Open access • Journal Article • DOI:10.1145/1966407.1966409

Effective handling of dialogue state in the hidden information state POMDP-based dialogue manager — Source link

Milica Gasic, Steve Young

Institutions: University of Cambridge

Published on: 06 Jun 2011 - ACM Transactions on Speech and Language Processing (ACM)

Topics: Partially observable Markov decision process, Markov decision process and Reinforcement learning

Effective Handling of Dialogue State in the Hidden Information State POMDP-based Dialogue Manager

MILICA GAŠIĆ and STEVE YOUNG
Cambridge University Engineering Department

Effective dialogue management is critically dependent on the information that is encoded in the dialogue state. In order to deploy reinforcement learning for policy optimisation, dialogue must be modelled as a Markov Decision Process. This requires that the dialogue state must encode all relevent information obtained during the dialogue prior to that state. This can be achieved by combining the user goal, the dialogue history and the last user action to form the dialogue state. In addition, to gain robustness to input errors, dialogue must be modelled as a Partially Observable Markov Decision Process (POMDP) and hence, a distribution over all possible states must be maintained at every dialogue turn. This poses a potential computational limitation since there can be a very large number of dialogue states. The Hidden Information State model provides a principled way of ensuring tractability in a POMDP-based dialogue model. The key feature of this model is the grouping of user goals into partitions that are dynamically built during the dialogue. In this paper, we extend this model further to incorporate the notion of complements. This allows for a more complex user goal to be represented and it enables an effective pruning technique to be implemented which preserves the overall system performance within a limited computational resource more effectively than existing approaches.

Categories and Subject Descriptors: H.1.2 [**Information Systems**]: Models and Principles—*User/Machine Systems*; I.2.6 [**Computing Methodologies**]: Artificial Intelligence—*Learning*

General Terms: Spoken Dialogue Systems, Dialogue modelling, POMDP, Reinforcement learning

Additional Key Words and Phrases: dialogue belief monitoring, dialogue state representation

1. INTRODUCTION

Statistical approaches to dialogue management enable extensible dialogue managers to be built based on data rather than hand-coded rules. In particular, the reinforcement learning approach enables a dialogue policy to be learnt in such a way as to optimise overall dialogue success. In order to deploy reinforcement learning for policy optimisation, dialogue is modelled as a Markov Decision Process (MDP) [Levin et al. 1998]. This requires the dialogue state to be Markovian, *i.e.,* the state has to encode everything that happened previously in the dialogue which might be potentially useful for selecting the next action. In addition, modelling dialogue as a Partially Observable Markov Decision Process (POMDP) allows the dialogue manager to be robust to speech recognition errors [Young et al. 2010; Thomson and Young 2010]. This approach requires that a distribution over all dialogue states, *the belief state*, is maintained at each dialogue turn.

According to [Williams et al. 2005], the combination of the user goal, the dialogue history and the last user action provides sufficient information to form a Markovian dialogue state. However, the set of all such dialogue states can be very large. This is particularly limiting in the case of the POMDP, where a distribution over all states has to be maintained at every dialogue turn. In order to deal with this, there are two main approaches which enable a tractable POMDP dia-

Authors' address: M. Gašić, S. Young, {mg436|sjy}@eng.cam.ac.uk, Trumpington Street, Cambridge, CB2 1PZ, UK
Submitted to ACM Transactions on Speech and Language Processing Special Issue on Machine Learning for Robust and Adaptive Spoken Dialogue Systems.

logue manager to be built for a real-world task. The Bayesian Update of Dialogue State (BUDS) model [Thomson and Young 2010], assumes that the state can be factored into independent elements and then the probability distribution can be independently updated for each element. The Hidden Information State (HIS) model, advocated in [Young et al. 2010], does not assume any independence relationship between different elements of the state. Instead, it maintains the probability distribution of only the most likely dialogue states. In order to do this efficiently, user goals are grouped together into *partitions*, on the assumption that every goal in the same partition is equally likely. The partitions are built using the N-best user dialogue acts from the speech understanding component, the system's output dialogue act and the domain ontology. In this way, the partitions are constrained to represent only the plausible goals from the domain. A similar approach is adopted in [Williams 2010] with the difference that an ontology is not used to determine possible goals. Instead, the partitions represent every possible combination of attribute-value pair and their complements.

In order to further exploit modelling of the input dependencies that the HIS model facilitates, we extend this framework to enable richer expressiveness via the notion of complements. Similar to [Kim et al. 2008; Williams 2010], the partitions are formed using not only the user input, but also the complements of the user input. In this way, coverage of potential user goals is improved. This is particularly useful for more complex dialogue structures, where the user goal evolves during the dialogue, influenced by the system's responses. Moreover, the notion of complements allows a variant of first order logic to be incorporated, *i.e,* the user can use negations, conjunctions and disjunctions to communicate with the system and the system can use quantifiers to express the result of the user query.

Since there is uncertainty in the user input at every turn, the number of possible partitions grows exponentially as the dialogue progresses. This poses computational issues, especially in domains where relatively long dialogues are expected. It also limits the length of the N-best list of hypothesised user dialogue acts input to the dialogue manager, which is crucial for robust belief monitoring in noisy conditions [Thomson et al. 2008]. In this paper, we show that the explicit notion of complements allows an efficient pruning technique to be implemented which enables arbitrarily long N-best lists of input acts and arbitrarily long dialogues to be supported, whilst preserving the most probable user goals.

The next section gives a brief overview of the Hidden Information State model with a focus on the structure of the ontology, the state representation and the belief update. Then, Section 3 provides a description of how the notion of complements is used in partitioning to support a more complex dialogue structure. A technique for reducing the number of partitions is presented in Section 4 and evaluation results are given in Section 5. Finally, conclusions are given in Section 6.

2. HIDDEN INFORMATION STATE MODEL

A HIS-based spoken dialogue system consists of three major components: speech understanding, speech generation and dialogue management - see Fig. 1. The speech understanding component consists of a speech recogniser and a semantic decoder. Its function is to map user utterances into an abstract representation of the user intention – the user action. Since this input might be corrupted with noise, an N-best list of possible user actions along with a confidence score for each is passed to the dialogue management component in each turn. Using this, the dialogue manager updates the estimate of the belief state. Then, based on the updated belief estimate and the dialogue policy, the dialogue manager chooses the system's action. The speech generation component normally consists of a natural language generator and a speech synthesiser. It maps the system's response first into text and then into speech. The overall structure of a statistical spoken dialogue system is shown in Fig. 1.

The core of the system is the HIS dialogue manager. In the remainder of this section we explain the main features of the HIS model based on [Young et al. 2010]. Firstly, we explain the structure of the ontology that is used for building the dialogue state. We then explain how the dialogue state is formed and, following that, we give a brief description of the belief update procedure. Finally, we conclude the section with a description of the action-selection process.

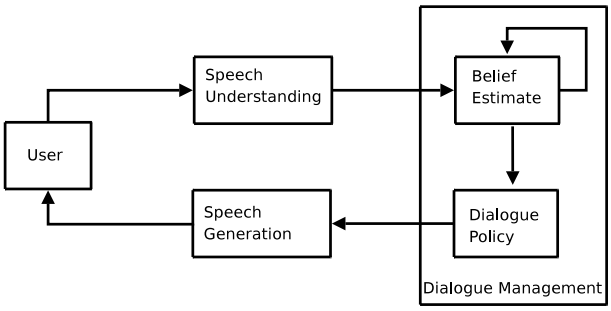


Fig. 1. Statistical Spoken Dialogue System Structure

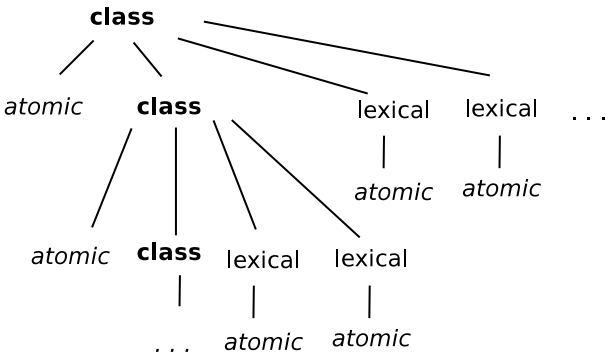


Fig. 2. Generic Ontology Structure

2.1 Domain Ontology

The HIS model is targeted at limited domain query dialogues. As far as dialogue management is concerned, the dialogue between the system and the user takes place at the level of dialogue acts where each dialogue act comprises a type and a list of attribute-value pairs. For example, `inform(type=restaurant, food=Chinese)` would be the representation at the dialogue act level corresponding to the user saying *"I'd like a Chinese restaurant"*. A domain ontology then defines all of the attributes and their possible values, as well as the structural relationship between different attributes.

The ontology has a tree structure. The tree nodes are divided in three groups: class nodes, lexical nodes and atomic nodes (see Fig 2). Class nodes can have many child nodes, the first is always atomic and defines a specific instance of the class, the remainder consist of an optional class node and one or more lexical nodes. Lexical nodes can have only a single atomic child node.[1]

As an example, a simple tourist information ontology is given in Table I, where examples of class nodes are `entity` and `type` (bold font), lexical nodes are `pricerange` and `food` (regular font) and atomic nodes are `restaurant` and `Chinese` (italic font).

The attributes listed in each dialogue act correspond to either class or lexical nodes in the ontology; and the values that they take are represented by atomic nodes. The tree root is a class node and it defines the user goal in the most general way. Other class nodes define the user query more specifically. In detail, each class node and its atomic child node define an additional set of attributes that are represented by lexical nodes and optionally a class node. For example, in the ontology from Table I, atomic node `restaurant` for class node `type` defines an additional set of lexical nodes: `food`, `pricerange`, `music`, `drinks`, and `stars`.

The Hidden Information State model makes use of the hierarchical relationship between the

[1]Note that attributes corresponding to lexical nodes are often referred to as *slots* in the dialogue systems literature since in simple systems, dialogues are designed with the aim of filling in a fixed set of slots with values from the user.

entity	←	*venue*(**type**, area, name, addr, near, phone, comment)
type	←	*hotel*(pricerange, stars, price, drinks)
type	←	*restaurant* (food, pricerange, price, music, drinks, stars)
type	←	*bar*(drinks, music, pricerange)
type	←	*amenity*
area	=	{ *central*, *east*, *west*, ... }
food	=	{ *Italian*, *Chinese*, *Indian*, ... }
...		

Table I. Ontology Rules

attributes to model the dependencies in each user input. For example, in the tourist information domain if the user specified `food=Italian` that implies that the user wants `type=restaurant` and `entity=venue`. Attribute value pair `pricerange=cheap` is associated with `type=restaurant`, `type=bar` and `type=hotel`, but not `type=amenity`. Therefore if no `type` was specified, applying `pricerange=cheap` would result in creating these all these three partitions. A full description of a real-world tourist information ontology is given in Appendix 7.2.

2.2 State Representation

In the Hidden Information State model, the dialogue state is represented as a combination of the user goal, the last user act and the dialogue history. Since the user goal and the true user act cannot be directly observed they are both part of the hidden state. Although the system actions are fully observable, the user actions are not and therefore the dialogue history is also part of the hidden state.

This combination can result in a vast number of dialogue states and it would not be computationally tractable to maintain a probability distribution over such a large state space. Therefore, user goals are grouped together into *partitions* on the assumption that all goals from the same partition are equally probable. Partitions are built using the attribute-value pairs from the N-best list of the user input and the previous system output. They are combined together using the dependencies defined by the domain ontology. In detail, each partition represents a realisation of a tree from the ontology. The dialogue history is represented in the form of a finite state machine that keeps track of the dialogue progress. The combination of a partition, a user act from the N-best list and the associated dialogue history if the user took that action for that representation of the user goal (the partition) forms a *hypothesis*, *i.e.,* a single member of the partitioned state space. Note that the dialogue act need not be consistent with the partition. This represents the case that the user apparently says one thing while wanting something else; and this can frequently happen when there are speech recognition errors. In this case, the user model assigns a low probability to the partition as explained further in Section 2.3. A probability distribution over the most likely hypotheses is maintained during the dialogue and this distribution constitutes the POMDP's belief state.

The main requirements for representing partitions are that each partition is unique and that they are represented in a way that allows a large number of partitions to be maintained efficiently and compactly. Since each partition is the realisation of a tree from the ontology, many partitions have common subtrees and hence partitions should be able to share nodes and achieve a compact representation [Young et al. 2010]. A problem arises, however, in negotiation-type dialogues when users change their goal and ask for something else. The implicit rejection of the current most probable goal means that the probability of all hypotheses consistent with this goal should go down and the probability of all other hypotheses should increase. However, without an explicit representation of complements, it is difficult to determine which set of hypotheses is which. For example, consider attribute value pairs `name=Char Sue`, `type=restaurant` and `pricerange=cheap`. In the representation without complements this would result in four partitions: one that contains `name=Char Sue`, `type=restaurant` and `pricerange=cheap` (1), another that contains `type=restaurant` and `pricerange=cheap` (2), one with just `type=restaurant` (3) and one with just `pricerange=cheap` (4). Thus, if the system makes an offer inform(name=Char Sue,type=restaurant,pricerange=cheap), this matches partition (1) most closely. However, if the

user asks for an alternative, the user goal could be represented by any of the partitions (2), (3), and (4) despite the fact that these could all yield the same entity Char Sue from the data-base. Preventing this to ensure that a genuine alternative is offered to the user is difficult and requires ad hoc hand-crafting. Hence, although sharing nodes among partitions allows for a vast number of partitions to be efficiently represented, it is difficult to identify and remove partitions that are represented in such a way. It has been shown in [Williams 2010] that partition recombination can overcome this problem with the notion of complements, which we further explore in this paper.

2.3 Belief Update

The Hidden Information State model maintains a probability distribution over hypotheses – *the belief state*. The probability of each hypothesis in the belief state is updated every turn using four components: the observation model, the user action model, the user goal model and the dialogue history model. It is updated according to:

$$\underbrace{b'(p', a'_u, s'_d)}_{\substack{h' \\ \text{new} \\ \text{hypothesis}}} = k \cdot \underbrace{P(o'|a'_u)}_{\substack{\text{observation} \\ \text{model}}} \underbrace{P(a'_u|p', a_m)}_{\substack{\text{user} \\ \text{action} \\ \text{model}}} \underbrace{P(p'|p)}_{\substack{\text{user goal} \\ \text{model}}} \sum_{s_d} \underbrace{P(s'_d|p', a'_u, s_d, a_m)}_{\substack{\text{dialogue history} \\ \text{model}}} b \underbrace{(p, a_u, s_d)}_{\substack{h \\ \text{old} \\ \text{hypothesis}}}, \quad (1)$$

where b is the current belief state, b' is the updated belief state, p is a partition of user goals, a_u is the user action, a_m is the system action and s_d is the dialogue history (primed are the elements of the next turn). The observation model is derived from a scored N-best list of the user input dialogue acts generated by the speech understanding component. The user action model consists of a dialogue act type bigram model and an item matching model. The act type bigram model determines how probable the user dialogue act type is given its preceding dialogue act type in the dialogue. The item matching model is a filter that gives a low probability to the dialogue act items inconsistent with the given partition and a high probability otherwise. The user goal model is derived from the domain ontology. The dialogue history, that is assigned to each hypothesis, represents a set of states for every node that belongs to the respective partition of that hypothesis. These states keep track of what has happened previously in the dialogue. The dialogue history model itself is deterministic and is implemented as a finite state machine. For example, if the partition contains elements *cheap*, *hotel*, *area* and the user says `inform(type=hotel, pricerange=cheap)` then *cheap* and *hotel* transit to the state of $UserInform$ while *area* remains in the $Initial$ state. Then, if the system says `request(area)` the node *area* transits to $SysRequest$ state. These are important for the action selection process which is described in the next section.

2.4 Policy Representation and Action Selection

The number of hypothesis for any real-world problem can be very large and applying POMDP learning algorithms directly to the full dialogue state to find an optimal policy would be computationally intractable. To overcome this problem, the belief state space (*master space*) is mapped into a smaller-scale *summary space*. The features of the summary space are: top hypothesis probability, next hypothesis probability, the last user act type from top hypothesis and an indicator of how many database entries match the top partition. This continuous summary space is discretised into a grid, so that reinforcement learning MDP learning algorithms can be tractably performed. The policy is a mapping from summary space grid points to summary actions. The policy optimisation is performed in interaction with a simulated user which gives a reward to the system at the end of every dialogue. In that way, every dialogue is a learning episode, which allows the Monte Carlo Control algorithm [Sutton and Barto 1998; Young et al. 2010] to be used to find the optimal policy. The optimal policy maps each summary grid point into a summary action selected to yield the highest expected reward. The summary action is then mapped back into a master action by adding additional information from the corresponding master belief state to give the required system dialogue act. For this purpose, the dialogue history information is used. For example, if the system makes an offer it mentions all the attribute-value pairs that the user requested so that the user can be assured that the offered entity has certain properties. Although

Representation	Atomic node for Lexical node *food*						
Set	*Chinese*	*English*	*Indian*	*Italian*	*Japanese*	*French*	*Thai*
	T	F	T	F	F	F	F
Disjunctions	*Chinese* $\vee$ *Indian*						
Conjunctions	$\neg$*English* $\wedge$ $\neg$*Italian* $\wedge$ $\neg$*Japanese* $\wedge$ $\neg$*French* $\wedge$ $\neg$*Thai*						

Table II. Different representations of the same atomic node in a partition

based on heuristics, the summary to master space mapping is relatively easy to implement and rarely leads to poor dialogue management behavior in the same way as the inappropreate choice of summary actions.

3. EXTENDED STATE REPRESENTATION

In order to exploit the HIS system's capability to model dependencies between different attribute-value pairs, we extend the standard model to include the explicit representation of complements.

3.1 Explicit Representation of Complements

A partition is a realisation of the ontology tree with an extended representation of values in the atomic nodes. In detail, class and lexical nodes take unique values, for example *type*, *area* or *food*. Logically, atomic nodes should be represented as a set of Boolean indicators for each possible value from the ontology. However, a potential drawback of this set representation is the need to enumerate every value that an attribute can take which can be a problem for classes with high cardinality. An alternative and more compact approach is to represent atomic nodes as a disjunction of the values which are true or a conjunction of the negation of the values which are false. An example is given in Table II.

3.2 Partitioning Process

Partitioning is the process of applying an attribute-value pair $s = v$ to a partition p that contains node s and creating its child partition c. In the ontology, s is either a class or a lexical node and v is an atomic node. In the partition p, node s has a child atomic node that has all possible values that attribute s can take. During the partitioning process, the value v in the atomic node of the partition p is set to false and the partition c is a copy of the partition p in which v of the corresponding node is set to true.

In order to apply attribute-value pair $s = v$ to an existing set of partitions, it must first be ensured that there is a partition that contains node s. For attribute s, the list of *superiors* is defined as all attribute-value pairs $s_i = v_i$ where s_i are class nodes on the path from the node s to the root of the ontology tree, and v_i are the values of their child atomic nodes that enable the attribute expansion leading to the occurrence of s in the tree. For example, for attribute-value pair `food=Italian` the list of superiors is `type=restaurant, entity=venue`, see Table I. The ontology automatically generates this list for each attribute s, so that they can be applied prior to applying $s = v$. In that way, it is ensured that there exists a partition with node s before $s = v$ is applied.[2]

The partitioning process starts by applying the list of attribute-value pairs from the N-best user input to the initial partition, which is just the root of the ontology tree. The process is then recursively repeated. In such a way, an ordered tree of partitions is created, where the order indicates when each partition was created. Attribute-value pairs from the system act are also applied during this partitioning process to ensure that the belief state includes attributes which have been introduced by the system as well as by the user.

It is important to note that this process guarantees that each partition that is created is unique. This is achieved by checking whether a partition contains v set to false before $s = v$ is applied to that partition. If it does contain v set to false, then $s = v$ must have already been used and

[2]This mechanism is essential for dialogues where the user takes the initiative. For example: Sys:"How may I help you", User:"I want some Italian food".

System	hello()
User	inform(=hospital)
System	inform(name="Addenbrooke's Hospital",type=amenity,amtype=hospital)
User	request(area)
System	inform(name="Addenbrooke's Hospital",area=addenbrookes)
User	inform(area!=addenbrookes)
System	inform(name=none,type=amenity,amtype=hospital,name!="Addenbrooke's Hospital")
User	request(addr)
System	inform(name="Addenbrooke's Hospital", addr="Cambridge University Hospitals NHS Foundation Trust, Hills Road")
User	bye()

Table III. Dialogue with Negations in System's Response

should not be applied again to that partition.
A step-by-step example of the partitioning process is given in Fig 4. The final tree of partitions represents the partitions that are created from the following attribute-value pairs: `entity=venue`, `type=restaurant`, `area=central`, `food=Italian` and `pricerange=cheap`. The ontology from Table I is used to determine the valid combinations. Therefore, there is no combination that involves `type!=restaurant` and `food=Italian`, since the lexical node `food` is specific to class node `type` in which the atomic child value `restaurant` is set to true.

3.3 Logical Expressions for Negotiation-type Dialogues

The explicit representation of complements in partitions, improves the model in a number of ways.

Firstly, it enables a simpler error recovery. For example, if attribute-value pair $s = v$ occurred in the N-best user input due to a recognition error, and it turns out later in the dialogue that the user does not want v, then the user modelling component will automatically increase the probability of the partition that contains $\neg v$. In that way, even if the system does not know exactly what the user wants for attribute s, the knowledge that the user does not want v is explicitly represented and the true user goal will be in the partition that has $\neg v$.

Secondly, this representation is particularly useful when the user goal evolves during the dialogue. For example, if the user wants a Chinese restaurant in the centre, the system may offer *"Charlie Chan is a Chinese restaurant in the centre"*, which is represented at the dialogue act level by `inform(name=Charlie_Chan, type=restaurant, food=Chinese, area=central)`. When the system makes such an offer, the partitioning results in some partitions containing `name=Charlie_Chan` and others `name!=Charlie_Chan`. In a real dialogue, a user might want to have more options and may ask *"Do you have anything else?'*, corresponding to the dialogue act `reqalts()`. Based on this, the user action model will increase the probability of partitions which have `name!=Charlie_Chan` and decrease the probability of partitions that have `name=Charlie_Chan`.

3.3.1 *Quantifiers in the System's Response.* By utilising the notion of complements, the system can provide more accurate responses to the user. Referring back to the example from the previous section, if the user wanted something other than Charlie Chan's, it may turn out that the partition with `name!=Charlie_Chan`, `type=restaurant`, `food=Chinese` and `area=central` does not have any matching entries in the database. In that case, the system may respond with `inform(name=none, type=restaurant, food=Chinese, area=central, name!=Charlie_Chan)`, meaning *"There is no restaurant that serves Chinese food and is in the centre and isn't Charlie Chan"*, or in a more natural form *"Charlie Chan is the **only** Chinese restaurant in the centre"*. An example of a Cambridge tourist information dialogue (Appendix 7.2) that utilises such expressions is given in Table III.

During the course of a dialogue with a high rate of speech recognition errors, the system might not know what the user wants, but it might be confident about what the user does not want. Coming back to the previous example, the speech recogniser might output *"No, I don't want the central area, I want ..."* corresponding to `deny(area=central)` or `inform(area!=central)`, but

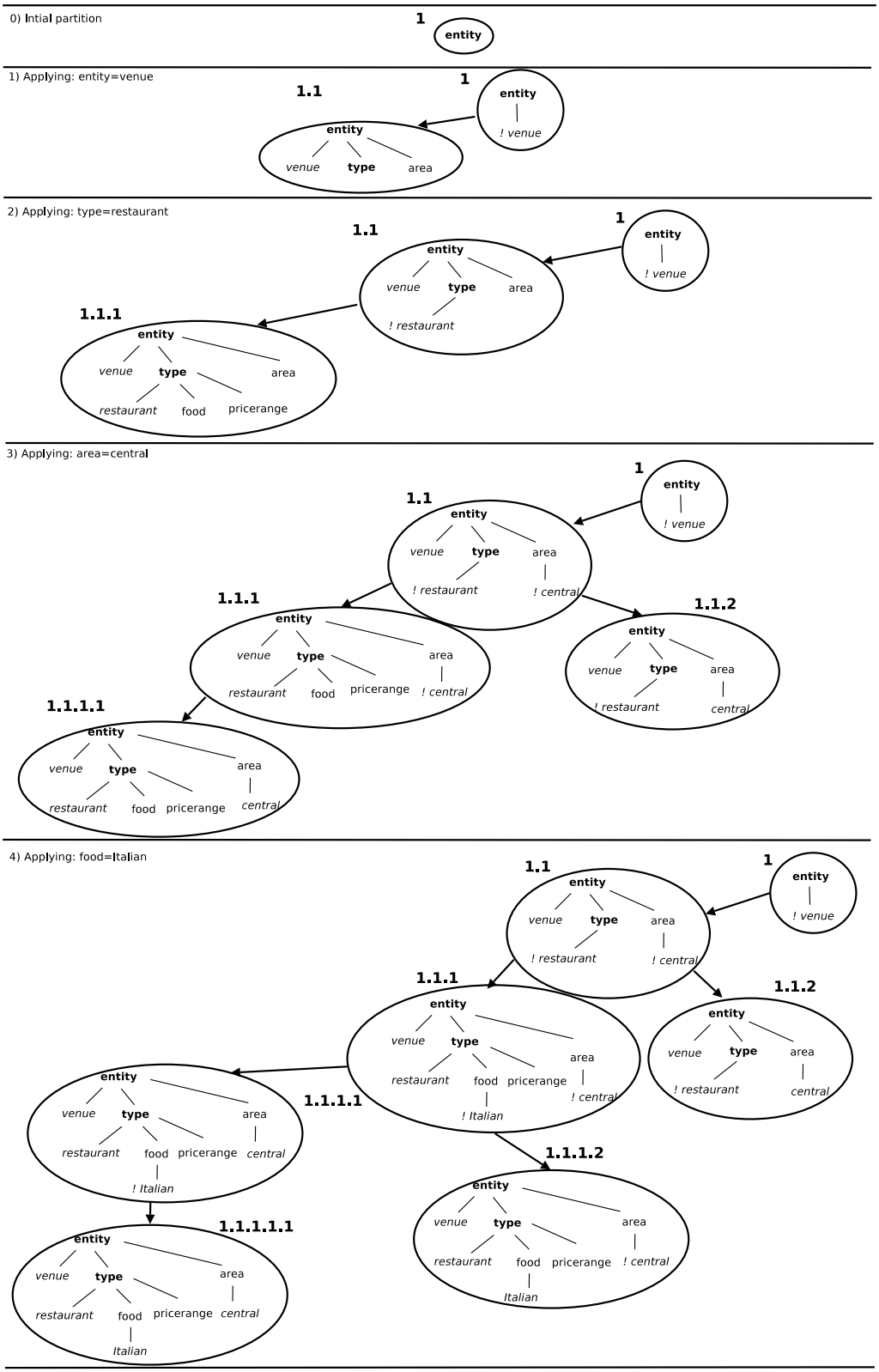


Fig. 3. Step-by-step Partitioning Process

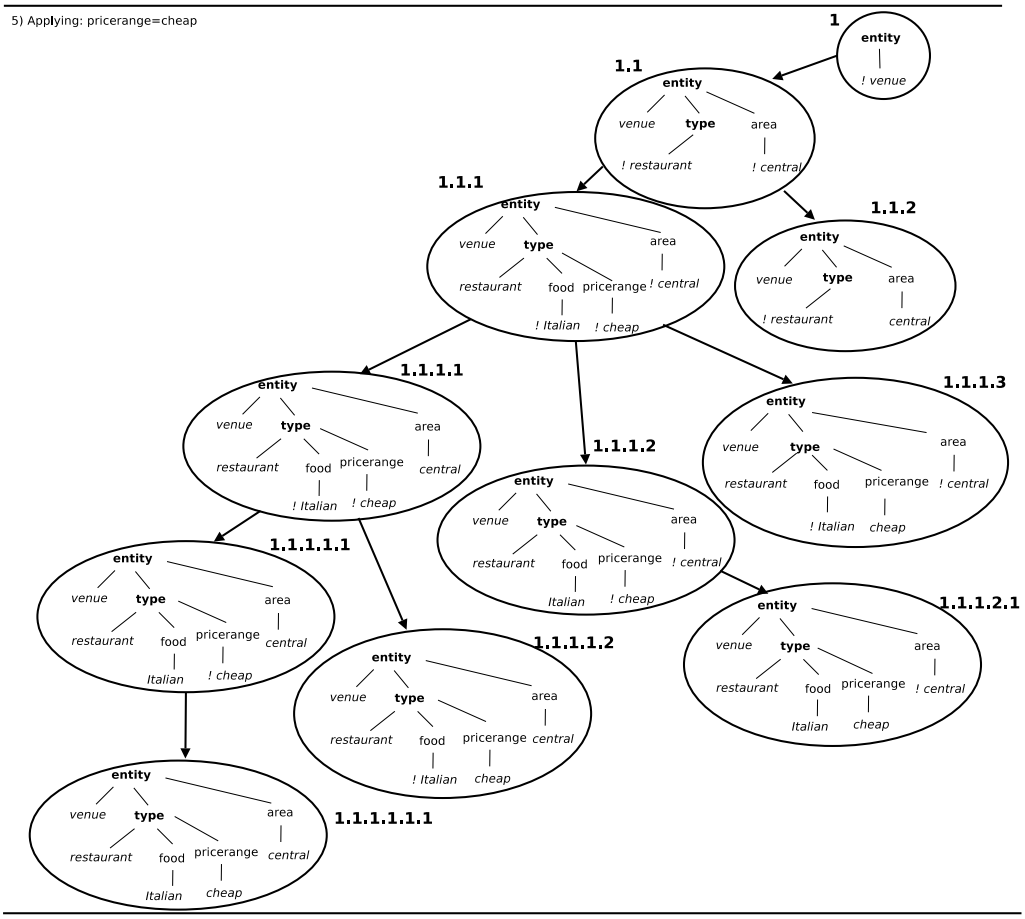


Fig. 4. Step-by-step Partitioning Process (cont.)

the actual value for the attribute `area` might not be recognised. Then, after applying the user action model, the partition containing `name!=Charlie_Chan`, `type=restaurant`, `food=Chinese` and `area!=central` might end up having the highest probability. If such a partition does not have any matching entries in the database, the system may inform the user about it rather then trying to find out the exact value of attribute `area` that the user wants. The resulting system dialogue act would then be `inform(name=none, type=restaurant, food=Chinese, area!=central)`, meaning *"There is no restaurant that serves Chinese food and is not in the central area"*, or alternatively *"**All** Chinese restaurants are in the centre"*.

These two examples demonstrate the capability the explicit representation of complements provides for expressing existential and universal quantifiers in the system's response. This can make the dialogue more efficient as there is no need for the system to try to resolve every single constraint that user might have, since knowing just what the user does not want might be sufficient to provide an acceptable response.

3.3.2 *Disjunctions and Conjunctions in the User's Request.* It was mentioned previously that atomic nodes in partitions can be represented in terms of conjunctions or disjunctions of values in partitions. So far we have only considered the case when the user has in mind one specific value for each attribute, and any cases of multiple values in partitions come from confusions caused by recognition errors. However, this representation also allows a richer expression of the user's constraints. For example, if the user wants a restaurant that serves Thai or Indian food `inform(type=restaurant, food=Thai∨Indian)`, partitions can directly represent this. Therefore, the system can handle such requests. An example of a Cambridge tourist information dialogue

System	hello()
User	inform(=restaurant, **food=Thai** | **Indian**, pricerange=cheap))
System	inform(name="The Rice Boat",type=placetoeat,eattype=restaurant,pricerange=cheap,food=Indian)
User	reqalts(food=Thai)
System	inform(name=none,type=placetoeat,pricerange=cheap,food=Thai)
User	request(name="The Rice Boat", addr)
System	inform(name="The Rice Boat",addr="37 Newnham Road")
User	bye()

Table IV. Dialogue with Disjunctions in User's Request

System	hello()
User	inform(=hotel, **stars!="4", stars!="5"**)
System	inform(name="Express by Holiday Inn Cambridge",type=placetostay,staytype=hotel,stars="2")
User	reqalts()
System	inform(name="Royal Cambridge Hotel",type=placetostay,staytype=hotel,stars="3")
User	request(area)
System	inform(name="Royal Cambridge Hotel",area=riverside)
User	bye()

Table V. Dialogue with Conjunctions in User's Request

where the user makes use of disjunctions is given in Table IV.

Since we assume that every entity has only one value for each attribute, for example a hotel can only have a certain number of stars, then conjunctions such as `stars=4∧5` would not be applicable. However, conjunctions in terms of negations are possible and effectively represent disjunctions of negated values. For example, the user constraint *"I want a hotel, but not with four or five stars"* corresponds to the dialogue act `inform(type=hotel,stars!=4∨5)`, or, alternatively, `inform(type=hotel,stars!=4, stars!=5)`. This can be directly represented in partitions and thus handled by the system, see the Cambridge tourist information dialogue in Table V.

4. PRUNING

Due to the nature of the partitioning process, the number of partitions grows exponentially as the dialogue progresses, which poses a potential computational limitation. A simple experiment on 3000 dialogues in interaction with a simulated user on 40% semantic error rate using a 10-best list shows the exponential nature of the growth of the number of partitions, see Fig. 5. The average time in seconds that is taken for the belief update in each turn is given at each point in the graph[3]. In order for the system to run in real time the total response time (including decoding and generation) should not be larger then 1 second, which means that the belief update time should be kept well below that threshold. Therefore, a pruning technique is needed to ensure this.

4.1 Partition Recombination

The number of partitions can be reduced simply by removing the low probability partitions. As noted earlier in Section 2.3, belief state hypotheses are formed from the combination of a partition, the last user action and the respective dialogue history, and the probability of each hypothesis is maintained through-out the dialogue. The probability of a specific partition can therefore be easily computed by marginalising out the user action and dialogue history simply by summing all hypotheses containing that partition *i.e.,* $b(p) = \sum_{h \in p} b(h)$. This allows for low probability partitions to be identified and removed. However, since the partitions represent groups of user goals, completely removing a user goal makes it impossible to recreate it and this is clearly not desirable.

Rather then removing the partitions, the method proposed in [Williams 2010] reduces the number of partitions by recombining the low probability leaf partitions with their parent partitions.

[3]The runtime results are obtained on 8 core Intel Xeon 2.83GHz processor and 24Gbytes RAM

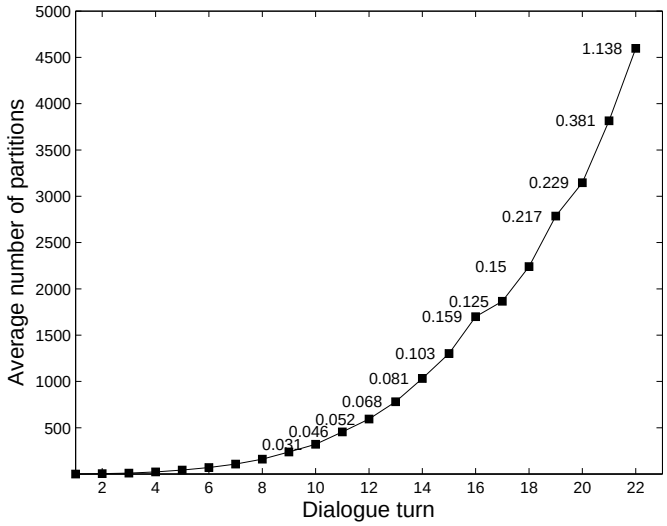


Fig. 5. Average number of partitions for dialogue turn. The values on the curve denote the average time to perform the belief update in seconds.

1: Let o' be an observation from the N-best input
2: Let p be a partition and its belief $b(p)$
3: Let h be a hypothesis and its belief $b(h)$
4: **repeat** for each dialogue turn
Belief Update
5: Partition each p using attribute-value pairs from the last system action a_m
6: Initialise $b'(p) = 0$ for all partitions p in the current set of partitions
7: **for** each o' in the N-best list **do**
8: Partition each p using attribute-value pairs from o'
9: **for** each partition p' in the current set of partitions **do**
10: **for** create new hypothesis h' from previous hypothesis h and o' **do**
11: $b'(h') = P(o'|a'_u)P(a'_u|p', a_m)P(h'|h, p', a_u, a_m)P(p'|p)b(h)$
12: $b'(p') = b'(p') + b'(h')$
13: **end for**
14: **end for**
Partition Recombination
15: Recombine partitions w.r.t the current updated belief $b'(p')$
16: **end for**
Action Selection
17: Choose the next system action a'_m according to $b'(h')$
18: **until** dialogue ended

Fig. 6. Belief Update with Recombination

The recombination is performed by removing the complementary value from the parent partition, updating its probability with the probability of its child partition and removing the child partition.

An outline of the belief update algorithm that utilises this partition recombination is given in Fig. 6. In each dialogue turn the partitioning is performed using the attribute-value pairs from the last system action (line 5). Then, for each observation in the N-best user input the partitioning is performed using its attribute-value pairs (line 8), the belief over new hypotheses is updated (line 11) and the updated belief over partitions is accumulated (line 12). If the number of partitions exceeds the threshold, the partitions are recombined according to the current updated belief (line 15). After the whole N-best list is processed, the next system action is chosen according to the updated belief.

This method is shown to be effective in domains that do not have many attributes [Williams 2010]. However, the method has its limitations in more complex domains. Firstly, a partition can

only be recombined with its parent even though there may be other partitions it is complementary to and which would be better candidates for recombination. Referring to the example from Fig 4, partition 1.1.1.1.1.1 is complementary both to partition 1.1.1.1.1 and to partition 1.1.1.1.2. Secondly, allowing only leaf partitions to be removed might not be desirable in long dialogues. Leaf partitions are usually the last to be created but in dialogues where the user goal evolves over time, the partitions that are created early on typically become less probable as the dialogue progresses. Thus, the more recent leaf partitions are often more useful. However, if one simply modifies the recombination technique to allow for non-leaf partitions to be recombined, as in for example 1.1.1 and 1.1.2, it would be difficult to determine the right position for the newly obtained partition. What is more, such a partition would not have any complements and thus it would be impossible to remove it before other partitions are recombined. Finally, the problem of partitions without any complements can occur even in the case of recombining the leaf partitions. For example, recombining 1.1.1.1.1.1 with its parent 1.1.1.1.1 results in a partition that does not have any complements. In complex dialogues, where the user can change the goal, it may be important that each partition has a complementary partition.

4.2 Pruning of the Applied Attribute-Value List

In this paper, we propose a new pruning method that is not constrained by the position of partitions in the tree and guarantees that every partition has a complement.

Rather then recombining the partitions, the number of partitions can be reduced by removing some of the applied attribute-value pairs. The marginal probability of attribute-value pair $s = v$ is the sum of probabilities of all partitions that have v set to true. In that way, a sorted list of the applied attribute-value pairs can be obtained. The lowest probability attribute-value pairs are likely to have the least impact on modelling the user goal and therefore can be removed.

Let $s = v$ be the attribute-value with the lowest probability in the list of applied attribute-value pairs. Assume that $s = v$ is not among the superiors for any other applied attribute-value pair $s_k = v_k$ (see Subsection 3.2). To remove $s = v$, the following procedure is taken. Starting from the root partition, the partition is examined to see if it contains node s with a child node containing $\neg v$. If not, the search is continued through its children starting from the oldest. If it does contain $\neg v$, it is marked as *upper*. Then the search is performed through its children, starting from the oldest until, one that contains node s with a child node containing v is found. It is marked as *lower*. Such a pair of partitions is guaranteed to exist, they are complementary and only differ in the atomic node that contains v and for every given *upper* there is only one lower (see Stat. 1 and 5 in Appendix 7.1) . What is more, if partitions *upper* and *lower* have child partitions, they have subtrees with the same structure where these partitions are roots. Each partition from the *upper* subtree will have its complement in the *lower* subtree. All that is needed is to add the belief of each partition in the *lower* subtree to its complement in the *upper* subtree, to remove $\neg v$ from the *upper* partition and then to delete the *lower* subtree (see Stat. 6 in Appendix 7.1). The procedure is continued until there is no partition matching $\neg v$ left. This can be easily performed using a stack structure. The pseudo code is given in the Appendix 7.3.

Every time a *lower* partition is removed all the hypotheses that are associated to that partition are removed. However, when a node is removed in an *upper* partition only the history related to that node is removed in each associated hypothesis.

An example of the pruning procedure is given in Fig. 7, where the attribute-value pair `food=Italian` is removed from the list of applied attribute-value pairs. The first partition that contains node `food` and `!Italian` is 1.1.1 and its child partition that contains `Italian` is 1.1.1.2. They are respectively marked as *upper* and *lower* and both of them have child partitions which are complementary, 1.1.1.3 and 1.1.1.2.1 respectively, similarly for partitions 1.1.1.1 and 1.1.1.1.1. Then, partitions with `Italian` are deleted and `!Italian` is removed from their complementary partitions.

If $s = v$ is among the superiors of some attribute-value pair $s_k = v_k$, then $s_k = v_k$ has to be pruned from the part of the tree that contains $s = v$ (see Stat. 8 in Appendix 7.1). The algorithm is same as the one described above, with the difference that *upper* and *lower* partitions are complementary in v_k and both contain v set to true. Referring to the example from Fig 4, if, for instance, `type=restaurant` is to be removed, then `food=Italian` and `area=central` have

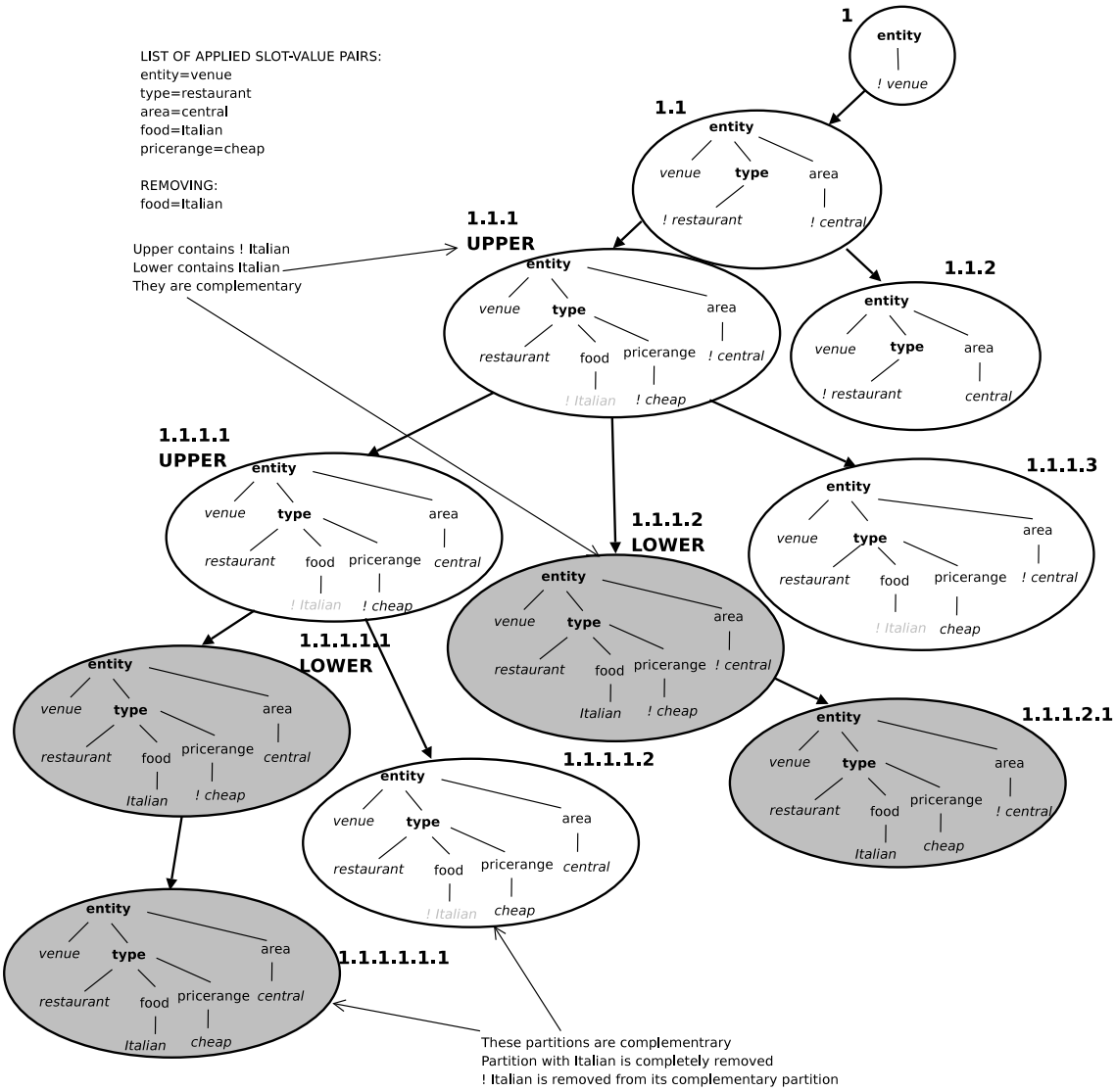


Fig. 7. Pruning an attribute-value pair from the tree of partitions

to be removed first. If the pruning is performed based on the lowest probability, it is never the case that `food=Italian` has higher probability than `type=restaurant`, since it can only occur in the partitions that have `restaurant` as true. However, some attributes can occur for different realisations of class nodes. For example, `type=restaurant` and `type=hotel` can both have `pricerange=cheap`, see Table I. In that case, if `type=hotel` is to be removed, `pricerange=cheap` has to be removed only from the partitions that have `hotel`.

In this way, it is guaranteed that the lowest probability attribute-value pair can be removed from the tree of partitions, regardless of when it was applied and how the partitions that contain it are structured in the tree. After pruning an attribute-value pair, the structure of the tree of partitions is the same as if that attribute-value pair had never been applied at all, so the pruning does not affect the existence of complements.

Whilst the partitioning process exponentially increases the number of partitions, the above pruning technique decreases it at the same rate, so the total number of partitions remains bounded. This allows dialogues of arbitrary length. Furthermore, it also enables large N-best inputs to be applied.

An outline of the belief update algorithm that utilises the above pruning method is given in

1: Let o' be an observation from the N-best input
2: Let p be a partition
3: Let h be a hypothesis and its belief $b(h)$
4: Let d be an attribute-value pair and $p(d)$ its marginal probability
5: **repeat** for each dialogue turn
Pruning
6: **for** each applied attribute-value pair d **do**
7: $p(d) = \sum_{p:d \in p} \sum_{h \in p} b(h)$
8: **end for**
9: Prune the list of the applied attribute-value pairs w.r.t. $p(d)$
Belief Update
10: Partition each p using attribute-value pairs from the last system action a_m
11: **for** each o' in the N-best list **do**
12: Partition each p using attribute-value pairs from o'
13: **for** each partition p' in the current set of partitions **do**
14: **for** create new hypothesis h' from previous hypothesis h and o' **do**
15: $b'(h') = P(o'|a'_u)P(a'_u|p', a_m)P(h'|h, p', a_u, a_m)P(p'|p)b(h)$
16: **end for**
17: **end for**
18: **end for**
Action Selection
19: Choose system action according to $b'(h')$
20: **until** dialogue ended

Fig. 8. Belief Update with Pruning

Fig. 8. In contrast to the algorithm in Fig. 6, pruning is applied before the processing of the input, so that no information from the current N-best list is lost before the next system action is chosen. The reason for this is to accommodate negotiation-type dialogues, where the user can change their mind and therefore put more importance on the most recent observations rather than the keeping of low probability older information. More precisely, in this framework what the user does not want and what the system is uncertain about are represented in the same way in order to retain tractability. For that reason, very low probability slot-value pairs are most probably the ones that the user does not want so it is safer to remove them rather that the ones that were obtained from the latest N-best input. At the beginning of each dialogue turn, the marginal probability of all applied attribute-value pairs is calculated (lines 6-8). Then pruning of the lowest probability attribute-value pairs is performed (line 9). Following that, partitioning is performed using the attribute-value pairs from the last system act (line 10). Then, partitioning is performed for each user dialogue act in the N-best input list using its attribute-value pairs and the belief is updated (lines 11-18). Finally, the next system action is chosen based on the updated belief (line 19).

In Appendix 7.4 an example of a long negotiation-type dialogue which incorporates this pruning technique is given together with the list of active attribute-value pairs and their marginal probabilities in each turn.

5. EVALUATION

The evaluation is divided in three parts. In Section 5.1, we examine how well the system can deal with user goals when the constraints are in the form of disjunctions. Then, in Section 5.2, we compare the performance of pruning in the applied attribute-value list algorithm (described in Section 4.2) and a version of the partition recombination algorithm [Williams 2010]. Finally, in Section 5.3 different pruning thresholds are examined.

For each experiment, the policy was trained in interaction with a simulated user at the dialogue act level. The application is the Cambridge tourist information system. The domain consists of more then 500 entities each of which has upto 10 attributes the user can query. The full description of the ontology is given in Appendix 7.2. The simulated user gives a reward at the end of each dialogue of 100 if the dialogue was successful and 0 otherwise, less the number of turn. The simulated user allows a maximum of 100 turns in each dialogue, terminating it when all the necessary information has been obtained or if the dialogue manager repeats the same dialogue

action more than three times in a row. The simulated user is able to generate user acts for a particular goal, but it can also change the goal during the dialogue. When the system makes an offer the simulated user changes its goal with probability 0.28. An error model is used to add confusion to the user input and it produces a scored N-best list of user dialogue acts with confidence scores consistent with the required error rate. Each error rate roughly represents the probability of the user input not being on the top of the N-best list [Thomson 2009]. In order to demonstrate the system's capability for dealing with reasonably long N-best lists, the length of the N-best list was set to 10. The system takes about 13 turns on average to complete a dialogue; Fig. 9 shows how the number of turns increases as the user input becomes more noisy. The policies were trained using the grid-based Monte Carlo Control algorithm in an incremental noise setting [Young et al. 2010]. The resulting policies were evaluated with the simulated user performing 2500 dialogues at each error rate.

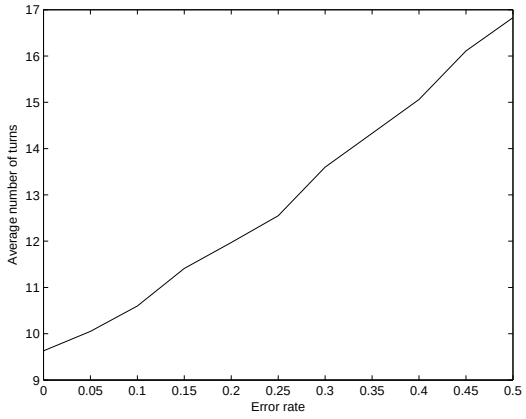


Fig. 9. Average number of turns as a function of error rate

5.1 Disjunctions in the User Goal

For this experiment, the simulated user was modified to produce constraints in the user goal such that on average 20% of them contain a disjunction of two values, for example, `type=restaurant`, `pricerange = moderate | cheap`, `food=Japanese`. The performance of the system is compared on both the tasks that contain disjunctions and regular tasks. The performance is measured as the average reward at different confusion levels and the results are presented in Fig. 10[4]. As can be seen from the graph, the system can deal with disjunctions in the user constraints at least as well as it can for standard user constraints.

5.2 Pruning vs Recombination

In this experiment, the two methods for reducing the number of partitions: the pruning of the applied attribute-value list and the partition recombination algorithm are compared. In both cases, the maximum number of partitions was set to 300. We examined the performance by measuring the average reward that the system obtained with each of the methods. The results are given in Fig 11 which shows that the pruning of the applied attribute-value list gives a better overall performance. As shown by the error bars, the results are statistically significant in the high noise regions, suggesting that it can more effectively manage user goal partitioning in noisy complex domains compared to the simpler partition recombination approach.

[4]The error bars represent a 95% confidence interval.

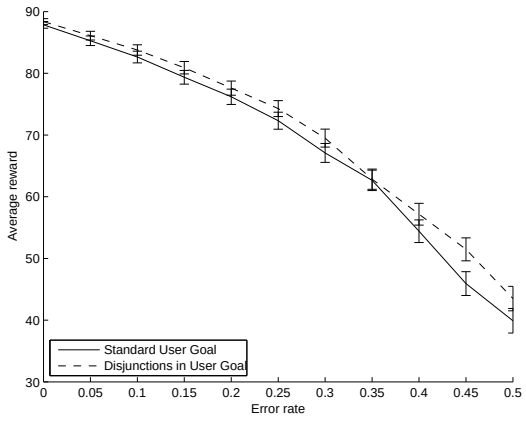


Fig. 10. System Performance with Disjunctions in the User's Request

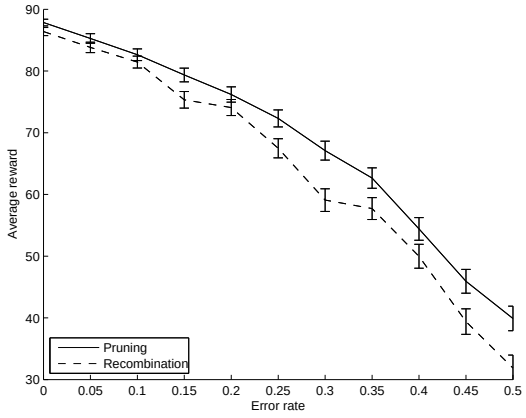


Fig. 11. System Performance Comparison of Attribute-Value List Pruning (Pruning) and Partition Recombination (Recombination). In both cases, the maximum number of active partitions was 300.

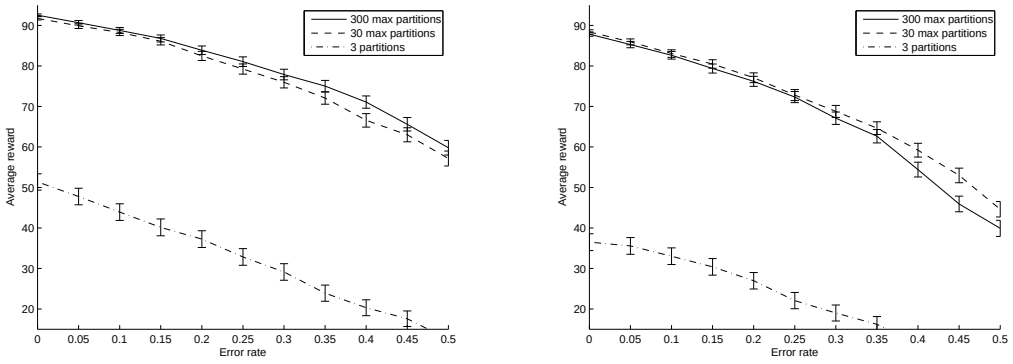
5.3 The Effect of Different Pruning Thresholds

In order to examine the effect that the pruning has on the system's performance, three different pruning thresholds were compared: 3, 30 and 300. In addition, a contrast is given between two different user simulator setting – one where the user goal stays constant during the dialogue, and one where the user changes its goal during the dialogue. The comparison is given in Fig. 12.

In the case when the user goal stays constant during the dialogue, increasing the number of partitions leads to improved system performance. This is in line with the findings in [Williams 2010]. It is important to note that, in contrast to a dramatic difference between 3 and 30, the difference between 30 and 300 is mostly not statistically significant. 3 partitions roughly correspond to only 2 attribute-value pairs representing the user goal and the user typically has about 4 constrains, so the goal in that case is not fully represented. Therefore, if the dialogue manager has a very low pruning threshold it is not able to represent the user goal even on zero error rate, which leads to low performance. The results from Fig. 12 suggest that increasing the number of partitions over 300 would not improve the performance further and this was confirmed by further tests at 3000.

In the case where the user goal changes during the dialogue, the threshold of 30 gives a more robust performance on higher error rates then the threshold of 300, see Fig. 12(b). This is probably a consequence of the fact that the HIS system does not have an explicit state transition matrix. Since

a change of user goal can also be achieved by discarding earlier evidence in favour of the most recent evidence, pruning helps achieve this. Thus, in the HIS model, pruning enables the dialogue to be more adaptive to inconsistent user behaviour. In real dialogues, users do not normally have a strictly defined goal but are likely to change their mind depending on the system's response, and pruning can facilitate this.



(a) User goal stays constant during the dialogue (b) User goal changes during the dialogue

Fig. 12. Influence of the pruning threshold on system's performance when user has a constant goal (a) and when user changes the goal (b)

6. CONCLUSION

This paper has described how enriching the dialogue state structure with the explicit representation of complements can improve POMDP-based dialogue modelling in a complex domain. It enables the use of disjunctions and conjunctions in the user request as well as quantifiers in the system's response. More importantly, the notion of complements provides a basis for a pruning technique that can effectively bound the number of partitions created during a dialogue and thereby ensure tractability. It supports N-best lists of user dialogue act hypotheses which are large enough to include all of the informative variants during noisy speech, and it can handle dialogues of arbitrary length. Furthermore, we have shown that this new pruning technique leads to better performance than the existing recombination method in practical real-world application domains.

Acknowledgements

This research 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). The authors would like to thank anonymous reviewers for their suggestions that helped improve the paper, as well as Rogier van Dalen, Filip Jurčiček, Simon Keizer, François Mairesse, Blaise Thomson and Kai Yu for useful comments and discussions.

REFERENCES

Kim, K., Lee, C., Jung, S., and Lee, G. G. 2008. A frame-based probabilistic framework for spoken dialog management using dialog examples. In *SIGdial '08: Proceedings of the 9th SIGdial Workshop on Discourse and Dialogue*. Association for Computational Linguistics, Morristown, NJ, USA, 120–127.

Levin, E., Pieraccini, R., and Eckert, W. 1998. Using Markov Decision Processes for Learning Dialogue Strategies. In *Int Conf Acoustics, Speech and Signal Processing*. Seattle,USA.

Sutton, R. and Barto, A. 1998. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, Mass.

Thomson, B. 2009. Statistical methods for spoken dialogue management. Ph.D. thesis, University of Cambridge.

Thomson, B. and Young, S. 2010. Bayesian update of dialogue state: A POMDP framework for spoken dialogue systems. *Computer Speech and Language* In press.

Thomson, B., Yu, K., Gašić, M., Keizer, S., Mairesse, F., Schatzmann, J., and Young, S. 2008. Evaluating semantic-level confidence scores with multiple hypotheses. In *Interspeech*. Brisbane, Australia.

Williams, J. 2010. Incremental Partition Recombiantion for Efficient Tracking of Multiple Dialogue States. In *Int Conf Acoustics Speech and Signal Processing ICASSP*. Texas.

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

Young, S., Gašić, M., Keizer, S., Mairesse, F., Schatzmann, J., Thomson, B., and Yu, K. 2010. The Hidden Information State Model: a practical framework for POMDP-based spoken dialogue management. *Computer Speech and Language 24(2),* 150-174.

7. APPENDIX

7.1 Consistency Proofs for Pruning Operations

DEFINITION 1. *An ontology is a forest of ordered trees such that all its trees have the same root node and if a tree contains a node that has multiple child nodes, then the first child node determines the other child nodes of that node.*

DEFINITION 2. *An attribute-value pair is a pair of nodes from some ontology tree where the value is a leaf node and the attribute is its parent node. It is denoted as $s = v$.*

DEFINITION 3. *Attribute-value pair $s = v$ is called superior to attribute-value pair $s' = v'$ if, in some ontology tree, value v is the first child node of node s and among other child nodes of s, $v_1, \cdots, v_n$, there exists node v_i so that nodes s and v_i appear on the path from s' to the root in that ontology tree.*

DEFINITION 4. *A partition is a tree where all its non-leaf nodes coincide with all non-leaf nodes of at least one ontology tree, the coinciding ontology tree. Each leaf node of a partition is a set of boolean indicators. There is one indicator for each coinciding ontology tree. Each indicator corresponds to the value in an attribute-value pair in the coinciding ontology tree, where the attribute coincides with the parent of the leaf node in the partition.*

DEFINITION 5. *The generic partition is the partition that has only two nodes. The root node of the generic partition coincides with the root node of every tree in the ontology and the leaf node has all indicators set to true.*

DEFINITION 6. *Two partitions p and c are complementary in attribute-value pair $s = v$ if they are the same in all nodes except in the nodes the child nodes of which have all indicators set to true in either p or c, and in the child node of node s, where in partition p all boolean indicators are set to false apart from the one for v, which is set to true, and in partition c the indicator for v is set to false. It is said that partition p contains $s = v$ and partition c contains $s = \neg v$.*

DEFINITION 7. *Attribute-value pair $s = v$ is applicable to partition p if partition p contains the boolean indicator for value v set to true in the child node of node s and at least one more indicator set to true in the child node of node s. Then, the process of applying attribute-value pair $s = v$ to partition p is setting the indicator for v to false in partition p and replicating partition p into partition c where all indicators are set to false in the child node of node s, apart from the indicator for v, which is set to true. In the case where s has multiple child nodes in one of the coinciding ontology tree of partition c and v is its first child node in that ontology tree, other child nodes of node s are created in that partition. Every newly created child node then has a child leaf node where all indicators are set to true. The same applies to partition p if only one indicator in the child node of node s remains true. Partition c is called a child of partition p and partition p is called the parent of c. The process of applying attribute-value pair to a partition is also called partitioning.*

STATEMENT 1. *Partition p and its child partition c created by partitioning p with $s = v$ to p are complementary in $s = v$.*

PROOF. Trivial from Defs. 6 and 7. □

DEFINITION 8. *A tree of partitions is an ordered tree where the nodes are partitions connected in parent-child relationship. The tree of partitions is created by applying ordered list of attribute-value pairs $s_1 = v_1, \cdots, s_n = v_n$ in such way that $s_1 = v_1$ is applied to the generic partition, $s_2 = v_2$ is applied to all resulting partitions and so on for every subsequent attribute-value pair from the list.*

DEFINITION 9. *A subtree of partitions is a tree of partitions where instead of the generic partition the partitioning process starts with an arbitrary partition.*

STATEMENT 2. *Two trees of partitions are the same if they are created using the same ordered list of attribute-value pairs.*

PROOF. Trivial from Def. 8. □

STATEMENT 3. *Two subtrees of partitions are the same if they are created using the same ordered list of attribute-value pairs starting from the same partition.*

PROOF. Trivial from Stat. 2. □

STATEMENT 4. *Attribute-value pair $s = v$ cannot be applied to partition p if partition p does not contain all the superior attribute-value pairs of $s = v$ that belong to one coinciding ontology tree of partition p.*

PROOF. Trivial from Defs. 3 and 7. □

STATEMENT 5. *Let T be the tree of partitions created with ordered list of attribute-value pairs $s_1 = v_1, \cdots, s_k = v_k, s = v, s_{k+1} = v_{k+1}, \cdots, s_n = v_n$. Then, there exists at least one pair of partitions p and c in T such that p is the parent of c, they are complementary in $s = v$ and for every such p there exists only one c.*

PROOF. Let T' be the tree of partitions created by attribute-value pairs $s_1 = v_1, \cdots, s_k = v_k$. Then, in order to apply attribute value pair $s = v$, $s = v$ has to be applicable to at least one partition p. Applying $s = v$ to partition p creates partition c such that p is the parent of c and they are complementary in $s = v$ (Def. 7 and Stat. 1).

Attribute-value pair $s = v$ is only applied to partition p if it contains the indicator for value v set to true. Once p is partitioned, the indicator is set to false and therefore $s = v$ cannot be reapplied to partition p (Def. 7). Hence, another such child partition c cannot be created. □

DEFINITION 10. *Pruning of attribute-value pair $s = v$ from tree of partitions T built using ordered list of attribute-value pairs $s_1 = v_1, \cdots, s_k = v_k, s = v, s_{k+1} = v_{k+1}, \cdots, s_n = v_n$ is the process of removing all partitions from the tree T that are built using $s = v$ or created from partitions that are built using $s = v$ so that the resulting T' is the same as if it was built using attribute-value pairs $s_1 = v_1, \cdots, s_k = v_k, s_{k+1} = v_{k+1}, \cdots, s_n = v_n$.*

STATEMENT 6. *Let T be the tree of partitions created by applying ordered list of attribute-value pairs $s_1 = v_1, \cdots, s_k = v_k, s = v, s_{k+1} = v_{k+1}, \cdots, s_n = v_n$. Let p and c be a pair of partitions such that p is the parent of c and they are complementary in $s = v$. Let $s = v$ be not among the superiors for any attribute-value pair $s_i = v_i$, $i = k+1, \cdots, n$. If p and c are roots of subtrees, these subtrees have the same structure and for every partition in one subtree there exists a partition in the same place in the other subtree and such that they are complementary in $s = v$.*

PROOF. Assume that every attribute-value pair $s_i = v_i$ from $s_{k+1} = v_{k+1}, \ldots s_n = v_n$, if applicable to a partition that contains $s = v$, is then also applicable to partition that contains $s = \neg v$. Then, sublist of applied attribute-value pairs $s_{i_1} = v_{i_1}, \ldots s_{i_m} = v_{i_m}$ that is applicable to a partition which contains $s = v$, is also applicable to a partition which contains $s = \neg v$. By Defs. 7 and 6 partition p contains $s = v$ and partition c contains $s = \neg v$. Since the sublist $s_{i_1} = v_{i_1}, \ldots s_{i_m} = v_{i_m}$ is applied in the same order to both of these partitions, it creates two subtrees with roots p and c respectively. These subtrees have the same structure since the list is applied in the same order and by Def. 8 a tree of partitions is an ordered tree. Had p and c been identical then these subtrees would have been identical too (Stat. 3). Since partitions p and c only differ in the child node of s, every partition of one subtree only differs in the child node of s from the partition in the same position in the other subtree. In that way, for every partition in the subtree with the root p there exists a partition in the same place in the subtree with the root c and such that they are complementary in $s = v$.

Let $s_i = v_i$ be an attribute-value pair that is applicable to partitions that contain $s = v$ but not applicable to partitions that contain $s = \neg v$. That means that $s_i = v_i$ can only appear in trees that contain $s = v$, which means that having indicator for v set to true enables creating other child nodes of s and thus allows for $s_i = v_i$ to be applied. Then, by Def. 3, $s = v$ is among the superiors for $s_i = v_i$. This is in contradiction with the assumption that $s = v$ is not among the superiors for any attribute-value pair $s_i = v_i$, $i = 1, \ldots, k$. □

STATEMENT 7. *Let T be the tree of partitions created by applying ordered list of attribute-value pairs $s_1 = v_1, \ldots, s_k = v_k, s = v, s_{k+1} = v_{k+1}, \ldots, s_n = v_n$ and $s = v$ is not among superiors for any attribute-value pair $s_i = v_i$, $i = 1, \ldots, k$. Then, $s = v$ can be pruned from T.*

PROOF. According to Stat. 6 there exists at least one pair of partitions p and c which are complementary in $s = v$ and if they are roots of subtrees, then for every partition in one subtree there exists a partition in the same place in the other subtree such that they are complementary in $s = v$. If for all such pairs p and c, the subtree with c as the root is removed and the indicator for v is set to true in each partition in the subtree with p as the root, the resulting tree does not have any partitions created by applying $s = v$ or created from the partitions that had $s = v$ applied to it. □

STATEMENT 8. *Any attribute-value pair that is used for building a tree of partitions can be pruned from that tree.*

PROOF. Let $s = v$ be the attribute value pair that is to be pruned from tree of partitions T created by $s_1 = v_1, \ldots, s_k = v_k, s = v, s_{k+1} = v_{k+1}, \ldots, s_n = v_n$. If $s = v$ is not among any superiors of $s_{k+1} = v_{k+1}, \ldots, s_n = v_n$ then Stat. 7 applies.

Assume $s = v$ is among the superiors for $s_{k_i} = v_{k_i}$ from $s_{k+1} = v_{k+1}, \ldots, s_n = v_n$ and assume $s_{k_i} = v_{k_i}$ is not among superiors for any of $s_{k_{i+1}} = v_{k_{i+1}}, \ldots, s_n = v_n$. Let partitions p and c be complementary in $s = v$ and be the roots of subtrees T_p and T_c. Then, according to Stat. 7 $s_i = v_i$ can be pruned from subtree T_c and the resulting subtree T'_c is the same one as the one created when $s_{k+1} = v_{k+1}, \ldots, s_{k_{i-1}} = v_{k_{i-1}}, s_{k_{i+1}} = v_{k_{i+1}}, \ldots, s_n = v_n$ are applied to partition c. Then, if $s = v$ is not among superiors for any of $s_{k+1} = v_{k+1}, \ldots, s_{k_{i-1}} = v_{k_{i-1}}, s_{k_{i+1}} = v_{k_{i+1}}, \ldots, s_n = v_n$, according to Stat. 7, it can be pruned. Otherwise the same applies to any such attribute-value pair from $s_{k+1} = v_{k+1}, \ldots, s_{k_{i-1}} = v_{k_{i-1}}, s_{k_{i+1}} = v_{k_{i+1}}, \ldots, s_n = v_n$, $s = v$ is the superior of.

If $s_{k_i} = v_{k_i}$ is among superiors for any of $s_{k_{i+1}} = v_{k_{i+1}}, \ldots, s_n = v_n$ the same procedure is applied recursively until the attribute-value pair that is not among superiors for any of the attribute-value pairs following in the list is found and pruned. The last attribute-value pair trivially satisfies that requirement.

□

7.2 CamInfo Domain

The CamInfo domain is a tourist information for Cambridge, whereby the user can ask for information about a restaurant, a bar, a hotel, a museum or other tourist attractions in the local area. The database consists of more than 400 entities each of which has up to 10 attributes that the user can query. The possible attribute-value pairs are organised in an hierarchical ontology, see Table VI.

entity	←	*venue*(**type**, name, area, near, addr, phone, postcode)
type	←	*placetostay*(**staytype**, hasinternet, hasparking, price, pricerange, stars)
type	←	*placetoeat*(**eattype**, pricerange, openhours, price)
type	←	*placetodrink*(**drinktype**, pricerange, openhours, price)
type	←	*placetosee*(**seetype**, pricerange, openhours)
type	←	*entsvenue*(**entstype**)
type	←	*univenue*(**unitype**, openhours)
type	←	*sportsvenue*(**sport**)
type	←	*transvenue*(**transtype**)
type	←	*shopvenue*(**shoptype**, openhours)
type	←	*amenity*(**amtype**)
amtype	←	*hospital*()
amtype	←	*policestation*()
amtype	←	*bank*(openhours)
amtype	←	*postoffice*(openhours)
amtype	←	*touristinfo*(openhours)
shoptype	←	*supermarket*()
shoptype	←	*shoppingcentre*()
transtype	←	*airport*()
transtype	←	*busstation*()
transtype	←	*trainstation*(openhours)
staytype	←	*guesthouse*()
staytype	←	*hotel*()
eattype	←	*restaurant*(food)
drinktype	←	*bar*(childrenallowed, hasinternet, hasmusic, hastv, openhours, price)
drinktype	←	*coffeeshop*()
drinktype	←	*pub*(childrenallowed, hasfood, hasinternet, hastv)
seetype	←	*architecture*()
seetype	←	*museum*()
seetype	←	*park*()
unitype	←	*college*()
unitype	←	*department*()
unitype	←	*library*()
entstype	←	*cinema*()
entstype	←	*theatre*()
entstype	←	*nightclub*(openhours, price, pricerange)
entstype	←	*entertainment*()
entstype	←	*boat*()
entstype	←	*concerthall*()
food	=	{ *American*, *Cafe food*, *Chinese*, ... }
pricerange	=	{ *free*, *cheap*, *moderate* , ... }
sport	=	{ *badmintoncourt*, *cricketfield*, *footballfield* , ... }
area	=	{ *girton*, *kingshedges*, *arbury* , ... }
...		

Table VI. CamInfo Ontology Rules

7.3 Pseudo Code for Pruning

```
void Partition::prune(AttributeValuePair d)
{
  PartitionPtr upper, lower;
  Stack<float>& s;
  if(this->has_children)
    for each child c of this starting from the oldest
      c->prune(d);
  if (this->contains(complement(d)))
  {
    upper=this;
    for each child c of upper starting from the oldest
      if(c->contains(d))
      {
        lower=c;
        break;
      }
    lower->deletePartitions(s);
    s.push(lower->belief);
    delete(lower);
    upper->updateBeliefandRemoveComplements(s,d);
  }
}
void Partition::deletePartitions(Stack<float>& s)
{
  for each child c of this starting from the oldest
  {
    c->deletePartitions(s);
    c->push(belief(c));
  }
  delete(children);
}
void Partition::updateBeliefandRemoveComplements(Stack<float>&d, AttributeValuePair d)
{
  this->belief+=s.pop();
  this->removeComplement(d);
  for each child c starting from oldest that contains negation of d
  {
    c->updateBeliefandRemoveNegations(s,d);
    if(s->size()==0)
     break;
  }
}
```

7.4 Typical Long Negotiative Dialogue

In this section an example of a typical long negotiative dialogue is given. It consists of 17 dialogue turns in each of which the system prompt and the true user action are given. In addition, the system's dialogue act and a scored list of the N-best user dialogue acts are provided, together with the probability of the most likely hypothesis and a sorted list of applied attribute-value pairs with their marginal probabilities. The attribute-value pairs that are in bold font match the most likely partition.

There are several points of the dialogue to note. Firstly, turns 4-6 show how the user goal changes during the dialogue. Secondly, turns 8 and 11 show the capability of the system to deal with long N-best inputs[5]. Furthermore, there are also examples of how the system deals with a range of problems, such as mis-recognitions in turns 9, 11 and 12, a user mistake in turn 10 and a silence detection problem in turn 13. Moreover, in turns 8, 9 and 11-16 pruning of the lowest probability attribute value pairs takes place. Finally, it is also interesting to note how the probability of the top hypothesis changes over time. For example, in turn 12, where a mis-recognition occurs, the probability of the top hypothesis is very low, only 0.013. This is due to the fact that the user through-out the dialogue asked for a place to stay and in turn 12 the top partition contains a restaurant. However, in turn 14 and 15 when the system offered a hotel and user asked for an address, the probability increases from 0.025 to 0.308 showing how the system became more confident about the user's requirements.

1	System act	hello()
	System prompt	Thank you for calling the Cambridge Information system. How may I help you?
	True user action	HI I'M LOOKING FOR A PLACE TO STAY
	N-best input	hello(type=placetostay,task=find) 1.00
		hello(type=placetostay) 0.00
		hello(type=placetostay,name="The Place",task=find) 0.00
	Top hypothesis prb	0.676
	Applied attributes	**task=find** 0.997; **entity=venue** 0.994; **type=placetostay** 0.987; name=The Place 0.312
2	System act	confreq(type=placetostay,near)
	System prompt	You are looking for a place to stay. Are you looking for somewhere near a particular venue?
	True user action	MAYBE SOMEWHERE IN THE CENTRE
	N-best input	inform(area=citycentre) 1.00
	Top hypothesis prb	0.900
	Applied attributes	**task=find** 0.999; **entity=venue** 0.998; **area=citycentre** 0.972
		type=placetostay 0.932; name=The Place 0.012
3	System act	inform(name="Gonville Hotel",type=placetostay,area=citycentre)
	System prompt	The Gonville Hotel is a great place to stay and it's in the central part of town.
	True user action	IS IT EXPENSIVE
	N-best input	confirm(pricerange=expensive) 1.00
		inform(pricerange=expensive) 0.00
	Top hypothesis prb	0.456
	Applied attributes	**task=find** 0.999; **entity=venue** 0.998; **type=placetostay** 0.991
		name=Gonville Hotel 0.978; **area=citycentre** 0.930; **pricerange=expensive** 0.495
		name=The Place 0.006

Table VII. A typical long negotiative dialogue (turns 1-3)

[5]The confidence scores are here rounded to two decimals, but in fact they are always greater than zero.

4	System act	inform(name="Gonville Hotel",pricerange=none)
	System prompt	I am sorry but there is no information about the price range for the Gonville Hotel.
	True user action	DO YOU HAVE ANY CHEAP HOTELS
	N-best input	inform(pricerange=cheap,=hotel) 0.96
		inform(=hotel) 0.04
	Top hypothesis prb	0.290
	Applied attributes	**task=find** 0.999; **entity=venue** 0.998; **type=placetostay** 0.987
		area=citycentre 0.806; **staytype=hotel** 0.708; **pricerange=cheap** 0.585
		name=Gonville Hotel 0.523; pricerange=expensive 0.201; name=The Place 0.014
5	System act	inform(name=none,type=placetostay,area=citycentre,pricerange!=dontcare)
	System prompt	There is no place to stay in the central area for which I have any information about the price range.
	True user action	HOW ABOUT RIVERSIDE
	N-best input	reqalts(area=riverside) 0.90
		inform(area=riverside) 0.05
		reqalts() 0.04
		reqalts(=bar,area=riverside) 0.01
		null() 0.00
	Top hypothesis prb	0.213
	Applied attributes	**task=find** 0.999; **entity=venue** 0.998; **area=riverside** 0.775; **type=placetostay** 0.752
		staytype=hotel 0.585; **pricerange=cheap** 0.419; name=Gonville Hotel 0.273;
		type=placetodrink 0.198; area=citycentre 0.152
		pricerange=expensive 0.150; drinktype=bar 0.092; name=The Place 0.068
6	System act	inform(name=none,type=placetostay,pricerange=cheap)
	System prompt	There is no place to stay in the cheap price range.
	True user action	HOW ABOUT SOMETHING MODERATE
	N-best input	reqalts(pricerange=moderate) 0.83
		inform(pricerange=moderate) 0.17
	Top hypothesis prb	0.065
	Applied attributes	**task=find** 0.999; **entity=venue** 0.998; **pricerange=moderate** 0.847
		area=riverside 0.612; **type=placetostay** 0.593; type=placetodrink 0.394
		staytype=hotel 0.338; name=Gonville Hotel 0.247; area=citycentre 0.218
		drinktype=bar 0.193; name=The Place 0.138; pricerange=cheap 0.037
		pricerange=dontcare 0.036; pricerange=expensive 0.033
7	System act	inform(name=none,type=placetostay,staytype=hotel,area=riverside,pricerange=moderate)
	System prompt	There is no hotel in the riverside area in the moderate price range.
	True user action	I DON'T CARE ABOUT THE AREA
	N-best input	inform(=dontcare) 1.00
		null() 0.00
	Top hypothesis prb	0.116
	Applied attributes	**task=find** 0.999; **entity=venue** 0.998; **name=dontcare** 0.569; type=placetostay 0.512
		area=dontcare 0.491; **type=placetodrink** 0.458; **pricerange=dontcare** 0.433
		pricerange=moderate 0.357; **drinktype=bar** 0.252; area=riverside 0.252
		staytype=hotel 0.249; area=citycentre 0.131; name=Gonville Hotel 0.124
		name=The Place 0.103; pricerange=cheap 0.061; pricerange=expensive 0.059

Table VIII. A typical long negotiative dialogue (cont., turns 4-7)

8	System act	inform(name="The Eagle",type=placetodrink,drinktype=bar, area=citycentre,pricerange=none)
	System prompt	The Eagle, which is in the central part of town, is a great bar. I am sorry but there is no information about the price range for it.
	True user action	I'M LOOKING FOR A PLACE TO STAY
	N-best input	inform(type=placetostay,task=find) 0.99 inform(type=placetostay,name="The Place",task=find) 0.01 inform(type=placetostay) 0.00 inform(=trainstation,name="The Place",task=find) 0.00 inform(type=placetostay,name="Cambridge Police Station",task=find) 0.00 hello(type=placetostay) 0.00 **3 attribute-value pairs are pruned**
	Top hypothesis prb	0.080
	Applied attributes	**task=find** 0.999; **entity=venue** 0.998; **type=placetostay** 0.814 **pricerange=moderate** 0.516; **staytype=hotel** 0.421; **area=riverside** 0.321; pricerange=dontcare 0.259; area=dontcare 0.254; area=citycentre 0.196 type=placetodrink 0.125; name=Gonville Hotel 0.093; drinktype=bar 0.065 **name=dontcare** 0.065; name=The Eagle 0.049; name=Cambridge Police Station 0.049 name=The Place 0.049; type=transvenue 0.039; transtype=trainstation 0.020
9	System act	inform(name=none,type=placetostay,staytype=hotel,area=riverside,pricerange=moderate)
	System prompt	There is no hotel in the riverside area in the moderate price range.
	True user action	I DON'T CARE ABOUT THE AREA
	N-best input	inform(=dontcare) 0.57 null() 0.28 inform() 0.15 **7 attribute-value pairs are pruned**
	Top hypothesis prb	0.069
	Applied attributes	**task=find** 0.998; **entity=venue** 0.995; **type=placetostay** 0.958 **pricerange=dontcare** 0.435; staytype=hotel 0.420; **area=dontcare** 0.386 pricerange=moderate 0.365; **name=dontcare** 0.205 area=riverside 0.187; area=citycentre 0.182; name=Gonville Hotel 0.117
10	System act	inform(name="City Centre North",type=placetostay,staytype=guesthouse, area=arbury,pricerange=none)
	System prompt	City Centre North is an outstanding guesthouse in the Arbury part of town, also I am sorry but there is no information about the price range for it.
	True user action	DO YOU HAVE ANY MODERATE RESTAURANT UH HOTEL
	N-best input	inform(pricerange=moderate,=restaurant,=hotel) 0.91 inform(=restaurant,=hotel) 0.05 inform(pricerange=moderate,=restaurant) 0.04
	Top hypothesis prb	0.023
	Applied attributes	**task=find** 0.999; **entity=venue** 0.998; **type=placetostay** 0.743 **pricerange=dontcare** 0.399; pricerange=moderate 0.362; name=City Centre North 0.301 **area=dontcare** 0.277; staytype=guesthouse 0.261; type=placetoeat 0.216 staytype=hotel 0.209; **name=dontcare** 0.188; area=citycentre 0.182; area=riverside 0.169 area=arbury 0.169; eattype=restaurant 0.124; name=Gonville Hotel 0.101

Table IX. A typical long negotiative dialogue (cont., turns 8-10)

11	System act	inform(name="Finches Bed and Breakfast",type=placetostay,area=girton, pricerange=none,staytype=guesthouse)
	System prompt	Finches Bed and Breakfast is a great guesthouse in the Girton part of town and I am sorry but there is no information about the price range for it.
	True user action	I'M LOOKING FOR A MODERATELY PRICED PLACE TO STAY
	N-best input	inform(pricerange=moderate,task=find) 0.79 inform(name="The Place",pricerange=moderate,task=find) 0.09 inform(type=placetostay,pricerange=moderate,task=find) 0.06 inform(pricerange=free,pricerange=moderate,task=find) 0.04 inform(type=placetostay,pricerange=moderate,name="The Place",task=find) 0.01 **3 attribute-value pairs are pruned**
	Top hypothesis prb	0.062
	Applied attributes	**task=find** 0.999; **entity=venue** 0.998 **type=placetostay** 0.974 staytype=guesthouse 0.344; staytype=hotel 0.270; **area=dontcare** 0.206 area=girton 0.168; area=riverside 0.155; area=arbury 0.150; area=citycentre 0.134 name=City Centre North 0.063; pricerange=dontcare 0.074; pricerange=free 0.072 name=dontcare 0.064; name=Finches Bed and Breakfast 0.063; name=The Place 0.063
12	System act	inform(name="Express by Holiday Inn",area=barnwell, type=placetostay,hasinternet=true,hasparking=true,stars="2")
	System prompt	The Express Holiday Inn has an internet connection and a parking lot and it's in the Barnwell part of town. This place, which has 2 stars, is a great place to stay.
	True user action	WHAT'S THE PRICERANGE
	N-best input	inform(=restaurant) 0.89 null() 0.11 **5 attribute-value pairs are pruned**
	Top hypothesis prb	0.013
	Applied attributes	**task=find** 0.999; **entity=venue** 0.999; type=placetostay 0.815 pricerange=moderate 0.594; hasinternet=true 0.408; hasparking=true 0.408; stars=2 0.407; name=Express by Holiday Inn 0.327; staytype=guesthouse 0.279 staytype=hotel 0.251; name=City Centre North 0.224; **type=placetoeat** 0.171 **area=dontcare** 0.161; **eattype=restaurant** 0.146; area=barnwell 0.145 area=riverside 0.142; area=girton 0.141; area=arbury 0.133; area=citycentre 0.129
13	System act	inform(name="Riverside Restaurant and Granta Bar",area=riverside,eattype=restaurant, type=placetoeat,food=Traditional,pricerange=expensive,near="Queens' College")
	System prompt	The Riverside Restaurant is an outstanding restaurant in the riverside part of town in the expensive price range near Queens' College, also it serves traditional food.
	True user action	I'M LOOKING
	N-best input	inform(task=find) 0.50 null() 0.50 **8 attribute-value pairs are pruned**
	Top hypothesis prb	0.008
	Applied attributes	**task=find** 0.999; **entity=venue** 0.997; type=placetostay 0.830; hasinternet=true 0.422; hasparking=true 0.422; stars=2 0.416; pricerange=moderate 0.414 staytype=guesthouse 0.408; pricerange=expensive 0.286; area=dontcare 0.260 **area=riverside** 0.252; area=barnwell 0.243; name=Express by Holiday Inn 0.170 name=Riverside Restaurant and Granta Bar 0.168; type=placetoeat 0.146 eattype=restaurant 0.095; food=Traditional 0.047

Table X. A typical long negotiative dialogue (cont., turns 11-13)

14	System act	inform(name="The Cricketers",area=riverside,near="Ruskin Gallery")
	System prompt	The Cricketers is in the riverside part of town near the Ruskin Gallery.
	True user action	I'M LOOKING FOR A FOR A HOTEL IN THE MODERATE PRICERANGE
	N-best input	inform(pricerange=moderate,=hotel,task=find) 0.96 inform(=hotel,pricerange=moderate) 0.04 **5 attribute-value pairs are pruned**
	Top hypothesis prb	0.025
	Applied attributes	**task=find** 0.999; **entity=venue** 0.999; **type=placetostay** 0.993 **pricerange=moderate** 0.722; **staytype=hotel** 0.722; hasinternet=true 0.503 hasparking=true 0.503; stars=2 0.493; textbfarea=dontcare 0.264; area=riverside 0.244 area=barnwell 0.244; name=The Cricketers 0.206; staytype=guesthouse 0.135 pricerange=expensive 0.135
15	System act	inform(name="The Cambridge Belfry",type=placetostay,staytype=hotel, pricerange=moderate,area=girton,hasinternet=true,hasparking=true)
	System prompt	The Cambridge Belfry, which has an internet connection and parking spaces, is an outstanding hotel. It's in the moderate price range and in the Girton part of town.
	True user action	WHAT'S THE ADDRESS
	N-best input	request(addr) 1.00 **3 attribute-value pairs are pruned**
	Top hypothesis prb	0.308
	Applied attributes	**task=find** 0.999; **entity=venue** 0.998; **type=placetostay** 0.990 **pricerange=moderate** 0.854; **staytype=hotel** 0.854 **name=The Cambridge Belfry** 0.794; **hasinternet=true** 0.791 **hasparking=true** 0.791; **area=dontcare** 0.390; **area=girton** 0.365 stars=2 0.199; area=barnwell 0.081; area=riverside 0.081
16	System act	inform(name="The Cambridge Belfry",addr="Back Lane, Cambourne")
	System prompt	The Cambridge Belfry is located on Back Lane, Cambourne.
	True user action	AND THE PHONE NUMBER
	N-best input	request(phone) 1.00 **2 attribute-values are pruned**
	Top hypothesis prb	0.427
	Applied attributes	**task=find** 0.999; **entity=venue** 0.998; **type=placetostay** 0.993 **hasinternet=true** 0.912; **hasparking=true** 0.912 **pricerange=moderate** 0.912; stars=2 0.081; **staytype=hotel** 0.912 **name=The Cambridge Belfry** 0.914; **area=dontcare** 0.482; area=girton 0.460
17	System act	inform(name="The Cambridge Belfry",phone="01954714600")
	System prompt	The phone number of The Cambridge Belfry is 01954714600.
	True user action	THANKS BYE Hanging up

Table XI. A typical long negotiative dialogue (cont., turns 14-17)