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

M. Gašić and S. Young
CUED/F-INFENG/TR.650
Cambridge University Engineering Department
Trumpington Street
Cambridge. CB2 1PZ
England
E-mail: {mg436,sjy}@eng.cam.ac.uk

Abstract

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 relevant 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.

Contents

1	Introduction	3
2	POMDP Dialogue models	4
3	Hidden Information State Model	4
3.1	Domain Ontology	4
3.2	State Representation	6
3.3	Belief Update	6
3.4	Policy Representation and Action Selection	7
4	Extended State Representation	7
4.1	Explicit Representation of Complements	7
4.2	Partitioning Process	8
4.3	Logical Expressions for Negotiative dialogue	8
4.3.1	Quantifiers in the System's Response	8
4.3.2	Disjunctions and Conjunctions in the User's Request	11
5	Pruning	12
5.1	Partition Recombination	12
5.2	Pruning of the Applied Slot-value List	13
6	Evaluation	15
6.1	Disjunctions in the User Goal	16
6.2	Pruning vs Recombination	16
6.3	The Effect of Different Pruning Thresholds	17
7	Conclusion	18
8	Appendix	20
8.1	CamInfo Domain	20
8.2	Pseudo Code for Pruning	21
8.3	Typical Long Negotiative Dialogue	21

List of Figures

1	Statistical Spoken Dialogue System Structure	5
2	Generic Ontology Structure	5
3	Step-by-step Partitioning Process	9
4	Step-by-step Partitioning Process (cont.)	10
5	Average number of partitions for dialogue turn	12
6	Belief Update with Recombination	13
7	Pruning a slot-value pair from the tree of partitions	15
8	Belief Update with Pruning	16
9	System Performance with Disjunctions in the User's Request	17
10	System Performance Comparison of Attribute-Value List Pruning (Pruning) and Partition Recombination (Recombination). In both cases, the maximum number of active partitions was 300.	17
11	Influence of Pruning on System's Performance	18

List of Tables

1	Ontology Rules	6
2	Different representations of the same atomic node in a partition	7
3	Dialogue with Negations in System's Response	11
4	Dialogue with Disjunctions in User's Request	11
5	Dialogue with Conjunctions in User's Request	11
6	CamInfo Ontology Rules	20
7	A typical long negotiative dialogue (turns 1-4)	22
8	A typical long negotiative dialogue (cont., turns 5-8)	23
9	A typical long negotiative dialogue (cont., turns 9-12)	24
10	A typical long negotiative dialogue (cont., turns 13-17)	25

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 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) [1]. 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 [8, 4]. This approach requires that a distribution over all dialogue states, *the belief state*, is maintained at each dialogue turn.

According to [7], 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 dialogue manager to be built for a real-world task. The Bayesian Update of Dialogue State (BUDS) model [4], 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 [8], 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 [6] 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 expressibility via the notion of complements. Similar to [6], 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 [5]. 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 4 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 5 and evaluation results are given in Section 6. Finally, conclusions are given in Section 7.

2 POMDP Dialogue models

Modelling dialogue as a POMDP crucially depends on the information that is included in the dialogue state. On one hand, it has to be sufficient to allow the state to be Markovian, and, on the other hand, it has to enable tractable belief monitoring and policy learning. A sufficient representation of state can be achieved by combining the user goal, the dialogue history and the last user action. However additional approximations are needed to achieve tractability.

The Bayesian Update of Dialogue State (BUDS) model [4] assumes that the dialogue state can be decoupled into conditionally independent concepts. In simple slot-filling dialogues where each slot is an independent concept, the information is extracted from the user request and filled in each slot. The problem with this approach arises when there are errors in the user input in which case correctly filled slots are not able to contribute to resolving ambiguities in related slots, since they are considered independent. In order to overcome this, the BUDS model represents the state as a set of concepts in the form of a Bayesian network that incorporates some of the dependencies between the concepts. Not having all the concepts dependent allows for a prior distribution to be defined on each concept separately and a posterior distribution to be tractably maintained.

The Hidden Information State (HIS) model [8], in contrast, maintains the full joint distribution without making any independence assumptions. For example, if the user said “*I want a cheap hotel*” there would be a joint probability for *cheap* and *hotel*. If later in the dialogue, the system obtains just “*The cheap one*” as the user input, the probability for both *cheap* and *hotel* would be increased, which gives more chance that the system would give the correct answer to the user request. Thus, this approach enables both errors in the input as well as anaphoric user requests to be handled. Tractability in the HIS model is achieved by grouping user goals into *partitions* and assuming that each goal in a partition is equally likely. The recombination of low probability partitions can further improve computational efficiency at the cost of a small loss in accuracy [6]. In this paper we explore another method for achieving tractability in the HIS framework, showing additional expressiveness as well as improved performance of the system.

3 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.

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 [8]. 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 the description of the action-selection process.

3.1 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

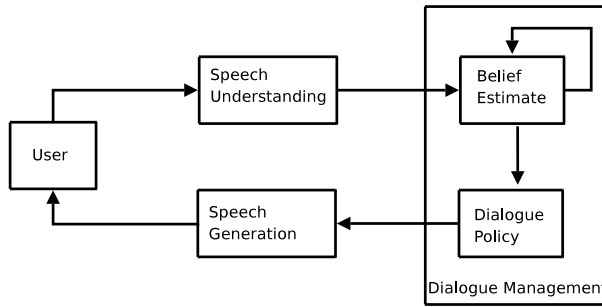


Figure 1: Statistical Spoken Dialogue System Structure

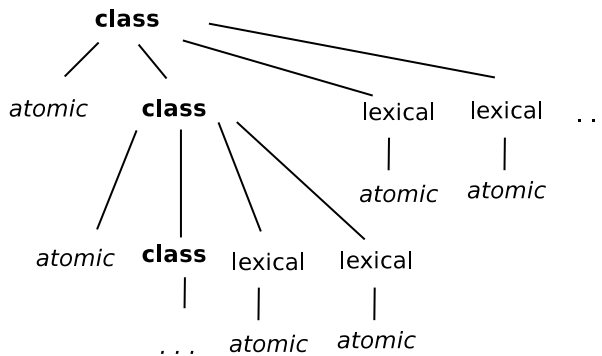


Figure 2: Generic Ontology Structure

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 1, 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 1, 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 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**. A description of a real-world tourist information ontology is given in Appendix 8.1.

¹Note 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	←	<i>venue</i> (type , area, name, addr, near, phone, comment)
type	←	<i>hotel</i> (pricerange, stars, price, drinks)
type	←	<i>restaurant</i> (food, pricerange, price, music, drinks, stars)
type	←	<i>bar</i> (drinks, music, pricerange)
type	←	<i>amenity</i>
area	=	{ <i>central, east, west, ...</i> }
food	=	{ <i>Italian, Chinese, Indian, ...</i> }
...		

Table 1: Ontology Rules

3.2 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, the associated user act and dialogue history forms a *hypothesis*, *i.e.*, a single member of the partitioned state space. 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 [8]. A problem arises, however, in negotiative 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. 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 [6] that partition recombination can overcome this problem with the notion of complements, which we further explore in this paper.

3.3 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:

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

h'
 h

new hypothesis
old hypothesis

Representation	Atomic node for Lexical node <i>food</i>						
Set	<i>Chinese</i>	<i>English</i>	<i>Indian</i>	<i>Italian</i>	<i>Japanese</i>	<i>French</i>	<i>Thai</i>
	T	F	T	F	F	F	F
Disjunctions	<i>Chinese</i> \vee <i>Indian</i>						
Conjunctions	\neg <i>English</i> \wedge \neg <i>Italian</i> \wedge \neg <i>Japanese</i> \wedge \neg <i>French</i> \wedge \neg <i>Thai</i>						

Table 2: Different representations of the same atomic node in a partition

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 in 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 model is deterministic and it determines whether updating a particular dialogue history results in a consistent hypothesis. A more detailed explanation is given in [8].

3.4 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 [2, 8] 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.

4 Extended State Representation

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

4.1 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*. Atomic nodes, on the other hand, are represented as a set of boolean indicators for each plausible value from the ontology. Alternatively, an atomic node can be represented 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 2. A potential drawback of the set representation is the enumeration of every single value that a slot can take, which can become cumbersome if the number of values is very large. The logical representation, either in the form of conjunctions or disjunctions, is more compact.

4.2 Partitioning Process

Partitioning is applying a slot 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 slot s can take. During the partitioning process, the value v in that atomic node of the partition p is set to false. The partition c is a copy of the partition p where v is set to true.

In order to apply slot-value pair $s = v$ for partitioning, it has to be ensured that there is a partition that contains node s . For slot s , the list of *superiors* is defined as all slot-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 slot-value pair `food=Italian` the list of superiors is `type=restaurant, entity=venue`, see Table 1. The ontology automatically generates this list for each slot 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.

The partitioning process starts by applying the list of slot-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 was each partition created. Slot-value pairs from the system act are also used for partitioning.

It is important to note that this process guarantees that each partition that is created is unique. This is achieved by checking if a partition contains v set to false before $s = v$ is applied to that partition. If it does contain it that means that $s = v$ has already been used and cannot be applied again to that partition.

A step-by-step example of partitioning process is given in Fig 4. The final tree of partitions represents the partitions that are created from the following slot-value pairs: `entity=venue, type=restaurant, area=central, food=Italian` and `pricerange=cheap`. The ontology from Table 1 is used to determine the valid combinations. Therefore, there is no combination that involves `type!=restaurant` and `food=Italian`, as lexical node `food` is only specific to class node `type` in which atomic child value `restaurant` is set to true.

4.3 Logical Expressions for Negotiative dialogue

The explicit representation of complements in partitions, improves the model in a number of ways. Firstly, it enables an easier error recovery. For example, if a slot-value pair $s = v$ occurred in the N-best 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 what exactly the user wants for slot s , the knowledge that user does not want v is explicitly represented so 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 one: *Charlie Chan is a Chinese restaurant in the centre*, which in the semantic representation is `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-world dialogue it is expected that user might want to have more options, so user can ask *Do you have anything else?*, which on semantic level is `reqalts()`. Based on this, the user action model increases the probability of partitions which have `name!=Charlie Chan` and decreases the probability of partitions that have `name=Charlie Chan`.

4.3.1 Quantifiers in the System's Response

Utilising the notion for complements, the system can provide a more accurate response to the user. Referring back to the example from the previous section, if the user wanted something else than Charlie Chan, it may turn out that the partition with `name!=Charlie Chan, type=restaurant,`

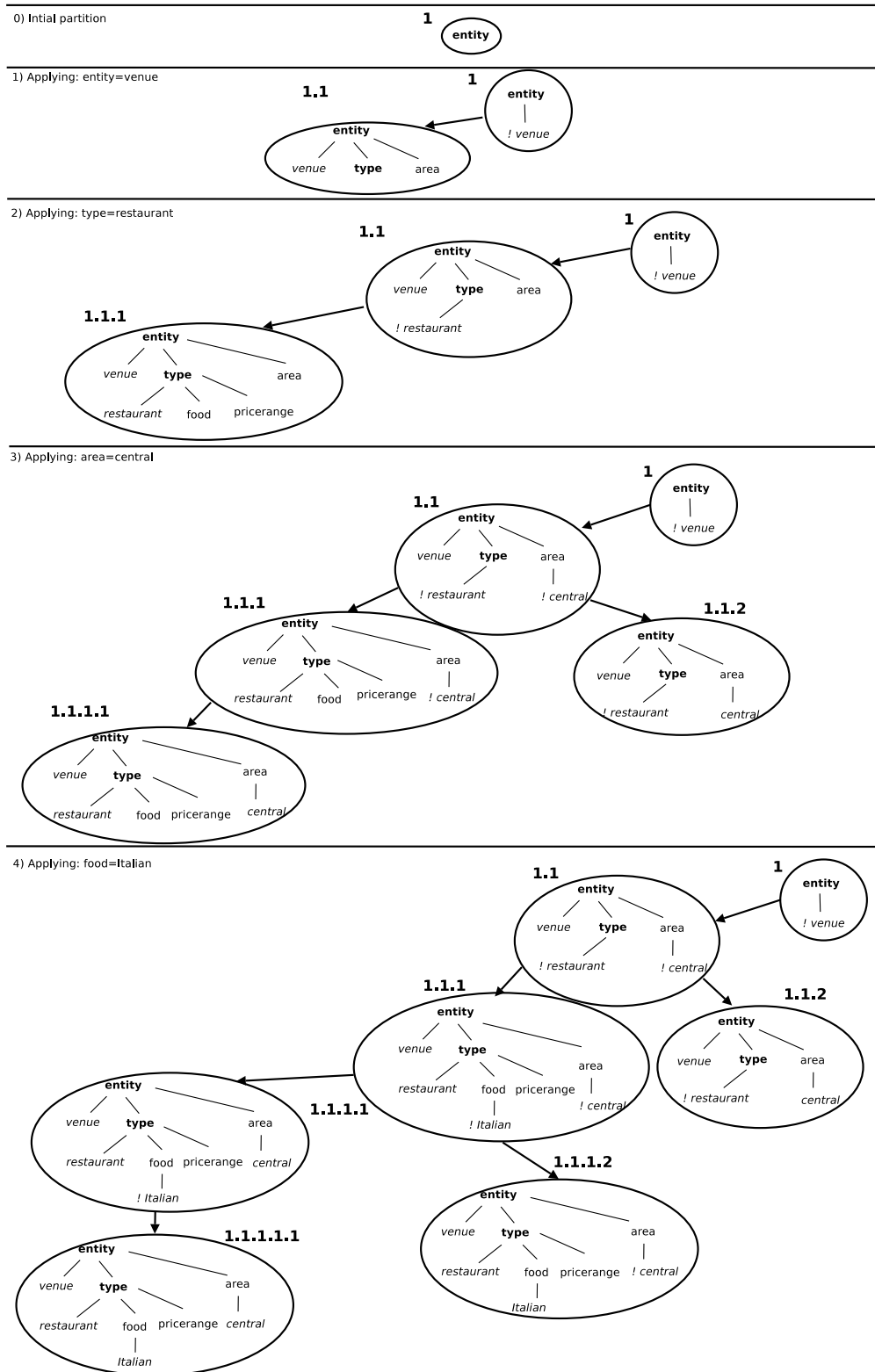


Figure 3: Step-by-step Partitioning Process

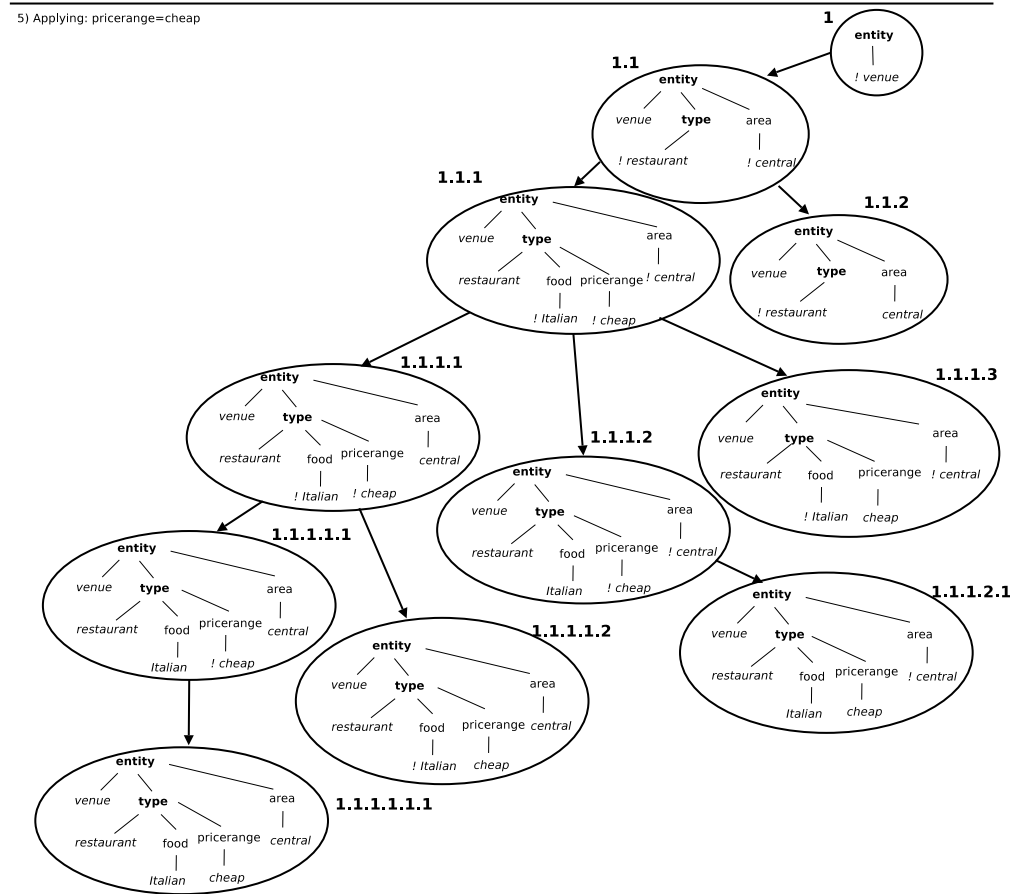


Figure 4: Step-by-step Partitioning Process (cont.)

food=Chinese and area=central does not have any matching entries in the database. Then the semantic output of the system is `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 8.1) that utilises such expressions is given in Table 3.

During the course of dialogue, the system, induced by the recognition errors, 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 outputs *No, I don't want central area, I want ...* which on semantic level could be `deny(area=central)` or `inform(area!=central)`, but the actual value for the slot `area` might not be obtained. 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 than trying to find out the exact value of slot `area` that the user wants. The resulting dialogue act can 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 in a more natural form ***All** Chinese restaurants are in the centre*.

These two examples demonstrate the capability of expressing existential and universal quantifier 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 direct the user.

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 3: Dialogue with Negations in System's Response

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 4: Dialogue with Disjunctions in User's Request

4.3.2 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 only considered the case when the user has in mind one specific value for each slot. Then, multiple values in partition come from the system's confusions. However, this representation allows a richer expression of the user's constrains. For example, if the user wants a restaurant that servers Thai or Indian food `inform(type=restaurant, food=Thai∨Indian)`, partitions can directly represent this. Therefore, the system can handle such request. An example of Cambridge tourist information dialogue where the user makes use of disjunctions is given in Table 4.

Since we assume that an 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, user constraint *I want a hotel, but not with four or five stars* is, on semantic level, `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 a Cambridge tourist information dialogue in Table 5.

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 5: Dialogue with Conjunctions in User's Request

5 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 800 dialogues in interaction with a simulated user in noise free conditions shows the exponential nature of the growth of the number of partitions, see Fig. 5. Note that there are only 11 example dialogues with 20 turns, which accounts for an inconsistent number of partitions for 20 turn length.

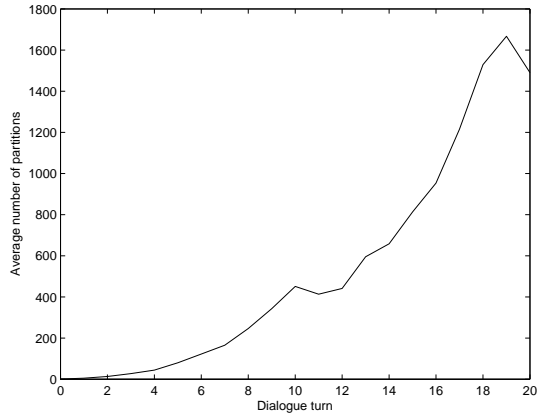


Figure 5: Average number of partitions for dialogue turn

The complexity issue becomes more apparent if the length of the N-best input is large. Constraining the N-best list to be small and setting a maximum number of dialogue turns can be very limiting for real-world dialogues. Therefore, a pruning technique is needed to deal with this.

5.1 Partition Recombination

The number of partitions can be reduced simply by removing the low probability partitions. The hypotheses, the probability of which is maintained through-out the dialogue, are formed as a combination of a partition, the last user action and the respective dialogue history, see Section 3.3. Therefore, each partition has a number of associated hypotheses, so the probability of a partition is a sum of the probabilities of each of its associated hypotheses $b(p) = \sum_{h \in p} b(h)$. This allows for the low probability partitions to be removed. However, since the partitions represent the groups of user goals, completely removing a user goal makes it impossible to recreate it, which is not desirable.

Rather than removing the partitions, the method proposed in [6] reduces the number of partitions by recombining the low probability leaf partitions with their parent partitions. 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 the partition recombination is given in Fig. 6. In each dialogue turn the partitioning is performed using the slot-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 slot-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 the domains that do not have many slots [6]. However, there are some considerations in more complex domains. Firstly, it may be limiting to allow a

```

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 slot-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 slot-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

```

Figure 6: Belief Update with Recombination

partition to be recombined only with its parent, since there may be other partitions it is complementary to. 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. In dialogues where the user goal evolves during the time, the partitions that are created early on are become less probable as the dialogue progresses, whereas the leaf partitions are more useful. If one amends the recombination technique to allow for non-leaf partitions to be recombined 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, by 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.

5.2 Pruning of the Applied Slot-value List

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 than recombining the partitions, the number of partitions can be reduced by removing some of the applied slot-value pairs. The marginal probability of slot-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 slot-value pairs can be obtained. The lowest probability slot-value pairs probably have the least impact on modelling the user goal and therefore can be removed.

Let $s = v$ be a slot-value with the lowest probability in the list of applied slot-value pairs. Assume that $s = v$ is not among superiors for any other applied slot-value pair $s_k = v_k$ (see Subsection 4.1). To remove $s = v$, the following procedure is taken. Starting from the root partition, it is examined if the partition 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 pair of partitions is guaranteed to exist, since if $s = v$ is applied to a partition and its child is created, that partition contains $\neg v$ and the child contains v . Since *upper* is the parent of *lower*, they are complementary and only differ in the atom node that contains v . What is more, if partitions *upper* and *lower* have child partitions there are subtrees of same structure with these partitions as 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 *upper* partition and to remove the *lower* subtree. The procedure is continued until there is no partition matching $\neg v$ left. This can be easily performed using the stack structure. The pseudo code is given in the Appendix 8.2.

An example of the pruning procedure is given in Fig. 7, where slot-value pair `food=Italian` is removed from the list of applied slot-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 removed and `!Italian` are removed from their complementary partitions.

If $s = v$ is among superiors of some slot-value pair $s_k = v_k$, then $s_k = v_k$ has to be pruned before $s = v$ can be pruned. Referring to the example from Fig 4, before removing `type=restaurant`, `food=Italian` and `area=central` have to be removed first. If the pruning is performed based on the lowest probability, it is never be 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 1. In that case, if `type=hotel` is to be removed, `pricerange=cheap` has to be removed only from the partitions that have `hotel`. This is performed by removing $s_k = v_k$ given $s = v$. The algorithm for removing $s_k = v_k$ given $s = v$ 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.

In this way, it is guaranteed that the lowest probability slot-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 a slot-value pair, the structure of the tree of partitions is the same as if that slot-value pair was not applied at all, so the pruning does not affect the existence of complements.

The partitioning exponentially increases the number of partitions, however, this pruning technique exponentially decreases it, so there is no danger that the number of partition grows faster than being reduced. This allows dialogues of arbitrary length. What is more it also enables large N-best inputs to be applied.

An outline of the belief update algorithm that utilises the pruning method is given in Fig. 8. In contrast to the algorithm in Fig. 6, the 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. At the beginning of each dialogue turn a marginal probability over applied slot-value pairs is calculated (lines 6-8). Then, the pruning of the lowest probability slot-value pairs is performed (line 9). Following that, the partitioning is performed using the slot-value pairs from the last system action (line 10). Then, for each observation from the N-best list of the user input the partitioning is performed using its slot-value pairs and the belief is updated (lines 11-18). Finally, the next action is chosen based on the updated belief (line 19).

In the the Appendix 8.3 an example of a long negotiative dialogue which incorporates this pruning technique is given together with the list of active slot-value pairs and their marginal probabilities in each turn.

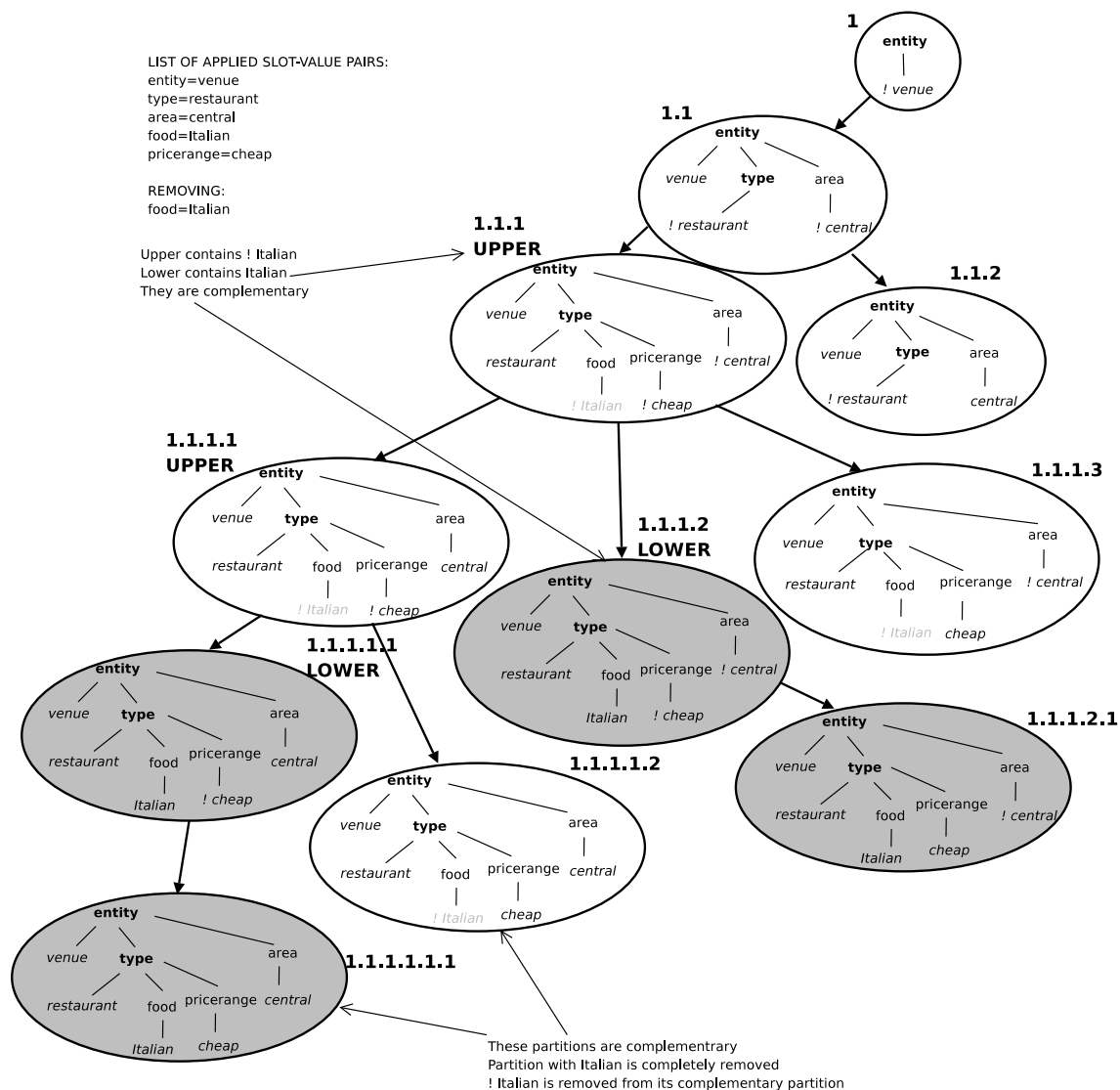


Figure 7: Pruning a slot-value pair from the tree of partitions

6 Evaluation

The evaluation is divided in three parts. In Section 6.1, we examine how well the system can deal with user goals when the constraints are in the form of disjunctions. Then, in Section 6.2, we compare the performance of pruning in the applied attribute-value list algorithm (described in Section 5.2) and a version of the partition recombination algorithm [6]. Finally, in Section 6.3 different pruning threshold 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 (Appendix 8.1). 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. An error model is used to add confusion to the user input and it produces a scored N-best list of


```

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 a slot-value pair and  $p(d)$  its marginal probability
5: repeat for each dialogue turn
    Pruning
6:   for each applied slot-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 slot-value pairs w.r.t.  $p(d)$ 
    Belief Update
10:  Partition each  $p$  using slot-value pairs from the last system action  $a_m$ 
11:  for each  $o'$  in the N-best list do
12:    Partition each  $p$  using slot-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

```

Figure 8: Belief Update with Pruning

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 [3]. 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 policies were trained using the grid-based Monte Carlo Control algorithm in an incremental noise setting [8]. The resulting policies were evaluated with the simulated user performing 2500 dialogues at each error rate.

6.1 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. 9². 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.

6.2 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 10 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

²The error bars represent a 95% confidence interval.

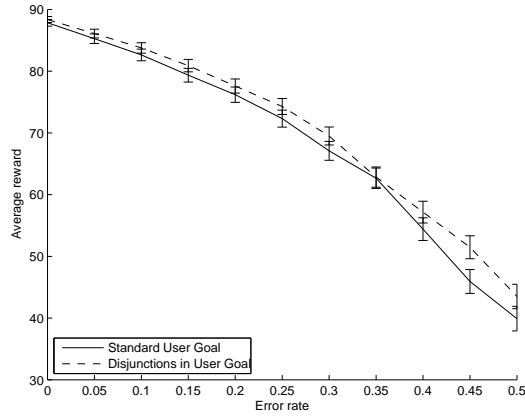


Figure 9: System Performance with Disjunctions in the User's Request

regions, suggesting that it can more effectively manage user goal partitioning in noisy complex domains compared to the simpler partition recombination approach.

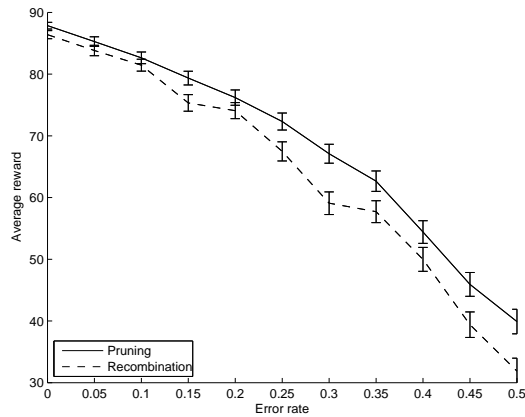


Figure 10: System Performance Comparison of Attribute-Value List Pruning (Pruning) and Partition Recombination (Recombination). In both cases, the maximum number of active partitions was 300.

6.3 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 it's goal during the dialogue. The comparison is given in Fig. 6.3.

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 [6]. 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. This suggests 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 than the threshold of 300, see Fig. 11(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.

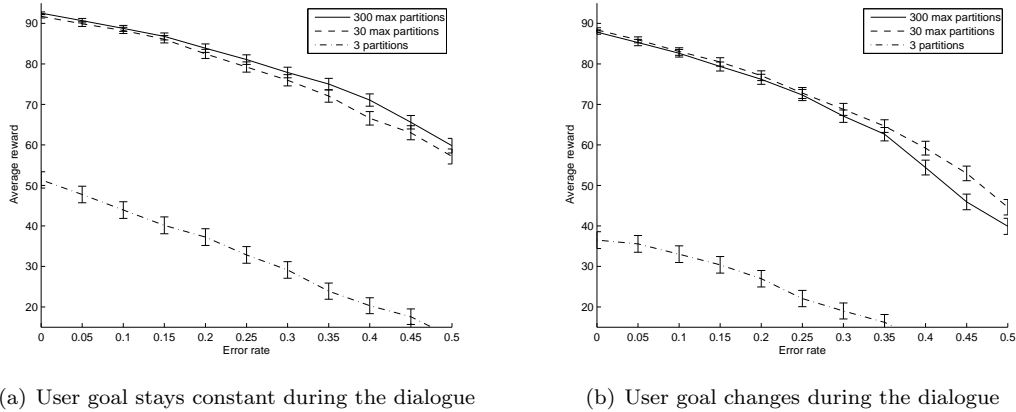


Figure 11: Influence of the pruning threshold on system’s performance when user has a constant goal (a) and when user changes the goal (b)

7 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. 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).

References

- [1] E Levin, R Pieraccini, and W Eckert. Using Markov Decision Processes for Learning Dialogue Strategies. In *Int Conf Acoustics, Speech and Signal Processing*, Seattle, USA, 1998.
- [2] RS Sutton and AG Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, Mass, 1998.

- [3] B Thomson. *Statistical methods for spoken dialogue management*. PhD thesis, University of Cambridge, 2009.
- [4] B Thomson and SJ Young. Bayesian update of dialogue state: A POMDP framework for spoken dialogue systems. *Computer Speech and Language*, (In press), 2010.
- [5] B. Thomson, K. Yu, M. Gašić, S. Keizer, F. Mairesse, J. Schatzmann, and S. Young. Evaluating semantic-level confidence scores with multiple hypotheses. In *Interspeech*, Brisbane, Australia, 2008.
- [6] JD Williams. Incremental Partition Recombination for Efficient Tracking of Multiple Dialogue States. In *Int Conf Acoustics Speech and Signal Processing ICASSP*, Texas, 2010.
- [7] 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.
- [8] SJ Young, M Gašić, S Keizer, F Mairesse, J Schatzmann, B Thomson, and K Yu. The Hidden Information State Model: a practical framework for POMDP-based spoken dialogue management. *Computer Speech and Language*, 24(2)(150-174), 2010.

8 Appendix

8.1 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 6.

entity	←	<i>venue</i> (type , name, area, near, addr, phone, postcode)
type	←	<i>placetostay</i> (staytype , hasinternet, hasparking, price, pricerange, stars)
type	←	<i>placetoeat</i> (eatype , pricerange, openhours, price)
type	←	<i>placetodrink</i> (drinktype , pricerange, openhours, price)
type	←	<i>placetosee</i> (seetype , pricerange, openhours)
type	←	<i>entsvenue</i> (entstype)
type	←	<i>univenue</i> (unitype , openhours)
type	←	<i>sportsvenue</i> (sport)
type	←	<i>transvenue</i> (transtype)
type	←	<i>shopvenue</i> (shoptype , openhours)
type	←	<i>amenity</i> (amtype)
amtype	←	<i>hospital</i> ()
amtype	←	<i>policestation</i> ()
amtype	←	<i>bank</i> (openhours)
amtype	←	<i>postoffice</i> (openhours)
amtype	←	<i>touristinfo</i> (openhours)
shoptype	←	<i>supermarket</i> ()
shoptype	←	<i>shoppingcentre</i> ()0.1
transtype	←	<i>airport</i> ()
transtype	←	<i>busstation</i> ()
transtype	←	<i>trainstation</i> (openhours)
staytype	←	<i>guesthouse</i> ()
staytype	←	<i>hotel</i> ()
eatype	←	<i>restaurant</i> (food)
drinktype	←	<i>bar</i> (childrenallowed, hasinternet, hasmusic, hastv, openhours, price)
drinktype	←	<i>coffeeshop</i> ()
drinktype	←	<i>pub</i> (childrenallowed, hasfood, hasinternet, hastv)
seetype	←	<i>architecture</i> ()
seetype	←	<i>museum</i> ()
seetype	←	<i>park</i> ()
unitype	←	<i>college</i> ()
unitype	←	<i>department</i> ()
unitype	←	<i>library</i> ()
entstype	←	<i>cinema</i> ()
entstype	←	<i>theatre</i> ()
entstype	←	<i>nightclub</i> (openhours, price, pricerange)
entstype	←	<i>entertainment</i> ()
entstype	←	<i>boat</i> ()
entstype	←	<i>concerthall</i> ()
food	=	{ <i>American, Cafe food, Chinese, ...</i> }
pricerange	=	{ <i>free, cheap, moderate, ...</i> }
sport	=	{ <i>badmintoncourt, cricketfield, footballfield, ...</i> }
area	=	{ <i>girton, kingshedges, arbury, ...</i> }
...		

Table 6: CamInfo Ontology Rules

8.2 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;
    }
}
}
```

8.3 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 inputs³. 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 System prompt True user action N-best input Top hypothesis prb Applied attributes	hello() Thank you for calling the Cambridge Information system. How may I help you? HI I’M LOOKING FOR A PLACE TO STAY hello(type=placetostay,task=find) 1.00 hello(type=placetostay) 0.00 hello(type=placetostay,name=“The Place”,task=find) 0.00 0.676 task=find 0.997; entity=venue 0.994; type=placetostay 0.987; name=The Place 0.312
2	System act System prompt True user action N-best input Top hypothesis prb Applied attributes	confreq(type=placetostay,near) You are looking for a place to stay. Are you looking for somewhere near a particular venue? MAYBE SOMEWHERE IN THE CENTRE inform(area=citycentre) 1.00 0.900 task=find 0.999; entity=venue 0.998; area=citycentre 0.972 type=placetostay 0.932; name=The Place 0.012
3	System act System prompt True user action N-best input Top hypothesis prb Applied attributes	inform(name=“Gonville Hotel”,type=placetostay,area=citycentre) The Gonville Hotel is a great place to stay and it’s in the central part of town. IS IT EXPENSIVE confirm(pricerange=expensive) 1.00 inform(pricerange=expensive) 0.00 0.456 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
4	System act System prompt True user action N-best input Top hypothesis prb Applied attributes	inform(name=“Gonville Hotel”,pricerange=none) I am sorry but there is no information about the price range for the Gonville Hotel. DO YOU HAVE ANY CHEAP HOTELS inform(pricerange=cheap,=hotel) 0.96 inform(=hotel) 0.04 0.290 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

Table 7: A typical long negotiative dialogue (turns 1-4)

³The confidence scores are here rounded to two decimals, but in fact they are always greater than zero.

5	System act System prompt True user action N-best input Top hypothesis prb Applied attributes	inform(name=none,type=placetostay,area=citycentre,pricerange!=dontcare) There is no place to stay in the central area for which I have any information about the price range. HOW ABOUT RIVERSIDE reqalts(area=riverside) 0.90 inform(area=riverside) 0.05 reqalts() 0.04 reqalts(=bar,area=riverside) 0.01 null() 0.00 0.213 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 System prompt True user action N-best input Top hypothesis prb Applied attributes	inform(name=none,type=placetostay,pricerange=cheap) There is no place to stay in the cheap price range. HOW ABOUT SOMETHING MODERATE reqalts(pricerange=moderate) 0.83 inform(pricerange=moderate) 0.17 0.065 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 System prompt True user action N-best input Top hypothesis prb Applied attributes	inform(name=none,type=placetostay,staytype=hotel,area=riverside,pricerange=moderate) There is no hotel in the riverside area in the moderate price range. I DON'T CARE ABOUT THE AREA inform(=dontcare) 1.00 null() 0.00 0.116 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
8	System act System prompt True user action N-best input Top hypothesis prb Applied attributes	inform(name="The Eagle",type=placetodrink,drinktype=bar,area=citycentre,pricerange=none) 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. I'M LOOKING FOR A PLACE TO STAY 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 0.080 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

Table 8: A typical long negotiative dialogue (cont., turns 5-8)

9	System act System prompt True user action N-best input Top hypothesis prb Applied attributes	inform(name=none,type=placetostay,staytype=hotel,area=riverside,pricerange=moderate) There is no hotel in the riverside area in the moderate price range. I DON'T CARE ABOUT THE AREA inform(=dontcare) 0.57 null() 0.28 nform() 0.15 7 attribute-value pairs are pruned 0.069 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 System prompt True user action N-best input Top hypothesis prb Applied attributes	inform(name="City Centre North B and B",type=placetostay,staytype=guesthouse, area=arbury,pricerange=none) 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. DO YOU HAVE ANY MODERATE RESTAURANT UH HOTEL inform(pricerange=moderate,=restaurant,=hotel) 0.91 inform(=restaurant,=hotel) 0.05 inform(pricerange=moderate,=restaurant) 0.04 0.023 task=find 0.999; entity=venue 0.998; type=placetostay 0.743 pricerange=dontcare 0.399; pricerange=moderate 0.362; name=City Centre North B and B 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
11	System act System prompt True user action N-best input Top hypothesis prb Applied attributes	inform(name="Finches Bed and Breakfast",type=placetostay,area=girton, pricerange=none,staytype=guesthouse) 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. I'M LOOKING FOR A MODERATELY PRICED PLACE TO STAY 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 0.062 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 B and B 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 System prompt True user action N-best input Top hypothesis prb Applied attributes	inform(name="Express by Holiday Inn Cambridge",area=barnwell, type=placetostay,hasinternet=true,hasparking=true,stars="2") 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. WHAT'S THE PRICERANGE inform(=restaurant) 0.89 null() 0.11 5 attribute-value pairs are pruned 0.013 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 Cambridge 0.327; staytype=guesthouse 0.279 staytype=hotel 0.251; name=City Centre North B and B 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

Table 9: A typical long negotiative dialogue (cont., turns 9-12)

13	System act System prompt True user action N-best input Top hypothesis prb Applied attributes	inform(name="Riverside Restaurant and Granta Bar",area=riverside,eatype=restaurant, type=placetoeat,food=Traditional,pricerange=expensive,near="Queens' College") 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. I'M LOOKING inform(task=find) 0.50 null() 0.50 8 attribute-value pairs are pruned 0.008 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 Cambridge 0.170 name=Riverside Restaurant and Granta Bar 0.168; type=placetoeat 0.146 eatype=restaurant 0.095; food=Traditional 0.047
14	System act System prompt True user action N-best input Top hypothesis prb Applied attributes	inform(name="The Cricketers",area=riverside,near="Ruskin Gallery") The Cricketers is in the riverside part of town near the Ruskin Gallery. I'M LOOKING FOR A FOR A HOTEL IN THE MODERATE PRICERANGE inform(pricerange=moderate,=hotel,task=find) 0.96 inform(=hotel,pricerange=moderate) 0.04 5 attribute-value pairs are pruned 0.025 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 System prompt True user action N-best input Top hypothesis prb Applied attributes	inform(name="The Cambridge Belfry",type=placetostay,staytype=hotel, pricerange=moderate,area=girton,hasinternet=true,hasparking=true) 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. WHAT'S THE ADDRESS request(addr) 1.00 3 attribute-value pairs are pruned 0.308 task=find 0.999; entity=venue 0.998; type=placetostay 0.990 pricerange=moderate 0.854; staytype=hotel 0.854; name=The Cambridge Belfry 0.79 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 System prompt True user action N-best input Top hypothesis prb Applied attributes	inform(name="The Cambridge Belfry",addr="Back Lane, Cambourne") The Cambridge Belfry is located on Back Lane, Cambourne. AND THE PHONE NUMBER request(phone) 1.00 2 attribute-values are pruned 0.427 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 System prompt True user action	inform(name="The Cambridge Belfry",phone="01954714600") The phone number of The Cambridge Belfry is 01954714600. in the expensive price range near Queens' College, also it serves traditional food. THANKS BYE Hanging up

Table 10: A typical long negotiative dialogue (cont., turns 13-17)