space Reinforcement Learning. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 769–775, 1998.

[20] MJ Kochenderfer and G Hayes. Adaptive Partitioning of State Spaces using Decision Graphs for Real-Time Modeling and Planning. In *Workshop on Planning and Learning in A Priori Unknown or Dynamic Domains, IJCAI-05*, Edinburgh, 2005.

[21] R Jaulmes, J Pineau, and D Precup. Active Learning in Partially Observable Markov Decision Processes. In *European Conference on Machine Learning (ECML)*, Porto, Portugal, 2005.

[22] RS Sutton and AG Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, Mass, 1998.

[23] JD Williams. *Partially Observable Markov Decision Processes for Spoken Dialogue Management*. PhD thesis, Cambridge University, 2006.