Using Dialogue Acts in dialogue strategy learning: optimising repair strategies

Matthew Frampton

Doctor of Philosophy

Institute for Communicating and Collaborative Systems
School of Informatics
University of Edinburgh

2008

Abstract

A Spoken Dialogue System's (SDS's) dialogue strategy specifies which action it will take depending on its representation of the current dialogue context. Designing it by hand involves anticipating how users will interact with the system, and/or repeated testing and refining, and so can be a difficult, time-consuming task. Since SDSs inevitably make understanding errors, a particularly important issue is how to design "repair strategies", the parts of the dialogue strategy which attempt to get the dialogue "back-on-track" following these errors.

To try to produce better dialogue strategies with less time and effort, previous researchers have modelled a dialogue strategy as a sequential decision problem called a Markov Decision Process (MDP), and then applied Reinforcement Learning (RL) algorithms to example training dialogues to generate dialogue strategies automatically. More recent research has used training dialogues conducted with simulated rather than real users and learned which action to take in all dialogue contexts, (a "full" as opposed to a "partial" dialogue strategy) - simulated users allow more training dialogues to be generated, and the exploration of new dialogue contexts not present in an original dataset. As yet however, limited insight has been provided as to which dialogue contextual features are important to include in the MDP and why. Indeed, a full dialogue strategy has not been learned from training dialogues with a realistic probabilistic user simulation derived from real user data, and then shown to work well with real users.

This thesis investigates the value of adding new linguistically-motivated contextual features to the MDP when using RL to learn full dialogue strategies for SDSs. These new features are recent Dialogue Acts (DAs). DAs indicate the role or intention of an utterance in a dialogue e.g. "provide-information", an utterance being a complete unit of a speaker's speech, often bounded by silence. An accurate probabilistic user simulation learned from real user data is used for generating training dialogues, and the recent DAs are shown to improve performance in testing in simulation and with real users. With real users, performance is also better than other competing learned and hand-crafted strategies. Analysis of the strategies, and further simulation experiments show how the DAs improve performance through better repair strategies. The main findings are expected to apply to SDSs in general - indeed our strategies are learned and tested on real users in different domains, (flight-booking versus tourist information). Comparisons are also made to recent research which focuses on handling understanding errors in SDSs, but which does not use RL or user simulations.

Acknowledgements

I wish to thank my thesis superviser Oliver Lemon for his expertise and guidance. I would also like to thank Kallirroi Georgila for technical assistance, and a number of people who provided me with useful feedback on my research at various stages. These include Steve Renals, Mark Hepple, Mirella Lapata, Hiroshi Shimodaira, Mark Steedman, Diane Litman, Joel Tetreault, and the anonymous reviewers of papers which I submitted. Finally I would also like to thank the people who acted as subjects in my user evaluation experiments.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Matthew Frampton*)

Table of Contents

Chapter 1

Introduction

1.1 Motivation

Spoken Dialogue Systems (SDSs) are human-computer interfaces that enable humans to have spoken dialogues with computers, and can be used either in place of, or as a complement to the traditional Graphical User Interface (GUI). The motivation for SDSs is that since spoken dialogue enables humans to enjoy efficient and productive interactions with one another, given sufficiently advanced technology, the same can be true for human-computer interactions. Figure 1.1 is a graphical representation of a generic SDS and shows the key basic components. One of these components is the Dialogue Manager (DM), the main functions of which are to maintain a representation of the current dialogue context, and based on this representation, to specify which action the system will take next. This is likely to be one or more utterances, (units of speech, often bounded by silence), but might also be some other kind of action such as a database query to retrieve information for the user. The DM's mapping between representations of dialogue contexts and system actions is known as a *dialogue strategy*, and clearly the dialogue strategy should be designed to maximise a dialogue's overall chances of success, however this may be defined. It is dialogue strategy design for SDSs to which this thesis directly relates.

The next section here introduces some of the main concepts and issues which are related to dialogue strategy design and which are particularly relevant to the experimental work of this thesis. In doing so, it tries to give an impression of how designing a dialogue strategy by hand can be a complicated and time-consuming task. This fact has

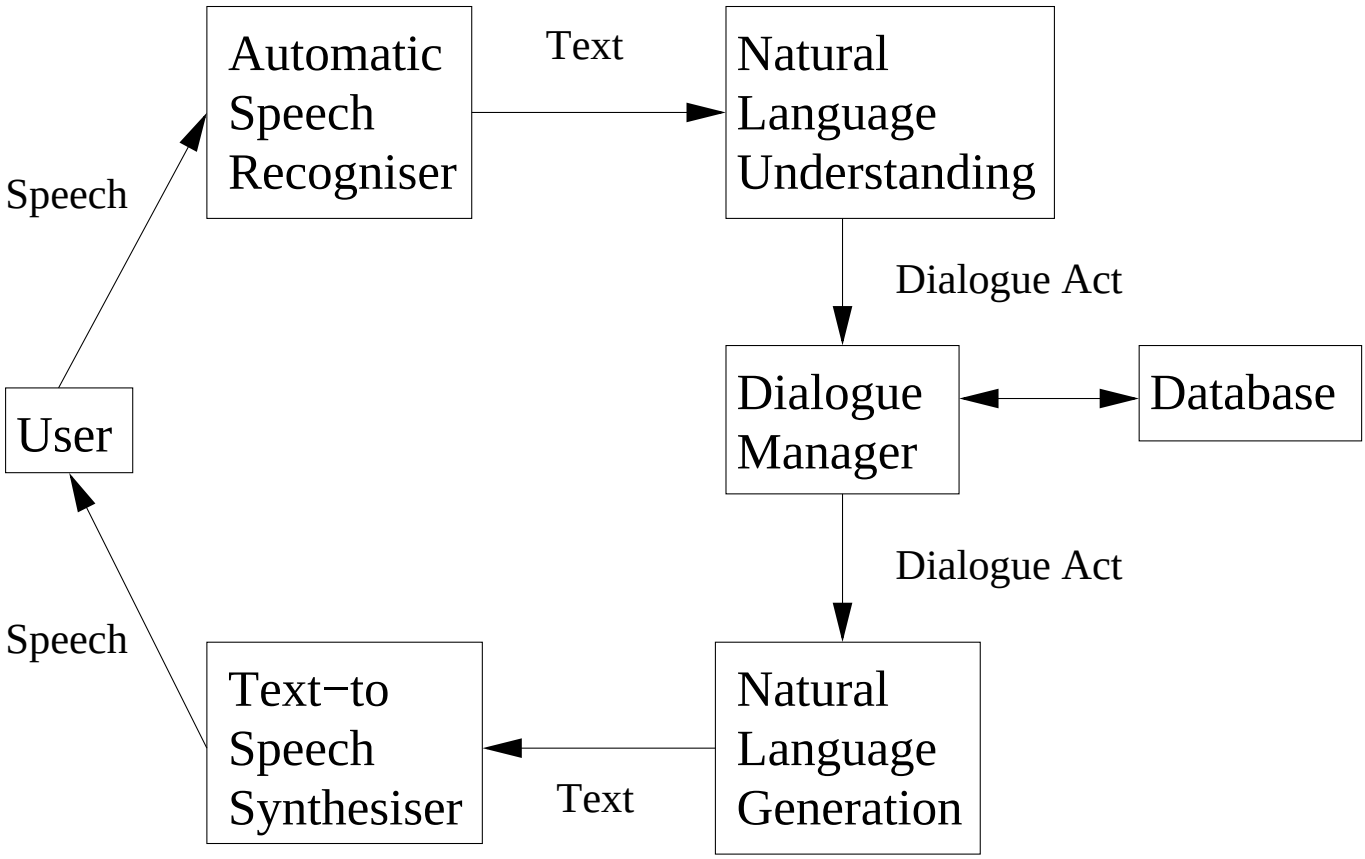

Figure 1.1: The basic architecture of a generic Spoken Dialogue System; The input components, (the Automatic Speech Processing and Natural Language Understanding components), try to interpret user utterances, and the output components, (Natural Language Generation and Text-To-Speech synthesiser), generate utterances; The Dialogue Manager (DM) maintains a representation of the dialogue context and executes a dialogue strategy; A Dialogue Act (DA) is defined in Section 1.2 as a representation of the role/intention of an utterance in a dialogue.

been the spur for a field of research whose goal is to produce better dialogue strategies with less time and effort, and whose basic approach has been to use a particular kind of algorithm to automatically derive dialogue strategies from appropriate datasets. It is this field of research that this thesis seeks to extend, and so Section 1.1.2 goes on to outline the scope of previous research and motivate the experimental work of later chapters.

1.1.1 Concepts and issues involved in designing a dialogue strategy

Designing a dialogue strategy by hand, even for a relatively simple Spoken Dialogue System (SDS), can be a difficult and time-consuming process. The limitations of an SDS mean that human-machine dialogue is quite different in nature to human-human dialogue (Fraser and Gilbert, 1991), and that action choices must be made based on sometimes highly imperfect representations of the dialogue context - representations that potentially include a lot of information, varying amounts of which will be either relevant, inaccurate (e.g. as a result of Automatic Speech Recognition (ASR) errors), or irrelevant. Thus, designers of a dialogue strategy may need to spend a large amount of time anticipating how potential users will interact with the system, and/or on repeated testing and refining. In this section, we use an example dialogue to introduce relevant concepts and terms, and to try to give an impression of the non-trivial nature of dialogue strategy design.

1.1.1.1 Information-slot-filling Spoken Dialogue Systems

Table 1.1 shows a hypothetical dialogue between a user and the "TownInfo" SDS (Lemon et al., 2006b) which is used in an experiment described in this thesis, and whose purpose is to help the user to find local restaurants, bars and hotels. The TownInfo system is an example of an *information-slot-filling* SDS, ("slot-filling" for short), and most of the relevant previous research, and all of the experimental work in this thesis concerns dialogue strategy design for information-slot-filling SDSs. Information-slot-filling SDSs attempt to obtain preferences from the user, (fill slots), and then based on these preferences, (slot-values), they retrieve and present suitable entries from a relevant information source such as a database. Preferences which the TownInfo system asks the user for include *type* e.g. type of food, *price-range* and *location*. Of course, if no options are found, then a sensible dialogue strategy will ask the user to relax one or more of their preferences, while if there are many options, then it will ask for further preferences, (assuming that there are further preferences to ask for).

Turn	System transcript	Turn	User (ASR)
1	Welcome to the TownInfo tourise information system, how may i help you?	2	i want a restaurant
3	Did you say restaurant?	4	yes
5	What kind of food are you looking for?	6	[asr_rejection]
7	Would you like it to be cheap, reasonable or expensive?	8	[out-of-domain]
9	Would you like it to be cheap, reasonable or expensive?	10	reasonably priced
11	A reasonably-priced restaurant, in what area of town?	12	[out-of-domain]
13	I can answer simple requests such as ..., how may I help you?	14	a chinese restaurant
15	A chinese restaurant, in what area of town?	16	no french
17	A french restaurant, in what area of town?	18	central
19	Did you say central?	20	yes
21	Please wait a moment while I find a reasonably-priced central French restaurant. The first of two options ...		

Table 1.1: A hypothetical example dialogue between a tourist information Spoken Dialogue System and a user. The transcript for the user's utterances is that generated by the speech recogniser.

1.1.1.2 Understanding errors in Spoken Dialogue Systems

Due to the limitations of their input components, (the Automatic Speech Recognition (ASR) and Natural Language Understanding (NLU) components), SDSs inevitably make understanding errors. *Non-understandings* are a first broad category of understanding error, and these occur when the system fails to obtain any interpretation for what the user just said, or is not confident enough to choose a specific interpretation. In this thesis, we define a non-understanding to mean that the system failed to obtain a "within-domain" interpretation - an interpretation which means something to the system in question e.g. for the TownInfo system, something relating to the type, price or location of a restaurant/hotel/bar, or one of a small set of simple commands such as "Start over"). Turns 6, 8, and 12 in Table 1.1 are therefore examples of non-understanding errors. In turn 6, there is an *ASR rejection*, which means that either the ASR component could not form a hypothesis for what the user just said with a sufficient degree of confidence to consider it reliable, or that it failed to form any hy-

pothesis at all. By contrast, in turns 8 and 12, the ASR component forms a hypothesis for what the user said, but no within-domain interpretation can be derived from it and so the utterance is considered "out-of-domain". Often in these cases, the original user utterance is in fact within-domain and is considered out-of-domain due to ASR errors (misrecognised words). The second broad category of understanding error is *misunderstandings*. These occur when the SDS forms an interpretation which is not in line with the user's intentions, and for an information-slot-filling SDS, this would mean that the system has obtained an incorrect slot-value. In turn 16 in Table 1.1, following the system's attempt to confirm an incorrect slot-value, the user indicates the misunderstanding error.

1.1.1.3 The non-trivial nature of dialogue strategy design: repair strategies and initiative

To try to give more of an impression of the non-trivial nature of dialogue strategy design for SDSs, we now consider "repair strategies", the parts of a dialogue strategy which try to get the dialogue "back-on-track" following understanding errors. We start with repair strategies for non-understandings.

Following a non-understanding error then, it is often unclear which repair strategy is best to use. For example, should the system repeat the prompt which caused the non-understanding, (turn 9), or is this a bad thing to do because it is likely to frustrate the user and cause another non-understanding? Alternatively, should the SDS switch focus to a different slot, (turn 7), but if so, which one, or should it instead use some kind of "give help" function, (turn 13) in order to inform/remind the user of the system's capabilities/limitations? As the number of possible repair strategies available to the dialogue strategy designer increases, the opportunities to make more appropriate choices for the current dialogue context may also increase, but then so will the complexity of the problem of finding these most appropriate choices. Like all action choices, the decision of which repair strategy to apply must be based on the information in the system's representation of the dialogue context, but what information is important is also likely to be unclear. Again, as the amount of information in the representation increases, the opportunities to make better repair strategy choices may also increase, but so will the complexity of the problem.

Which repair strategies are best to use following different user indications of misun-

derstanding errors is another potential area for uncertainty - for example, in turn 17 it seems sensible for the system to re-confirm the problem slot, (not doing so might disconcert the user), but had the system already failed a number of times to correctly fill this slot, would it instead be better to give up and move on? In order to try to detect misunderstanding errors in the first place, a dialogue strategy may use implicit and explicit confirmation, (turn 11 is an example of the former, and turn 19, of the latter). However, the dialogue strategy designer may not wish the system to over-use confirmation for fear of irritating the user, and so this then presents the problem of deciding when it is necessary to confirm, and when not.

A further important concept in dialogue strategy design which is related to understanding errors and repair strategies, and which will be mentioned again at various points in this thesis, is *initiative*. The concept of "having the initiative" can be loosely defined as having the greater control in directing the dialogue. In turns 5 and 7, the system has the initiative because it asks for a specific piece of information. By contrast, in turns 1 and 13 it asks open-ended questions and so invites the user to supply any within-domain information that they please, thus giving the initiative to the user. Table 1.1 may be referred to as a *mixed-initiative* dialogue, because sometimes the system has the initiative, and sometimes the user. Another problem for the dialogue strategy designer then, is to decide when it is appropriate for the system to take the initiative, and when to give it to the user. For example, if the system asks an open-ended question at the start of the dialogue, then this might often induce users to supply a greater number of preferences in their first turn. This could be a desirable or undesirable outcome depending on whether such user turns are likely to be subject to understanding errors. If they are, then it will probably set the system back in its goal of successfully completing the dialogue, but if they are not, then the opposite is true.

1.1.2 Using Reinforcement Learning in dialogue strategy design and the scope of previous research

Due to the difficult nature of designing dialogue strategies, previous researchers have begun to investigate how *machine learning* can be applied to the problem. Machine learning is a broad subfield of artificial intelligence, and it concerns algorithms and techniques that allow computers to "learn" to perform a task by extracting rules and patterns from (usually large) appropriate datasets. For dialogue strategy design, start-

ing with Levin and Pieraccini (1997), previous researchers have chosen to use a machine learning approach which involves modelling a dialogue strategy as a sequential decision problem called a Markov Decision Process (MDP). An MDP is defined in terms of *states*, *actions* and numerical *rewards*, and in the case of a dialogue strategy, states are system representations of the dialogue context, actions correspond to system actions, and higher rewards are assigned to dialogues which have favourable conclusions e.g. task completion, short length, high user satisfaction. The goal then is to find the action in each state which,on average will lead to the greatest long-term reward. Happily, Reinforcement Learning (RL) algorithms, which work using trial-and-error search, can be applied to example training dialogues in order to try to solve this problem, and in doing so, generate a dialogue strategy automatically.

If we are to use RL to learn which is the best action to take in all possible contexts, ("full" as opposed to "partial" dialogue strategies), then we require a large number of training dialogues. In general, generating a sufficiently large number of training dialogues with real users will be infeasible due to time and possibly cost constraints, and so a user simulation, a predictive user model for simulating user responses (Schatzmann et al., 2006), must be used instead. This user simulation must simulate real users as accurately as possible. If it does not accurately simulate real users, then we risk learning a dialogue strategy that may work well with the user simulation, but not with real users. Ideally, to create an accurate user simulation, an empirical approach should be taken, and this means creating a stochastic user simulation whose probabilities are derived from real user data via *Supervised Learning (SL)* e.g. Georgila et al. (2005a). SL algorithms are machine learning algorithms which generate a function[1] that maps inputs to desired outputs - in the case of a user simulation, inputs are representations of the dialogue context, and outputs are user responses. If taking this approach to create a user simulation requires first collecting new data, then it could become very time-consuming. However, the finding made by Lemon et al. (2006a) that a strategy learned in one information slot-filling domain, (flight-booking), can work well in another, (tourist information) is encouraging. This suggests that at least for information-slot-filling systems, it should be unnecessary to collect new training data for every different domain, and indeed that data from different slot-filling domains can be pooled.

In any case, in using RL to learn full dialogue strategies, only Scheffler and Young

[1] If the output of the function is a continuous value, then it is called *regression*, and if it predicts a class label of the input object, then it is *classification*.

(2002) has used a stochastic user simulation whose probabilities are derived from real user data, but Scheffler and Young's learned strategy was not then tested and shown to work well with real users. Additionally, like other previous research which involved learning full dialogue strategies with a user simulation (e.g. Pietquin and Renals 2002), Scheffler and Young included little contextual information in the RL state - only *slot-status* features (e.g. for each information slot, whether it is filled and/or confirmed), and no linguistically-motivated features. Hence it provides little insight as to what contextual features should be represented in the state for learning a better dialogue strategy and why. Henderson et al. (2005, 2008), where a dialogue strategy was learned from a fixed dataset, is the only example of previous research in which a large amount of contextual information is included in the RL state. Such an amount would have been intractable for standard RL, and so Henderson et al. applied a Hybrid RL/Supervised Learning (SL) approach where the SL component restricted the learned strategy to states for which there was data, and additionally the RL component was augmented with a generalisation technique in order to generalise from observed to unobserved states. However, including all of the available contextual features in the RL state did not provide insights as to which were important and why, and there is no qualitative analysis of the learned strategy. Indeed there is also the question of how well the learning approach really worked - would it be possible to learn as good as, or an even better dialogue strategy with much less information in the state and standard RL?

1.2 Thesis contributions

The work of this thesis concerns the issues described above. The main aims are to identify new contextual features which can improve the learned strategy if included in the RL state, and to understand why they cause improvements. Given what was said above, our basic approach must therefore be to train full dialogue strategies with accurate user simulations, and then to confirm that they work well with real users by testing them on real users. Following a set of preliminary RL experiments, this is what we go on to do. The new contextual features which we choose to focus on are linguistically-motivated, namely recent Dialogue Acts (DAs). A Dialogue Act (DA), alternatively referred to as a *speech act*, or a *dialogue move*, is a concept from semantic and prag-

matic theory of language[2] (Searle, 1969), and is used to indicate the role or intention of an utterance within a dialogue. For example, if we were to define a DA-schema for utterances in TownInfo dialogues, we might represent turn 5 in Table 1.1 with a DA such as "request_food_type", and turn 18 with "provide_food_location". Various DA-schemas have been designed to fit the needs of previous research, and descriptions can be found in the literature e.g. Dialogue Act Markup in Several Layers (DAMSL) (Core and Allen, 1997), Dialogue Act Tagging scheme for Evaluation of SDSs (DATE) (Walker and Passonneau, 2001). Besides investigating the usefulness of recent DAs, other aims of this thesis are to investigate whether our learned strategies outperform a state-of-the-art hand-crafted strategy, and the Hybrid Strategy of Henderson et al. (2008), and if so, to understand why.

The main contributions of this thesis can be summarised as follows:

- A complete literature review of previous research on using RL to learn dialogue strategies is provided.
- Analysis is made of suitable RL parameter settings for learning dialogue strategies, and of the effect on the learned strategy of different reward functions based on task completion and dialogue length.
- When training and testing with different stochastic user simulations whose probabilities are derived from real user data, results show that adding the DAs of the last user and then system turn to the slot-status features already in the state produces significant incremental improvements in the learned strategy. Our learned strategies are also shown to achieve better evaluation scores than the Hybrid Strategy of Henderson et al..
- Results from real user tests show that a strategy trained with one of these user simulations, and with a state containing slot-status and recent DA features, significantly outperforms both a strategy learned with only slot-status features, and a state-of-the-art hand-crafted strategy. Again, our learned strategy also achieves better evaluation scores than the Hybrid Strategy of Henderson et al..
- Analysis is made of the Hybrid Strategy of Henderson et al., and it is compared to our learned strategies to show that it is apparently nowhere near optimal. An

[2]Semantics is loosely defined as the study of meaning in natural language, and pragmatics, as the study of the ability of speakers to communicate more than what is explicitly said.

explanation for why this is the case is provided - this explanation centres on the size of the state-action space used by Henderson et al..

- Analysis of the learned strategies and the real user experiment, and further RL experiments show how the recent DAs are improving the learned strategy:
  - The DAs are only making significant improvements to the learned strategy with respect to better repair strategies in states in which the slot-status features are unchanged, (most often due to non-understanding errors) - they are not producing improvements with respect to dealing with user indications of misunderstanding errors, nor in portions of dialogue in which there is smooth progress i.e. slots are being filled and confirmed. However, we do observe a general trend for the reinforcement learner to learn to maintain focus on the problem slot following user indications of misunderstanding errors.
  - The DAs are important both because they can be used to identify whether the slot-status features are unchanged and hence that a repair strategy is required, and also for then choosing *which* repair strategy to apply. For example, our best learned strategy is more likely to repeat its previous action following an ASR rejection, but to switch focus to a new slot following a user utterance that is recognised as out-of-domain.
  - Choosing an optimal repair strategy is not simply a case of choosing any "sensible" repair strategy which avoids repeating the previous system action. The hypothesis that it was originated from analysis of the real user experiment which showed that both a strategy learned with only slot-status features and the hand-crafted strategy over-used repetition in states in which the slot-status features were unchanged - this irritated the users, causing them to hyperarticulate / adopt an irritated tone, which in turn led to more ASR errors and hence longer dialogues and lower average task completion.
  - Including the DAs of both the last system and user turns produces better repair strategies - taking out the DAs of the last user turn and using only the DA of the last system turn causes a deterioration in performance.
- Our findings are compared to those of relevant previous research on handling non-understanding errors in SDSs - Skantze (2003)/Chapter 4 of Skantze (2007) and Bohus and Rudnicky (2005).

1.3 Thesis overview

This section provides an overview of the thesis. It summarises the contents of each chapter, and where this includes experimental work, it also gives some information about the methodology.

1.3.1 Chapter 2: Markov Decision Processes and Reinforcement Learning

This chapter introduces Markov Decision Processes (MDPs) and Reinforcement Learning (RL) in the context of their application in this thesis.

1.3.2 Chapter 3: Previous research on Reinforcement Learning of dialogue strategies

This chapter summarises and analyses the work of the different research groups who have made significant contributions with regards to using Reinforcement Learning (RL) to learn dialogue strategies for Spoken Dialogue Systems (SDSs). Throughout the presentation, a comparative analysis of the previous research is provided, lessons learned are presented, and further work as carried out in the research described in Chapters 4 to 7 of this thesis is motivated. See also Frampton and Lemon (2008b) for an extended version of the literature review contained in this chapter.

1.3.3 Chapter 4: The Reinforcement Learning setup and proof-of-concept experiments

This chapter first introduces our basic Reinforcement Learning (RL) experimental setup and then goes on to describe three preliminary RL experiments, (see also Frampton and Lemon 2005). In introducing the basic experimental setup, we discuss how best to set the reinforcement learner's parameters for learning dialogue strategies. The experiments here are considered preliminary because they use a stochastic user simulation whose probabilities are not learned from real user data - instead they are set based on intuition and an initial analysis of relevant data. The experiments of this chapter may be preliminary, but nevertheless, they enable us to investigate a number

of relevant issues. These include how and why adding recent Dialogue Acts (DAs) to the state might improve the learned strategy, whether it is possible to scale up to a commercially-realistic number of slots, how different types of reward function affect the learned strategy and the implications of this for designing reward functions for learning dialogue strategies.

1.3.4 Chapter 5: Learning with real user data: n-gram user simulation experiments

This chapter describes two experiments in which slot-filling dialogue strategies are learned and tested with stochastic user simulations whose probabilities are learned via Supervised Learning (SL) from real user data i.e. n-gram models whose probabilities are learned from the COMMUNICATOR data (Walker et al., 2001a) (flight-booking domain). An n-gram model models sequences of n items, (here n dialogue turns represented as Dialogue Acts (DAs), and predicts the next item based on the previous n-1 items.

In the first experiment of this chapter, dialogue strategies are learned for a system with three information slots, and in the second, for a system with four information slots, (see also Frampton and Lemon (2006) for a summary of the four-slot experiment). These experiments investigate whether the learned strategy can be improved if the DAs of the last system and user turns are added to the state. Since the DAs are found to improve the learned strategy, detailed analysis is conducted in order to investigate why. The main reasons for conducting the second 4-slot experiment include to accumulate more evidence for how the recent DAs improve the learned strategy and to show that the RL problem remains tractable with a more commercially-realistic number of slots. Another main reason is to compare performance with the Hybrid RL/SL Strategy of Henderson et al. (2008). A meaningful comparison is possible here because the Hybrid Strategy was tested with a user simulation derived from COMMUNICATOR data which has been shown to produce very similar simulated dialogues to the n-gram simulations (Georgila et al., 2006). Prior to describing the experiments, this chapter also discusses the limitations of the n-gram simulations with respect to accurately simulating real users, and the implications this has for learning dialogue strategies.

1.3.5 Chapter 6: Testing the learned strategies on real users

This chapter describes an experiment in which we implement two of our learned three-slot strategies and a state-of-the-art hand-crafted strategy in a slot-filling Spoken Dialogue System (SDS) and test them on real users. The SDS which we use here - the TownInfo SDS (Lemon et al., 2006b) - operates in the tourist information domain and so we must first transfer our learned strategies to this alternative domain by treating them as generic slot-filling strategies. The first learned strategy tested here was learned with a state containing only the slot-status features, and the second, with additional state features for the Dialogue Acts (DAs) of the last system and user turns. Hence here we are investigating the relative performance of these different strategies in testing with real users. Various evaluation measures, both objective and subjective, are collected, and we provide analysis to explain performance differences between the strategies. Again we are also able to compare results with the Hybrid RL/Supervised Learning (SL) Strategy of Henderson et al. (2008), because a three-slot version of the Hybrid strategy was tested on real users with the same SDS, and the same evaluation measures were collected, (see Lemon et al. 2006a). We provide analysis of the Hybrid strategy itself in order to explain its relative performance, and discuss why the learning approach used by Henderson et al. produced the strategy which it did.

1.3.6 Chapter 7: Investigating the role of Dialogue Acts in learning repair strategies

Based on the results and analysis of the experiments in Chapters 5 and 6, a number of hypotheses were formed as to why the recent DAs improved the learned strategy. In this chapter, these hypotheses are investigated further in four new experiments where new strategies are learned and tested with the n-gram simulations, (see also Frampton and Lemon (2008a) for a summary of these experiments). The use of the n-gram simulations here is justified because of the positive result obtained in the real user experiment of Chapter 6. The hypotheses tested here all relate to repair strategies, and the role of recent DAs in learning more effective repair strategies. Having described these experiments, we then compare our findings with relevant previous research on repair strategies for Spoken Dialogue Systems (SDSs) i.e. Skantze (2003) and Bohus and Rudnicky (2005). This previous research did not involve Reinforcement Learning (RL) or user simulations, and so the experimental methodology is quite different to

that employed in the experimental work of this thesis.

1.3.7 Chapter 8: Summary and conclusions

The final chapter discusses the contributions made in this thesis to the research field of using Reinforcement Learning (RL) to design dialogue strategies, and to dialogue management for Spoken Dialogue Systems (SDSs) in general. It summarises these contributions and assesses their impact. Based on how they extend existing knowledge, their limitations, and the overall state of the field, the chapter also suggests areas for future research.

Chapter 2

Markov Decision Processes and Reinforcement Learning

2.1 Introduction

A Markov Decision Process (MDP) is a *decision-theoretic stochastic planning* model - it provides a mathematical framework for modelling decision-making in situations where outcomes are partly random and partly under the control of the decision-maker, (also known as the agent). Based on an MDP representation of interactions between the agent and its environment, a Reinforcement Learning (RL) algorithm can then be used to try to learn which action the agent should take in different situations in order to achieve a specified long-term goal. In the experimental work of this thesis, the MDP-RL framework is applied to dialogue management. It is used to learn a slot-filling dialogue strategy as a Dialogue Manager (DM) interacts with a stochastic simulation which simulates both a user, and the input components of a slot-filling Spoken Dialogue System (SDS) - the Automatic Speech Recognition (ASR) and Natural Language (NLU) components. Here the agent is the dialogue manager, its goal is to fill and confirm the slots in as few turns as possible, and the environment is the stochastic user/ASR/NLU simulation.

The purpose of this first background chapter then is to introduce MDPs and RL in the context of their application in the experimental work of this thesis. It will cover all of the required concepts and formal definitions, but we start in the next section by providing reasons why the MDP-RL framework is applied to dialogue management.

2.2 Advantages of using Reinforcement Learning for dialogue management

Markov Decision Processes (MDPs) and Reinforcement Learning (RL) are used for designing dialogue strategies because they are thought to be able to produce better strategies with less time and effort than the standard rule-based hand-coding approach. As a statistical learning approach, the MDP-RL framework offers several key potential advantages over rule-based hand-coding (Lemon and Pietquin, 2007):

- data-driven development cycle,
- provably optimal action policies,
- a precise mathematical model for action selection,
- reduced development and deployment costs for industry,
- greater robustness in the face of noise/uncertainty.

Amongst the various statistical learning approaches which might be applied to dialogue management, RL is made very attractive by its two main features. The first of these features is that RL is *planning* so as to maximise long-term reward. Planning is a branch of Artificial Intelligence that concerns the realisation of strategies or action sequences. A sequence of actions which maximises reward in the long-term is exactly what we want from a dialogue strategy - in general, and certainly in task-oriented dialogues, it is only at the end of the dialogue that we can say whether or not the dialogue was successful i.e. the task was successfully completed, the user is satisfied. For example, in a flight-booking dialogue, we cannot say whether the system has provided the user with a suitable flight until the end. The second of RL's main features is its use of trial-and-error search. This is important in learning a dialogue strategy, because there are likely to be a very large number of possible strategies to explore, and so some degree of automated trial-and-error experimentation will be required.

We now start a thorough introduction to MDPs and RL. In the next section, we begin by providing a formal definition of the problem which the MDP-RL framework is used to tackle.

2.3 Defining the problem

The agent and environment interact at each of a sequence of discrete time steps, $t = 0,1,2,3...$ [1]. At each time step t, the agent receives some representation of the environment's state, $s_t \in S$, where S is the set of possible states, and on that basis selects an action, $a_t \in A(s_t)$, where $A(s_t)$ is the set of actions available in state s_t. The mapping from states to probabilities of selecting each possible action is called the agent's *policy* and is denoted π_t, where $\pi_t(s,a)$ is the probability that $a_t = a$ if $s_t = s$. In the case of dialogue management then, states represent the dialogue context [2], actions correspond to system actions, (e.g. generating a particular utterance), and a policy is a dialogue strategy. One time step later, in part as a consequence of its action, the agent receives a real number as a reward, $r_{t+1} \in \Re$, and finds itself in a new state, s_{t+1}. This reward is computed by a function of the state, which we refer to as the *reward function*. The purpose or goal of the agent is formalised in terms of this reward function - in dialogue management, it is designed so that it gives higher total reward to dialogues with favourable outcomes e.g. task completion, short length, high user satisfaction.

Informally then, the problem which we are trying to solve is to find the policy which maximises the total amount of reward which the agent receives i.e. not immediate reward, but cumulative reward in the long run. More specifically, we seek to find the *optimal policy* π^*, which is the policy that maximises the *expected return* for all states. The return, denoted R_t, is some particular function of the reward sequence, denoted $r_{t+1}, r_{t+2}, r_{t+3}, ...$, and since the return is a random variable, the expected return is its population mean.

The return can be defined differently for two different types of task - *episodic* and *continuing*. In an episodic task, the agent-environment interaction naturally breaks down into a sequence of separate episodes. For example, dialogue management is an episodic task in which each dialogue is an episode. In episodic tasks then, the return is simply the sum of rewards:

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + ... + r_T \tag{2.1}$$

[1] We restrict attention to discrete time to keep things as simple as possible, even though many of the ideas can be extended to the continuous-time case (e.g. Bersekas and Tsitsiklis 1996, Werbos 1992, Doya 1996).

[2] The dialogue context is represented by information derived from the Spoken Dialogue System's (SDS's) representation of the dialogue context.

where T is a final time step. By contrast, in a continuing task, the agent-environment interaction does not break down into identifiable episodes, but goes on continually without limit e.g. a robot with a long life-span. The above formulation is problematic for continuing tasks because the final time step would be $T = \infty$, and the return which is what we are trying to maximise could itself easily be infinite. Therefore, we need to use *discounting* in order to determine the present value of future rewards, and so seek to maximise the expected discounted return, rather than the expected return. The expected *discounted return* is:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma r_{t+3} + ... + = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.2}$$

where γ is a parameter, $0 \leq \gamma \leq 1$, called the *discount rate*. If $\gamma < 1$, the infinite sum has a finite value as long as the reward sequence $\{r_k\}$ is bounded. When $\gamma = 0$, the agent is only concerned with maximising immediate rewards i.e. r_{t+1}. As γ approaches 1, it takes future rewards into account more strongly i.e. it becomes more far-sighted. We can then unify the definition of the return for episodic and continuous tasks by writing it as:

$$R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1} \tag{2.3}$$

and including the possibility that $T = \infty$, (i.e. we are dealing with a continuing task), or $\gamma = 1$, (i.e. we are dealing with an episodic task). As stated above, dialogue management is an episodic task, and so we set $\gamma = 1$ in the RL experiments described in this thesis.

Here we have provided a formal definition of the kind of problem to which the MDP-RL framework is applied. We now move on to describe MDPs, and start in the next section by introducing their defining characteristic - the Markov property.

2.4 The Markov Property

The previous section stated that the agent chooses which action to take based on a function of a signal from the environment called the environment's state. Hence, an ideal state signal is one which contains all of the relevant information and nothing

more. A state signal that succeeds in retaining all relevant information is said to be *Markov*, or to have the *Markov property* (Sutton and Barto, 1998).

We now formally define the Markov property for the RL problem. The mathematics can be kept simple if it is assumed that there are a finite number of states and reward values - this enables us to work in terms of sums and probabilities rather than integrals and probability densities, but we could easily extend to include continuous states and rewards if required. When the agent takes an action a at time t, the environment will respond at time $t+1$ with a reward r_{t+1} and a new state s_{t+1}. It is possible that this response depends on everything that has happened earlier, and so the dynamics must be defined in terms of the complete probability distribution. This is shown in Equation 2.4 where the notation $Pr(A|B)$ means "the probability of event A occurring given that event B occurs".

$$Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t, r_t, s_{t-1}, a_{t-1}, ..., r_1, s_0, a_0\} \tag{2.4}$$

However, if the state signal has the Markov property, then this means that the environment's response depends only on the state and action at time t, and hence the environment's dynamics can be defined as:

$$Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t\} \tag{2.5}$$

for all s',r,s_t and a_t. In other words, a state signal has the Markov property, and is a Markov state, if and only if Equation 2.5 is equal to 2.4. Hence, if an environment has the Markov property, then its one-step dynamics enable us to predict the next state and expected reward given the current state and action, and it follows that Markov states provide the best possible basis for choosing actions.

RL algorithms can still be applied when the state signal is non-Markov, but assuming tractability, their performance will improve as the state approaches being Markov. Hence in any problem to which the MDP-RL framework is applied, there is a challenge to identify all or as much as possible of the relevant information required for producing a Markov state signal. By identifying contextual features which when represented in the state enable an RL algorithm to learn an improved dialogue strategy, this thesis can be said to make a contribution towards producing a more Markov MDP state for dialogue management.

Now that we have introduced the defining characteristic of MDPs - the Markov property - we move on in the next section to provide a formal definition of an MDP.

2.5 Markov Decision Processes

A Markov Decision Process (MDP) is a Reinforcement Learning (RL) task that satisfies the Markov property (Sutton and Barto, 1998). If the state and action spaces are finite, then it is called a *finite MDP*. Finite MDPs are particularly important to the theory of RL, and the theory presented in the rest of this chapter implicitly assumes that the environment is a finite MDP. An MDP for an agent is defined by a tuple, $\{S, A, T, R\}$, where S is the set of states that the agent can be in, A is the set of possible actions which the agent can take, T defines a transition probability distribution over the state space (sometimes called the transition matrix), and R is the expected reward distribution. As previously stated, in the case of dialogue management, states represent the dialogue context, and actions correspond to system actions, (e.g. generating a particular utterance, presenting some information to the user). The couple $\{T, R\}$ defines the one-step dynamics of the system:

$$T^a_{ss'} = Pr\{s_{t+1} = s' | s_t = s, a_t = a\} \tag{2.6}$$

$$R^a_{ss'} = E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\} \tag{2.7}$$

where E denotes the expected value, (population mean).

Having defined both the problem to which the MDP-RL framework is applied, and MDPs themselves, we now shift focus to RL. Since almost all RL algorithms are based on estimating *value functions*, value functions are our starting point.

2.6 Value functions

Almost all RL algorithms are based on estimating *value functions* - either functions of states that estimate how good it is for the agent to be in a given state, or functions of state-action pairs that estimate how good it is to perform a given action in a given state. The notion of "how good" here is defined in terms of expected return, and since the rewards that the agent can expect to receive in the future depend on what

actions it will take, value functions are defined with respect to particular policies. In the first subsection here, we provide formal definitions for state and state-action value functions.

2.6.1 Definitions for state and state-action value functions

$V^{\pi}(s)$ denotes the value of an individual state s under a policy π - it is the expected return R when starting in s at time t and following π thereafter. For MDPs, $V^{\pi}(s)$ is defined formally as:

$$V^{\pi}(s) = E_{\pi}\{R_t|s_t = s\} = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t = s\} \tag{2.8}$$

$Q^{\pi}(s,a)$ denotes the expected return when starting in s at time t, taking action a, and following π thereafter. It is defined as:

$$Q^{\pi}(s,a) = E_{\pi}\{R_t|s_t = s, a_t = a\} = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t = s, a_t = a\} \tag{2.9}$$

In the next subsection, we now go on to introduce Bellman equations for value functions. We do this because Bellman equations are fundamental to Dynamic Programming (DP), and although DP is not used in the experimental work of this thesis, it is introduced a little later in this chapter because it provides an essential foundation for the understanding of RL algorithms.

2.6.2 Bellman equations for value functions

Reinforcement Learning (RL) algorithms make use of the fact that value functions satisfy particular recursive relationships. For any policy π and any state s, the consistency condition given below in Equation 2.10 holds between the value of s and the value of its possible successor states. The reader can refer back to Equations 2.6 and 2.7 for the respective definitions of T and R.

$$
\begin{aligned}
V^{\pi}(s) &= E_{\pi}\{R_t|s_t = s\} \\
&= E_{\pi}\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s_t = s\} \\
&= E_{\pi}\{r_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^k r_{t+k+2}|s_t = s\} \\
&= \sum_a \pi(s,a)\sum_{s'} T^a_{ss'}[R^a_{ss'} + \gamma E_{\pi}\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+2}|s_{t+1} = s'\}] \\
&= \sum_a \pi(s,a)\sum_{s'} T^a_{ss'}[R^a_{ss'} + \gamma V^{\pi}(s')] \qquad (2.10)
\end{aligned}
$$

The last line of Equation 2.10 is the *Bellman Equation* for V^{π}, and it tells us the value of a state in terms of the values of its successor states. If we start in state s, then the agent can take any of a set of actions, and this in turn will lead to one of several next states, s', along with a reward r. The Bellman equation averages over all the possibilities, weighting each by its probability of occurring, and tells us that the value at the start state must equal the (discounted) value of the expected next state, plus the reward expected along the way.

In the same way as for the state value function, the state-action value function can also be re-written:

$$
\begin{aligned}
Q^{\pi}(s,a) &= E_{\pi}\{R_t|s_t = s, a_t = a\} \\
&= E_{\pi}\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s_t = s, a_t = a\} \\
&= E_{\pi}\{r_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^k r_{t+k+2}|s_t = s, a_t = a\} \\
&= \sum_{s'\varepsilon S} T^a_{ss'}\{R^a_{ss'} + \gamma E_{\pi}\{\gamma\sum_{k=0}^{\infty}\gamma^k r_{t+k+2}|s_t = s\}\} \\
&= \sum_{s'\varepsilon S} T^a_{ss'}\{R^a_{ss'} + \gamma V^{\pi}(s')\} \qquad (2.11)
\end{aligned}
$$

Above, the last line of Equation 2.11 is the Bellman equation for Q^{π}

We now go on to describe Bellman optimality equations for value functions i.e. Bellman equations for the value function of the optimal policy.

2.6.3 Bellman optimality equations for value functions

Value functions can be used to rank policies. A policy π is defined to be better than or equal to another policy π' if its expected return is greater than or equal to that of π for all states i.e. if and only if $V^{\pi}(s) \geq V^{\pi}(s)$ for all $s \in S$. There is always at least one policy that is better than or equal to all other policies - the *optimal policy*, denoted π^*. Optimal policies share the same state-value function, which is called the optimal value function and is denoted as V^*. V^* is defined as, for all $s \in S$:

$$V^*(s) = max_{\pi} \ V^{\pi}(s) \tag{2.12}$$

where the notation max_x specifies the value of x which produces the greatest value for the expression that follows it. Optimal policies also share the same optimal state-action value function, denoted $Q*$, which is defined as, for all $s \in S$ and $a \in A(s)$, (possible actions in s):

$$Q^*(s,a) = max_{\pi} \ Q^{\pi}(s) \tag{2.13}$$

We can write Q^* in terms of V^* as follows:

$$Q^*(s,a) = E\{r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a\} \tag{2.14}$$

Since V^* is the value function for a policy, it must satisfy the self-consistency condition given by the Bellman equation for state values, but because it is the optimal value function, its consistency condition can be written in a special form without reference to any specific policy. This leads us to the Bellman equation for V^*, or the *Bellman optimality equation*, which intuitively expresses the fact that the value of a state under an optimal policy must equal the expected return for the best action from that state.

$$
\begin{aligned}
V^*(s) &= max_a \; Q^{\pi^*}(s,a) \\
&= max_a \; E_{\pi^*}\{R_t | s_t = s, a_t = a\} \\
&= max_a \; E_{\pi^*}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\} \\
&= max_a \; E\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s, a_t = a\} \\
&= max_a \; E\{r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a\} \qquad (2.15) \\
&= max_a \; \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^*(s')] \qquad (2.16)
\end{aligned}
$$

The last two equations (2.15 and 2.16) are two forms of the Bellman optimality equation for V^*. The Bellman optimality equation for Q^* is given by Equation 2.17:

$$
\begin{aligned}
Q^*(s,a) &= E\{r_{t+1} + \gamma \, max_{a'} \; Q^*(s_{t+1}, a')\} \\
&= \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma \, max_{a'} \; Q^*(s'a')] \qquad (2.17)
\end{aligned}
$$

For finite MDPs, the Bellman optimality equation has a unique solution independent of the policy.

We now go on to describe how to derive the optimal policy from an optimal state or state-action value function.

2.6.4 Deriving the optimal policy from a value function

If we have V^*, then it is relatively easy to determine an optimal policy - for each state, the optimal action or actions are those which lead to the new state with highest value. Of course, to know what the possible new states are, we need to know about the environment's dynamics, (see Section 2.5). The optimal policy is said to be *greedy* with respect to V^*. In computer science, the term greedy describes any search or decision procedure that selects alternatives based only on local or immediate considerations, without taking account of the possibility that such a selection may prevent future access to even better alternatives. However, in the case of V^*, the greedy policy is optimal in the long-term sense because V^* already takes into account the reward consequences of all possible future behaviour. If we have Q^*, then finding the optimal action(s)

in each state is even easier - for any state *s*, the optimal action or actions maximise $Q^*(s,a)$. Since we do not need to know about possible successor states, we do not require knowledge of the environment's dynamics.

Now that we have described Bellman equations, we are in a position to introduce Dynamic Programming (DP). As stated previously, although DP is not used in the experimental work of this thesis, we describe it here because it provides an essential foundation for the understanding of RL algorithms.

2.7 Dynamic Programming

Dynamic Programming (DP) refers to a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as a Markov Decision Process (MDP). They are of limited utility in Reinforcement Learning (RL) both because of their assumption of a perfect model, and because of their great computational expense, but they have been used to learn partial dialogue strategies from a corpus of human-machine dialogues e.g. Singh et al. (1999, 2002), (Section 3.3), Walker (2000), (Section 3.4), and Tetreault and Litman (2006), (Section 3.9). They are important theoretically - other RL methods can be viewed as attempts to achieve much the same as DP, only with less computation and without assuming a perfect model of the environment.

DP involves two interacting processes - *policy evaluation* and *policy improvement*. These are described in the next two subsections respectively. Subsection 2.7.3 then describes how DP uses these two processes in combination in order to learn an optimal policy.

2.7.1 Policy evaluation

Policy evaluation is the process of computing the state-value function V^π for an arbitrary policy π. How then can we do this? If the environment's dynamics are completely known, then the Bellman equation, (Equation 2.10), is a system of $|S|$ simultaneous linear equations in $|S|$ unknowns, ($|S|$ denotes the number of states in *S*), and policy evaluation is achieved by solving this system. One way to find the solution is by using *iterative policy evaluation*, which is an example of an *iterative method* for solving a system of linear equations. Unlike *direct methods*, which attempt to solve the system

in one-shot, iterative methods find successive approximations to the solution starting from an initial guess. An iterative method is preferred for policy evaluation because of the number of variables involved - iterative methods are able to cope with a much larger number, and the number involved in policy evaluation is likely to make direct methods prohibitively expensive.

The first step in iterative policy evaluation is to make an arbitrary choice for an initial approximation of the value function, (N.B. the terminal state, if there is one, must be given the value 0). Note, the first approximation of the value function is denoted V_0, the second V_1, the third V_2 etc. On each iteration, iterative methods generate successive approximations using what is called an *update rule*, and in the case of generating successive approximations of V^π, we can use the Bellman Equation as an update rule. In each iteration then, iterative policy evaluation applies Equation 2.18 to every state in the state space in what is often referred to as a *sweep* through the state space, and thus produces a new approximation of the value function i.e. V_{k+1}.

$$\begin{aligned} V_{k+1}(s) &= E_\pi\{r_{t+1} + \gamma V_k(s_{t+1}) | s_t = s\} \\ &= \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V_k(s')] \end{aligned} \tag{2.18}$$

The sequence V_k can be shown in general to converge to V^π as k tends to infinity ($k \to \infty$), under the same conditions that guarantee the existence of V^π. In practice, iterative policy evaluation must be halted short of this. After each iteration / sweep through the state space, we can find the greatest amount by which the value of any state has changed between this and the last approximation of the value function i.e. $max_{s \in S} \ |V_{k+1}(s) - V_k(s)|$. A typical *stopping criterion* is to then stop when this amount is considered sufficiently small.

We now go on to describe the other process required to learn the optimal policy - *policy improvement*.

2.7.2 Policy improvement

Policy evaluation is necessary for *policy improvement* - the process of finding a better policy. As we shall see, we improve on an original policy by for each state, choosing an action which is greedy with respect to this original policy's value function.

Policy improvement is performed by making use of a general result called the *policy improvement theorem* [3]. The policy improvement theorem tells us that if π and π' are any pair of deterministic policies such that for all $s \in S$,

$$Q^{\pi}(s, \pi'(s)) \geq V^{\pi}(s) \tag{2.19}$$

then the policy π' must be as good as, or better than π i.e. it must obtain greater or equal expected return from all states $s \in S$:

$$V^{\pi'} \geq V^{\pi} \tag{2.20}$$

Consider a scenario then where we take an original policy π and change the action in one state s to produce a new policy π'. If as a result of this single change, strict inequality holds in Equation 2.19, then we will have succeeded in producing an improved policy. We can easily extend from considering such a single change in π, to considering changes at *all* states to *all* possible actions, selecting at each state the action that appears best according to $Q^{\pi}(s,a)$. In other words, we consider the new greedy policy, π', given by Equation 2.21.

$$\begin{aligned} \pi'(s) &= max_a \, Q^{\pi}(s,a) \\ &= max_a \, E\{r_{t+1} + \gamma V^{\pi}(s_{t+1}) | s_t = s, a_t = a\} \\ &= max_a \, \sum_{s} P^a_{ss'} [R^a_{ss'} + \gamma V^{\pi}(s')] \end{aligned} \tag{2.21}$$

This greedy policy π' takes the action that looks the best after one step of lookahead according to V^{π}. By construction, the greedy policy meets the conditions of the policy improvement theorem (Equation 2.19), and so we know that it is as good as, or better than the original policy. Hence we make a new policy which improves on an original policy by choosing actions which are greedy with respect to the value function of the original policy.

Having introduced policy evaluation and policy improvement, we are now ready to explain how Dynamic Programming (DP) finds the optimal policy by combining these two processes under a framework called *Generalized Policy Iteration (GPI)*.

[3] The interested reader can refer to page 95 of Sutton and Barto (1998) to see how the policy improvement theorem is derived.

2.7.3 Generalized Policy Iteration

The term *Generalized Policy Iteration* (GPI) refers to the general idea of letting the policy evaluation and policy improvement processes interact in order to find an optimal policy. If both the evaluation and improvement processes stabilise, then the value function and policy must be optimal. The value function stabilises only when it is consistent with the current policy, and the policy stabilises only when it is greedy with respect to the current value function. Thus both processes stabilise only when a policy has been found that is greedy with respect to its own evaluation function. This implies that the Bellman optimality equation (Equation 2.16) holds, and thus that the policy and the value function are optimal. Almost all Reinforcement Learning (RL) methods are well described as GPI - they all have identifiable policies and value functions, with the policy always being improved with respect to the value function, and the value function always being driven toward the value function for the policy.

Having introduced Dynamic Programming (DP), we are now in a better position to describe RL algorithms which are able to learn from sample returns, and hence which do not require a complete model of the environment's dynamics. However, before doing this, we now provide an overview of the two main learning approaches for using RL to train dialogue strategies, one of which involves DP, and the other, RL algorithms that learn from sample returns. We discuss the relative merits of the two approaches.

2.8 Model-based and simulation-based approaches to learning dialogue strategies

Two different learning approaches can be distinguished in previous research on using RL to train dialogue strategies, and like Schatzmann et al. (2006), we refer to these as the *model-based approach* and the *simulation-based approach*. Here, we will begin by describing the model-based approach.

2.8.1 The model-based approach

As we will see in Chapter 3, the model-based approach was used more in earlier research between 1998-2002 e.g. Singh et al. (1999, 2002), Walker (2000). The model-based approach uses a corpus of dialogues in which state transitions have been logged

in order to estimate the transition probabilities T (see Section 2.5), and so build a complete model of the environment. Parameter estimation can be done based on the relative frequency of occurrence of each transition, (simple *Maximum Likelihood Estimation (MLE)*):

$$T(s',a,s) = \frac{count(s',a,s)}{count(s,a)} \tag{2.22}$$

Dynamic Programming (DP) can then be used to learn the optimal policy, (here dialogue strategy). In practice, DP is only used to learn the optimal action in certain states for which the corpus contains exploratory data e.g. states in which the system must choose whether or not to take the initiative, (see Section 1.1.1.3 for a definition of initiative). For the corpus to be suitable for model-based learning of full dialogue strategies, it should ideally contain exploratory data for all states, and this is very unlikely to be the case. Even if the corpus did contain such data, there is then the issue of DP's high computational expense to contend with. Apart from this, the model-based approach has a number of other significant deficiencies, which include:

1. Available corpora may well not be large enough to reliably estimate transition probabilities for practical systems.

2. When learning from a fixed corpus, the Dialogue Manager (DM) can only use state-action combinations that were explored at the time of the corpus data collection - it cannot try out new strategies since no transition probabilities can be computed for unseen state-action combinations.

3. It is necessary to know the state-space and action set representation in advance so that the corpus can be annotated correspondingly for estimating the state transition probabilities.

We now move on to describe the simulation-based approach to learning dialogue strategies with RL.

2.8.2 The simulation-based approach

The simulation-based approach involves using an RL algorithm which is able to learn from sample returns to learn a dialogue strategy as the Dialogue Manager (DM) interacts with a stochastic user simulation. This approach therefore requires an accurate

user model which generalises to unseen dialogue situations, and for this reason, it is more complex than model-based approaches. As stated in Section 1.1.2, a stochastic user simulation can be produced by training on a dialogue corpus using Supervised Learning (SL). Assuming that the resulting user model is reliable, the simulation-based approach offers the following advantages:

1. The simulated user allows any number of training episodes to be generated so that the learning dialogue manager can exhaustively explore the space of possible strategies, and indeed learn full rather than partial strategies.

2. It enables strategies to be explored which are not in the training data. The learning DM can deviate from known strategies and try out new and potentially better strategies.

3. The system state space and action set do not need to be fixed in advance, because the system is not trained on corpus data. If the given representation turns out to be problematic, then it can be changed and the system re-trained using the simulated user.

As a result of these advantages, in general, simulation-based approaches to learning dialogue strategies have been preferred in more recent research e.g. Pietquin and Renals (2002), Scheffler and Young (2002), Frampton and Lemon (2005), Frampton and Lemon (2006), Frampton and Lemon (2008a). Using SL to produce accurate user models, and how to evaluate them is a an open and active research area (see Schatzmann et al. 2006). We will discuss these issues in the following chapter where we review previous research on using RL to learn dialogue strategies.

We now return to our introduction to RL itself, and shift focus to RL algorithms which are able to learn from sample returns, and hence do not require a complete model of the environment's dynamics. We will describe two different categories of these algorithms - Monte Carlo (MC) methods first, and then Temporal Difference Learning (TDL). However, since RL algorithms that learn from sample returns must use "soft" training policies i.e. training policies which have a non-zero probability of selecting each action in a state, in the next section, we first describe different approaches for generating and using soft training policies in RL.

2.9 Generating and using soft training policies in RL

As stated above, Reinforcement Learning (RL) algorithms that learn from sample experience e.g. *Monte Carlo (MC)* methods (see Section 2.10), and *Temporal Difference Learning (TDL)* methods (see Section 2.11) require "soft" training policies i.e. training policies which have a non-zero probability of selecting each action in a state. In the first subsection here then, we describe two different action selection approaches for producing a soft training policy.

2.9.1 Action selection for soft training policies

Two action selection methods for producing a soft training policy are ε-*greedy* and *softmax* action selection. If using ε-greedy action selection, the parameter ε is set to a value $0 < \varepsilon < 1$, and then for each state, the learning agent has a probability of ε of selecting the action which currently has the highest Q-value. When the learning agent *explores* i.e. it does not select the highest Q-value action, it chooses equally among the other possible actions, and so it is as likely to choose the worst-appearing action as it is to choose the next-to-best. If some actions are clearly better than others, then this will produce redundant exploration and slower learning. The RL experiments of Chapter 4 use ε-greedy action selection.

Softmax action selection can do better in this respect, because it varies the action selection probabilities as a graded function of their corresponding current Q-values. This can be implemented via a *Gibbs* or *Boltzmann* distribution which chooses action a in state s with probability

$$\frac{e^{(Q(s,a)/\tau)}}{\sum_{b \in A(s)} e^{(Q(s,b)/\tau)}} \tag{2.23}$$

where e is the exponential function, and τ is a positive parameter called the *temperature*. If we lower the temperature, then this increases the difference in selection probabilities for actions that differ in their current Q-values. The RL experiments of Chapters 5 and 7 use softmax action selection.

We now go on to describe two alternative frameworks for implementing soft training policies - "on-policy" methods and "off-policy" methods.

2.9.2 On-policy versus off-policy methods

This section will briefly explain the difference between on and off-policy methods and then give the reason for our decision to use an on-policy rather than off-policy algorithm in the Reinforcement Learning (RL) experiments of Chapters 4, 5 and 7.

In explaining on-policy versus off-policy methods, it is useful to refer to a *behaviour* policy and an *estimation* policy. The behaviour policy is the policy which is used to generate the training episodes, while the estimation policy is the resulting learned policy. The difference between on-policy and off-policy methods is that for an on-policy method, one policy acts as both the behaviour and estimation policy, while for an off-policy method, the behaviour and estimation policies are separate. Hence in an on-policy approach, the behaviour policy is necessarily guided by the estimation policy e.g. if we are using ε-greedy action selection with ε set to 0.7, then 70% of the time, the behaviour policy will select the action with the highest Q-value according to the estimation policy. However in an off-policy approach, since the behaviour policy is separate, it does not have to be guided by the estimation policy. As a result, on-policy methods tend to converge faster than off-policy methods. Since our RL experiments were relatively slow to run, this is why we chose to use an on-policy algorithm called Sarsa, (see Section 2.11.2), rather than an off-policy alternative.

Now that we have described how to generate soft training policies, we are ready to introduce a first category of RL algorithms that learn from sample returns - Monte Carlo (MC) methods.

2.10 Monte Carlo Learning

Monte Carlo (MC) learning methods can be used to estimate value functions and find optimal policies without complete knowledge of the environment. Instead, they require only experience - sample sequences of states, actions and rewards from on-line or simulated interaction with an environment, (in the case of dialogue strategy learning, a simulation of a user and the input components of a Spoken Dialogue System (SDS)). MC methods have been used by previous researchers to learn full dialogue strategies e.g. Levin et al. (2000) (Section 3.2), Pietquin and Renals (2002), (Section 3.5), and English and Heeman (2005), (Section 3.7). As with Dynamic Programming (DP), MC methods can be described in terms of policy evaluation and policy improvement.

Policy improvement is performed in the same way as for DP i.e. by choosing actions which are greedy with respect to the value function (see Section 2.7.2). This approach works whether the policy which we are trying to improve is deterministic or soft, and the interested reader can refer to pages 122 - 124 in Sutton and Barto (1998) where equations are presented to show that the policy improvement theorem assures that for any ε-soft policy, π, any ε-greedy policy with respect to Q^{π} is guaranteed to be better than or equal to π. MC methods are instead distinguished by the way in which they perform policy evaluation, and it is this that we focus on in the next subsection.

2.10.1 Policy Evaluation

Monte Carlo (MC) policy evaluation involves estimating the value of a state/state-action pair by averaging the returns observed after visits to that state/state-action pair. An example of an MC policy evaluation method is the *every-visit* method, and if we initially consider policy evaluation for states, rather than state-action pairs, then this estimates $V^{\pi}(s)$ as the average of the returns following all of the visits to s, (occurrences of s), in a set of episodes. A simple every-visit MC method uses Equation 2.24 as the update rule for a visited state, where R_t is the actual return following time t and α is a step-size parameter. Here the arrow notation is used to indicate that what is on the left of the $\leftarrow$ is re-estimated as what is on its right. The step-size parameter α has not been introduced before in this chapter, and the reader will see that in Equation 2.24, it determines the degree to which the new estimate of a state's value moves towards the return that follows its visit. For an environment that does not change over time, as is the case in the experimental work of this thesis, it is appropriate to set α to decrease as the number of visits to a state increases: $\frac{1}{k}$ where k is the number of times that the state in question has been visited. However for a non-stationary environment, this is not appropriate, and so for these cases, α is often set to a constant value $0 < \alpha \leq 1$.

$$V(s_t) \leftarrow V(s_t) + \alpha[R_t - V(s_t)] \tag{2.24}$$

The *First-visit* MC method is an alternative to the *Every-visit* MC method which averages just the returns following the first visit to each state s within each episode. Note that if we lack knowledge of the environment's dynamics, then we can apply the first and every-visit methods to evaluate state-action values rather than state values.

We now go on to give an example of a full MC algorithm

2.10.2 Example of a Monte Carlo algorithm

The following on-policy Monte Carlo (MC) algorithm evaluates the value of state-actions by averaging returns following first-visits. It improves the policy towards the optimal ε-greedy policy by making the training policy ε-greedy with respect to the Q-value.

For all $s \in S$ and $a \in A(s)$ set:
$Q(s,a)$ arbitrarily;
$Returns(s,a)$ to an empty list;
π to the ε-greedy policy.

Repeat forever:
(1) Generate an episode using π.
(2) For each pair s,a appearing in the episode:
add the return following the first occurrence of s,a in the episode to $Returns(s,a)$;
set $Q(s,a)$ to the average of all of the returns in $Returns(s,a)$.
(3) For each s in the episode:
Set the learned action a^* to the action a which produces the greatest value for $Q(s,a)$;
for all $a \in A(s)$, set $\pi(s,a)$, the probability that π will take a, to:
ε if $a = a^*$;
else $(1-\varepsilon)/(|A(s)|-1)$.

This then completes the introduction to MC methods. We now move on to describe another category of RL algorithms that like MC methods learn from sample returns and hence do not require a model of the environment's dynamics.

2.11 Temporal Difference Learning

This section will introduce a category of Reinforcement Learning (RL) algorithm called *Temporal Difference Learning* (TDL). The RL experiments of Chapters 4, 5 and 7 use a TDL algorithm. As for Dynamic Programming (DP) and MC methods, policy improvement is performed by choosing actions which are greedy with respect to the value function. However, unlike MC methods, TDL algorithms perform policy

evaluation based on partial rather than complete returns, and so it is this that we focus on in the next subsection.

2.11.1 Policy evaluation

While Monte Carlo (MC) policy evaluation involves estimating the value of a state s by averaging the returns observed after visits to that state, Temporal Difference Learning (TDL) algorithms re-estimate the value of a state $V(s_t)$ based on the observed reward r_{t+1} and the estimate of the value of the subsequent state $V(s_{t+1})$. For example, the simplest TDL method, known as $TD(0)$ uses the update rule in Equation 2.25, where the step-size parameter α determines the degree to which the new estimate moves in the direction of the sum of r_{1+1} and $V(s_{t+1})$.

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \tag{2.25}$$

Since the TDL method bases its update in part on an existing estimate, it is said to be a *bootstrapping* method like Dynamic Programming (DP). Hence while MC methods wait until the end of the episode to determine the increment to $V(s_t)$ (only then is R_t known), TDL methods only wait until the next time step. This then provides a potential advantage over MC methods in terms of rate of learning. If training episodes are long, which they can be in training dialogue strategies, then delaying all learning until an episode's end can have a significant impact. Rate of learning was a major concern of ours in the RL experiments of Chapters 4, 5 and 7, and so this is the reason why we chose to use a TDL algorithm. Note that TDL methods are sound, meaning that the $TD(0)$ algorithm has been proved to converge to V^{π} with probability 1 if the step-size parameter decreases according to the usual stochastic approximation conditions given on page 39 of Sutton and Barto (1998).

We now move on to describe a full TDL algorithm - the on-policy algorithm, Sarsa.

2.11.2 An on-policy TDL algorithm: Sarsa

Sarsa is an on-policy Temporal Difference Learning (TDL) algorithm which learns the values of state-action pairs by considering transitions from state-action pair to state-action pair. It performs the following update after every transition from a non-terminal state.

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)] \tag{2.26}$$

where α is a constant step-size parameter. If s_{t+1} is terminal, then $Q(s_{t+1},a_{t+1})$ is defined as zero. This rule uses every element of the quintuple of events, $(s_t,a_t,r_{t+1},s_{t+1},a_{t+1})$ that make up a transition from one state-action pair to the next, and this gives rise to the name *Sarsa*. Below is the general form of the Sarsa algorithm, which like all on-policy methods, continually estimates Q^π while at the same time changing π towards greediness with respect to Q^π:

For all $s \in S$ and $a \in A(s)$ set:
    $Q(s,a)$ arbitrarily
    π to the ε-greedy policy

Repeat for each episode
    Choose action a from start-state s using π.
    Repeat for each step of the episode until s is a terminal state:
        Take action a and observe the reward r and new state s';
        choose action a' from s' using π;
        re-estimate $Q(s,a)$: $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$;
        set the learned action a^* to the action a which gives the greatest value for $Q(s,a)$;
        for all $a \in A(s)$, set $\pi(s,a)$, the probability that π will take a, to:
            ε if $a = a^*$;
            else $(1-\varepsilon)/(|A(s)|-1)$.
        s' now becomes s, and a' becomes a;

We have now introduced two different types of RL algorithm which learn from sample returns - Monte Carlo (MC) methods and Temporal Difference Learning (TDL). Since they are used in our RL experiments, the next section introduces *Eligibility Traces* (ETs). ETs can be combined with almost any RL method in order to speed up learning.

2.12 Eligibility Traces

Almost any Reinforcement Learning (RL) method, such as Sarsa, can be combined with *Eligibility Traces* (ETs) to obtain a more general method that may learn more

efficiently. When ETs are combined with Sarsa, the algorithm is called Sarsa(λ), and we use Sarsa(λ) in all of the RL experiments described in Chapters 4, 5 and 7.

We begin our introduction to ETs in the next section, by describing how they can be seen as forming a bridge between Monte Carlo (MC) and Temporal Difference Learning (TDL) methods.

2.12.1 Eligibility Traces as a bridge between MC and TDL methods

Recall that a Monte Carlo (MC) policy evaluation method updates the estimate of a state/state-action pair in the direction of the complete return i.e. the *target* of the update is the complete return:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ...\gamma^{T-t-1} r_T \tag{2.27}$$

where T is the last step of the episode. On the other hand, for simple Temporal Difference Learning (TDL) updates, the target (correctly denoted $R_t^{(1)}$) is the first reward plus the discounted estimated value of the next state/state-action pair e.g.

$$R_t^{(1)} = r_{t+1} + \gamma V_t(s_{t+1}) \tag{2.28}$$

where $\gamma V_t(s_{t+1})$ takes the place of the remaining terms $\gamma r_{t+2} + \gamma^2 r_{t+3} + ... + \gamma^{T-t-1} r_T$. In fact, it is possible to consider any number of steps, and in general, the n-step target is:

$$R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... + \gamma^{n-1} r_{t+n} + \gamma^n V_t(s_{t+n}) \tag{2.29}$$

An n-step back-up is TDL because it still changes an earlier estimate based on how it differs from a later estimate. TD(λ) is one particular way of averaging n-step back-ups. This average contains all the n-step updates, each weighted proportional to λ^{n-1}, where $0 \leq \lambda \leq 1$. A normalisation factor of $1-\lambda$ ensures that the weights sum to 1. The resulting update is toward a return, called the $\lambda-$*return*, defined by

$$R_t^{\lambda} = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)} \tag{2.30}$$

If $\lambda = 1$, then updating according to the λ-return is the same as the MC algorithm, while if $\lambda = 0$, then it reduces to $R_t^{(1)}$. It can be shown formally that TDL policy evalu-

ation using n-step updates converges to the correct values under appropriate technical conditions, (see Chapter 7.1 of Sutton and Barto 1998).

The view of ETs presented in this section is often called the *forward view*, because it is as if for each state visited, we are looking forward in time to all the future rewards and deciding how best to combine them. We now compare different kinds of ETs - *accumulating* and *replacing*.

2.12.2 Accumulating versus replacing Eligibility Traces

If $e_t(s,a)$ denotes the Eligibility Trace (ET) for the state-action pair s,a, then an *accumulating* ET can be defined as:

$$e_t(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s,a) + 1 & if\ s = s_t\ and\ a = a_t; \\ \gamma\lambda e_{t-1}(s,a) & otherwise. \end{cases} \qquad \text{for all s,a}$$

Hence at the start of each episode, $e_t(s,a)$ has a value of zero, but then increases by 1 each time action a is taken in state s. If (s,a) is re-visited within an episode before $e_t(s,a)$ has fully decayed to zero, then $e_t(s,a)$ will be driven greater than 1.

This is not the case with *replacing* ETs. The following definition for replacing ETs shows that $e_t(s,a)$ is set to 1 every time (s,a) is visited:

$$e_t(s,a) = \begin{cases} 1 & if\ \ s = s_t\ and\ a = a_t; \\ \gamma\lambda e_{t-1}(s,a) & otherwise. \end{cases} \qquad \text{for all s,a}$$

This definition is sometimes modified so that the ETs for the unselected actions in a visited state are set to zero:

$$e_t(s,a) = \begin{cases} 1 + \gamma\lambda e_{t-1}(s,a) & if\ s = s_t\ and\ a = a_t; \\ 0 & if\ s = s_t\ and\ a \neq a_t; \\ \gamma\lambda e_{t-1}(s,a) & if s \neq s_t. \end{cases} \qquad \text{for all s,a}$$

This is the specific kind of ET that we use in the RL experiments of Chapters 4, 5 and 7. Section 4.2.3.3 will explain why we choose this kind of ET over the alternatives. and Section 4.2.3.4 will explain why when using this kind of ET, we must be careful not to set the initial Q-values too high.

2.13 Summary

This chapter introduced Markov Decision Processes (MDPs) and Reinforcement Learning (RL) in the context of their application in the experimental work of this thesis. The MDP-RL framework is used for dialogue management because it is thought able to produce better dialogue strategies with less effort than the traditional rule-based hand-coding approach. As a statistical learning method, it has a number of key potential advantages over the traditional approach e.g. greater robustness to noise and uncertainty. Amongst other statistical learning methods, it is very attractive because RL tries to produce a plan to maximise long-term reward, and this is exactly what we want for a dialogue strategy. In addition, RL uses trial-and-error search which will be necessary given a large number of possible dialogue strategies to explore.

An MDP is defined in terms of actions (available to the agent - in our case the Dialogue Manager (DM)), states (representations of the state of the environment - in our case the dialogue context), and numerical rewards as determined by a reward function, (a function of the state which formalises the agent's goal). A policy is a mapping between states and actions, (in our case, a dialogue strategy). MDPs bear the Markov property which means that the new state which an agent transitions to, and the reward it receives as a result, depends only on the immediately prior state and the action taken in that state. The dynamics of an MDP are defined by the one-step transition probabilities, (denoted here as T), and the expected rewards, (denoted here as R), for all states and their allowable actions. RL algorithms are used to try to learn the optimal policy i.e. the policy which maximises the expected return. The return is a function of the sequence of rewards produced by each interaction between the agent and the environment within a single episode e.g. dialogue. Since the return is a random variable, its expected value is its population mean.

Value functions of policies are fundamental to all RL algorithms - for each state or state-action, they estimate the expected return when starting from that state or state-action and then following a particular policy thereafter. The optimal value function assigns to each state, or state-action, the largest expected return achievable by any policy, and a policy whose value function is optimal is an optimal policy. Dynamic Programming (DP) is not used for learning full dialogue strategies because it requires complete knowledge of the environment, i.e. T and R, and it is computationally expensive. However, it has been used in what we referred to as a model-based approach in

order to learn partial dialogue strategies from a corpus of human-machine dialogues. It was also described in this chapter because it provides a good basis for understanding other RL algorithms. DP learns the optimal policy by using two different processes - policy evaluation, and policy improvement. Policy evaluation computes the value function for a particular policy. The Bellman equation for a value function expresses a relationship between the value of a state and the values of its successor states. If we substitute all of the possible states into this equation, and then solve the resulting set of simultaneous linear equations, (typically through an iterative method), then we have performed policy evaluation. Policy improvement is the process of improving a policy by making it greedy with respect to its own value function. The processes of policy evaluation and policy improvement interact under a framework called Generalised Policy Iteration (GPI), and both stabilise only when the optimal policy is found.

RL algorithms which learn from sample returns e.g. Monte Carlo (MC) methods and Temporal Difference Learning (TDL) do not require a full model of the environment, and are less computationally expensive. They have been used to learn full dialogue strategies with a user and Automatic Speech Recognition (ASR) / Natural Language Understanding (NLU) simulation, in what we referred to as a simulation-based approach. Learning from sample returns requires soft training policies i.e. training policies which have a non-zero probability of selecting each action in a state, and these can be produced using ε-greedy or softmax action selection. On-policy and Off-policy methods are two different methods for using soft training policies in order to learn a policy. The fundamental difference between MC and TDL methods is in the way in which they perform policy evaluation - whereas MC methods evaluate a policy using complete returns, TDL methods use partial returns. Eligibility Traces (ETs) can be seen as forming a bridge between MC and TDL methods, and can be used to speed up learning.

Chapter 3

Previous research on Reinforcement Learning of dialogue strategies

3.1 Introduction

This chapter will summarise and analyse previous research on using Reinforcement Learning (RL) to learn dialogue strategies for Spoken Dialogue Systems (SDSs). Several research groups have been working in this area in the past 10 years (see Table 3.1), and significant progress has been made. Different user simulation approaches for training RL approaches to dialogue have previously been surveyed by Schatzmann et al. (2006), and where relevant, this chapter also discuss issues related to user simulations, but here our primary focus is on surveying the different RL systems themselves. This includes surveying very recent work such as Henderson et al. (2008), highlighting the main advances, and pointing out open problems. For a version of the literature review contained in this chapter which is extended to cover Frampton and Lemon (2005, 2006, 2008a), the reader should refer to Frampton and Lemon (2008b).

For each research system developed by the groups, our analysis will compare:

- application domain of the SDS,
- RL technique,
- data set/corpus used,
- state features and action set,

Research publications	**Summary**
Levin and Pieraccini (1997)	**Proof-of-concept** that a DM can be modelled as a MDP & RL applied to learn a dialogue strategy.
Singh et al. (1999) & Singh et al. (2002)	Learned/tested **partial ISF strategies** with **real users**; state features for slot-status & which ASR grammar used last.
Walker (2000)	As for Singh et al. but using **PARADISE reward:** predicts **user satisfaction** from dialogue efficiency/quality & task success.
Pietquin and Renals (2002)	Learned **full ISF strategy**; goal-directed, part-stochastic US; stochastic **ES**; **hand-coded probabilities**; no real user tests; only slot-status state features.
Scheffler and Young (2002)	As for Pietquin & Renals but **probabilities for US & ES learned from data**; US & ES evaluated (unconvincing).
English and Heeman (2005)	Learned **both** the US & system strategies simultaneously via RL, hence problem over accuracy of US.
Paek and Chickering (2005)	Learned v. simple strategies with MDPs, **non-Markov models** & **model-specific automatic FS**.
Tetreault and Litman (2006)	Learned partial strategies for a **tutor system**; evaluated usefulness of state features; no real user tests.
Henderson et al. (2005) & Henderson et al. (2008)	Hybrid RL/SL to learn full ISF strategy with v. large state-space from fixed dataset; real user tests & comparison to hand-coded strategy; no insights for which state features important & why.

Table 3.1: Timeline of previous research on RL of dialogue strategies for SDSs; DM = Dialogue Manager; MDP = Markov Decision Process; ISF = Information-Slot-Filling; U/ES = User/Error Simulation; FS = Feature Selection; SL = Supervised Learning.

- reward function,
- user simulations,
- Automatic Speech Recognition (ASR)/ Natural Language Understanding (NLU) error simulations,
- experimental results.

Throughout the presentation a comparative analysis of the previous research will be

given, lessons learned will be presented, and future research directions will be motivated.

3.1.1 Properties shared by all approaches: slot status features, initiative and confirmation actions

In almost all of the systems discussed below, the task of the dialogue system is "slot-filling", which as stated in Section 1.1.1, involves collecting a set of preferences or search constraints from the user (e.g. destination city, preferred food type). In general, dialogue management action decisions such as initiative, (see Section 1.1.1), and confirmation strategies are studied by all groups. In addition, all prior research uses "slot-status" features in dialogue states, e.g. for each information slot in the particular domain, whether it is filled, the associated Automatic Speech Recognition (ASR) Confidence Level, whether it is confirmed. A CL is a number between 0 and 1 based on acoustic measurements and defines how sure the system is to have performed correct recognition. Some approaches also differentiate states based on the particular values of filled slots. In the presentation below, we note cases where research has used features *in addition to* slot filled/confirmed status for the relevant task domain.

3.2 Early theory and proof-of-concept: Levin and Pieraccini 1997 and 2000

Levin and Pieraccini (1997) contains the first presentation of the concept of using a Markov Decision Process (MDP) and Reinforcement Learning (RL) to learn a dialogue strategy. Levin et al. (1998) and Levin et al. (2000) then described a first attempt at putting the theory into practice. They used a Monte Carlo (MC) algorithm to learn a dialogue strategy for an Air Travel Information System (ATIS).

3.2.1 State features and action set

In learning strategies for the ATIS task, Levin et al. (1998) and Levin et al. (2000) used state vectors consisting of the following fields, (in addition to slot status):

1. the number of data tuples retrieved from the database according to the user request,
2. a feature which records whether the system has already presented data to the user.

Possible system actions for learning include:

1. an open-ended question i.e. “How can I help you?”,
2. ask the user to provide information about a slot/specific attribute of the task (e.g. origin, airline, departure time etc.,
3. retrieve data from the database according to the current user request,
4. present the retrieved data to the user,
5. ask the user to relax a particular constraint e.g “Do you mind considering other airlines?”,
6. close the dialogue.

3.2.2 Reward function

In the ATIS domain, the system’s goal is to provide the user with information about flights in an efficient way. Efficiency here involves the duration of the dialogue (in turns), the cost of external resources (database retrieval) and the effectiveness of the system output to the user. Hence Levin et al. (2000) used a reward function that was a weighted sum of the following:

1. length of the dialogue in number of turns,
2. expected number of tuples retrieved from the database,
3. a data presentation cost function, $f_0(N_0)$,
4. overall task success measure.

For the third term, the data presentation cost function, N_0 is the number of records that are presented to the user, and generally, $f_0(N_0)$ is zero for N_0 smaller than a reasonable N^*, and increases rapidly thereafter, where N^* depends on the medium used to output information to the user, (it is generally small for voice based communication, and higher for display). The fourth term is an overall binary task success measure that is changed to "successful" if any data is presented to the user - it is assumed that there have been no misunderstanding errors and so that the data matches the user request.

3.2.3 User simulation

Unlike other earlier research between 1998-2002, Levin et al. (2000) applies the simulation -based approach for using Reinforcement Learning (RL) to learn a dialogue strategy, (see Section 2.8 for definitions of model and simulation-based approaches). Levin et al.'s user simulation is partly-stochastic - it uses unigram and bigram models which generate a user response based on the previous system action. As in all cases where a user simulation has been used for RL of dialogue strategies, training dialogues between the system and user simulation are conducted via abstract representations of utterances such as Dialogue Acts (DAs), (see Section 1.1.1 for a definition of a DA). This is because such abstract representations are easier to generate than word sequences, let alone speech signals, and they also make it easier to simulate Automatic Speech Recognition (ASR) and Natural Language Understanding (NLU) errors. Note that Levin et al. (2000) does not use any kind of error simulation. Levin et al.'s user simulation supplies slot values rather than abstracting away from them. (For example, if it is trying to fill the "destination_city" slot, it will output something of the form "destination_city(Pittsburgh)" rather than just "destination_city".) This necessitates the deterministic part of the simulation which ensures *consistent or goal-directed* behaviour, so that within a dialogue, the simulation always supplies the same values for each slot e.g. the simulated user does not change its mind half-way through so that it wants to fly to "Philadelphia" instead of "Pittsburgh".

The simulation used unigram and bigram models for the following actions, where X and Y can refer to any slot, including the same slot:

1. the number of values to supply in response to a greeting e.g. $P(n)$, $n = 0, 1, 2$,
2. which slot to supply a value for in response to a greeting e.g. $P(ORIGIN)$, $P(AIRLINE)$,

3. which value to supply given the slot e.g. $P(Boston|ORIGIN)$,$P(Delta|AIRLINE)$,
4. supplying a value for slot X given that the system asked about slot Y e.g. $P(AIRLINE|AIRLINE)$, $P(AIRLINE|DEPARTURE\_TIME)$,
5. supplying values for N unsolicited slots given that the system has just asked about slot X e.g. $P(2|AIRLINE)$,
6. accepting the relaxation prompt given which slot the system wishes to relax e.g. $P(yes|AIRLINE) = 1 - P(no|AIRLINE)$.

The original ATIS dialogue corpus (Walker et al., 1997) could only be used to estimate the parameters for actions 1 and 2, because in this corpus, the system never takes the initiative, and does not ask constraining or relaxing questions. Hence the parameters for the other models were set using intuition. Levin et al. (2000) did not simulate ASR/NLU errors. This simulation was not evaluated to assess how realistic it is.

As we will see, this is a fairly typical approach to user simulation for RL in SDS, featuring:

- partly deterministic and partly-stochastic behaviour,
- consistent/goal-driven behaviour,
- some probabilities are derived from appropriate data, some are hand-coded,
- communication between the user simulation and system via abstract representations of utterances e.g. Dialogue Acts (DAs),
- no evaluation of the simulation quality.

Later in this survey, more recent research will be described in which efforts are made to establish the accuracy of the user simulation e.g. Scheffler and Young (2002), Henderson et al. (2008).

3.2.4 Experimental results

Levin et al. (2000) state that by the end of the training, the system had explored 111 states, and converged to the optimal strategy. A summary is provided of how the strategy behaves. Firstly, the system always starts the dialogue by greeting. Depending on

the system state after getting the user response to this greeting, the system, if needed, proceeds by asking constraining questions until the origin, destination and airline are specified. Note that the strategy does not take into account the number of database entries that match the user's constraints after every user turn (i.e. the strategy continues to ask for all constraints even if there is only 1 or 0 current results). Next, the strategy retrieves data from the database. After the retrieval, if the resulting data set is empty, (because the query was over-constrained), then the system, depending on the current state, relaxes the airline or the departure time, and retrieves again. If there are too many flights in the data set, it asks for additional constraints (e.g. the departure time) and then retrieves again. If at any point during the dialogue the retrieved data set has a reasonable number of flights, then the data is output and the dialogue is closed.

Levin et al. (2000)'s evaluation is unsatisfactory for the following reasons:

- There is no quantitative evaluation of the learned strategy based on average final reward.
- There are no quantitative comparisons to any other strategy e.g. a hand-crafted strategy, a random baseline strategy or some other kind of learned baseline strategy, (only one strategy is learned in any case).
- As a result of the above deficiencies, no statistical significance results can be reported.
- The learned strategy is tested with the same simulation with which it was trained.
- The learned strategy is not tested on real users.

As we shall see later in this chapter, evaluation methodologies have become more sophisticated in recent years e.g. Lemon et al. (2006a).

3.2.5 Summary

This early work was very important in pioneering the basic concepts and methods in RL for Spoken Dialogue Systems (SDSs). All subsequent work builds on this approach to some degree, but as we shall show, many aspects of the methodology have been improved upon. We now describe work which closely followed the initial presentation by Levin and Pieraccini (1997).

3.3 RL using data from real users: initial results from Singh et al. 1999-2002

This section describes the work of Singh et al. (1999, 2002), which differs from that of Levin et al. (2000) in that it uses the model-based as opposed to simulation-based approach for using Reinforcement Learning (RL) to learn a dialogue strategy, (see Section 2.8 for definitions of model and simulation-based approaches). In addition, partial rather than full strategies are learned, (action choices are learned only in certain states, not all states), and the exploratory data used by the reinforcement learner is generated by real, not simulated user interactions. Since the probabilities for the stochastic user simulation used by Levin et al. (2000) were set using intuition, rather than learned from data, we can say that Singh et al. (1999, 2002) pursue a more strongly data-driven approach. As we shall see further on in this survey, other researchers e.g. Walker (2000), Tetreault and Litman (2006) have also used the model-based approach and real user interactions in order to learn partial strategies. Learning full strategies is perhaps not realistic when RL is directly applied to dialogues collected with real users. Learning full strategies rather than partial strategies obviously requires a greater number of training dialogues, but collecting dialogues with real as opposed to simulated users is costly in terms of time and possibly money. Even if these costs were not prohibitive, real users cannot be expected to interact with a system which is exploring different actions in every state, many or most of which will be unreasonable.

Singh et al. (1999) describes 6 experiments in which they apply their software tool "RLDS" (Reinforcement Learning for Dialogue Systems) to the TOOT train schedule system. The TOOT system is a slot-filling system whose goal is to find the user a suitable train in the Amtrak train schedule. RLDS takes a set of transcribed sample dialogues, builds a Markov Decision Process (MDP) and then uses a standard Dynamic Programming (DP) algorithm called *value iteration* in order to find the optimal value function and strategy. RLDS was applied to a corpus of 146 sample dialogues between real users and TOOT. In appropriate states, action choices were learned for information presentation, confirmation (whether and how to confirm user utterances) and initiative (system vs. mixed), while in other states the action choice was fixed. The main aims of the experiments described in Singh et al. (1999) were to:

1. confirm that the RLDS methodology and software produces intuitively sensible policies,

2. use the value functions computed by the RLDS software to discover and understand correlations between dialogue properties and performance.

In related work, Singh et al. (2002) collected sample dialogues between real users and another slot-filling system called NJFun. The NJFun system provides users with information about "fun" things to do in New Jersey. Here, there were 54 subjects for training and 21 for testing, and this provided 311 training dialogues and 124 test dialogues. Like Singh et al. (1999), Singh et al. (2002) only attempted to learn which action to take in certain states. In some states, they wanted to learn whether to confirm a slot value, and in others, whether the system should take the initiative.

3.3.1 State features and action sets

Singh et al. (1999) used different state features depending on the aim of the experiment. The state features used for TOOT in the first experiment were the slot-status features only. Here the aim was simply to check that RLDS was working and could learn a sensible policy i.e. one which filled all of the slots, confirmed them and then queried the database. Subsequent experiments also used the following two state features:

1. number of filled slots,
2. length of the dialogue.

One interesting experiment aimed to find a correlation between the value function and the number of "distress indicators" in a dialogue - indicators that the dialogue is potentially in trouble e.g. *timeouts*, *resets*, *user requests for help*. Hence a feature that kept track of the number of distress indicators was added to the state representation.

Singh et al. (2002) aimed to learn which of 2 actions (initiative and confirmation type) to take in 42 different states (the other action choices were hand-coded). Each of these states was represented using the following features:

1. whether the system has greeted the user (0 or 1),
2. which slot is being worked on (1-4),

3. confidence/confirmed (0,1,2 for low, medium and high ASR confidence[1], 3,4 for explicitly confirmed, and disconfirmed),

4. whether a value has been obtained for current slot (0 or 1),

5. how many times the current slot has been asked (0,1,2),

6. whether a non-restrictive or restrictive grammar was used (0 or 1),

7. whether there was trouble on any previous slot (0 or 1).

Some of the 42 states occurred when the system needed to ask or re-ask a slot, and then the action choices were to retain the initiative or to give it to the user. The rest of the 42 states occurred when the system has just obtained a slot value, and then the action choices are to confirm, or to move onto another slot. The system was trained with 54 users (311 dialogues) by taking random choices at these points, (the "Exploratory for Initiative and Confirmation" strategy), and collecting rewards via task completion.

3.3.2 Reward functions

Singh et al. (1999) and Singh et al. (2002) only ever gave a reward in terminal dialogue states. For the TOOT experiments, this terminal state reward was obtained from a question in the user satisfaction survey. This reward was $+1$ if the user said that they would use the system again, 0 if they said "maybe", and -1 if they said "no". In Singh et al. (2002) dialogue reward was automatically labelled by a $+1$ in the case of a completed task, or -1 otherwise.

3.3.3 Experimental results

The main findings of the 6 experiments described in Singh et al. (1999) were the following:

1. RLDS is capable of learning a sensible basic dialogue policy,

2. the value function grows roughly linearly with the number of confirmed attributes,

[1] As stated in Section 3.1.1, a speech recogniser can use acoustic measurements to indicate how sure it is to have performed a correct recognition.

3. dialogues with a higher number of distress features have a lower value,
4. within the same length dialogue it is better to have obtained more attributes,
5. system initiative has higher value than mixed initiative,
6. results are extremely similar using a reward function based on whether the user perceived the task to have been completed, rather than actual task completion.

Singh et al. (2002) found that the value function of the learned strategy was higher than the average value of the random "Exploratory for Initiative and Confirmation" strategy used during training. Task completion increased from 52% in training to 64% in testing ($p < 0.059$ in an independent samples t-test over subject means) [2]. The experiments had also involved collecting subjective evaluations from the users, but these were not significantly different between the learned and random policies.

This work was the first to provide significant results showing that a learned policy can perform well with real users of a dialogue system. However, there are some unsatisfactory elements to this result:

- the baseline strategy for comparison was random action choice, rather than a state-of-the-art hand-crafted strategy,
- the strategy was only learnt for a small number of choice points, rather than for the entire state-action space (all actions in all possible states),
- only small state spaces were used (e.g. compared to Henderson et al. 2008).

We now discuss the related work of Walker (2000), which used a different, data-driven, methodology for determining the reward function for learning.

[2] A t-test is any statistical hypothesis test in which the test statistic has a Student's t distribution if the null hypothesis is true i.e. no difference exists between two groups for the variable being compared, (in this case the means). Population data from which sample data are drawn are assumed to be normal, and variances of the populations, equal. The p value gives the probability that the null hypothesis is true. For information on independent samples t-tests, (two randomly selected groups), see page 427 of Sheskin (2007), and on dependent samples, (two groups matched on some variable or the same people tested twice i.e. repeated measures), see page 743.

3.4 Predicting user satisfaction and defining reward: Walker 2000

Like Singh et al. (1999, 2002), Walker (2000) also describes an experiment in which a partial slot-filling strategy is learned based on exploratory data generated by real user interactions. However, Walker makes an important novel contribution with respect to the reward function by proposing "PARADISE" (PARADIgm for System Evaluation), which is a method for predicting a dialogue-end *user satisfaction* score based on metrics that can be easily collected by the system itself. Section 3.4.2 will introduce PARADISE in detail.

The particular Spoken Dialogue System (SDS) used by Walker (2000) is ELVIS (Email Voice Interactive System) (Walker et al., 1998), the purpose of which is to support access to email over the phone. Q-learning, (an off-policy Temporal Difference Learning (TDL) algorithm - see Sutton and Barto 1998), is applied to a corpus of 219 dialogues between ELVIS and 73 different real users, (each user carries out a set of three email tasks). In generating these dialogues, the system randomly explored alternate strategies in appropriate states for initiative, reading messages and summarising folders, and used fixed strategies elsewhere e.g. for requesting and providing information. Hence Reinforcement Learning (RL) is being used to learn action choices for initiative, reading messages and summarising folders. The learned strategy is tested in 18 dialogues with 6 new users. Training and testing dialogues are all evaluated with a user satisfaction score, which is computed from the user's answers to a number of questions about how the dialogue went.

3.4.1 State features and action sets

As stated above, Walker (2000) explores different action choices with regard to initiative, and summarising and reading messages. ELVIS explores two different types of initiative action:

1. The system-initiative action constrains what the user can say by requesting a particular item of information.

2. The user-initiative action allows the user to take control of the dialogue and specify exactly what s/he wants to do next.

For the implementation of ELVIS used in Walker (2000), the choice of initiative is made early in the dialogue and then kept to for the remainder in order to avoid confusing the user. If the system is using a user-initiative strategy but the user fails to provide a recognisable response, then the system will take the initiative to repair the situation before switching back to user-initiative actions. ELVIS explores 3 alternative summarisation actions:

1. The Summarize-Both (SB) action uses both the sender and the subject attributes in the summary.
2. The Summarize-System (SS) action summarises by subject or by sender based on the current context.
3. The Summarize-Choice-Prompt (SCP) action asks the user to specify which of the relevant attributes to summarise by.

Finally, ELVIS explores 2 different Read actions for reading multiple messages following a user request e.g. “Read my messages from Kim.”:

1. The Read-First (RF) action involves summarising all of the messages from Kim, and then taking the initiative to read the first one.
2. The Read-Summary-Only (RSO) action provides information that allows users to refine their selection criteria.

Walker uses the following state features, which are described further below:

1. KnowUserName (U): 0,1,
2. InitStrat (I): 0,SM,MI,
3. SummStrat (S): 0,SS,SCP,SB,
4. ReadStrat (R): 0,RF,RSO,RCP,
5. TaskProgress (P): 0,1,2,
6. CurrentUserGoal (G): 0, Read, Summarize,
7. NumMatches (M): 0, 1, N>1,

8. WhichSelection (W): 0, Sender(Snd), Subject(Sub), InOrder(InO),
9. KnowSelectionCriteria (SC): 0,1,
10. Confidence (C): 0,1,
11. Timeout (T): 0,1,
12. Help (H): 0,1,
13. Cancel (L): 0,1.

The KnowUserName (U) feature keeps track of whether ELVIS knows the user's name or not. The InitStrat (I), SummStrat (S) and ReadStrat (R) features keep track of whether ELVIS has already employed a particular initiative strategy, summarise strategy, or a reading strategy in the current dialogue, and if so, which strategy it was. The TaskProgress (P) feature tracks how much progress the user has made in completing the experimental task. The CurrentUserGoal (G) feature corresponds to the system's belief about what the user's current goal is. The WhichSelection (W) feature tracks whether the system knows what type of selection criteria the user would like to use to read her messages. The KnowSelectionCriteria (SC) feature tracks whether the system believes it understood either a sender name or a subject name to use to select messages. The NumMatches (M) feature keeps track of how many messages match the user's selection criteria. The Confidence (C) feature is a threshold variable indicating whether the speech recogniser's confidence that it understood what the user said was above a pre-set threshold. The Timeout (T) feature represents the system's belief that the user said Help, and leads to the system providing context-specific help messages. The Cancel (L) feature represents the system's belief that the user said Cancel, which leads to the system resetting the state to the state before the last user utterance was processed. Walker reports that these state features produced 110592 possible states but that not all of these states occur.

3.4.2 PARADISE and the reward function

As stated previously, Walker (2000) proposes a methodology called PARADISE for developing predictive models of SDS performance (see also Walker et al. 2000). Walker (2000) describes an application of the methodology to the training dialogues collected with ELVIS. Recall that the training dialogues are generated using a strategy which

randomly explores action choices for initiative, reading messages and summarising folders. At the end of each dialogue with ELVIS, the user's satisfaction is seen as the sum of the following features, where each feature has some positive weight:

1. Actual Task Completion (0 or 1),
2. Perceived Task Completion (0 or 1),
3. Task Ease ($0-4$),
4. Comprehension Ease ($0-4$),
5. System behaved as Expected ($0-4$),
6. Future Use ($0-4$).

The value for Actual Task Completion was obtained from the system logs, but the values for the other user satisfaction features, ($2-6$ in the list above) were supplied by the users. The modelling technique, *multivariate linear regression* [3] is then used to learn to predict the user satisfaction score based on a number of metrics that can be directly measured from the system logs. These metrics include:

1. Dialogue Efficiency Metrics - elapsed time, system turns, user turns,
2. Dialogue Quality Metrics - mean ASR confidence score, number of timeouts, (a timeout is when user response is detected within a certain amount of time), Automatic Speech Recognition (ASR) rejections, user requests for help, user requests to restart the dialogue, barge-ins, (interruptions of the system by the user).

The resulting model is a PARADISE model for predicting user satisfaction/SDS performance.

In learning the action choices for initiative, reading messages and summarising folders, the actual user satisfaction score was used as reward, not the user satisfaction score as computed by the PARADISE model.

[3] Multivariate linear regression, (see page 1433 of Sheskin 2007), models numerical data by a *least squares function* which is a linear combination of the model parameters and depends on > 1 independent variables. A least squares function fits a model so that the sum of the squared residuals has its least value, a residual being the difference between an observed value and the value given by the model.

3.4.3 Experimental results

Regarding PARADISE, a stepwise linear regression on the training data showed that Task Completion, Mean Recognition Score (MRS), Barge-in% and Rejection% were significant contributors to User Satisfaction, accounting for 39% of the variance in *R-squared*. [4]. How well the model generalised to unseen data was tested with a ten-fold cross-validation. [5] The average R^2 for the training set was 37% with a standard error of 0.005, while the average R^2 for the held-out 10% of the dialogues was 38% with a standard error of 0.06. This suggests that the model will generalise to new ELVIS dialogues.

Regarding the learned strategy, statistical analysis indicated a significant increase in user satisfaction from training to test (p = 0.047). Wherever the choice arises, the learned strategy uses the System-Initiative and Read-First actions. The learned strategy uses the Summarize-Both action at the beginning of the dialogue, and then switches to the Summarize-System action in later phases.

This work then shows that using a more data-driven definition of reward leads to a better learned strategy than a random strategy. We now turn to another strand of research, which has focused on *simulated* users and ASR systems rather than training with real user data, (the simulation-based approach to learning dialogue strategies - see Section 2.8.2).

3.5 Learning with simulated users and ASR errors: Pietquin and Renals 2002

This section describes the work of Pietquin and Renals (2002), which uses a Monte Carlo (MC) algorithm and a stochastic user simulation to learn a strategy for a slot-filling computer-dealing system. The novel feature of this work is that a simulated Automatic Speech Recognition (ASR) system is introduced into the Reinforcement

[4] R^2, the "coefficient of determination", (see page 1230 of Sheskin 2007), is the proportion of variability in a data set that is accounted for by a statistical model. $R^2 = 1$ indicates that the fitted model explains all variability, $R^2 = 0$, no 'linear' relationship between the dependent and independent variables, and $R^2 = 0.39$, that approximately 39% of the variation in the dependent variable can be explained by the independent variables, and the remaining 61% by unknown variables/inherent variability.

[5] In an n-fold cross-validation, the data is first divided into n (usually equal-sized) portions, and then in each of n folds, a different one of these portions is used for testing, while the remainder of the data is used for training. Results are averaged across the n folds.

Learning (RL) environment. This ASR simulation simulates both speech recognition errors and Confidence Levels (CLs), (recall that CLs were introduced in Section 3.1.1). However, the probabilities for the user and ASR simulations are not learned from data.

3.5.1 State features and action sets

For the computer-dealing application, each state is represented with the following slot-status feature:

- A confidence feature for each of the 7 slots - undefined if slot unfilled, low or high.

Hence each slot can be represented as either (0,undefined), (1,low) or (1,high), meaning that there are 3^7 possible states.

The action set contains 6 generic actions:

1. GREETING e.g. “How may I help you?”,
2. ASK: ask to constrain the value of a slot,
3. CONF: ask to confirm the value of a slot,
4. RELAX: ask to relax the value of a slot,
5. DBQUERY: perform a database query,
6. CLOSE: present data and close the dialogue session.

With 7 slots, this gives 24 different actions. When the database was queried, only values with a high confidence level were used.

3.5.2 Reward functions

After each turn, the reinforcement learner receives a reward that is a weighted sum of the following:

1. a negative reward if the final state has not yet been reached,

2. the number of database accesses,
3. the number of presented records,
4. the ASR Confidence Level (CL) for the user's most recent utterance,
5. a "function of the modelled user's satisfaction".

Pietquin and Renals (2002) and Pietquin (2004) do not appear to provide details about the "function of the modelled user's satisfaction".

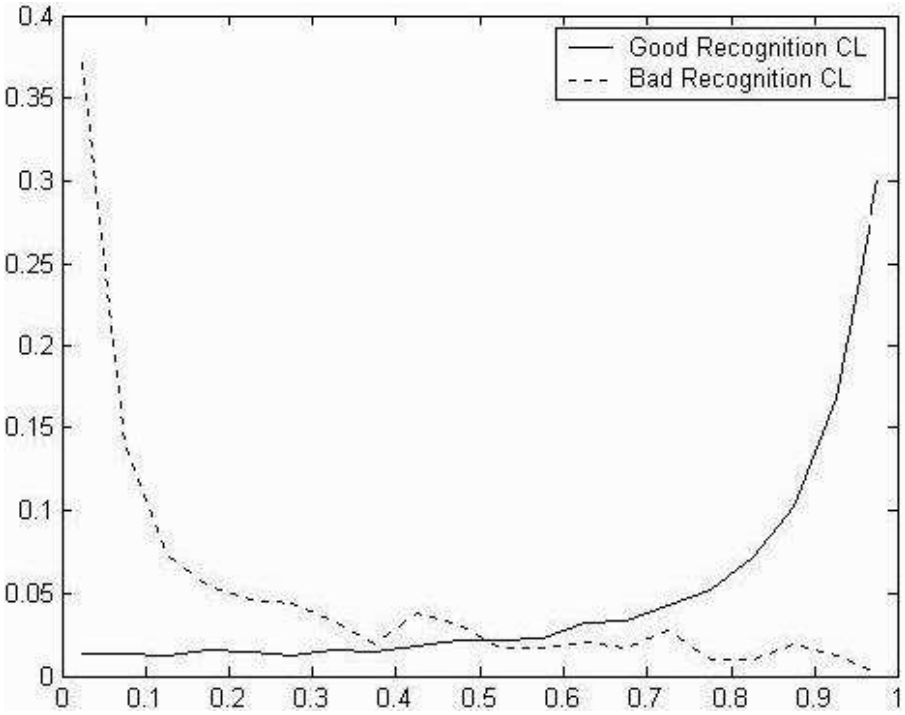

Figure 3.1: A Confidence Level (CL) distribution for good and bad recognitions.

3.5.3 A stochastic user and error simulation

Like the simulation used by Levin et al. (2000), the simulation used by Pietquin and Renals (2002) is partly-stochastic and able to simulate mixed-initiative behaviour. Another common feature is that the purpose of its deterministic element is to ensure that the simulation maintains the same goal within an individual dialogue - a main user goal is randomly defined at the start of each dialogue and user actions are consistent with this goal. Here, since the domain is computer-dealing, a user goal describes a set of specifications for a computer. A difference to the Levin et al. (2000) simulation is

that for the Pietquin and Renals simulation, none of the probabilities are learned from data - they are all supplied via intuition.

Pietquin and Renals (2002) and Pietquin (2004) denote n to mean the number of slots, g = the user goal, k_t = the user knowledge at time t, s^β = the slot which the system has just asked about, u^α = the slot which the user simulation provides a value for in response to the system prompt. The user goal g defines the user's preferred value for each slot, and k_t records how many times the user has supplied a value for each slot. The probabilities used by the user simulation then include:

1. probabilities associated with responses to greeting e.g. $P(n|Greeting, g), P(u^\alpha|k_t, g)$,
2. probabilities associated with responses to constraining questions e.g. $P(u^\alpha|s^\beta, k_t, g)$, $P(n|s^\beta)$,
3. probabilities associated with responses to relaxation prompts e.g. $P(yes|s^\beta, k_t, g)$, $P(no|s^\beta, k_t, g)$,
4. probability associated with user satisfaction: $P(close|s^\beta, k_t, g)$ i.e. the user simulation can indicate its dissatisfaction by closing the dialogue early, and so this probability is set to increase with the number of times that the simulation must supply a particular slot value.

As stated at the start of Section 3.5, the novel feature of this work is that it uses a stochastic ASR simulation which simulates ASR errors and outputs Confidence Level (CL) scores. Just as for the user simulation, the probabilities for the ASR simulation are not learned from data - they are set using intuition. Pietquin and Renals (2002)'s ASR simulation uses different CL and error rate distributions for a finite number of recognition tasks, which include:

1. digits,
2. numbers,
3. dates,
4. unrestricted continuous speech.

A CL distribution is composed of two distinct curves respectively for good and bad recognition results - Figure 3.1 represents a CL distribution output from a real ASR

system obtained using some of its training data (isolated words). As the two curves cover each other, it is unavoidable to reject some well-recognised utterances as well as to accept a few bad recognition results by defining a single CL threshold. The ASR simulation used here receives lists of one or more slot-value pairs from the user simulation, which it then splits into individual slot-value elements. The probability of it simulating an ASR error for a particular slot-value pair is then dependent on the average Word Error Rate (WER) for the task in question. Note that the simulation assumes that recognition errors only affect values of the slot-value pairs and that only words occurring in the same context can be substituted with each other. If the simulation simulates an ASR error, then it produces a partial CL according to the "bad recognition" curve of the corresponding CL distribution, and if it does not, then it produces a partial CL according to the "good recognition" curve. A global CL is generated for the list by multiplying all partial CLs.

3.5.4 Experimental results

Pietquin and Renals (2002) reports that after several thousand simulated dialogues, the learned strategy stabilises and appears to be optimal. A summary description of the strategy is provided: after greeting the user, the system uses the ASK and RELAX actions until it has enough information with a high Confidence Level (CL) to query the database and return a set which is not empty, but not "too large". No details are given as to what "too large" means in practice.

Pietquin and Renals report that the ASR simulation affected the order in which the ASK action was applied for each slot. The learned strategy first asks questions about values that present better recognition results e.g. numbers. For example, it will ask for a value for the RAM size slot before the computer brand slot.

This work then shows that reasonable dialogue strategies can be trained in simulation rather than with real data, and that with only slot-status features represented in the state, the reinforcement learner can learn to ask the slots in an order which is sensitive to the likelihood of ASR errors. However, the evaluation here is lacking in the same way as that of Levin (see Section 3.2.4) e.g. there is no quantitative evaluation of the learned strategy or testing with real users. We now present similar work.

3.6 A goal-directed user simulation and error model with probabilities learned from data: Scheffler and Young 2002

Scheffler and Young (2002), Scheffler (2002) use Q-learning (an off-policy Temporal Difference Learning (TDL) algorithm - see Sutton and Barto 1998), Eligibility Traces (ETs), a goal-directed user simulation and a system error model to learn a dialogue strategy for a slot-filling cinema information Spoken Dialogue System (SDS). The error model simulates both Automatic Speech Recognition (ASR) and Natural Language Understanding (NLU) errors. Unlike Pietquin and Renals (2002) the user simulation and error model are both trained on a corpus of real user data. The user simulation is also described in Scheffler and Young (2001), and is an extension of a previous user model described in Scheffler and Young (2000). Unlike in Levin et al. (2000) or Pietquin and Renals (2002), there is some evaluation done to assess how realistic the simulations are, and the performance of the learned strategy is compared to hand-crafted baselines.

3.6.1 State features and action sets

Scheffler and Young (2002) used 5 different state representations composed from different combinations of the following features (in addition to slot-status):

1. slot currently in focus (Type, Day, Film, Cinema),
2. “InfoSource” (Default, Negated, Elicited, Unelicited),
3. confidence (Default,0.2,0.3,...,1.0),
4. “ConfLevel” (Default, Low, High).

Of these 5 state representations, the one with the greatest number of state-action pairs included features 1 and 3, in addition to standard slot-status features. Once the impossible state-action pairs had been ruled out, this state-representation was left with 1298 possible state-action pairs.

The different actions which the learner had to choose between were:

1. **Mixed Initiative** - mixed initiative query for more information (available if current slot is empty),
2. **System Initiative** - query for information on the current slot (available if current slot is empty),
3. **Explicit confirmation** - confirm the contents of the current slot explicitly,
4. **Confirm all** - confirm the contents of all slots,
5. **Implicit confirmation** - confirm the contents of the current slot implicitly while querying for information on the next slot,
6. **Accept** - accept information in the current slot without confirmation at present, and terminate if a complete transaction has been specified.

This action set allows for automatic design of the choice between mixed and system initiative and the confirmation strategy, (a choice between explicit confirmation, implicit confirmation, and delaying confirmation until later).

3.6.2 Reward function

Scheffler and Young (2002) use a simple reward function that gives a reward at the end of each dialogue. This reward function includes a per-turn penalty and a task failure penalty.

3.6.3 User simulation and error model

As in Levin et al. (2000) and Pietquin and Renals (2002), the user simulation of Scheffler and Young (2001, 2002) is partially-stochastic and is able to simulate mixed-initiative behaviour. Again, the deterministic element of the simulation is that it is goal-directed - a main user goal is randomly defined at the start of each dialogue and user actions are consistent with this goal. There is also a probabilistic error model.

The user simulation generates utterances using lattices which are made up of nodes and paths between these nodes. Nodes are probabilistic or deterministic choice points relating to user behaviour. The deterministic choices are based on user state and so ensure consistent goal-directed behaviour. The parameters for the probabilistic user

behaviour choice points and error model are estimated based on training data collected with a prototype cinema information SDS. In all cases, probabilities are estimated from the data using *Maximum Likelihood Estimation (MLE)* [6]. For the probabilistic user behaviour choice points, an *event* is a user action, and the *context* is the previous system action(s) and a representation of the internal user state. For the error model, an event would be whether or not a recognition/understanding error occurred. Counts are obtained for both specific and more general contexts, so that it is possible to *back-off* to a more general case whenever the number of training examples falls below a certain threshold.

The user simulation and error model are shown to simulate different scenarios well enough to perform relative predictions of their durations. Hence some evaluation is undertaken in order to assess how realistic the user and error simulations are, but being based only on the gross metric of dialogue duration, this evaluation is not very convincing. How best to evaluate the accuracy of user simulations is in fact an open research question (see Schatzmann et al. 2006), and it is an issue which we will refer to again in Section 3.10.

3.6.4 Experimental results

In evaluating the learned strategies, Scheffler and Young (2002) used two hand-crafted strategies as baselines. The first used a small state space containing slot-status features and feature 1 from Section 3.6.1, while the second used a large state space which included additional features such as a confidence feature, and a feature which counts how many times each slot has been asked. The performance of the learned and hand-crafted strategies was evaluated in test dialogues with the user simulation according to the reward function in Section 3.6.2. Of the learned strategies, those learned with larger state spaces tended to outperform those learned with smaller state spaces, but the improvement was not great. The learned strategies outperformed the small-state-space-hand-crafted strategy by a large margin, and performed roughly as well as the large-state-space-hand-crafted system.

Like Levin et al. (2000) and Pietquin and Renals (2002), this work then shows that reasonable strategies can be learned using simulated users, but again:

[6] $P(Event|Context) = \frac{count(Event,Context)}{count(Context)}$

- testing and training are performed with the same user simulation,
- no tests with real users were performed,
- the quality of the simulated users is not established convincingly - we are only told that the user simulation and error model are shown to simulate different scenarios well enough to perform relative predictions of their durations,
- the learned strategies are not shown to be better than hand-coded strategies.

We now discuss an alternative approach which treated both simulated user and dialogue manager as RL systems.

3.7 Reinforcement Learning for both user and system: English and Heeman 2005

English and Heeman (2005) use Reinforcement Learning (RL) to learn the system dialogue strategy for a collaborative task which requires the system and user to agree on 5 pieces of furniture to place in a room. Both the system and user have private preferences about which furniture items they want in the room e.g. “if there is a red couch in the room, I also want a lamp”. The main novel feature of this work is that RL is used to learn the user strategy simultaneously. The same RL algorithm is used for both strategies - an on-policy Monte Carlo (MC) method. The authors argue that their approach is preferable to the more generally-accepted approach of using a stochastic user simulation for which the probabilities have already been derived from a human-human or human-machine dialogue corpus. They state that two disadvantages of the more generally-accepted approach are:

1. significant time and effort is required to collect the sample dialogues for the dialogue corpus, and then model user behaviour to produce a user simulation,
2. the strategy that can be learned for the system is limited by the complexity and flexibility of the simulated user.

They claim that their approach avoids these disadvantages, but as will be explained in Section 3.7.4, it seems to be fundamentally flawed if it is to be used for learning dialogue strategies for interacting with real users.

3.7.1 State features and action sets

The state representation for each of the system and user agents includes the following binary features:

1. Pending-Proposal,
2. I-Proposed,
3. Violated-Preference,
4. Prior-Violated-Preferences,
5. Better-Alternative.

A **Pending-Proposal** indicates whether an item has been proposed but not accepted or rejected. **I-Proposed** indicates whether the agent made the most recent proposal. **Violated-Preference** indicates that the pending proposal has caused one or more violations of the conversant's private preferences. **Prior-Violated-Preferences** indicates whether the conversant had one or more violated preferences when the pending proposal was made. **Better-Alternative** indicates that the agent thinks it knows an item that would achieve a better score than the item currently proposed.

The action set for each of the system and user agents includes:

1. propose,
2. accept,
3. reject,
4. inform,
5. release turn.

The **propose**, **accept** and **reject** actions refer to proposing, accepting and rejecting different items of furniture for the room. An agent uses the **inform** action to inform the other conversant of preferences that are violated by the current proposal. Since a turn does not finish until the speaker uses the **release turn** action, a single turn can include multiple actions e.g. a **reject**, followed by an **inform** and then a **propose**.

3.7.2 Reward functions

The agents only receive non-zero rewards at the end of each dialogue. The reward function is a linear combination of the solution quality (S) and the dialogue length (L), taking the form:

$$Reward = w_1 S - w_2 L \tag{3.1}$$

where w_1 and w_2 are positive constants. The authors explore the effects of different values for the constants.

3.7.3 User simulation using Reinforcement Learning

The user simulation is also an RL agent and so which action it takes at any given time is determined by:

1. the action selection method e.g. ε-greedy and the relevant parameter's value e.g. ε (see Sutton and Barto (1998) or Section 2.9.1),
2. the Q-values for the different actions in the current state.

3.7.4 Experimental results

English and Heeman report that they succeeded in learning system and user dialogue strategies that achieved comparable performance with hand-crafted system and user strategy pairs. The authors also claim that the learned system strategies are robust - when the learned system strategies "conversed" with the hand-crafted user strategies, the resulting dialogues had comparable solution quality to what the hand-crafted and user strategies achieved together. They acknowledge that there was a lack of convergence in the Q-values over a number of learning trials, presumably because the Reinforcement Learning (RL) problem becomes more complex with two interacting learning agents.

However, if the goal is to produce system strategies for interacting with real users, then the approach of using RL to simultaneously learn both the system and user strategies seems to be fundamentally flawed. To learn a strategy which works well with real users, we need to train with a simulation which accurately simulates real users. For

example, we will see in the description of the experimental work of this thesis, that real users seem to respond better to different repair strategies in different contexts. Hence, unless we train with an accurate simulation, we cannot expect to learn these appropriate repair strategies. Therefore, it seems that the type of co-training used by English and Heeman cannot be relied on to produce optimal strategies for real users.

We now discuss work which presents a method for automatically selecting features to include in the state, and which investigates alternatives to Markov Decision Processes (MDPs) for modelling the dialogue management problem.

3.8 Alternative learning approaches and feature selection: Paek and Chickering 2005

Alternatives to Markov Decision Processes (MDPs) for modelling the dialogue management problem have been investigated by Paek and Chickering (2005). Unlike MDPs, their models do not constrain the state space by the Markov assumption, (see Section 2.4). Paek and Chickering are interested in whether it is possible to learn better strategies with these alternative models, i.e. ones which obtain higher reward. Paek and Chickering also present a data-driven method for identifying which features should be represented in the MDP state. This data-driven method generalises to the alternative non-Markovian models. First, an MDP is viewed as a special case of an *influence diagram*, which is a more general framework for graphical modelling that facilitates decision-theoretic optimisation. There are techniques for learning the parameters and structure of a Bayesian Network [7] that have been extended for influence diagrams (Heckerman, 1995, Chickering and Paek, 2005), and it is possible to use these in order to learn which features should be represented in the state.

In this way Paek and Chickering (2005) learned strategies for a speech-enabled web browser. The data was generated using a simulation environment where all possible system actions relating to a user command were systematically explored. Dialogues were limited in length to 3 system turns due to the typically low tolerance users have in command-and-control settings for extended repairs. The authors state that using Dynamic Programming (DP) for a state space that includes more than a handful of variables can be computationally expensive. Hence they use *forward-sampling* to ap-

[7] See Section 3.3 of Pietquin (2004) for an introduction to Bayesian Networks.

proximate the DP solution for the MDP (Kearns et al., 1999).

3.8.1 State features and action sets

The potential state features in the data fell into the following three broad categories:

1. **Within-utterance ASR features**: features pertaining to a single utterance such as the number of hypotheses in an n-best list [8] of variable length, the mean of the Confidence Levels (CLs) etc.,
2. **Between-utterance ASR features**: features pertaining to matches across utterances, such as whether the top rule in the n-best list matched the previous top rules, etc.,
3. **Dialogue features**: features pertaining to the overall dialogue such as the number of repairs so far, whether the system has engaged in a confirmation yet, etc.

Four different Dialogue Manager (DM) models were constructed:

1. a zero-order Markov model,
2. a first-order Markov model, i.e. an MDP,
3. a second-order Markov model,
4. a cumulative total reward model.

As we already know, in an MDP, a time slice's state variables depend on those from the previous time slice. For a second order Markov model, the third time slice state variables can also depend on those in the first time slice, while for a zero-order Markov model, there are no dependencies between time slices. For a cumulative total reward model, state features accumulate for each slice.

Here is a summary of the state features which Paek and Chickering (2005)'s data-driven method learned should be represented in the MDP:

1. a number of features related to the n-best list Confidence Levels (CLs),

[8] A speech recogniser may provide a list, in order, of its top *n* hypotheses for a user utterance according to their Confidence Levels (CLs).

2. the mean CL in the n-best list,
3. the sum of all the CLs from the n-best list,
4. range of score values from n-best list,
5. whether all the rules in the list were the same though the actual phrases or wording were different,
6. the grammar rule that was observed, e.g. the first or top rule in the n-best list.

Hence this includes a number of features related to the n-best list CLs, which seems sensible given that most hand-crafted dialogue management strategies use some kind of confidence threshold for taking actions, e.g. "Do the top recognised command if its confidence is greater than 95%". When applied to the other models, Paek and Chickering's method learned different feature sets, e.g. feature sets including between-utterance Automatic Speech Recognition (ASR) features.

The action choices for the first turn were:

1. **DoTop** - execute the most likely command in the n-best list,
2. **Confirm** - confirm among the top three choices while giving the option that it may not be any of them,
3. **Ignore** - ignore the utterance as spurious,
4. **Repeat** - ask for a repetition.

For the second turn they were:

1. **DoTop**,
2. **Confirm**,
3. **Repeat**.

Finally, the action choices for the third turn were:

1. **DoTop**,

2. **Bail** - make an apology and terminate the dialogue.

In generating the training data, all possible system actions were explored for each user command - Section 3.8.3 will provide more details about this training simulation environment.

3.8.2 Reward functions

If the simulation selected either the DoTop, Ignore or Bail action, then the session finished and a reward was given. When the final action was DoTop, if the system executed the correct command, then it received a reward of $+100$, else it received -100. If the final action was Ignore and there was no command, then it received $+100$, else -100. For Bail, it received -100. If either the repair action Confirm or Repeat was selected, a penalty of -75 was received.

3.8.3 User simulation

The simulation randomly selects a command from the command-and-control grammar for the browser (e.g., “go back”, “go forward”, “go to link x”). Using state-of-the-art Text-To-Speech (TTS) generation, an utterance is then produced for the command, varying all possible TTS parameters, such as engine, pitch, rate and volume. Since Paek and Chickering (2005) were interested in building models that were robust to noise, they included empty commands and added various types of background noise to see if a model could learn to ignore spurious commands. The produced utterance was then recognised by a Microsoft Speech API (SAPI) recognition engine. All possible SAPI events were logged, and these events, and functions of these events constituted the potential state feature set already provided in Section 3.8.1.

3.8.4 Experimental results

Paek and Chickering (2005) found that the strategy learned for the cumulative total reward model outperformed the strategies learned for the other models, including the MDP. Hence it seems that the cumulative total reward model may offer an attractive alternative to the MDP. However Paek and Chickering say they cannot draw any strong conclusions for the following reasons:

1. The domain was small with a relatively restricted command-and-control grammar and a small action space.

2. It is possible that certain features which would have enabled the MDP to perform best were not annotated.

3. The results depend on the techniques used for learning the structure of the state space and so these results may have turned out differently with other model selection techniques. Feature selection techniques such as Correlation-based Feature Selection (CFS) subset evaluation Hall (1999) offer an alternative. [9]

The work described here in this section proposed a method for automatically identifying features which should be included in the state. We now present some alternative work which has used RL itself to assess the importance of different state features. The basic methodology is to learn strategies with different state features, and to then subject these strategies to quantitative and qualitative analysis.

3.9 Feature selection in RL for tutorial dialogue systems: Tetreault and Litman 2006

Like (Singh et al., 1999, 2002, Walker, 2000), Tetreault and Litman (2006) also describes an experiment in which a partial strategy is learned based on data generated by real user interactions. However the work of Tetreault and Litman has two main novel features. Firstly, the utility of three new state features is investigated by learning action choices with and without each of these features, and then subjecting the learned strategies to quantitative and qualitative analysis. Secondly, the action choices are not learned for a slot-filling system, but instead for a Spoken Dialogue System (SDS) which acts as a tutor for undergraduate-level physics i.e. the ITSPOKE SDS (Litman and Silliman 2004). The action choices learned in appropriate states relate to whether to ask the user a question, and if so, what kind of question e.g. a simple versus a more complex question. The first state feature whose utility is investigated represents whether the system is forced to re-visit a particular concept, the second, how frustrated the user seemed to be in their last response, and the third is a measure of the student's

[9] As potential future work, Section 8.2 proposes a possible method for applying CFS to select a subset of contextual features to include in the RL state.

performance thus far on the current problem. Note that these features could potentially be important to any SDS. Tetreault and Litman focus on question strategies because what type of question a tutor should ask is of great interest to the Intelligent Tutoring Systems community. In addition, it is also possible that the results will generalise, because in any domain, asking users questions of varying complexity is likely to elicit different responses.

The action choices are learned by applying the Dynamic Programming (DP) algorithm *policy iteration* to an annotated corpus of 20 human-ITSPOKE SDS sessions. Each of these 20 sessions consists of an interaction with one student over 5 different physics problems, so giving a total of 100 dialogues. Before each session, the student read physics material for 30 minutes and then took a pretest based on that material. The system starts each dialogue by giving the student a problem and then the student writes a short essay response. The system assesses the essay for potential flaws in the reasoning and then asks questions to help the student understand the confused concepts. Its next action e.g. question is based only on the correctness of the student's last answer, and once the student has successfully completed the dialogue, they are asked to correct the initial essay. Finally, at the end of a session, the student is given a post-test similar to the pre-test, and this is then used to calculate the student's *normalised learning gain*:

$$Gain = \frac{posttest - pretest}{1 - pretest} \tag{3.2}$$

This is a standard evaluation metric in the intelligent tutoring systems community.

3.9.1 State features and action sets

The state is always represented by a subset of the following features, which all relate to the student (user):

1. **Correctness** - Correct (C),Incorrect or Partially Correct (IPC),
2. **Certainty** - Certain (cer),Uncertain (unc),Neutral (neu),
3. **Concept Repetition** - Concept is new (0),Concept is repeated (R),
4. **Frustration** - Frustrated (F),Neutral (N),
5. **Percent Correct** - 50-100% = (H)igh, 0-49% = (L)ow).

Forbes-Riley and Litman (2005) describes how the emotion-related features, **Certainty** and **Frustration**, were annotated manually in the corpus. **Certainty** describes how confident a student seemed to be in their answer, while **Frustration** describes how frustrated the student seemed to be when they responded. The other three features, (**Correctness**, **Concept Repetition**, **Percent Correct**) are automatically extracted. **Correctness** describes whether the student was correct or not, **Percent Correct**, the percentage of correctly answered questions so far for the current problem, and **Concept Repetition**, whether the system is forced to cover a concept again which reflects an area of difficulty for the student. Tetreault and Litman (2006) were interested in the utility of **Concept Repetition**, **Frustration**, and **Percent Correct**. Hence they learned a baseline with only features 1 and 2 in the state, and then three further strategies with features 1 and 2 and then one of 3 to 5.

As stated previously, where it is thought appropriate to ask a question, Tetreault and Litman use RL to learn to choose between the following actions:

1. Simple Answer Question (SAQ) e.g. “Good. What is the direction of that force relative to your first?”,
2. Complex Answer Question (CAQ) e.g. “What is the definition of Newton’s Second Law?”,
3. Mix of SAQ and CAQ e.g. “Good. If it doesn’t hit the centre of the pool what do you know about the magnitude of its displacement from the centre of the pool when it lands? Can it be zero? Can it be nonzero?”,
4. No Question (NoQ) e.g. “So you can compare it to my response...”.

For other states, the action is fixed as either giving some kind of feedback or some other type of helpful measure e.g. a hint or a restatement.

3.9.2 Reward functions

The 10 students with the highest normalised learning gain scores were labelled high learners and their respective 5 dialogues were given a final reward of 100. The other ten students were labelled low learners and their respective dialogues were given a final reward of -100.

3.9.3 Experimental results

Tetreault and Litman (2006) provide qualitative analysis of the effects of adding the new state features. They state that in general, the **Concept Repetition** feature causes the reinforcement learner to learn a Complex Answer Question (CAQ) after a concept has been repeated, and especially if the student is correct when addressing a question about the repeated concept. This seems to make sense, because if a concept has been repeated, this should signal that the student did not grasp the concept, and a clarification dialogue was initiated to help the student understand it better. Once the student answers the repeated concept correctly, this indicates that the student understands the concept and that the tutor can once again ask more difficult questions to challenge the student. The **Frustration** feature changes the policies most when the student is frustrated, but when the student is not frustrated (*neutral*), the policy stays the same as the baseline with the exception of when the student is either *Correct* and *Certain*, or *Incorrect* and *Uncertain*. Finally, the **Percent Correctness Feature** does not produce a large policy change - most learned actions remain "Mix of SAQ and CAQ", just as they were for the Baseline.

Tetreault and Litman go on to provide quantitative analysis of the learned strategies in order to compare the utility of the three new state features. Three metrics are used: (1) "Diff's" (2) % Policy Change and (3) Expected Cumulative Reward (ECR). The number of Diff's is the number of states whose learned action differs from the Baseline. % Policy Change weights each difference by the number of times that state-action sequence actually occurs in the data and then divides by the total number of state-action sequences. This more accurately measures the utility of a feature because although a first feature may produce a higher number of Diff's than a second feature, its overall impact could actually be lower if the states which it affects occur less frequently. However, % Policy Change still does not take into account the effect which the state feature has on the state values, (see Section 2.6). The third metric, ECR, does take this into account. ECR is calculated by normalising the value of each state by the number of times it occurs as a start state in a dialogue and then summing over all states. Tetreault and Litman report that according to all three metrics, **Concept Repetition** has the greatest utility, followed by **Frustration**, and then **Percent Correctness**. For example, the **+ Concept Repetition** strategy obtained an ECR of 39.52, the **+ Frustration** strategy, 31.30, and the **+ Percent Correctness**, 28.17. Note that the learned strategies are not compared to a hand-crafted strategy.

This work then demonstrates a simple but useful approach to performing feature selection for state features. As stated in section 3.8.4, the use of automatic feature selection methods such as Correlation-based Feature Selection (CFS) subset evaluation Hall (1999) is very much an open problem.

3.10 Learning in large state spaces via a generalisation method: Henderson, Lemon and Georgila 2005-2008

One of the main themes that we have observed in the above survey is the use of relatively limited state spaces, which do not include much linguistically-motivated detail regarding the dialogue context. This restriction has been due to researchers wishing to avoid the "curse of dimensionality" (Bellman, 1957), in which learning in large state spaces becomes intractable. One approach to large state spaces is to use *generalisation* methods in which previously unobserved states are seen as "similar" (by some metric) to states which have been observed. This section will describe Henderson et al. (2005, 2008) (henceforth "HLG") - work which explores using a generalisation method called *linear function approximation* in order to cope with a very large state space that includes linguistically-motivated features e.g. the entire dialogue history represented as Dialogue Acts (DAs).

HLG train a strategy on data collected with real users of the DARPA COMMUNICATOR systems. The COMMUNICATOR corpora (Walker et al., 2001a, 2002) consist of approximately 2300 "slot-filling" human-machine dialogues in the travel-booking domain. The systems in these corpora play the role of a travel agent, and the user always tries to book either a single or return flight, and sometimes also tries to book a hotel or rent a car. The data contains PARADISE (Walker et al., 2000) evaluation scores, which HLG use as reward. Recall that PARADISE was introduced in Section 3.4.2. Prior to applying Reinforcement Learning (RL), an automatic system was used to assign DATE Dialogue Act (DA) tags (Walker et al., 2001b) to the user utterances, and to compute *information states*, (representations of the dialogue context), for each dialogue turn. Section 3.10.1 will introduce DATE. The new contextual features produced a very large state space: 10^{386} states are theoretically possible. Since the number of dialogues was relatively small, but the state-action space extremely large, the data only

provided information about a small portion of the state-action space. To address this problem, HLG applied a hybrid learning model which combines RL with Supervised Learning (SL), (see Section 1.1.2 for a definition of SL), where the role of the SL component is to restrict the learned strategy to the portion of the space for which there is data. Hence, like model-based approaches to learning dialogue strategies, HLG learn a strategy from a fixed dataset, but their use of a Hybrid learning approach and a very large state-action space are new. Note that Henderson et al. (2005) reports preliminary results for an experiment which used the same methodology as Henderson et al. (2008), but was only conducted on a subset of the 2001 COMMUNICATOR corpus, and hence produced inferior learned strategies.

3.10.1 DATE Dialogue Act tagging scheme

This section describes DATE, (Dialogue Act Tagging Scheme for Evaluation of Spoken Dialogue Systems), (Walker and Passonneau, 2001), (see Section 1.2 for a definition of a Dialogue Act (DA)). DATE was developed to provide finer-grained quantitative dialogue metrics for comparing and evaluating COMMUNICATOR Spoken Dialogue Systems (SDSs). DATE tags each utterance according to three dimensions:

1. SPEECH ACT,
2. TASK/SUBTASK,
3. CONVERSATIONAL DOMAIN.

The SPEECH ACT dimension characterises the utterance's communicative goal, and the different types are:

1. REQUEST-INFO,
2. PRESENT-INFO,
3. OFFER,
4. ACKNOWLEDGEMENT,
5. STATUS-REPORT,
6. EXPLICIT-CONFIRM,

7. IMPLICIT-CONFIRM,
8. INSTRUCTION,
9. APOLOGY,
10. OPENINGS/CLOSINGS.

The TASK/SUBTASK dimension gives the task or subtask to which the utterance relates. This dimension is present so that the amount of effort a system expends on particular sub-tasks can be quantified. Below are the different tasks/sub-tasks:

1. TOP-LEVEL-TRIP,
2. ORIGIN,
3. DESTINATION,
4. DATE,
5. TIME,
6. AIRLINE,
7. TRIP-TYPE,
8. RETRIEVAL,
9. ITINERARY,
10. GROUND,
11. HOTEL,
12. CAR.

Finally, the CONVERSATIONAL DOMAIN dimension characterises the utterance as relating to one of the following three conversational domains:

1. ABOUT-TASK,
2. ABOUT-COMMUNICATION,

3. ABOUT SITUATION FRAME.

An ABOUT-TASK utterance will typically directly ask for or present task-related information, or offers a solution to a task goal. ABOUT-COMMUNICATION utterances are concerned with managing the verbal channel and providing evidence of what has been understood, e.g. via implicit confirmation. Finally, the SITUATION-FRAME domain pertains to the goal of managing the culturally relevant framing expectations. Most of the ABOUT-FRAME DAs fall into the speech-act category of INSTRUCTIONS, utterances directed at shaping the user's behaviour and expectations about how to interact with a machine.

3.10.2 State features and action set for Linear Function Approximation

As stated previously, prior to learning the strategy, HLG first annotated the COMMUNICATOR data with additional information. An automatic system which was implemented using DIPPER (Bos et al., 2003) and several OAA agents (Cheyer and Martin, 2001) was used to assign DATE tags to the user utterances, and to compute information states for each point in the system, (see Georgila et al. (2005b) for more details). The new state features produced a very large state space - as already stated, 10^{386} are theoretically possible), and there were 74 different system actions.

State features relate to the following:

1. Word Error Rate (WER),
2. complete speech acts history,
3. complete tasks history,
4. complete filled slots history,
5. complete filled slots values history,
6. complete grounded slots history.

The majority of the 74 different action choices are for asking or confirming a slot value, where confirming can be implicit or explicit. Other actions include querying the database of flights/hotels/cars, presenting database query results to the user, and giving some kind of help to the user.

3.10.3 Reward function

For learning the strategy, each system turn received a reward of -1, and the user satisfaction score was the dialogue final reward. The PARADISE user satisfaction features listed in Section 3.4.2 were used to compute this user satisfaction score. They were summed with the following positive weights:

1. Actual Task Completion: 100,
2. Perceived Task Completion: 100,
3. Task Ease: 9,
4. Comprehension Ease: 7,
5. System behaved as Expected: 8,
6. Future Use: 9.

The weights for features $3-6$ were determined by the application of PARADISE to the COMMUNICATOR data, reported in Walker et al. (2001a).

When testing the strategy in simulation, HLG used a reward function based only on dialogue length and task completion. We refer to this reward function as "HLG05", (it is first introduced in Henderson et al. 2005), and it is as follows:

1. Database query, $+25$ for each filled slot, another $+25$ for each slot which is confirmed,
2. System turn penalty: -1.

The training reward function could not be used when testing in simulation because the simulations do not generate PARADISE user satisfaction scores. Learned strategies are also evaluated with the HLG05 reward function in this thesis. This is so as to enable performance comparisons with the Hybrid strategy and the hand-crafted COMMUNICATOR systems.

3.10.4 Stochastic User simulations

HLG used a user simulation in order to test the learned strategy. This user simulation was trained on the COMMUNICATOR data using linear function approximation. The simulation is given a vector of features representing the current context of the dialogue, and outputs one or more actions, which consist of DATE speech-act-task pairs. Automatic Speech Recognition (ASR) and Natural Language Understanding (NLU) errors are incorporated since the model is built from the user utterances as they are recognised by the ASR and NLU components of the original COMMUNICATOR systems. Note that the user simulation does not provide instantiated slot values e.g. a response to provide a destination city is the speech act-task pair *"[provide info] [dest city]"*. It cannot be assumed that two such responses in the same dialogue refer to the same destination cities, and so slot-status features in the Dialogue Manager's (DM's) information state are only updated from *filled* to *confirmed* when the slot value is implicitly or explicitly confirmed.

HLG also report how the COMMUNICATOR data was used to estimate the parameters for user simulations based on n-grams. These n-gram models take as input the speech act-task pairs of the $n-1$ most recent turns in the dialogue history and then output a user utterance as one or more new speech act-task pairs. They simulate ASR and NLU errors in the same way as the linear function simulation, and also like the linear function simulation, they do not provide instantiated slot values. A criticism of some stochastic user models is that they do not do a good job of simulating the different types of user in the data from which they are derived (e.g. users with varying levels of expertise and cooperativeness), and instead simulate an average user [10]. However, this criticism is perhaps less true of n-gram user models. This is because an n-gram model generates a new action based on some amount of dialogue history, and so can generate a new action which bears common traits with previously generated actions. The degree to which it is able to do this is obviously limited by the size of n. We use the n-gram user simulations introduced here for training and testing strategies in experimental work in Chapters 5 and 7. Hence, in Chapter 5, we discuss how the accuracy of these simulations is limited by the size of n, and by the fact that they only use DA information. We also discuss the implications which these limitations have for learning dialogue strategies.

[10] This is obviously also a very important issue for how best to evaluate the accuracy of user simulations.

In order to establish their accuracy, Georgila et al. (2005a,b) evaluated the linear function approximation and the n-gram simulations on the further-annotated COMMUNICATOR data using *perplexity*, (see Schatzmann et al. (2006) for a formal definition of perplexity). Perplexity is a measurement in Information Theory which is used here for determining whether the simulated dialogues contain similar action sequences (or dialogue state sequences) to the real human-computer dialogue in the COMMUNICATOR data. If they do, then the resulting perplexity will be lower. However, perplexity is not necessarily a good indicator for how likely the user model is to predict a realistic response in the context of an unseen dialogue situation. This is problematic, since in simulation-based learning, our goal is to apply new strategies to the user model, and there is no guarantee that a user model with a low perplexity will produce reasonable user responses in such situations. Nevertheless, the model based on linear feature combination gave the best *perplexity*, followed by the 4-gram. Each one of the user models was also run against a system strategy learned with purely Supervised Learning, (linear function approximation), no RL. The quality of the simulated dialogues produced was then measured as a function of the filled slots, confirmed slots, and number of actions performed by the system in each dialogue. In this experiment, both the linear feature combination model and the best n-grams (5-gram and 4-gram) produced similar results. Hence the accuracy of the simulations is evaluated more thoroughly here than in Scheffler and Young (2002), (see Section 3.6.3). Recall that Scheffler and Young only showed that their user and error models simulate different scenarios well enough to perform relative predictions of their durations. As stated before, how best to evaluate the accuracy of user simulations is an open and active research area (see Schatzmann et al. 2006).

3.10.5 Experimental results

A dialogue strategy was learned from the further annotated COMMUNICATOR data, and then tested with the linear function approximation user simulation. The strategy outperformed all of the COMMUNICATOR systems - it scores 35.42% higher than the average COMMUNICATOR system. The authors find that when the relative importance of the Reinforcement Learning (RL) and Supervised Learning (SL) components are adjusted, the best hybrid policy performs 302% better than the standard RL policy, and 1.4% better than the supervised policy. Lemon et al. (2006a) shows that the Hybrid Strategy also outperforms a state-of-the-art hand-crafted system when tested

on real users with the "TownInfo" Spoken Dialogue System (Lemon et al., 2006b). Since the "TownInfo" system operates in a different domain, (the Tourist Information domain as opposed to the COMMUNICATOR domain), this result also demonstrates that it is possible to learn effective generic slot-filling strategies.

These results apparently show that large state spaces can be handled, and that good policies can be learned by doing so. However, the approach of simply using all available state information for COMMUNICATOR systems does not tell us which aspects of the state (in particular, of the dialogue history) are really important for the dialogue management task. Additionally, no qualitative analysis of the learned strategy is provided. Despite the positive results described above, the very small improvement of the hybrid policy over the purely supervised policy suggests that it is sub-optimal. It would also be interesting to know whether this hybrid strategy is better than what could have been learned using a smaller state-space and standard RL with or without a generalisation method. These are all issues which will be addressed by the experimental work of this thesis.

We now provide a summary and comparison of the research described in this survey, and open problems for the field.

3.11 Summary and open problems

This chapter has summarised the previous research in using Markov Decision Processes (MDPs), Reinforcement Learning (RL), and related methods, to learn dialogue strategies for Spoken Dialogue Systems (SDSs).

Most of the research described in this chapter involved learning strategies for systems which filled slots before querying some information source (e.g. a database), and presenting the results to the user. The exceptions were Tetreault and Litman (2006)'s ITSPOKE physics tutor system, English and Heeman (2005)'s system for collaborating in deciding how to arrange furniture in a room, and Paek and Chickering (2005)'s speech-enabled web browser.

As discussed above, the main dimensions in which approaches differ are:

- size and detail of the state spaces,
- whether Dialogue Acts (DAs) are represented in the state,

- size and complexity of action sets,
- amount of the state-action space actually explored in system training/ versus amount hand-coded,
- use of simulations or real user data for training,
- use of an Automatic Speech Recognition (ASR) simulation for training,
- evaluation with real or simulated users,
- statistical significance of results,
- learning methods e.g. Dynamic Programming (DP) versus Monte Carlo (MC) versus Temporal Difference learning (TDL), RL with and without a generalisation method,
- application domains (search, browsing, tutorial, negotiation....),
- reward functions: data-driven (e.g. PARADISE) versus hand-coded,
- degree to which feature selection methods are employed.

Table 2 compares the research based on important elements of the experimental methodology. We now summarise the various options covered in the prior work.

3.11.1 Learning methods

Researchers followed one of two general methods, these being:

1. **Model-based approaches**: Learn partial strategies from exploratory data generated by dialogues with real users, (Singh et al. 1999, 2002, Walker 2000, Tetreault and Litman 2006). (Generating dialogues with real users will generally give an insufficient number for learning full strategies). Due to the smaller number of variables, Dynamic Programming (DP) can be an option for the learning algorithm.
2. **Simulation-based approaches**: Learn full strategies from exploratory data generated by dialogues with simulated users, (Levin et al. 2000, Pietquin and Renals

Research group	**Slot fill**	**Full strats**	**Data-driven sims**	**DAs in state**	**Feature analysis**	**Real user tests**
Levin & Pieraccini 1997-2000	✓	✓	×	×	×	×
Singh et al. 1999-2002	✓	×	×	×	×	×
Walker 2000	✓	×	×	×	×	×
Pietquin & Renals 2002	✓	✓	×	×	×	×
Scheffler & Young 2002	✓	✓	✓	×	×	×
English & Heeman 2005	×	✓	×	×	×	×
Paek & Chickering 2005	×	✓	×	×	×	×
Tetreault & Litman 2006	×	×	×	×	✓	×
Henderson et al. 2005-2008	✓	✓	✓	✓	×	✓

Table 3.2: Comparison of work of different research groups:
"Slot-fill" column indicates whether the strategy was learned for a slot-filling system or not;
"Full-strats" indicates whether full as opposed to partial strategies were learned;
"DAs in state" indicates whether Dialogue Acts were represented in the state;
"Data-driven sims" indicates whether stochastic user and error simulations, probabilities for which are learned from data, were used to learn the strategy;
"Feature analysis" indicates whether there was both quantitative and qualitative analysis of the effect of different state features on the learned strategy;
"Real user tests" indicates whether the learned strategy was tested with real users and its performance compared to a state-of-the-art hand-crafted strategy.

2002, Scheffler and Young 2002, English and Heeman 2005, Paek and Chickering 2005). A larger number of variables means some approximation to DP must be applied, e.g. Monte Carlo (MC) or Temporal Difference Learning (TDL).

The exception to this is HLG. Like a model-based approach, HLG do not use a user simulation for training, and instead, learn a dialogue strategy from a fixed dataset of real user dialogues. However, they do not try to directly model the state transition probabilities and apply DP. Instead they use an alternative learning approach which means that they can learn full strategies with a very large state-action space: "hybrid" Supervised Learning/RL (TDL) with a generalisation method.

3.11.2 Comparing the State spaces

In constructing their state spaces, researchers must try to include as much important contextual information as possible without making it too large and making the RL problem intractable. All of the researchers apart from Paek and Chickering (2005) hand-picked their state features. Paek and Chickering treated the MDP as a special case of an *influence diagram*, and then applied techniques for learning which features should be represented in the state. Of those that learned full strategies for slot-filling systems (Levin et al. 2000, Pietquin and Renals 2002, Scheffler and Young 2002), only used slot-based features, e.g. whether a slot is filled, any associated confidence score etc. HLG used additional features e.g. features for the complete dialogue history, which produced a very large state space and necessitated using a hybrid supervised RL method with linear function approximation. Tetreault and Litman (2006) learned strategies in the tutorial domain, and used state features for user **Correctness** (in their most recent answer), user **Certainty**, **Concept Repetition** (whether the system has repeated a concept), user **Frustration** and **Percent Correct** (the proportion of questions on the current topic answered correctly by the user).

3.11.3 Different reward functions

Most of the researchers use a simple reward function which rewards *task completion* and penalises *dialogue length*. Walker (2000) and HLG use data which contains PARADISE (Walker et al., 2000) user satisfaction scores at the end of each dialogue, and so use this measure of user satisfaction as their reward function.

3.11.4 User simulation and error modelling methods

Of the user simulations used by researchers, all are at least partially-stochastic and are capable of simulating mixed-initiative dialogues. The user simulations of Levin et al. (2000), Pietquin and Renals (2002) and Scheffler and Young (2002) supply actual slot values, and so have a deterministic element to ensure that their behaviour is consistent within a dialogue. The simulations developed by HLG, (linear function approximation and n-grams derived from COMMUNICATOR data Walker et al. 2001a) do not supply actual slot values and so do not have this deterministic element. All of the probabilities for the user simulations of Scheffler and Young (2002) and HLG are learned from data,

and some of the probabilities for the user simulation of Levin et al. (2000) are. The probabilities for the simulation of Pietquin and Renals (2002) are entirely supplied using intuition.

To simulate ASR/NLU errors Pietquin and Renals (2002), Scheffler and Young (2002) and HLG also have a probabilistic error model. The probabilities for the error model of Scheffler and Young and HLG are learned from data, while the probabilities for the error model of Pietquin and Renals are supplied using intuition.

Efforts were made to establish the accuracy of the simulations used by Scheffler and Young, and those used and developed by HLG: linear function approximation simulation and n-gram simulations derived from COMMUNICATOR data (Walker et al., 2001a). Scheffler and Young showed that their user and error models simulate different scenarios well enough to perform relative predictions of their durations. The accuracy of the linear function approximation and n-gram simulations is evaluated more thoroughly (Georgila et al. 2005a,b). These simulations were evaluated on the COMMUNICATOR data in terms of *perplexity*. Each one of the user models was also run against a system strategy learned with purely Supervised Learning (SL), (linear function approximation), and the quality of the simulated dialogues which this produced was then measured as a function of the filled slots, confirmed slots, and number of actions performed by the system in each dialogue.

3.11.5 Open problems

The prior work brings to light a number of open problems for the field, which include:

- Can we develop user simulations which are accurate enough to reliably train RL Dialogue Managers (DMs)? What metrics can be used to evaluate user simulations?
- Using data-driven user simulations and error models, can a reinforcement learner learn full dialogue strategies which outperform state-of-the-art hand-crafted strategies in evaluation with real users?
- Can linguistically motivated features e.g. Dialogue Acts (DAs) enable the reinforcement learner to learn better strategies?

- If the answer to the above question is yes, why are these new strategies better, and what does this tell us about human-machine dialogue and dialogue in general.
- Is additional contextual information only useful in certain portions of the policy space? If it is, then this would be a useful finding - the additional contextual information could be sacrificed where it is known to be redundant, so helping to reduce the size of the policy search-space.
- Speech recognition/natural language understanding errors are a major obstacle in human-machine dialogue, and better repair/error-recovery strategies should lead to significant improvements in overall performance. Given that we are already using slot-status features, can additional context be used to learn better repair/error recovery strategies?
- If additional contextual information does lead to better repair strategies, what are these better repair strategies, and why are they better?
- Can automatic feature selection methods be applied to dialogue data to provide an automatic method for identifying the important contextual features?

The reader should also be aware that this chapter did not describe very recent work by Williams and Young (2007) which used *Partially Observable MDPs (POMDPs)* for learning dialogue strategies. A POMDP is an extension of an MDP, and is used for choosing actions when the entire world, or state-space is not always directly observable. Since the true state of the world cannot be uniquely identified, a POMDP reasoner must maintain a probability distribution, called the *belief state*, which describes the probabilities for each true state of the world. Maintenance of the belief state is Markovian in that it only requires knowledge of the previous belief state and the action taken. POMDPs are therefore able to handle uncertainty in a principled way, and since Automatic Speech Recognition (ASR) and Natural Language Understanding (NLU) are error prone, this makes them theoretically appealing for dialogue management. However, at present POMDPs are computationally intractable to solve for optimal behaviour for dialogue problems of realistic size. This then is another very interesting avenue for future research.

3.12 Conclusion

We summarised and analysed the work of the different research groups who have made significant contributions in using Reinforcement Learning (RL) techniques to learn dialogue strategies for Spoken Dialogue Systems (SDSs). We surveyed the most important developments in this emerging field, and the open research issues. Given the research content of this thesis, our primary focus was on the different RL systems themselves, but where appropriate, we also provided discussion related to user simulation approaches.

This then ends the portion of this thesis which is devoted to background material, and so in the next chapter, we begin description of the research content.

Chapter 4

The Reinforcement Learning setup and proof-of-concept experiments

4.1 Introduction

This chapter first introduces the experimental setup which we use for Reinforcement Learning (RL) of full dialogue strategies. This setup involves a reinforcement learner, a Dialogue Manager (DM), and a stochastic user simulation. The reinforcement learner controls the action choices of the DM, and learns a full dialogue strategy based on the resulting interactions between the DM and the user simulation. This chapter also goes on to describe three preliminary RL experiments which use this experimental setup in order to learn full dialogue strategies for information slot-filling systems. These experiments are also described in Frampton and Lemon (2005). The behaviour of the user simulation used here is determined by a probabilistic model called a Bayesian Network, and the experiments are considered preliminary because the Bayesian Network's structure and probabilities are hand-coded, rather than learned from real user data. We prefer, (as we do in the experiments of Chapters 5 and 7), to use a simulation whose probabilities are learned from real user data, because assuming that the data is of sufficient quality, then the simulation ought to be more realistic. An obvious but key point to make, is that if we want to learn dialogue strategies which are to be directly implemented in a real system, then the user and ASR simulations used for generating training dialogues must be accurate.

Given that these RL experiments used a stochastic user simulation whose probabili-

ties were not learned from real user data, why then did we conduct them? The first reason was to provide a proof-of-the-concept that representing recent DAs in addition to slot-status information in the state can potentially enable the reinforcement learner to learn a more effective strategy. Another important reason was to gain insights into how best to set the reinforcement learner's parameters. Finally, conducting these experiments also allowed us to to investigate the effects of different training reward functions, and whether any tractability problems are encountered as we scale up to a more commercially-realistic number of information slots.

The remainder of this chapter will proceed as follows. First, Section 4.2 describes our experimental setup for using RL to learn full dialogue strategies. This description covers the choice of RL algorithm (Section 4.2.1), how the DM, user simulation and reinforcement learner interact (Section 4.2.2) and the reinforcement learner's parameter settings (Section 4.2.3). Section 4.2.4 describes the Bayesian Network user simulation which is used in the preliminary experiments of this chapter. Sections 4.3, 4.4 and 4.5 then describe the three preliminary RL experiments. The first two experiments involve learning full strategies for a two-slot system, and the aim is to provide a proof-of-the-concept that representing the DA of the user's last turn in the state can improve the learned strategy. The first experiment fails in this aim, but analysis is provided to explain why, while the second experiment succeeds. In the third experiment, we then scale up to a more commercially-realistic 4 slots and investigate the effects on the learned strategy of different training reward functions.

4.2 The experimental setup for Reinforcement Learning

4.2.1 Introduction

In the experiments described in this chapter, the following three software agents were used to simulate dialogues and learn and test dialogue strategies:

1. the DIPPER Information State Update Dialogue Manager (DM) (Bos et al., 2003),

2. a Bayesian Network user simulation,

3. a reinforcement learner.

I created the reinforcement learner and Bayesian Network user simulations using Java. The original version of the reinforcement learner stored the Q-values in a *hashmap*, but in the experiments described in Chapter 5, the number of Q-values became too large, and so I created a second version which stored the Q-values in a file, and then a third which stored them in a database. In the work described in Chapters 5 and 7, the Bayesian Network user simulation is replaced with an n-gram user simulation.

The three "agents" communicate via the Open Agent Architecture (OAA) (Cheyer and Martin, 2001). DIPPER is used to track and update the dialogue context. It supplies the reinforcement learner with a reward following each system turn, and the reinforcement learner uses these rewards to update the Q-values using the Sarsa(λ) algorithm (see Sections 2.11 and 2.12). We chose to use the Sarsa(λ) algorithm because it is able to learn from raw experience without a model of the environment's dynamics, and it has been found to converge faster than other RL algorithms e.g. Monte Carlo (MC) algorithms. The reinforcement learner is also responsible for deciding which action DIPPER should take next. Figure 4.1 shows this experimental setup. The task domain is flight-booking, and the aim for the DM is to obtain values for the user's flight information "slots" i.e. *departure city*, *destination city*, *departure date* and *departure time*, before making a database query.

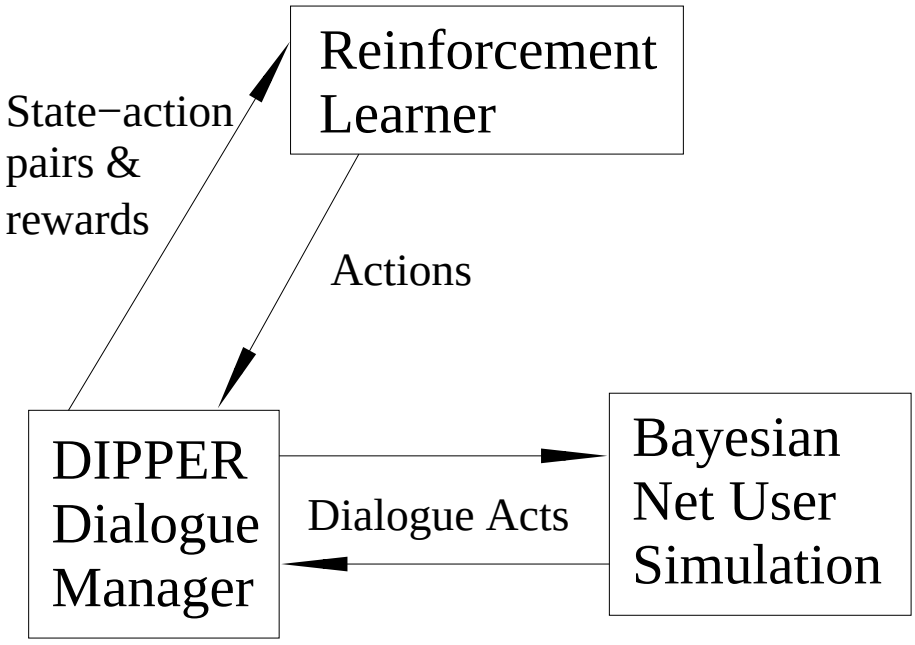

Figure 4.1: The basic experimental setup

Next, Section 4.2.2 gives an overview of what happens in a single exchange between the reinforcement learner, DIPPER, and the Bayesian Network simulation. Sections 4.2.3 and 4.2.4 give more details about the reinforcement learner and Bayesian Network simulation.

4.2.2 Overview of a single exchange

In a single exchange between the DIPPER Dialogue Manager (DM), (whose action choice is determined by the reinforcement learner), and the simulated user, the DM will first call the reinforcement learner's main function - *doRLearning(State, PossibleActions,Reward,NextAction)* - with the first 3 variables instantiated. For a non-terminal system turn, the reward will be specified as "systemTurnPenalty". This represents a negative value stored internally by the reinforcement learner, and so causes the reinforcement learner to learn strategies that do not prolong a dialogue longer than necessary. In the experiments described in this chapter, the value of "systemTurnPenalty" was always -1. The reinforcement learner will update its value estimates for actions, (Q-values), internally and provide the DM with the next action to be taken in the 4^{th} variable. The DM carries out the action in the 4^{th} variable and then calls the user simulation's main function - *generateResponse(SysPrompt,UserResponse,ConfScores, AbstractResponse)* - with only the 1^{st} variable instantiated. The user simulation, (see Section 4.2.4), returns a Dialogue Act (DA) as a response, and the DM updates its representation of the dialogue context accordingly.

This sequence of events repeats until the conclusion of the dialogue, and the system and user simulation may have any number of dialogues. A dialogue concludes when either the user simulation 'hangs-up' (with final reward 0), or the system makes a *database query*. In the case of a database query, the database query is compared to the user goal defined by the user simulation at the start of the dialogue i.e. particular values for slots such as the departure and destination cities. DIPPER will check whether the database query and the user goal are the same, and if they are, then the reinforcement learner will be given a relatively large dialogue-final reward. If the database query is partially correct, then depending on the reward function being used, the reinforcement learner may be given some lesser positive reward.

At set intervals of dialogues, the reinforcement learner will output a file containing the Q-values, and the number of times each corresponding state-action pair has been

visited. Such a file can be "loaded" into the reinforcement learner and this makes it possible to continue learning from this point, or to test the learned strategy using the reinforcement learner's function *getLearnedAction(State,Action)*, which given the state, returns the action with the highest Q-value. The reinforcement learner is also able to produce a graph of average reward-per-dialogue from set intervals of training dialogues, and this graph is useful for indicating when the learned strategy has stabilised.

4.2.3 The reinforcement learner's parameter settings

For the experiments described in this chapter, the reinforcement learner's parameters, (introduced in Chapter 2), were set as follows:

1. Step-size parameter: $\alpha = decreasing$,
2. Discount factor: $\gamma = 1$,
3. Action selection type = ε-*greedy* (alternative is *softmax*),
4. Action selection parameter: $\varepsilon = 0.7$,
5. Eligibility Trace parameter: $\lambda = 0.9$,
6. Eligibility Trace = *replacing* (alternative is *accumulating*),
7. Initial *Q-values* = 0.

The remainder of this section will explain why these are appropriate settings.

4.2.3.1 The step and discount parameters

As stated in Section 4.2.1, the reinforcement learner updates its Q-values using the *Sarsa(λ)* algorithm (see Sections 2.11 and 2.12). The first parameter is the step-size parameter, α, which was introduced in Sections 2.10.1 and 2.11.1, and this may take a value between 0 and 1, or specified as *decreasing*. If it is *decreasing*, as it is in our experiments, then for any given Q-value update, α is $\frac{1}{k}$ where k is the number of times that the state-action pair for which the update is being performed has been visited. This kind of step parameter will ensure that given a sufficient number of training dialogues, each of the Q-values will eventually converge. The second parameter, γ, (the discount

factor), may take a value between 0 and 1. Since dialogue management is an episodic task, we can set γ to 1 so that future rewards i.e. rewards for *task completion*, are taken into account as strongly as possible (see Section 2.3).

4.2.3.2 The action selection parameters

Apart from updating Q-values, the reinforcement learner must also choose the next action for the Dialogue Manager (DM) to take, and the third parameter specifies whether it does this by ε-*greedy* or *softmax* action selection (see Section 2.9.1). In the preliminary experiments described later in this chapter, we have used ε-greedy. If we are using ε-greedy action selection, then the fourth parameter gives the value of ε, the probability of selecting the action with currently the highest Q-value, while if using softmax, then it gives the *temperature*.

4.2.3.3 The Eligibility Trace parameters

The fifth parameter, the Eligibility Trace (ET) parameter λ, may take a value between 0 and 1, and the sixth parameter specifies whether the ETs are *accumulating* or *replacing* (see Section 2.12 for definitions of accumulating and replacing ETs). We used *replacing* ETs because they produce faster learning for the slot-filling task, especially when there is little information represented in the state e.g. only slot-status features. To understand why, consider how during a training dialogue, the DM may find a particular action in a particular state that often does not produce a change in state i.e. a *dead-end* state-action/action. For example, consider a scenario where the system re-asks a slot-value that has already been correctly filled/confirmed. The user will probably often reply with the same value again, and so assuming that the value is correctly recognised, the slot-status features will be unchanged. If the slot-status features are all that is represented in the state, then the state itself will be unchanged, and so on the next system turn, the same dead-end state-action pair can be visited again. Note that in general, the less information which is represented in the state, and so the less fine-grained the representation of the context, the more frequent visits to such dead-end state-actions/actions will be. If the system does re-visit the same dead-end state-action on its next turn, then with accumulating ETs, the ET increases to greater than 1. When a *non-dead-end action* is taken from this state, its ET will be less than that for the *dead-end* action. If a state-action's ET increases, then so does the amount

by which its Q-value will be updated in the direction of the most recently received reward. Hence, if task completion is achieved and so a positive reward is received at the end of the dialogue, then the Q-value for the dead-end state-action will increase by more than that for the non-dead-end. In the next dialogue, the reinforcement learner will be even more likely to choose the dead-end action, making it even more likely that the dead-end action will have the larger ET. Eventually all of this may be corrected, but learning will have been slowed down significantly. This problem does not occur with *replacing* ETs. No matter how many times the dead-end action is taken, its ET is always less than that for a non-dead-end action after the non-dead-end action has been taken. Indeed with the kind of *replacing* ETs used here, when the non-dead-end action is taken, the trace for the dead-end action is set to zero.

4.2.3.4 The initial Q-values parameter

The seventh parameter is for supplying the initial value for the Q-values. There is a trade-off to consider here, especially when the state contains little information e.g. only slot-status features. On the one hand, we want to set this value high in order to encourage the reinforcement learner to explore unvisited state-actions. However, when using the kind of replacing ETs described above, if we set it too high, then at least in the short term, the reinforcement learner will learn "dead-end" actions, (as stated above, actions which do not lead to a change in the state). This happens because early on in training a larger proportion of dialogues end in task failure and hence low reward. This low reward will be propagated back to the non-dead-end state actions, but not to the dead-end state-actions because their ETs will have been set to zero. Thus while the Q-values for the non-dead-end state-actions decrease significantly, those for the dead-end state-actions remain at their initial high value. This may be corrected after further dialogues but in any case, learning will be slowed. Again, this is more of an issue when less information is represented in the state e.g. only slot-status features, because in general, with a less fine-grained representation of the dialogue context, dead-end state-actions will be visited more often.

Having provided details about how we set the reinforcement learner's parameters, and the issues involved, we now move on to describe the Bayesian Network user simulation.

4.2.4 Bayesian Network user simulation

Here a *user simulation*, rather than real users, interacts with the Dialogue Manager (DM) via Dialogue Acts (DAs) (see Section 1.2 for a definition of a DA). The Bayesian Network user simulation is capable of simulating goal-directed mixed-initiative dialogue. When the user simulation's main solvable *generateResponse (SystemPrompt, UserResponse, ConfidenceScores, AbstractResponse)* is called with the first variable instantiated, it first generates an abstract response via a *Bayesian Network* (implemented using NeticaJ Norsys 2002). A Bayesian Network is a probabilistic graphical model that represents a set of variables and their probabilistic dependencies, and for a good introduction, the reader can refer to Section 3.3 of Pietquin (2004). For the Bayesian Network used here, the user's response is dependent on the values of the System Prompt, Slot Counter and History variables as shown in Figure 4.2. We chose this structure after analysing COMMUNICATOR data (Walker et al., 2001a) and identifying what seemed to be the most important factors in determining the next user response. These were:

1. the last system turn,
2. how many times each slot has been asked about,
3. whether or not the system has attempted to confirm any slot values incorrectly due to a misunderstanding error, (recall from Section 1.1.1, that a misunderstanding error occurs when the system obtains an incorrect slot-value),
4. whether the user has asked for help.

Below is an example of where the system tries to confirm an incorrect slot value as a result of a misunderstanding error i.e. an Automatic Speech Recognition (ASR) error earlier in the dialogue.

```
User Simulation: I want to fly from Edinburgh.
System: So you want to fly from Eindhoven?
```

Whether the system has attempted to confirm any slot values incorrectly, and whether the user has asked for help is represented by the 'history' node. As previously stated, we supplied the probabilities for this simulation based on an initial analysis of relevant

data - the COMMUNICATOR data (Walker et al., 2001a) -, and intuition. By contrast, the RL experiments described in Chapters 5 and 7 use n-gram user simulations, the probabilities for which are learned from the COMMUNICATOR data.

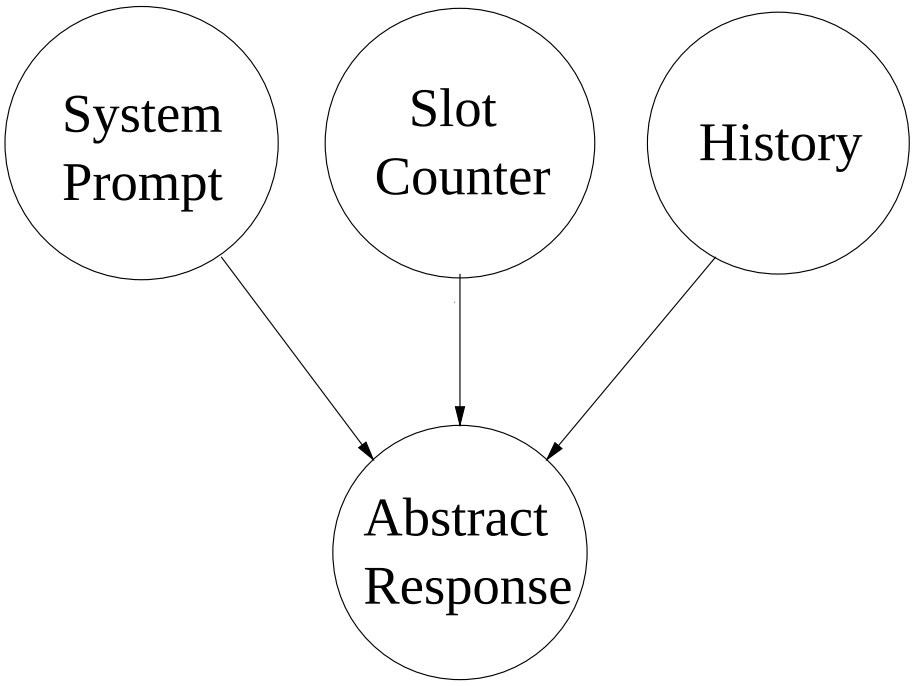

Figure 4.2: The Bayesian Network used by the user simulation: the abstract response depends on the values of the System Prompt, Slot Counter and History variables.

We now provide the possible values for each node in the Bayesian Network.

4.2.4.1 The possible values for each node

The possible values for the *system prompt node* are all of the different possible system actions. For Experiments 2 and 3, the list of possible system actions includes all of those given below, represented as Dialogue Acts (DAs). Unlike in Experiments 2 and 3, in Experiment 1 the system does not have the give help action available, and the user simulation does not ask for help. Note that *i* is the number of slots which is 2 in Experiments 1 and 2, but 4 in Experiment 3.

1. an open question e.g. ‘How may I help you?’,
2. ask the value for any of slots 1…*i*,
3. explicitly confirm any of slots 1…*i*,

4. ask for the i^{th} slot whilst implicitly confirming slot value $i-1$, (where $i=1$ we implicitly confirm the final slot e.g. 'So you want to fly from Edinburgh to where?'),
5. give help,
6. database query.

The different possible values for the *slot counter node* represent:

1. the slot asked about in the most recent system turn has been asked about ≤ 1 time before,
2. the slot asked about in the most recent system turn has been asked about ≥ 2 times before.

The different possible values for the *history node* represent:

1. Null: There are no outstanding incorrect system confirmations.
2. The dialogue system has attempted to confirm slot *i* incorrectly.
3. The user has asked for help.

Finally, the different possible values for the *abstract response* are DAs which represent the following:

1. stay quiet,
2. give the value for any of slots 1...*i*,
3. give the values for all of the slots,
4. say "no" and give the correct slot value in response to an incorrect system confirmation,
5. say "no",
6. say "yes",
7. ask for help,

8. end the dialogue prematurely i.e. ‘hang-up’.

Abstract responses 1, 4, 5 and 6 will include the number of the slot to which they refer e.g. “no(slot1)”.

We now give an overview of the settings for the probabilities themselves.

4.2.4.2 The probabilities

Appendix A shows the probability table for the Bayesian Network used in Experiment 3. The probability table used for Experiment 2 is the same except that there are just 2 slots rather than 4. The probability table used for Experiment 1 is the same as for Experiment 2 except that there are no probabilities related to asking for / receiving help.

The probabilities are designed so that the user simulation may take the initiative by giving more than one slot value when it is asked for only one, and may decide to close the dialogue if asked the same question twice or more. If the system tries to incorrectly confirm a particular slot value, the user simulation will generate abstract response 1, 4 or 5 to alert the system to this fact. Should the system then shift focus away from this problem slot on its subsequent turn, there will be a 90% probability that the user simulation will hang up. This is based on the intuition that in general, a user would find it disconcerting if having indicated a misunderstanding error, the system moves on without first acknowledging the corrected slot value. Unlike in Experiment 1, in Experiments 2 and 3, the user simulation can ask for help and the system provide it. The user simulation has a 20% chance of asking for help if:

1. the system has just asked an open question, (this is only possible at the start of a dialogue),
2. the system asks for slot 1 for the first time,
3. the system asks for slot 2 for the first time.

The user simulation will not ask for help in any other situations, and it now has a 90% chance of hanging up if it asks for help and then is not given help in the next system turn.

4.2.4.3 Translating the abstract response via the user goal

Having generated an abstract user response via the Bayesian Network, the user simulation next translates this abstract response into a concrete response (*userResponse* in the *generateResponse* solvable) by referring to a random user goal that it selected at the beginning of the dialogue. This random goal consists of values for *departure city*, *destination city*, *date of travel* and *preferred airline*. Hence if the abstract response is to give the value for slot 1, and the goal departure city is 'Edinburgh', then the concrete user response, (*userResponse*), will be $departure(edinburgh)$.

4.2.4.4 Automatic Speech Recognition simulation

Next this concrete user response is subject to an Automatic Speech Recognition (ASR) simulation. No data is used to help construct this simulation e.g. COMMUNICATOR (Walker et al., 2001a) or the SACTI corpora (Williams and Young, 2004). Words are assigned to groups with other similar-sounding words, and an ASR error is simulated by substituting a word for another from the same group. The 2^{nd} variable is then instantiated with the concrete response which now possibly contains a simulated ASR error, and the 3^{rd} variable with the associated ASR confidence score(s). The ASR confidence score may be either *low* or *high*. If a filled slot has low ASR confidence and is unconfirmed, then there is only a 55% chance that the value is correct, while if its ASR confidence is high, then it is correct.

This then completes the description of our basic experimental setup for RL of full dialogue strategies, and of the Bayesian Network user and ASR simulations. Hence, we now move on to describe the first of the three RL experiments with the Bayesian Network user simulation.

4.3 Experiment 1: a first attempt at improving the learned strategy with the user's last Dialogue Act

In this experiment, a first attempt is made at proving the concept that representing recent Dialogue Acts (DAs) in the state can enable a reinforcement learner to learn an improved dialogue strategy. Here, as stated in Section 4.1 and Section 4.2.4.1, the user simulation does not ask for help or the system provide it. If the system tries to

incorrectly confirm a particular slot value, the user simulation will generate a response to alert the system to this fact e.g. “no”. Should the system then shift focus away from this problem slot on its subsequent turn, the user simulation will have a 90% chance of hanging up. If the reinforcement learner does not know that the last user utterance was a rejection of an attempted confirmation, then it follows that it cannot learn to re-ask or reconfirm the problem slot value on its next turn.

4.3.1 Hypothesis: Adding the user’s last Dialogue Act to the state will improve the learned strategy

The hypothesis is that adding the Dialogue Act (DA) of the last user turn will enable the reinforcement learner to learn a more effective strategy.

4.3.2 State representations

In order to test the hypothesis, two strategies are learned using different state representations. The first, which we refer to as the Slot-Status Strategy uses a state representation which contains only slot-status features:

1. whether the 1^{st} slot is filled,
2. whether the 2^{nd} slot is filled,
3. the ASR confidence score for the 1^{st} slot value,
4. the ASR confidence score for the 2^{nd} slot value,
5. whether the 1^{st} slot value has been confirmed,
6. whether the 2^{nd} slot value has been confirmed.

Next, a second strategy is learned using a state representation which contains the above slot-status features and then an additional feature for the DA of the last user turn. Hence we refer to this as the “DA Strategy”. If the hypothesis is correct, then the DA Strategy will outperform the Slot-Status Strategy.

4.3.3 The action set for the learner

Below is a list of all of the different actions that the Reinforcement Learning (RL) Dialogue Manager (DM) can take and must learn to choose between based on the representation of the dialogue context in the state:

1. an open question e.g. 'How may I help you?',
2. ask the value for any of slots 1...i,
3. explicitly confirm any of slots 1...i,
4. ask for the i^{th} slot whilst implicitly confirming slot value $i-1$, (where $i=1$ we implicitly confirm the final slot e.g. 'So you want to fly from Edinburgh to where?'),
5. database query.

There are a couple of restrictions regarding which actions can be taken in which states. Action 1 (open question) is only possible at the start of the dialogue, and DIPPER can only confirm non-empty slots.

4.3.4 The reward function

This experiment used an "All-or-nothing" reward function which gave $+100$ if all the slot values are correct, otherwise 0. Such an "All-or-nothing" reward function was used because in a series of preliminary experiments, (not described here), it had been found to produce faster learning for a 2-slot problem when using ε-greedy action selection. Each system turn received a penalty of -1.

4.3.5 Results

Figure 4.3 shows how after 18000 training dialogues, each of the learned strategies seemed to have stabilised. Interacting with the user simulation during testing runs, the DA Strategy then achieved an average reward-per-dialogue of 94.05 over 500 dialogues, and the Slot-Status Strategy, 93.80. A one-tailed independent samples t-test showed that this improvement was not statistically significant. Hence in this case the

hypothesis was false - including the user's last Dialogue Act (DA) in the state did not improve the learned strategy. In the next section, we provide analysis to explain why.

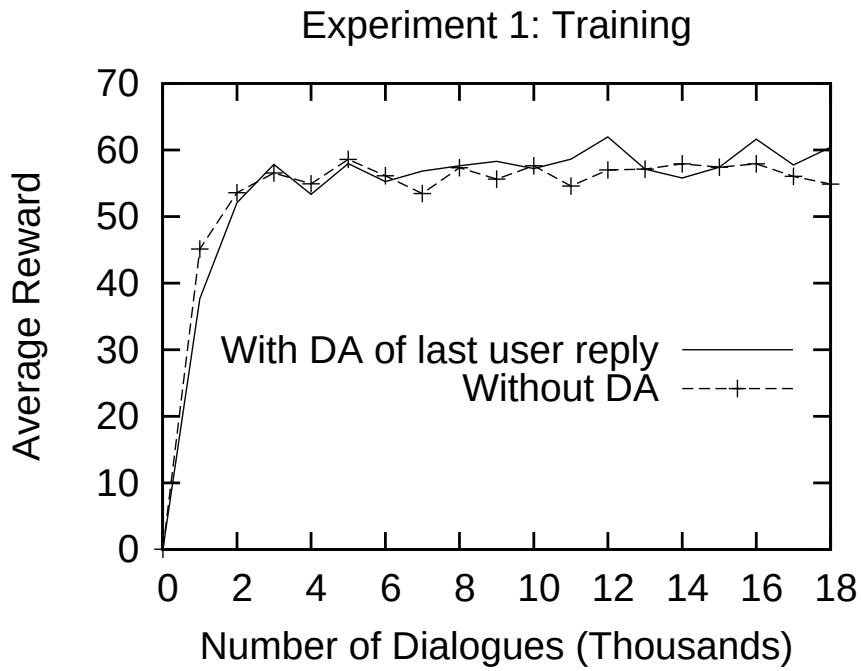

Figure 4.3: Learning in experiment 1: 2 slot system, reward function 2 (All-or-nothing Reward), no user "help" action.

4.3.6 Analysis

4.3.6.1 Why the hypothesis was false

In order to avoid a likely user hang-up, whenever the user simulation indicated a misunderstanding error i.e. an incorrect slot-value, it was necessary for the Dialogue Manager (DM) to maintain focus on the problem slot. The hypothesis was false because knowledge of the user's last DA was unnecessary for learning to do this. An optimal strategy here was one which attempted to fill and confirm the slots in a set order, not moving on until the slot currently in focus is filled/confirmed. For such a strategy, the problem slot can be inferred from the slot-status features alone, as shown by the example in Table 4.1. When the user indicates the misunderstanding error, the state does not change and so the learned action is the same as on the previous system turn i.e. the Slot-Status Strategy maintains focus on the problem slot. Hence there is no advantage to be gained from representing the last user DA in the state.

Speaker	**Slot-status features**	**Dialogue Act**	**Transcript**
System	[0,0,low,low,no,no]	askSlot1	"Where do you want to fly from?"
User			"From Edinburgh"
System	[1,0,low,low,no,no]	icSlot1_askSlot2	"From Eindhoven to where?"
User			"No Edinburgh"
System	[1,0,low,low,no,no]	icSlot1_askSlot2	"From Edinburgh to where?"

Table 4.1: On the final system turn here, the user's last DA is unnecessary for identifying the problem slot; ic = implicit confirm.

Speaker	**Slot-status features**	**Dialogue Act**	**Transcript**
System	[0,0,low,low,no,no]	askSlot1	"Where do you want to fly from?"
User			"[Asr rejection]"
System	[0,0,low,low,no,no]	askSlot2	"What is your destination?"
User			"Eindhoven"
System	[0,1,low,low,no,no]	icSlot2_askSlot1	"To Eindhoven from where?"
User			"No"
System	[0,0,low,low,no,no]	askSlot2	"Where do you want to fly to?"

Table 4.2: Due to switching focus from slot 1 to 2 earlier in the dialogue, on the final system turn here, the slot-status features cannot identify the problem slot with total certainty; ic = implicit confirm.

Had the optimal strategy been one which took a more flexible course through the dialogue, then contexts would have occurred in which the slot-status features alone were not enough to infer the problem slot. Table 4.2 shows an illustrative example with a hypothetical optimal strategy. Here the hypothetical optimal strategy switches focus from slot 1 to slot 2 following a non-understanding error, (see Section 1.1.1 for a definition of non-understanding errors). It then obtains a value for slot 2, but the user then indicates a misunderstanding error and so the slot-status features are now as they were in the first state. Hence we cannot tell from the slot-status features alone whether the problem slot is slot 1 or slot 2. It would be possible to learn this hypothetical optimal strategy if the last user DA is represented in the state, but not if the state only contains

the slot-status features. If the state only contains the slot-status features, then it would not be possible to learn the optimal action of switching focus from slot 1 to slot 2 following a non-understanding error. As a result, the ambiguous context following the misunderstanding error would not occur, but the overall strategy would be sub-optimal.

It will be interesting to see whether any of the issues discussed here crop up when we learn strategies with real user data in Chapter 5. Now however, we move on to describe further general characteristics of the learned strategies.

4.3.6.2 General characteristics of the learned strategies

Below is a list of further general characteristics of the learned strategies from this experiment. These characteristics are also true of the learned strategies from Experiments 2 and 3.

1. An open question is asked at the start of the dialogue i.e. the system gives the user the initiative.
2. Slot values with low ASR confidence scores are confirmed (implicitly or explicitly) rather than re-asked.
3. When at least 1 but not all slots are filled, an empty slot is asked while a filled slot is implicitly confirmed.
4. Once all slots are filled, explicit is preferred to implicit confirmation for slot values with low ASR confidence.

The reinforcement learner learns to ask an open question at the start of a dialogue because the user simulation is more likely to reply with > 1 slot value, so enabling a shorter dialogue. To explain Characteristic 2, we must consider slots that have been filled with low ASR confidence but are unconfirmed. The advantage of using confirmation over re-asking comes when the confirmation turns out be correct, which is the majority of cases. Then the user simulation replies with 'yes' which is always recognised correctly, whereas if the system re-asks the slot value, then the user simulation replies with the value and this has a 45% chance of being misrecognised. Characteristic 3 appears because if an implicit confirmation is correct, then the system can confirm the slot and obtain a value for an empty slot in just one turn, and this allows

shorter dialogues. However, asking for one slot value while implicitly confirming another is not desirable if all slots are filled because slot values provided by the user simulation may be incorrectly recognised and so cause longer dialogues and/or incorrect database queries. Hence, in these states the learned strategy chooses explicit confirmation (Characteristic 4).

We now move on to describe the second preliminary experiment in which we add Bayesian Network probabilities relating to a new system "Give Help" action. This experiment tests the same hypothesis.

4.4 Experiment 2: a second attempt at improving the learned strategy with the user's last Dialogue Act

In this experiment, a second attempt is made at proving the concept that representing recent Dialogue Acts (DAs) in the state can enable a reinforcement learner to learn an improved dialogue strategy. Here, the user simulation is now able to ask for help and the system to provide it. The user simulation has a 20% chance of asking for help if:

1. the system has just asked an open question (this is only possible at the start of a dialogue),
2. the system asks for slot 1 for the first time,
3. the system asks for slot 2 for the first time.

The user simulation will not ask for help in any other situations, and it now has a 90% chance of hanging up if it asks for help and then is not given help in the next system turn. Hence, adding the user's last DA to the state will enable the reinforcement learner to learn to give help when the user simulation asks.

4.4.1 Hypothesis: Adding the user's last Dialogue Act to the state will improve the learned strategy

As in Experiment 1 (see Section 4.3), the hypothesis is that adding the Dialogue Act (DA) of the last user turn will enable the reinforcement learner to learn a more effective strategy.

4.4.2 State representations

Again, a baseline Slot-Status Strategy is learned using a state representation which contains the following slot status features:

1. whether the 1^{st} slot is filled,
2. whether the 2^{nd} slot is filled,
3. the ASR confidence score for the 1^{st} slot value,
4. the ASR confidence score for the 2^{nd} slot value,
5. whether the 1^{st} slot value has been confirmed,
6. whether the 2^{nd} slot value has been confirmed.

A second strategy, the DA Strategy, is learned using a state representation which contains these features, plus an additional feature for the DA of the last user turn.

4.4.3 The action set for the learner

Below is a list of all of the different actions that the Reinforcement Learning (RL) Dialogue Manager (DM) can take and must learn to choose between based on the context:

1. an open question e.g. 'How may I help you?',
2. ask the value for any of slots 1...i,
3. explicitly confirm any of slots 1...i,
4. ask for the i^{th} slot whilst implicitly confirming the value for slot $i-1$, (where $i=1$ we implicitly confirm the final slot e.g. 'So you want to fly from Edinburgh to where?'),
5. give help,
6. database query.

Hence, the action set is the same as in Experiment 1, except that there is now an additional "give-help" action. Again, Action 1 (open question) can only be used at the start of the dialogue, and DIPPER can only confirm non-empty slots.

4.4.4 The reward function

This experiment used the same "All-or-nothing" reward function as the first experiment, awarding $+100$ for totally correct database queries, and 0 otherwise, and giving a penalty of -1 for each system turn.

4.4.5 Results and analysis

Learning lasted for 18000 dialogues after which both learned strategies were stable, as shown by Figure 4.4. The DA Strategy gave help whenever the user asked for it, while the Slot-Status strategy did not give help in any state. As a result, the DA Strategy achieved an average reward-per-dialogue of 90.55 over a 500 dialogue test run interacting with the user simulation, and the Slot-Status Strategy had an average reward of only 59.64. Hence the DA Strategy shows a 52% improvement, and an independent samples one-tailed t-test, (see page 427 of Sheskin 2007), was conducted to find that the significance level was $p < 0.05$. Note that the rewards in the training graph, (Figure 4.4), are less than in testing because of exploration i.e. during training the reinforcement learner sometimes explores actions which at present it has learned to be sub-optimal. This experiment then successfully demonstrates a first simple case in which adding the DA of the last user utterance allows the reinforcement learner to learn a superior strategy. Hence it is a proof-of-the-concept that including more than only the slot-status features in the state can lead to a more effective learned strategy.

We now move on to an experiment where we investigate firstly, whether the RL problem remains tractable if we scale up to a more commercially-realistic number of slots, and secondly, the effects on the learned strategy of different training reward functions.

4.5 Experiment 3: scaling-up and investigating different training reward functions

Although the previous experiment proved the concept that including the user's last Dialogue Act (DA) in the state can improve the learned strategy, we have not yet shown that the Reinforcement Learning (RL) problem remains tractable for a more commercially-realistic number of slots e.g. 4. In addition, while an "all-or-nothing" training reward function has been found to produce the fastest rate of learning for a

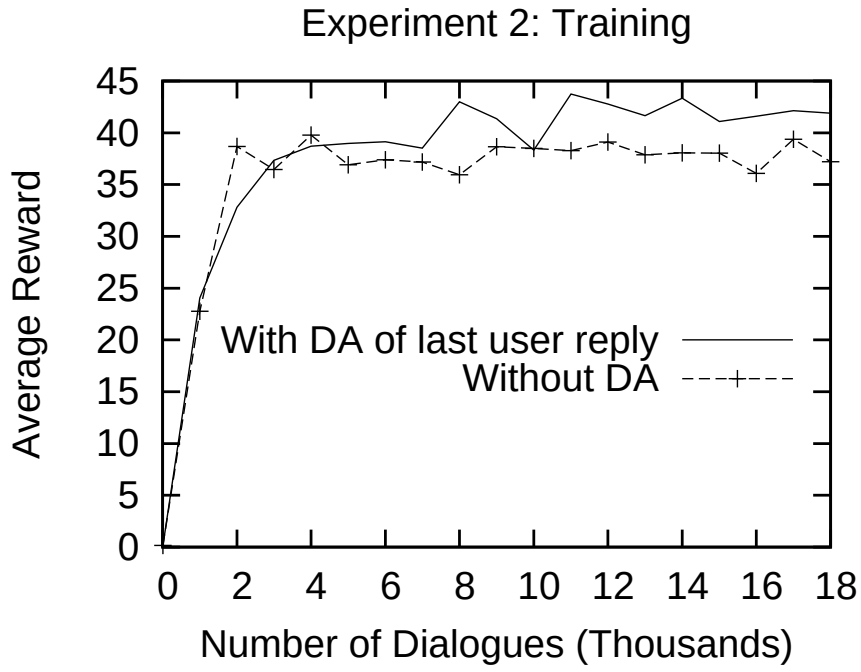

Figure 4.4: Learning in Experiment 2: 2 slot system, reward function 2 (All-or-nothing Reward), with user 'help' action.

2-slot system, it would be useful to know if this is also the case for a 4-slot system. As the number of slots increases, it would seem reasonable to expect that at some point, a reward function that gives some positive reward to partially-correct database queries, a "partial reward function", will produce faster learning than an all-or-nothing reward function. Partial reward indicates to the reinforcement learner that it is headed in the right direction for the optimal strategy, and so should become more important as the number of slots and hence incorrect solutions increases. This experiment investigates these issues.

4.5.1 Hypothesis 1: Scaling up to a more commercially-realistic number of slots does not cause intractability

Following the success of Experiment 2, the first hypothesis here is that with the Dialogue Act (DA) of the user's last turn in the state representation, the reinforcement learner is still able to learn an effective strategy if the number of slots is increased from 2 to a more commercially-realistic 4.

4.5.2 Hypothesis 2: Rate of learning will be fastest with an "all-or-nothing" training reward function

The second hypothesis is that with 4 slots, the rate of learning will still be fastest when using an all-or-nothing training reward function. We explore the impact of the following training reward functions on rate of learning/quality of the learned strategies:

1. Reward Function 1 (Partial Reward): $+100$ for each correct slot value,
2. Reward Function 2 (All-or-nothing Reward): IF all slot values are correct, $+100$, ELSE 0,
3. Reward Function 3 (Mixed Reward): IF all slot values are correct, $+100$, ELSE $+10$ for each correct slot value.

Note that as usual, each system turn receives a penalty of -1. Reward function 3 (Mixed Reward) can be seen as a halfway-house between the partial and all-or-nothing reward functions.

4.5.3 State Representation

Each state is represented in terms of the following variables:

1. Slot 1 confidence score,
2. Slot 2 confidence score,
3. Slot 3 confidence score,
4. Slot 4 confidence score,
5. the Dialogue Act (DA) of the last user utterance.

Variables 1-4 can take the values *empty*, *low* or *high*. If a value for the slot has been supplied, and if its Automatic Speech Recognition (ASR) confidence score is low and it is unconfirmed, then the value for the variable will be *low*. If it is confirmed or if its ASR confidence score is *high*, then it will be *high*. There were 19 possible values for variable 5 (see the user simulation actions in Subsection 4.2.4.1), and so a total of $3^4 \times 19 = 1539$ states.

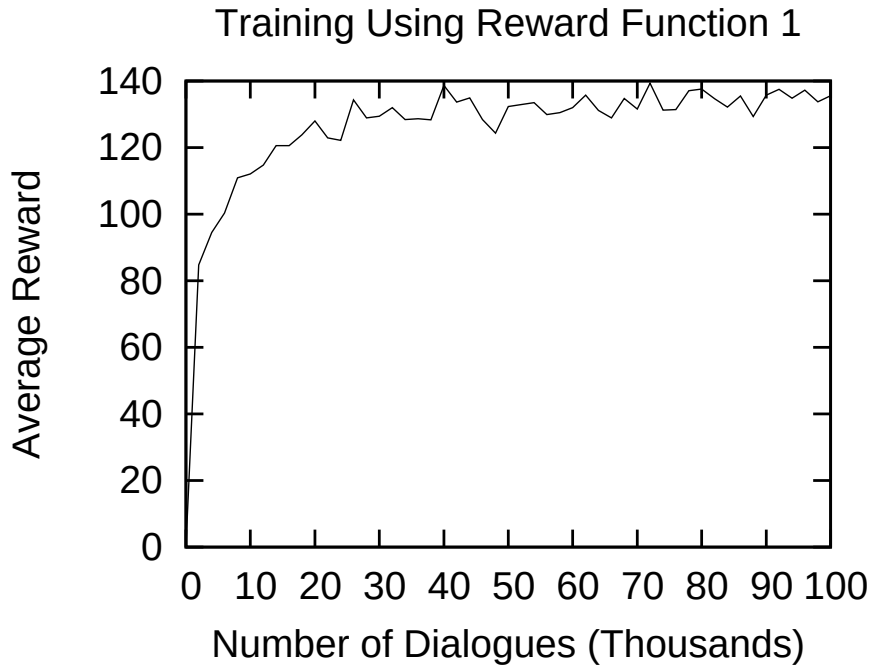

Figure 4.5: Learning in experiment 3: 4-slot system, reward function 1 (Partial reward), with user 'help' action, ε-greedy action selection, $\varepsilon = 0.7$

4.5.4 The action set for the learner

Below is a list of all the different actions that the Reinforcement Learning (RL) Dialogue Manager (DM) can take and must learn to choose between based on the context:

1. an open question e.g. 'How may I help you?',
2. ask the value for any of slots 1...*i*,
3. explicitly confirm any of slots 1...*i*,
4. ask for the i^{th} slot whilst implicitly confirming slot value $i-1$, (where $i=1$ we implicitly confirm the final slot e.g. 'So you want to fly from Edinburgh to where?'),
5. give help,
6. database query.

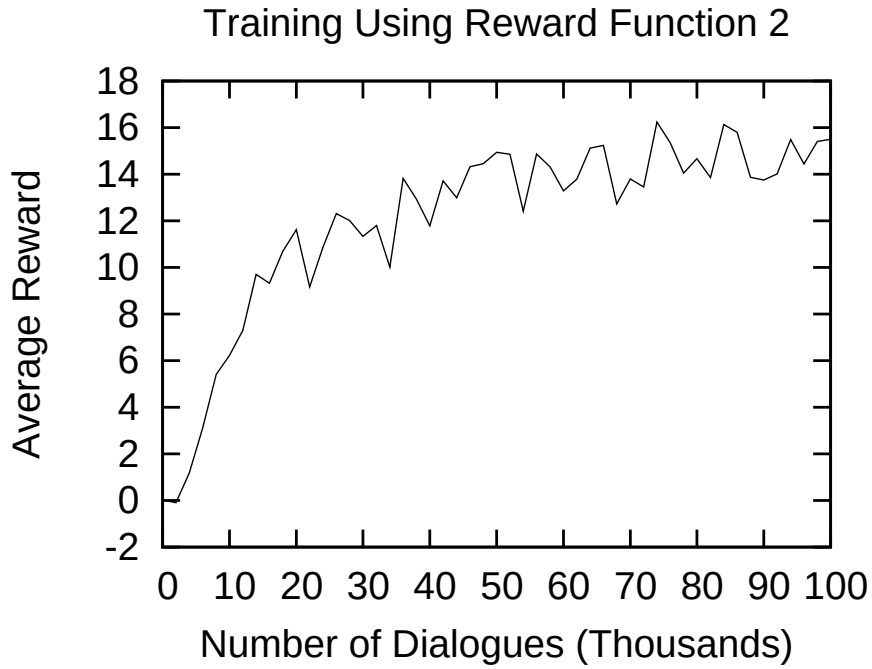

Figure 4.6: Learning in experiment 3: 4-slot system, reward function 2 (All-or-nothing Reward), with user 'help' action, ε-greedy action selection, $\varepsilon = 0.7$

Whereas in Experiments 1 and 2 there were two slots, now there are four. Again, action 1, (open question), is only possible at the start of the dialogue, and DIPPER can only confirm non-empty slots.

4.5.5 Results and analysis

Learning continued for 100000 dialogues, at which stage the learned strategies were still improving, but the one using Reward Function 1 (Strategy 1: Partial Reward) at a slower rate than those for Reward Functions 2 and 3 (All-or-nothing and Mixed Rewards, Strategies 2 and 3 respectively). This is shown in Figures 4.5, 4.6, and 4.7.

Here we can see that the general dialogue strategy learning problem remained tractable for the 4 slot case, and that after 30000 and 100000 dialogues, with ε-*greedy* action selection and ε set to 0.7, the best strategy was learned using Reward Function 2, (All-or-nothing Reward), the 2^{nd} best strategy using Reward Function 3, (Mixed Reward), and the worst using Reward Function 1, (Partial Reward). The experiment was repeated twice and this result was the same each time. The performance of the learned

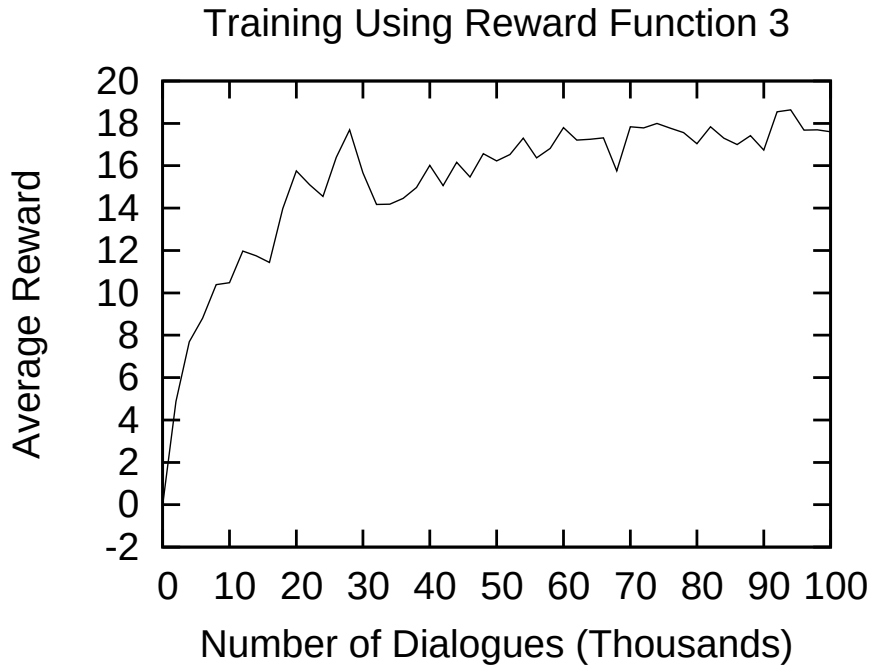

Figure 4.7: Learning in experiment 3: 4-slot system, reward function 3 (Mixed Reward), with user 'help' action, ε-greedy action selection, $\varepsilon = 0.7$

strategies from the first run are summarised in Tables 4.3 and 4.2 (columns represent dialogue strategies trained with the 3 different reward functions, rows show testing of those strategies based on the 3 ways of computing reward). The numbers represent the average reward achieved per dialogue over 500 test dialogues.

		Strategy 1	**Strategy 2**	**Strategy 3**
Reward	**1**	247.89	373.90	334.06
Function	**2**	20.12	84.42	62.74
(Testing)	**3**	35.65	85.30	71.90

Table 4.3: Testing the 3 strategies after 100000 training dialogues using each of the 3 reward functions, average reward achieved per dialogue over 500 test dialogues.

When learning with Reward Function 1, the reinforcement learner has learned to query the database in many more states where < 4 slots have been filled with high confidence than when learning with Reward Functions 2 or 3. This is apparently because it still receives a substantial reward for doing so: $+100$, $+200$, or $+300$ rather than the max-

		Strategy 1	**Strategy 2**	**Strategy 3**
Reward	**1**	186.76	320.04	301.91
Function	**2**	-4.54	78.69	40.32
(Testing)	**3**	14.58	75.16	56.79

Table 4.4: Testing the 3 strategies after 30000 training dialogues using each of the 3 reward functions, average reward achieved per dialogue over 500 test dialogues.

imum attainable $+400$. Note that learned Strategy 2 outperformed learned Strategies 1 and 3 on all performance metrics i.e. including Reward Functions 1 and 3.

Note that this experiment does not consider the fact that for a real Spoken Dialogue System (SDS) there are situations where it is preferable to query the database with only a subset of the slots filled. For example, the speech recogniser may simply be unable to recognise a particular slot value, perhaps due to the user's accent. Then it is clearly better to query the database with the slot values that can be obtained rather than asking for the problem slot over and over again before the user becomes so frustrated that they hang-up. Given a user simulation which is able to simulate unobtainable slot values, it may be possible to learn such a strategy by including a *dialogue length* feature in the state representation. This is a point which we will return to in Chapter 5.

4.6 Summary

This chapter first described our experimental setup for Reinforcement Learning (RL) of full dialogue strategies (Section 4.2). This included a description of a Bayesian Net user simulation (Section 4.2.4), for which the structure was based on the findings of an initial analysis of COMMUNICATOR dialogues (Walker et al., 2001a), and for which the probabilities were set based on intuition. The chapter then went on to detail three preliminary RL experiments involving this Bayesian Net user simulation (Sections 4.3, 4.4 and 4.5). These experiments are considered preliminary because for learning dialogue strategies, we prefer, (as we do in the experiments of Chapters 5 and 7), to use stochastic user simulations whose probabilities are learned from real user data. This is because such a simulation ought to be more realistic and for a learned

strategy to be directly applicable in a Spoken Dialogue System (SDS), the simulation must be realistic. The first aim of these experiments was to provide a proof of the concept that including more than just the slot-status variables in the state can potentially enable the reinforcement learner to learn a more effective strategy. Secondary aims included gaining knowledge for how best to set the reinforcement learner's parameters and investigating the effects of different training reward functions.

To begin summarising the results of the experiments, in the first experiment, adding the Dialogue Act (DA) of the last user utterance to the state representation does not produce a superior learned strategy. The Bayesian Network user simulation was such that it would hang up if the system attempted to confirm a slot value incorrectly, and then having been alerted to the fact, it switched focus to a different slot on its next turn. The hypothesis was that the reinforcement learner would need knowledge of the DA of the user's last turn so that it could learn not to switch focus to another slot in these situations. However, in this case, the hypothesis was false, and the explanation centred on the fact that here an optimal strategy was one which asked the slots in a set order, not moving on until it had filled/confirmed the slot currently in focus. As a result, when the user indicated a misunderstanding error, it was always possible to infer the problem slot based on the slot-status features alone.

The second experiment, where the system can respond to user "help" requests, demonstrates a first simple case in which including this 'high-level' information does result in a superior learned strategy - there is a 52% improvement in average reward with $p < 0.05$. This then is the proof-of-concept that we were seeking. The third experiment then shows that we can scale up to a more commercially realistic 4-slot problem, and that when using ε-greedy action selection, a reward function which rewards only totally correct database queries produced the best strategy i.e. all-or-nothing as opposed to partial rewards.

In the next chapter, we now go on to describe RL experiments which use a stochastic user simulation whose probabilities are learned from real user data.

Chapter 5

Learning with real user data: n-gram user simulation experiments

5.1 Introduction

Like the Reinforcement Learning (RL) experiments of the previous chapter, the RL experiments of this chapter also involve training and testing dialogue strategies with stochastic user simulations. However, whereas in the previous chapter, the user simulation probabilities were hand-coded, here they are learned from real user data. Instead of a Bayesian Network model, we now use n-gram models, (4 and 5 grams), and the probabilities for these n-grams are learned from the COMMUNICATOR data (Walker et al., 2001a), (see Section 3.10.4). As a result, these n-gram models ought to simulate real user behaviour more accurately than the Bayesian Network.

This chapter describes two RL experiments which were conducted using the n-gram user simulations - in the first, dialogue strategies are learned for a flight-booking system with three information slots (*departure city*, *destination city*, *departure date*), and in the second, for a flight-booking system with four information slots (*departure city*, *destination city*, *departure date*, *departure time*). The four-slot experiment is also described in Frampton and Lemon (2006). The main hypothesis tested by these experiments is that the learned strategy can be improved by adding recent Dialogue Acts (DAs) to the slot-status features already in the state. Should we find this hypothesis to be true, we are then obviously interested in how the recent DAs improve the learned strategy. We start with the three-slot experiment because using three rather than four

slots obviously makes the RL problem more tractable, and so means that we can train dialogue strategies to test the hypothesis in a shorter amount of time. Motivation for the four-slot experiment then includes gathering more evidence with respect to the hypothesis, and in doing so, making sure that the state-action space remains tractable when the number of slots is increased to this more commercially-realistic number. Conducting the four-slot experiment also means that we can compare performance with the four-slot Hybrid RL/Supervised Learning (SL) Strategy of Henderson et al. (2008), and with the hand-crafted COMMUNICATOR systems - in many COMMUNICATOR dialogues, the user required just a single-leg flight and so only four slots were relevant. The evaluation measures for the COMMUNICATOR systems come from dialogues with real users, and so this comparison should be taken with a pinch-of-salt. However, the comparison with the Hybrid Strategy is much more meaningful because the Hybrid Strategy was tested with a simulation derived from COMMUNICATOR data using linear function approximation (see Section 3.10.4), and this simulation has been shown to simulate very similar dialogues to the n-gram simulations (Georgila et al., 2006)[1].

The remainder of this chapter proceeds as follows. Section 5.2 will first describe the experimental design and methodology used in both experiments. This includes the basic RL experimental setup, description of the reinforcement learner and n-gram simulations, and details concerning training, testing and evaluation. Where we describe the n-gram simulations, we discuss their limitations, one of which is the limit that they place on the amount of dialogue history which is potentially useful in learning a dialogue strategy. Sections 5.3 and 5.4 then describe the three and four-slot experiments respectively. Both include detailed analysis of the effects of the recent DAs on the learned strategies. Finally, Section 5.5 will draw conclusions based on the findings of the two experiments.

5.2 Experimental methodology

In the experiments described in this chapter, the following four agents were used to simulate dialogues and learn and test dialogue strategies:

1. the DIPPER Information State Update (ISU) Dialogue Manager (DM) (Bos et al., 2003),

[1] Indeed, although the numbers are not reported, Henderson et al. (2008) states that the Hybrid Strategy was tested with the n-gram simulations and that this produced similar results.

2. n-gram user simulations (Georgila et al., 2005b,a), (one for training, one for testing),

3. a Reinforcement Learning (RL) program,

4. a mapping agent.

The roles of DIPPER, the user simulations and the Reinforcement Learning (RL) program were the same as in the Bayesian Network user simulation experiments of the previous chapter, (see Section 4.2), and all of the agents again communicated via the Open Agent Architecture (OAA) (Cheyer and Martin, 2001). The mapping agent is required to map the responses generated by the user simulations into a form that can be interpreted by DIPPER. Figure 5.1 shows this experimental setup. The task domain is again flight-booking, and the aim for the Dialogue Manager (DM) is to obtain values for the user's flight information "slots" before making a database query. As stated in the previous section, the slots in the (first) three-slot experiment are *departure city*, *destination city*, *departure date*, and the (second) four-slot experiment uses an additional slot for *departure time*.

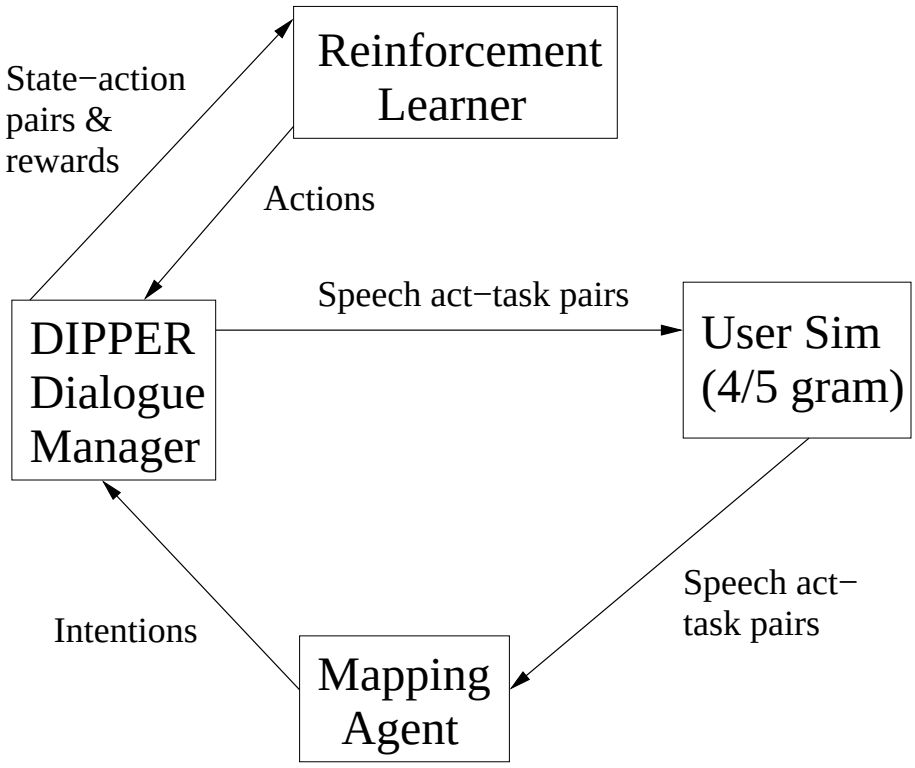

Figure 5.1: The basic experimental setup

5.2.1 The reinforcement learner's parameter settings

For the two experiments described in this chapter, the reinforcement learner's parameters were set as follows:

1. Step-size parameter: $\alpha = decreasing$,
2. Discount factor: $\gamma = 1$,
3. Action selection type = *softmax* (alternative is ε-*greedy*),
4. Action selection parameter: in the three-slot experiment, *temperature* = 15, and in the four-slot experiment, 45 for states visited < 5 times, otherwise 15,
5. Eligibility Trace (ET) Parameter: $\lambda = 0.9$,
6. Eligibility Trace (ET) = *replacing* (alternative is *accumulating*),
7. Initial *Q-values* = 25.

As in the experiments described in Chapter 4, the reinforcement learner updates its Q-values using the *Sarsa(λ)* algorithm (see Sutton and Barto (1998) or Section 2.11). Section 4.2.3 explains why these parameter settings are suitable for learning dialogue strategies.

5.2.2 The n-gram user simulations and their limitations

This chapter's experiments use the 4 and 5-gram versions of the n-gram simulations (Georgila et al. 2005a,b), which were previously introduced in Section 3.10.4. Recall that these take as input and output DATE (Walker and Passonneau, 2001) Dialogue Act (DA) tags, (see Section 3.10.1 for an introduction to DATE), and that the quality of the 4 and 5-gram user simulations has been established through a variety of metrics and against the behaviour of the actual users of the COMMUNICATOR systems, (see Georgila et al. 2005a). Also recall that the n-gram simulations incorporate the effects of Automatic Speech Recognition (ASR) and Natural Language Understanding (NLU) errors because they are built from the user utterances as they were recognised by the input components of the original COMMUNICATOR systems. Since they do not provide instantiated slot values e.g. a response to provide a destination city is the speech

act-task pair *"[provide info] [dest city]"*, we cannot assume that two such responses in the same dialogue refer to the same destination cities. Hence in the Dialogue Manager's (DM's) "Information State" where we record whether a slot is "empty", "filled", or "confirmed", we only update from "filled" to "confirmed" when the slot value is implicitly or explicitly confirmed. The additional mapping agent maps the user speech-act-task pairs to a form that can be interpreted by the DM. Post-mapping user responses are made up of one or more of the following types of utterance, each represented by a Dialogue Act (DA):

1. an ASR rejection i.e. the speech recogniser cannot form a hypothesis / is not confident enough in any of its hypotheses,
2. provide 1 or more slot values,
3. yes,
4. no,
5. ask for help,
6. hang-up,
7. out-of-domain: Note that in human-machine dialogue, a user utterance can often be within-domain but ASR / Natural Language Understanding (NLU) errors cause the system to consider it out-of-domain.

The nature of the user and input component simulation determines the nature of the learned strategies. If a learned strategy is to be directly implemented in a real dialogue system, then the user and input component simulations used to produce it should be as realistic as possible. For this reason we should note how the n-gram simulations might not always produce totally realistic behaviour. A first cause of unrealistic behaviour is imperfect training data. Perhaps in some dialogue contexts, user behaviour in the COMMUNICATOR data does not provide an accurate representation of user behaviour in human-machine flight-booking dialogues in general. For example, if the system fails to recognise anything at all from the user at a particular point in the dialogue, users in the data may respond best to one kind of repair strategy e.g. repetition, but in general users prefer an alternative e.g. switch focus to a different slot. In this case, training with the n-gram simulations can be expected to produce a repair strategy for this context which is not the best in practice.

Another cause of unrealistic behaviour is the way in which the n-grams model the data - firstly, they use a limited amount of dialogue history ($n-1$ turns), and secondly, this dialogue history is represented only in terms of DAs. On this second point, we could certainly speculate as to what other contextual features from the last $n-1$ turns apart from DAs might be important. However, the problems inherent in only modelling a limited amount of dialogue history are clearer, and so this is what we focus on now. One undesirable consequence of modelling a limited amount of dialogue history is that an n-gram simulation might unnecessarily repeat a particular slot value because it has "forgotten" that it already supplied this value at some point beyond $n-1$ turns ago in the dialogue. A real user is more likely to remember and so not re-supply the slot value. A further undesirable consequence concerns failure to simulate impossible-to-fill slots, and another, the amount of dialogue history which is potentially useful to the reinforcement learner in learning a dialogue strategy. We focus on these two issues in the following two subsections.

5.2.2.1 Impossible-to-fill slots

On rare occasions, the real COMMUNICATOR systems did seem to encounter impossible -to-fill slots e.g. because the speech recogniser had difficulty with the user's accent. However, in preliminary Reinforcement Learning (RL) experiments with the n-gram simulations, (not reported here), it was found that the n-gram simulations fail to simulate impossible-to-fill slots, and this must be because they use only a limited amount of dialogue history. Before discovering this, we had intended to experiment with using a dialogue length state feature and a partial reward function. The hope was that the reinforcement learner would then be able to learn a strategy which at some stage gave up trying to fill an impossible-to-fill slot, and instead queried the database with the other slots filled/confirmed. This then is an example of how state features can assume more or less importance based on the kind of behaviour which the simulation is able to generate.

5.2.2.2 How much DA history should ideally be given to the reinforcement learner

Following a system action, which new RL state we transition to depends on the user response. Hence for learning a dialogue strategy, if we assume that there are no problems with tractability, then the RL state should ideally represent all of the information

which in combination with the current system action, affects the next user response. For the n-gram simulations here, we know that the next user action is potentially affected by the DAs of the last $n-1$ turns, which for the 4-gram is the DAs of the last 3 turns, and for the 5-gram, the DAs of the last 4 turns. Figure 5.2 represents a portion of dialogue where time is discrete, each unit of time contains one system-user exchange, and the current time is t. Here, if the user actions are being generated by a 4-gram simulation, then the user action at time t i.e. $User_t$ is potentially affected by Sys_t, $User_{t-1}$ and Sys_{t-1}, but not by $User_{t-2}$, (a 5-gram would potentially be affected by $User_{t-2}$). Hence, ideally, Sys_t should combine with Sys_{t-1} and $User_{t-1}$ to produce a 3-gram which maximises the likelihood of eliciting the desired response in $User_t$. This means that until they have been shown not to improve the learned strategy, the DAs of the last two turns should be represented in the RL state because in combination with the current system action, they potentially affect the next user response, and so could potentially improve the learned strategy. In general, when training with an n-gram user simulation, if we start with a state containing only the slot-status features, then adding the DAs of the last $n-2$ turns will potentially improve the learned strategy i.e. the last 2 turns for a 4-gram, the last 3 turns for the 5-gram, and for argument's sake, the last 98 turns for a 100-gram.

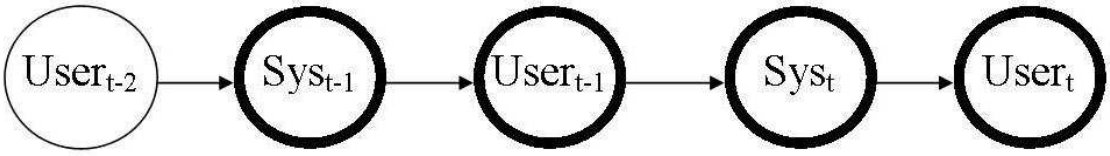

Figure 5.2: A representation of a portion of dialogue in which time is discrete, each unit of time contains one system-user exchange, and the arrows indicate the flow of the dialogue. A 4-gram simulation would output a new action for $User_t$ depending on the actions at the previous turns that have bold circles.

In our experiments here, we train with the 4-gram user simulation and test with the 5-gram and vice versa. Hence, the 4-gram is always involved in either training or testing, and this means that having added the DAs of the last 2 turns to the state, the learned strategy's test performance cannot be improved by also adding the DAs of any earlier turns. When we train with the 5-gram, adding the DAs of the 3^{rd} turn back will potentially affect the learned strategy - we may observe different learned actions where the DAs are the same in the 1^{st} and 2^{nd} turns back, but different in the 3^{rd}. However, since the 3^{rd} turn back cannot affect the 4-gram's next action, we will have

produced two different learned actions for what the 4-gram otherwise considers to be the same context. Therefore, unless both of these actions are optimal in this context with the 4-gram, this will cause a negative impact on the strategy's test performance. Hence, including the 3^{rd} turn back in the state can only reduce the learned strategy's performance in testing with the 4-gram, not improve it.

Note that the DAs of a turn earlier than n-2 turns could potentially improve the learned strategy if the state does not yet include the DAs for all of the turns as far back as n-2. This is because the DAs of the earlier turn could be used to predict the DAs for the missing turn, and if this missing turn is important, then being able to predict its value to some degree might improve the learned strategy. If slot-status features are not included in the state, and if we pretend for a moment that there would be no problem with tractability, then greater than n-2 turns can potentially improve the strategy. This is because turns further back than n-2 in the dialogue history can give the reinforcement learner information about which slots are still unfilled/unconfirmed. Of course, the tractability problem means that doing away with the slot-status features is not a practical approach.

5.2.3 Training and evaluation

In each experiment, two strategies were learned for each state representation - the first with the 4-gram user simulation (the 'a' strategy) and the second with the 5-gram user simulation (the 'b' strategy). The average reward obtained over each 1000 training dialogues was recorded. This allowed graphs to be generated that tracked the improvement of the learned strategies during training. If a strategy was trained with the 4-gram simulation then it was tested with the 5-gram and vice versa. Each strategy was tested over 10 sets of 100 dialogues. For each test dialogue, the total reward obtained was calculated according to two different reward functions. The first was the reward function used for training (which will be described in Section 5.3.4), and the second was the reward function used by Henderson et al. (2005) to compute the task completion scores for the original hand-crafted COMMUNICATOR systems. This will be referred to as the HLG05 reward function and is based on *task completion* and *dialogue length* rewards as determined by the PARADISE evaluation (Walker et al., 2000). This function is as follows:

1. database query, $+25$ for each filled slot, another $+25$ for each slot which is confirmed;

2. system turn penalty: -1.

The maximum possible score for a single dialogue is 197 i.e. 200 minus 3 actions, (the system prompts the user, the user replies by filling all of the slots in one turn, the system then asks for confirmation of a slot, the user gives confirmation for all of the slots, and then the system queries the database for an appropriate flight). The average score for the 1242 dialogues in the COMMUNICATOR data-set where the aim was to fill and confirm the same four slots as we have used in the first experiment was 115.26. Both of the 'a' and 'b' strategies learned for each state representation were tested over 10 sets of 100 dialogues. A one-tailed independent samples t-test, (see page 427 of Sheskin 2007), was first applied to the test data for the 'a' strategies in order to determine whether adding the Dialogue Act (DA) features to the state led to a significant improvement in performance. The same was then done for the 'b' test data, and finally to the average of the 'a' and 'b' test data.

This then completes the general description of our experimental setup and methodology, and so we now go on to describe the first RL experiment with the n-gram simulations - the three slot experiment.

5.3 Three slot experiment to investigate the usefulness of recent Dialogue Acts

5.3.1 Hypothesis: Adding recent Dialogue Acts to the state will improve the learned strategy

The main hypothesis tested by this experiment is that adding the Dialogue Acts (DAs) of the last user turn, and then last system turn to the slot-status features already in the state will produce significant incremental improvements in the performance of the learned strategy when tested with the n-gram simulations.

5.3.2 State representations

In order to test the hypothesis, strategies were trained with both the 4 and 5-gram simulations using the following three state representations:

- **Slot-Status**: The state contains a slot-status feature for each of the three information slots. Each has three possible values - “empty”, “filled” and “confirmed”.
- **DA1**: The state contains the slot-status features and the Dialogue Act(s) (DAs) of the last user turn i.e. the DA(s) of the last 1 turn and hence the name DA1.
- **DA2**: The state contains the slot-status features and the DAs of the last user and system turns i.e. the DAs of the last 2 turns and hence the name DA2.

A strategy learned with the Slot-Status state representation is referred to as a Slot-Status Strategy, a strategy learned with the DA1 state representation, as a DA1 Strategy, and a strategy learned with the DA2 state representation, as a DA2 Strategy. If the hypothesis is true then, DA1 Strategies ought to significantly outperform Slot-Status Strategies, and DA2 Strategies ought to significantly outperform DA1 Strategies.

5.3.3 The action set for the learner

Below is a list of all of the different actions that the RL Dialogue Manager (DM) can take and must learn to choose between based on the context:

1. an open question e.g. ‘How may I help you?’,
2. ask the value for any of slots 1...i,
3. explicitly confirm any of slots 1...i,
4. ask for the i^{th} slot whilst implicitly confirming slot value $i-1$ or $i+1$, (if $i=1$, $i-1$ is considered to be the final slot, and if i is the final slot, $i+1$ is considered to be the first slot e.g. ‘So you want to fly from Edinburgh to where?’),
5. give help,
6. pass to human operator,
7. database query.

There are two restrictions regarding which actions can be taken in which states:

1. An open question is only available at the start of the dialogue.
2. The DM can only confirm non-empty slots.

5.3.4 The reward function

This experiment used the following "all-or-nothing" reward function.

1. database query, all slots confirmed: $+100$,
2. any other database query: -75,
3. user simulation hangs-up: -100,
4. DIPPER passes to a human operator: -50,
5. each system turn: -5.

This reward function rewards confirmed slots-values because in general, these are more likely to be correct than unconfirmed slot-values. The maximum reward that can be obtained for a single dialogue is 85, (the DM prompts the user; the user replies by filling all of the slots in a single utterance; the DM asks for confirmation; the user confirms all of the slots; the DM submits a database query). The reward function is "all-or-nothing" - it only gives positive reward when a database query is made with all of the slots confirmed. This is due to the results of the final Bayesian Network user simulation experiment, (see Section 4.5), where for a 4-slot system, learning was faster with an "all-or-nothing" reward function than it was with a "partial reward" function.

5.3.5 The Dialogue Manager's context update rules

We were unsure whether to follow Henderson et al. (2005, 2008) and to update a slot's status from "filled" to "confirmed" following an implicit confirmation of that slot and a user response that was out-of-domain or an Automatic Speech Recognition (ASR) rejection. Hence we learned two versions of each strategy - a first where the Dialogue Manager (DM) did update the slot's status in these cases, and a second in which it did not. We investigated whether this had any effect on the learned strategy.

The next section provides the test results for the learned strategies.

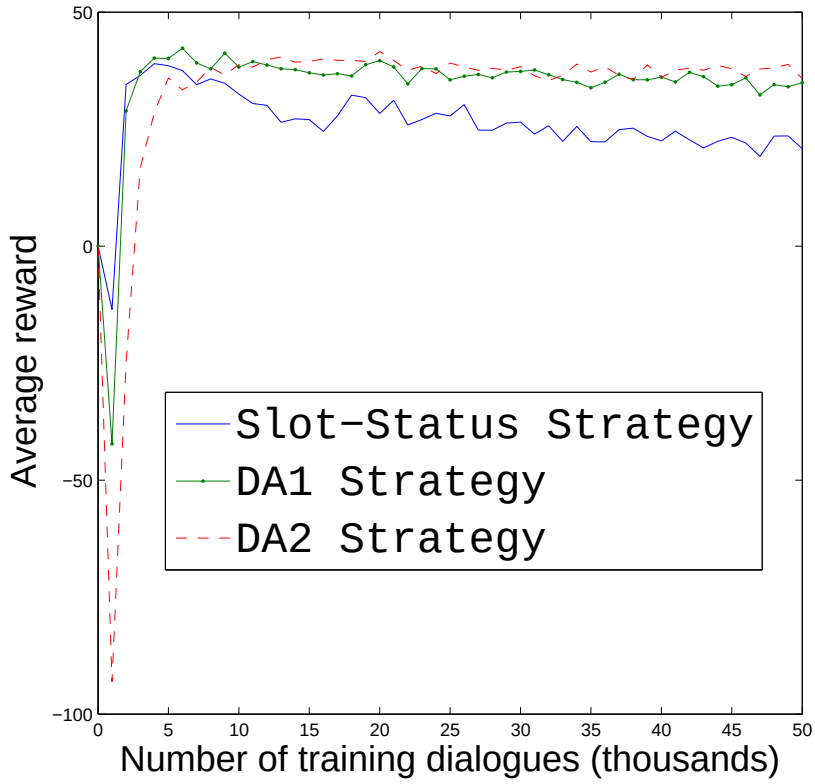

Figure 5.3: Training the three-slot dialogue strategies with the 4-gram simulation

5.3.6 Results

We did not find that the change in context update rule for implicit confirmations described in the previous section had any significant effects on scores obtained by the learned strategies in testing. Hence, here we provide the results for the first set of learned strategies i.e. those learned using a rule that updated a slot's status from "filled" to "confirmed" following an implicit confirmation of that slot and a user response that was out-of-domain or an ASR rejection.

Figure 5.3 tracks the improvement of the three learned strategies during training with the 4-gram simulation, and Figure 5.4, their improvement during training with the 5-gram simulation. They show, according to the training reward function described in Section 5.3.4, the average score per dialogue over intervals of 1000 training dialogues. After 50000 training dialogues with the 4-gram simulation, 164 state-action pairs had been visited for the Slot-Status Strategy, 2538 for DA1 and 5981 for DA2. After 50000

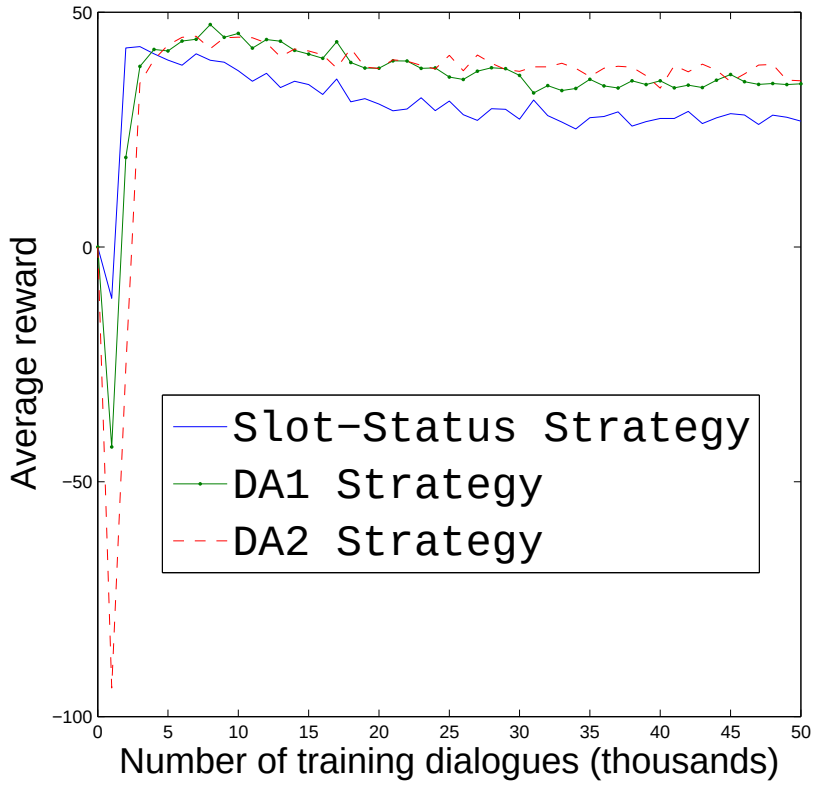

Figure 5.4: Training the three-slot dialogue strategies with the 5-gram simulation

training dialogues with the 5-gram simulation, 164 state-action pairs had been visited for the Slot-Status Strategy, 2393 for DA1, and 5403 for DA2.

Table 5.1 displays the test results for the strategies learned after 50000 training dialogues, (the Slot-Status Strategy, DA1 Strategy, DA2 Strategy). Column 3 shows the average scores according to the training reward function obtained per dialogue by each strategy over 1000 test dialogues. Columns 4 and 5 show the average scores per dialogue for the component parts of the reward function - the score for the database query and the length of the dialogue respectively.

Let us first concentrate on column 3. The 1000 test dialogues for each strategy were divided into 10 sets of 100. As stated in Section 5.2.3, one-tailed independent samples t-tests were used to test for significance. The DA2 Strategy improves over the DA1 Strategy by 1.53% ($p < 0.05$) and over the Slot-Status Strategy by 15.28% ($p < 0.005$), and DA1 improves over the Slot-Status Strategy by 13.53% ($p < 0.005$). Hence we can

	Features	**Av. Score**	**Conf. Slots**	**Length**
$4 \rightarrow 5$ **gram = (a)**				
Slot-Status Strategy(a)	Slot-status	47.55	100	10.49
DA1 Strategy(a)	+ Last User DA(s)	58.00**	100	8.40
DA2 Strategy(a)	+ Last System DA	59.15*	100	8.17
$5 \rightarrow 4$ **gram = (b)**				
Slot-Status Strategy(b)	Slot-status	55.60	100	8.88
DA1 Strategy(b)	+ Last User DA(s)	59.10**	100	8.18
DA2 Strategy(b)	+ Last System DA	59.75*	100	8.05
Slot-Status Strategy(av)	Slot-status	51.60	100	9.68
DA1 Strategy(av)	+ Last User DA(s)	58.55**	100	8.29
DA2 Strategy(av)	+ Last System DA	59.45*	100	8.11

Table 5.1: Testing the learned strategies after 50000 training dialogues, average reward achieved per dialogue over 1000 test dialogues. a = strategy trained using 4-gram and tested with 5-gram; b = strategy trained with 5-gram and tested with 4-gram; av = average; * significance level of improvement on above strategy $p < 0.05$; ** significance level of improvement $p < 0.005$

already say that the main hypothesis as stated in Section 5.3.1 has been successfully tested - adding the DAs of the last user turn, and then last system turn to the slot-status features already in the state does produce significant incremental improvements in the performance of the learned strategy when tested with the n-gram simulations.

Let us now consider the fourth and fifth columns which break up the reward into its two component parts - the dialogue query reward, and dialogue length penalty respectively. The fourth column shows that all five of the strategies are always able to fill and confirm all of the 3 slots when conversing with the simulated COMMUNICATOR users. As stated in Section 5.2.2, the n-gram simulations do not simulate the case of a particular user goal utterance being always unrecognisable/unobtainable. The variation in performance of the strategies was due to the average dialogue length i.e. the number of system turns required to fill and confirm all of the slots. This was 8.11 turns for the DA2 Strategy, 8.29 turns for the DA1 Strategy, and 9.68 turns for the Slot-Status Strategy.

We now move on to provide analysis in order to explain the relative performance of the different learned strategies in the tests reported here.

5.3.7 Analysis

In testing, the DA2 Strategy filled and confirmed all of the slots in the least number of turns on average, the DA1 Strategy the second least, and the Slot-Status Strategy third. The vast majority of this section is concerned with analysing the learned strategies in order to form hypotheses as to why this was the case. These hypotheses will be investigated further in the experiments of Chapter 7. However we start by looking at the effect of the change in the Dialogue Manager's (DM's) context update rules, as described in Section 5.3.5.

5.3.7.1 The effect of the different context update rules

As stated in Section 5.3.6, the change in context update rules for implicit confirmations, (see Section 5.3.5), did not have any significant effects on the scores obtained by the learned strategies in testing. However we did notice that this change had apparently important effects on the Slot-Status Strategy's behaviour in certain states. Some of these were good, and some bad, and given that there was no significant change in the test scores, it must be that these good and bad effects cancelled each other out.

These important behavioural effects occurred because often, if the last system turn had involved an implicit confirmation, different actions were then optimal on the next system turn depending on whether the user response had been out-of-domain / an ASR rejection, or an unambiguous acceptance of the implicit confirmation. Changing the update rule affected whether the learner was able to distinguish between such cases, and hence learn alternative actions. Tables 5.2 and 5.3 show example state-learned action pairs which illustrate this point. For Table 5.2, assume that a slot's status is being updated to "confirmed" following an implicit confirmation and a user response of "user(out-of-domain)" or "user(asr_rejection)". Here then, the third slot, was "filled", but is updated to "confirmed" following the system action "impConfSlot3_askSlot2" and user responses of "user(out-of-domain)" / "user(asr_rejection)" / "user(slot2)". Since the second slot was already "filled", for each of the different user responses, the slot-status features end up with the same values. Hence the Slot-Status Strategy cannot distinguish between these different contexts and so the learned action is the same for

each - “exp_conf_slot2”. However, when the user response is “user(out-of-domain)”/ “user(asr_rejection)”, a better action is to re-ask slot 2 because “exp_conf_slot2” is likely to elicit another unrecognised / out-of-domain user utterance. The DA2 (and DA1) state produced the better learned action because the representation of the DAs of the last user turn enabled the learner to distinguish between these contexts.

DA2 state representation	**DA2 Strategy**	**Slot-Status Strategy**
[conf,fill,conf,icSlot3_askSlot2,user(out-of-domain)]	ask_slot2	exp_conf_slot2
[conf,fill,conf,icSlot3_askSlot2,user(asr_rejection)]	ask_slot2	exp_conf_slot2
[conf,fill,conf,icSlot3_askSlot2,user(slot2)]	exp_conf_slot2	exp_conf_slot2

Table 5.2: Here the Dialogue Manager (DM) updates 3^{rd} slot’s status to “confirmed” following an implicit confirmation and a user response that is out-of-domain / an ASR rejection, and as a result the Slot-Status state is the same as if the user response was an unambiguous acceptance of the confirmation; ic = implicit confirm.

DA2 state representation	**DA2 Strategy**	**Slot-Status Strategy**
[fill,emp,emp,ask_slot1,user(slot1)]	icSlot1_askSlot2	icSlot1_askSlot2
[conf,emp,emp,icSlot1_askSlot2,user(out-of-domain)]	ask_slot3	ask_slot3
[conf,emp,emp,icSlot1_askSlot2,user(asr_rejection)]	ask_slot3	ask_slot3
[conf,fill,emp,icSlot1_askSlot2,user(slot2)]	ask_slot3	ask_slot3

Table 5.3: Here the Dialogue Manager (DM) updates the 1^{st} slot’s status to “confirmed” following an implicit confirmation and a user response that is out-of-domain / an ASR rejection, and as a result, the Slot-Status state is **not** the same as if the user response was an unambiguous acceptance of the confirmation; ic = implicit confirm.

In Table 5.2 then, we have seen a bad effect of updating a slot’s status following “user(out-of-domain)” / “user(asr_rejection)”, but in Table 5.3, we see a good effect. Here, the first slot is originally “filled”, and the other two slots are “empty”. The optimal action is then apparently “impConfSlot1_askSlot2”. However, if the next user response is “user(out-of-domain)” or “user(asr_rejection)” as opposed to “user(slot2)”, then although the first slot’s status is updated to “confirmed”, the second slot’s status does not become “filled”. Hence on this occasion, updating has allowed the learner to distinguish between the two contexts, and so learn alternative actions. Had the learner not updated, then the slot-status features would have remained as they were originally, and so the learned action would have been “impConfSlot1_askSlot2”. This was

observed to be a relatively poor action choice because it was likely to elicit another unrecognised / out-of-domain user utterance. Again, the DA2 (and DA1) state produced the better learned actions because the representation of the DAs of the last user turn enabled the learner to distinguish between these contexts.

We now move on to the rest of the analysis of our learned strategies, which concentrates on finding an explanation for why the DA2 Strategy filled and confirmed all of the slots in the least number of turns on average, the DA1 Strategy the second least, and the Slot-Status Strategy third.

5.3.7.2 Different repair strategies in SSFU states

Observing the test dialogues, I noticed that the strategies often behaved differently in the following two categories of states, the vast majority falling into the 1^{st} category:

1. States in which the Slot-Status Features are Unchanged by the previous user turn (SSFU states): SSFU states are most often caused by a user response that is out-of-domain or an ASR rejection. Other causes are seemingly uncooperative user DAs of "yes" or "no" in response to a straightforward question for a slot value. Consider an example where the slot-status features are initially [filled,empty,empty], and then the user turn is "user(slot1)". As a result, the DM "re-fills" the 1^{st} slot by setting it to "empty" and then "filled", and so although the slot-status features end up as [filled,empty,empty], such states are not considered SSFU states.

2. Other states in which the last user response was "user(out-of-domain)" or "user(asr_rejection)", but in which the slot-status features have changed because the last system DA involved an implicit confirmation i.e. a slot has been updated from "filled" to "confirmed" due to this implicit confirmation, (see Table 5.2 for an example).

Thus the strategies behaved differently when dialogue progress stalled i.e. they used different repair strategies. It seemed that depending on the Dialogue Acts (DAs) of the last two turns, certain repair strategies were more "effective" than others - here "effective" means that a repair strategy is good at getting the dialogue "back-on-track" by eliciting a recognisable slot-value/confirmation from the user. Note that the category 2 states are category 1 states for the strategies learned with a Dialogue Manager (DM)

that does not update a slot's status from "filled" to "confirmed" following an implicit confirmation and a user response "user(out-of-domain)" / "user(asr_rejection)". Hence for simplicity's sake, the analysis here is based on these strategies.

To further investigate what the different learned strategies were doing in SSFU states, we collected relevant statistics based on five new sets of 100 test dialogues for each strategy. Again, if a strategy had been trained with the 4-gram user simulation, then it was tested with the 5-gram and vice-versa. The different repair strategies which were observed in SSFU states can be categorised as the following:

1. **Repeat**: repeat the question/confirmation which led to the first SSFU state in the current sequence of SSFU states e.g. if following smooth progress, the system action "ask_slot1" then leads to an SSFU state, so long as the system remains in an SSFU state, the action "ask_slot1" is classified as *repeat*,
2. **Switch focus**: switch focus to ask/confirm a different unfilled/unconfirmed slot e.g. if the last system DA was "ask_slot1" but the user failed to supply a value, assuming that slots 2 and 3 are "empty", "ask_slot2" or "ask_slot3" would be switching focus,
3. **Give help**: use the give help function,
4. **Backtrack**: re-ask an already filled slot.

For each learned strategy, Table 5.4 shows a number of statistics relating to the occurrence of states in which the slot-status features are unchanged i.e. Slot-Status Features Unchanged (SSFU) states. Notice first of all, that there is very little difference between the learned strategies in the number of first-visits to an SSFU state i.e. in a series of two or more consecutive SSFU states, only the first is counted. However there are significant differences in the overall number of visits to an SSFU state - the Slot-Status strategy produces the most such turns, the DA1 Strategy the second most, and the the DA2 Strategy the least. This suggests that in general, when in an SSFU state, the DA2 Strategy employs repair strategies which are the most likely to ensure that the next state is not also an SSFU state, the DA1 Strategy employs the second most effective repair strategies, and the Slot-Status Strategy, the least effective. In other words, once progress stalls, the DA2 Strategy is the best at getting the dialogue back-on-track, the DA1 Strategy the second best, and the Slot-Status Strategy, the worst. Notice also that there are different levels of variation in the kinds of repair strategies employed by each

of the learned strategies - DA1 employs a greater variety than the Slot-Status, and the DA2 Strategy a greater variety still. This is better illustrated by Figure 5.5. Given the statistics for first and total visits to SSFU states, the different levels of variation suggest that different repair strategies are more effective depending on the Dialogue Acts (DAs) of the last two turns.

Strategy	**First-visits to SSFU states**	**Total visits to SSFU states**	**%Repeat**	**%Switch focus**	**%Back -track**	**%Give help**
Slot-Status	0.90	2.51	90.82	9.18	0	0
DA1	0.91	1.19*	59.24	29.83	10.92	0
DA2	0.90	1.06*	53.99	37.79	7.98	0.23

Table 5.4: The number of first and total visits to SSFU states per dialogue, and the proportion of times which each of the four possible repair strategies is applied; * Improvement level on strategy above with significance level $p < 0.05$

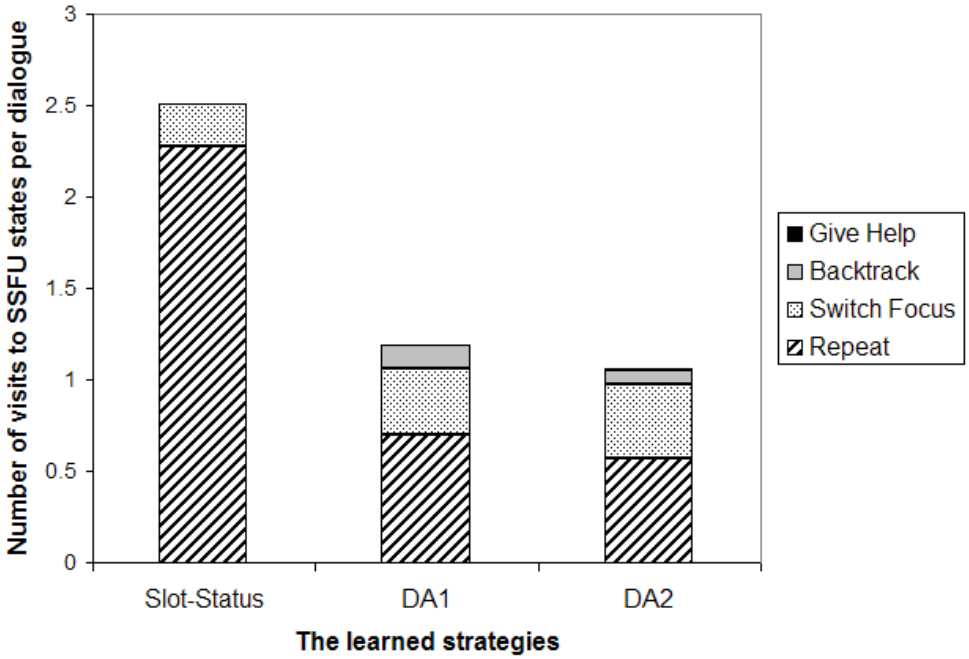

Figure 5.5: The number of total visits to SSFU states per dialogue, and the proportion of times which each of the four possible repair strategies is applied.

Let us now consider the frequency with which different user responses cause SSFU states. This is shown in Figure 5.6. The reader can see that for all three strategies, the vast majority are caused by non-understanding errors: user responses which are recognised as out-of-domain, or are ASR rejections, (see Section 1.1.1.2 for a definition of

non-understanding errors). Notice that in particular, there is a much larger number of SSFU states caused by out-of-domain user responses for the Slot-Status Strategy than there is for the DA1 and DA2 Strategies, (1.67 on average per dialogue versus 0.24 and 0.44 respectively). In the real user experiments of the next chapter, we find that the Slot-Status Strategy's over-use of the repeat repair strategy causes user frustration, and so hyper-articulate speech, which in turn causes more ASR errors and hence longer dialogues with a lower chance of task completion. This then is presumably what the n-gram simulations are simulating here - the Slot-Status Strategy is eliciting more user responses which are being misrecognised and hence classified as out-of-domain.

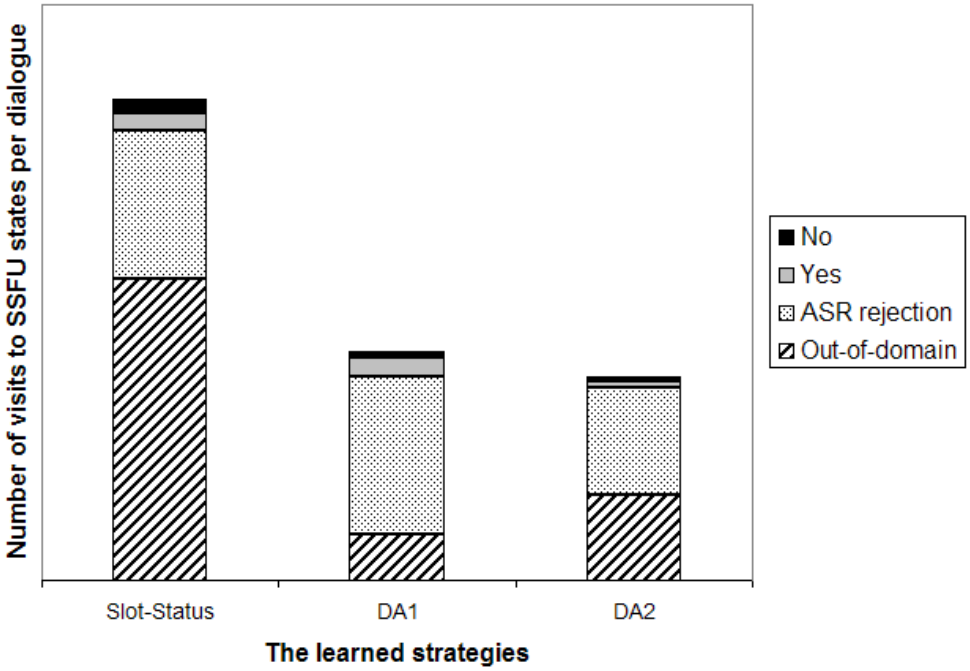

Figure 5.6: The frequency with which different user responses cause SSFU states for each of the three learned strategies.

Figure 5.7 shows the relative frequency with which the DA2 Strategy employs each of the four possible repair strategies depending on which user DA(s) caused the current SSFU state. Since Figure 5.6 showed that out-of-domain user utterances and ASR rejections are the main cause of SSFU states, these are of most interest here. Notice then that the DA2 Strategy uses the *switch focus* repair strategy most often for user utterances that are recognised as out-of-domain (66.67%), but the repeat repair strategy most often for ASR rejections (83.04%). Given what we said in the previous paragraph about out-of-domain user utterances and potential error spirals, it makes sense that the *switch focus* repair strategy would usually be more effective for out-of-domain

utterances. On the other hand, when the system has failed to recognise anything from the user i.e. there is an ASR rejection, potential causes are that the user was not ready to speak / was distracted, or that there was some background noise such as a door-slam. In such circumstances, it would seem more appropriate to repeat the previous question / confirmation and so give the user another opportunity to answer. Hence the general trends suggested by Figure 5.7 seem to make sense.

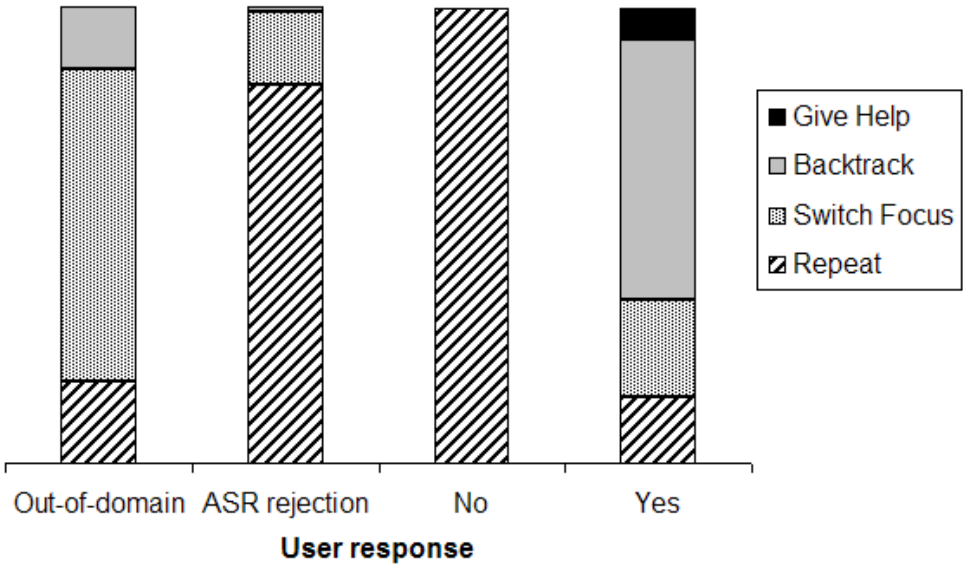

Figure 5.7: The relative frequency with which the DA2 Strategy uses each of the four possible repair strategies in SSFU states following different user responses.

To summarise the key point of this section then, it seems that adding recent DAs into the state has improved the learned strategy by producing better repair strategies for SSFU states. In Chapter 7, the experiment described in section 7.2 investigates whether this is the only way in which the recent DAs are improving the learned strategy.

In the next section here, we now go on to provide example dialogues which involve the kinds of repair strategies which we have just described.

5.3.7.3 Example dialogues involving different repair strategies

Let us now consider some example dialogues in order to see examples of the different repair strategies in Slot-Status Features Unchanged (SSFU) states. Table 5.5 shows an example dialogue conducted using the Slot-Status Strategy, (learned with the 5-gram simulation, tested with the 4-gram). At a number of points in the dialogue, a

System state	System action	Recognised user response
[emp,emp,emp]	askSlot1	Out-of-domain
[emp,emp,emp]	askSlot1	Out-of-domain
[emp,emp,emp]	askSlot1	“New York” (slot 1)
[fill,emp,emp]	impConfSlot1_askSlot2	Out-of-domain
[fill,emp,emp]	impConfSlot1_askSlot2	ASR rejection
[fill,emp,emp]	impConfSlot1_askSlot2	“April 18th” (slot 3)
[fill,emp,fill]	impConfSlot3_askSlot2	“Miami” (slot 2)
[fill,fill,conf]	impConfSlot2_askSlot1	“Newark” (slot 1)
[fill,conf,conf]	expConfSlot1	Out-of-domain (slot 1)
[fill,conf,conf]	expConfSlot1	“New York” (slot 1)
[fill,conf,conf]	expConfSlot1	“yes”
[conf,conf,conf]	dbQuery	

Table 5.5: An example Slot-Status Strategy dialogue. Whenever the slot-status features are unchanged, the Slot-Status Strategy’s repair strategy is to repeat.

user utterance fails to fill/confirm a slot value, and so on its next turn, the system must employ a repair strategy. In all of these SSFU states, the Slot-Status Strategy’s learned action is simply to repeat its previous action e.g. ask for slot 1 again in the first and second cases. Without any representation of the previous user turn, the state is unchanged, and hence the reinforcement learner could not learn a different action for this context. Indeed the only SSFU Slot-Status state in which the reinforcement learner could have learned to do anything other than repeat is a state with the representation “[empty,empty,empty]”, and which immediately follows the start state i.e. “[start]”. Here the state representation has changed and so the learned action can also change. When training with the 5-gram simulation, the reinforcement learner learned to repeat in this state (see Table 5.5), but when training with the 4-gram, it learned to switch focus.

Table 5.6 shows an example dialogue conducted using the DA1 Strategy (again learned with the 5-gram simulation, tested with the 4-gram). This time, a user utterance fails to fill/confirm a slot value at four different points in the dialogue. In the first two cases,

System state	**System action**	**Recognised user response**
[emp,emp,emp,none]	askSlot1	ASR rejection
[emp,emp,emp,asr_rejection]	askSlot1	ASR rejection
[emp,emp,emp,asr_rejection]	askSlot1	"New York" (slot 1)
[fill,emp,emp,giveSlot1]	impConfSlot1_askSlot2	Out-of-domain
[fill,emp,emp,out-of-domain]	askSlot3	"26th June" (slot 3)
[fill,emp,fill,giveSlot3]	impConfSlot3_askSlot2	"Dulles" (slot 2)
[fill,fill,conf,giveSlot2]	impConfSlot2_askSlot1	"Newark" (slot 1)
[fill,conf,conf,giveSlot1]	expConfSlot1	Out-of-domain
[fill,conf,conf,out-of-domain]	askSlot1	"New York" (slot 1)
[fill,conf,conf,giveSlot1]	expConfSlot1	"yes"
[conf,conf,conf,yes]	dbQuery	

Table 5.6: An example DA1 strategy dialogue (learned with 5-gram simulation), showing three different types of repair strategy - repeating, switching focus and backtracking.

the repair strategy used is simply to repeat the previous action, in the third case, it is to switch focus from slot 2 to slot 3, and in the fourth it is to "backtrack" one step and re-ask the second slot value. Since it is now represented in the state, for this strategy, the reinforcement learner has been able to learn repair strategies which are sensitive to the last user Dialogue Act(s). This produces a greater variety of repair strategies than for the Slot-Status strategy - we see more switching focus, and backtracking.

Table 5.7 shows an example dialogue conducted using the DA2 Strategy (again learned with the 5-gram simulation, tested with the 4-gram). Here, a user utterance fails to fill/confirm a slot value at four different points in the dialogue, and we see all four of the different types of repair strategy - give help, repeating, switching focus and backtracking. Since it is now represented in the state, the reinforcement learner has been able to learn repair strategies which are sensitive to both the last system and user Dialogue Act(s). This again seems to increase the variety of repair strategies - we now see a little giving help, more switching focus and again, backtracking.

We now move on to consider the effect of the DAs in states where the user has indicated a misunderstanding error, (the system has obtained an incorrect value for a slot - see

System state	System action	User response
[emp,emp,emp,none,none]	askSlot1	"yes"
[emp,emp,emp,askSlot1,yes]	giveHelp	ASR rejection
[emp,emp,emp,giveHelp,asr_rejection]	askSlot1	"New York" (slot 1)
[fill,emp,emp,askSlot1,giveSlot1]	icSlot1_askSlot2	Out-of-domain
[fill,emp,emp,icSlot1_askSlot2,out-of-domain]	askSlot3	"26th June" (slot 3)
[fill,emp,fill,askSlot3,giveSlot3]	icSlot3_askSlot2	"Dulles" (slot 2)
[fill,fill,conf,icSlot3_askSlot2,giveSlot2]	icSlot2_askSlot1	"Newark" (slot 1)
[fill,conf,conf,icSlot2_askSlot1,giveSlot1]	expConfSlot1	Out-of-domain
[fill,conf,conf,expConfSlot1,out-of-domain]	askSlot1	"New York" (slot 1)
[fill,conf,conf,askSlot1,giveSlot1]	expConfSlot1	"yes"
[conf,conf,conf,expConfSlot1,yes]	dbQuery	

Table 5.7: An example DA2 Strategy dialogue (learned with 5-gram simulation), showing four different types of repair strategy - repeating, switching focus, giving help and backtracking; ic = implicit confirm.

Section 1.1.1).

5.3.7.4 Misunderstandings

Recall from Section 1.1.1 that for an information slot-filling system, we consider a misunderstanding error to have occurred when due to Automatic Speech Recognition (ASR) / Natural Language Understanding (NLU) errors, the system obtains an incorrect slot-value. Table 5.8 shows DA2 Strategy (trained with 5-gram) actions for states where there has been a misunderstanding, and the user indicates the misunderstanding directly after the system attempts to confirm the incorrect value. These state-learned action pairs are all those that were visited 25 or more times during the 50000 training dialogues. The table has five columns - the second contains the DA2 Strategy state representations, the third the DA2 Strategy learned action, the fourth how many times that state was visited during training, and the fifth the Slot-Status Strategy learned action. The table shows that in 16 out of 17 states, the learned action for the Slot-Status and DA2 Strategies is the same. Given then that there is just this one difference in the learned actions, it seems then that the DAs are not making significant improvements

to the learned strategy with respect to dealing with these user indications of misunderstanding errors.

	DA2 state representation	**DA2 Strategy**	**No. visits**	**Slot-Status**
1	[emp,conf,conf,ec1,user(no(slot1))]	ask1	1107	ask1
2	[conf,emp,conf,ec2,user(no(slot2))]	ask2	2022	ask2
3	[conf,emp,conf,ec2,user(no)]	ask2	612	ask2
4	[emp,emp,emp,ic1ask2,user(no(slot1))]	ask1	436	ask1
5	[fill,emp,emp,ec1,user(no(slot2))]	ic1ask2	161	ic1ask2
6	[conf,fill,fill,ec3,user(no,slot3)]	ic3ask2	997	ic3ask2
7	[conf,fill,emp,ic3ask2,user(no,yes)]	ask3	143	ask3
8	[conf,emp,fill,ec3,user(no,slot3)]	ic3ask2	507	ic3ask2
9	[conf,emp,emp,ic3ask2,user(no,yes)]	ask3	49	ask3
10	[emp,emp,conf,ec1,user(no(slot1))]	ask1	28	ask1
11	[fill,emp,conf,ec1,user(no(slot2))]	ic1ask2	47	ic2ask1
12	[emp,fill,conf,ec2,user(no(slot1))]	ic2ask1	47	ic2ask1
13	[conf,emp,emp,ic3ask2,user(no)]	ask3	37	ask3
14	[conf,fill,emp,ec3,user(no)]	ask3	28	ask3
15	[conf,conf,fill,ec3,user(no,slot3)]	ecSlot3	239	ecSlot3
16	[emp,emp,emp,ec1,user(no(slot1))]	ask1	2782	ask1
17	[conf,fill,emp,ic3ask2,user(no)]	ask2	125	ask3

Table 5.8: Example states for misunderstandings where the user indicates the misunderstanding directly after the system tries to confirm the incorrect value; emp = empty; fill = filled; conf = confirmed; ic = implicit confirm; ec = explicit confirm.

Table 5.8 shows example states where the system attempts to confirm an incorrect slot value, and the simulation immediately indicates the misunderstanding error, but what about contexts in which the simulation delays one or more turns before indicating the misunderstanding error? Does the DA2 strategy employ different actions to the Slot-Status strategy in these states? As expected, we found such cases to be much less common than those in which the simulation immediately indicates the misunderstanding error. Table 5.9 shows the only three example state-learned action pairs which were visited over 25 times in the 50000 training dialogues. Here again, the reader can see that the learned actions for the Slot-Status and DA2 strategies are the same - the DA information has not made any difference. Hence again, it seems that the DAs are not improving the learned strategy here.

DA2 state representation	DA2 Strategy	No. visits	Slot-Status
[fill,emp,emp,ec1,user(no(slot2))]	ic1ask2	161	ic1ask2
[emp,fill,conf,ask2,user(no(slot1))]	ic2ask1	48	ic2ask1
[emp,fill,conf,ec2,user(no(slot1))]	ic2ask1	47	ic2ask1

Table 5.9: Example states for misunderstandings where the user indicates the misunderstanding later in the dialogue.emp = empty; fill = filled; conf = confirmed; ic = implicit confirm; ec = explicit confirm

DA2 state representation	Slot-Status	DA1 Strategy	DA2 Strategy
[conf,fill,emp,ask3,user(slot2)]	ask3	ask3	**ask2**

Table 5.10: The user simulation re-supplies a slot value and the learned actions are different for the DA1 and DA2 Strategies.

We did find a couple of example states of the kind shown in Table 5.10, which are non-SSFU states where the user re-supplies a slot value, and the DA2 learned action is different from the DA1 and Slot-Status learned actions. It is unclear what exactly is being simulated in such states - the user could be re-supplying the slot-value as a self-correction, or alternatively, because they are concerned for some reason that the system has made a misunderstanding error and obtained an incorrect value. Here, while the DA1 Strategy asks for slot 3, the DA2 Strategy backtracks and asks for slot 2. It could be that backtracking is the best action to take here, but to learn this action requires knowledge of the last system DA. The DA1 Strategy asks slot 3 because on average, the most likely previous system DA was "impConfSlot1_askSlot2", which means that the dialogue is progressing, and the best action to take is to plough on and ask about the as-yet-unfilled slot 3. Perhaps then we have found an example of a non-SSFU state where the recent DAs are improving the learned strategy, namely the last system DA. However, this state-action, and the other similar example which we found were very infrequently visited - less than 25 times during the 50000 training dialogues. Hence, we cannot be sure that the DA2 Strategy's action is in fact better, and even if it is, the fact that it is so infrequently applied means that is highly unlikely to have a significant impact in testing. If states occur very infrequently in general then they are obviously of less concern in dialogue strategy design. However, if when they do occur, they can

have a significant impact on the likelihood of the dialogue being successful, then they become more important. The same is true if they occur infrequently in general, but frequently with a particular kind of user. This is because the ideal dialogue strategy should be robust to all of the different kinds of users which may interact with the system e.g. naive versus expert, native versus non-native speaker etc.

It seems then that the DAs are not making significant improvements to the learned strategy in the contexts under discussion in this section, but is there anything else to say about the learned actions? Notice that in all but the case of the DA2 Strategy in state 17 of Table 5.8, and the Slot-Status and DA1 Strategy actions in Table 5.10, the learned action maintains focus on the problem slot. Intuitively, this seems to make sense, because for example, having indicated a misunderstanding error, a user is likely to find it disconcerting if the system simply moves on with the dialogue as if everything is okay. As a result, the user may lose all confidence in the system and quit immediately, or adopt an irritated/hyperarticulated tone of voice which causes more ASR errors and so a longer dialogue with a lower chance of task completion. Recall that in the Bayesian Network user simulation experiment described in Section 4.3, we set the probabilities so that the user simulation was very likely to "hang up" if after indicating a misunderstanding error, the system failed to maintain focus on the problem slot. In the section containing analysis of the results (Section 4.3.6), we gave an example misunderstanding error state in which the problem slot could not be inferred with total certainty from the slot-status features alone. This was because on a previous turn, following a non-understanding error, the system had switched focus from slot 1 to slot 2. Clearly in such states, if it is important to maintain focus on the problem slot, the DA(s) of the last user turn would improve the learned strategy because they would resolve the ambiguity regarding the identity of the problem slot. Hence it looks as if in the three-slot experiment here, this kind of scenario is occurring at most very infrequently, (e.g. in Table 5.10). Note that as the amount of switching focus in SSFU states and the number of slots increases, then there will also be an increase in the occurrence of misunderstanding error states in which it is impossible to infer the problem slot with total certainty based on the slot-status features alone. Thus, if it is true that the system ought to maintain focus on the problem slot, then as a result, we would expect the DAs to become more important for learning actions to react to indications of misunderstanding errors.

We can also speculate that given a very accurate user simulation, optimising for deal-

ing with indications of misunderstanding errors may require information about more dialogue history than was used here. For example it may be that in general, following the first or second consecutive indication of a misunderstanding error for a particular slot, users prefer the system to maintain focus on the problem slot, but then if there is a further misunderstanding error, they become exasperated and prefer the user to switch focus. Optimising for dealing with these contexts may also be made easier by using a more sophisticated reward function i.e. one which uses a measure of "user satisfaction". Of course as explained earlier, the n-gram simulations used here place a limit on the amount of dialogue history which can potentially improve the learned strategy, and they do not generate user satisfaction scores. Attempting to optimise learned strategies for dealing with misunderstanding errors using information about more dialogue history and/or user satisfaction scores could be an avenue for future research.

To summarise then, the analysis in this section strongly suggests that at least in the case of the n-gram COMMUNICATOR user simulations, the last system and user DAs are not significantly improving the learned strategy with respect to dealing with user indications of misunderstanding errors. However, there does seem to be a general trend to maintain focus on the problem slot. We now move on to compare the behaviour of the learned strategies in contexts in which the user turns are filling/confirming slot-values i.e. the dialogue is progressing smoothly.

5.3.7.5 Behaviour when user turns fill/confirm slot values

	Slot-Status Strategy	**DA1 Strategy**	**DA2 Strategy**
System:	askSlot1	askSlot1	askSlot1
User:	slot1	slot1	slot1
System:	impConfSlot1AskSlot2	impConfSlot1AskSlot2	impConfSlot1AskSlot2
User:	slot2	slot2	slot2
System:	impConfSlot2AskSlot3	impConfSlot2AskSlot3	impConfSlot2AskSlot3
User:	slot3	slot3	slot3
System:	expConfSlot3	expConfSlot3	expConfSlot3
User:	yes(slot3)	yes(slot3)	yes(slot3)

Table 5.11: System-initiative dialogues in which each user turn fills/confirms a slot value.

	Slot-Status Strategy	**DA1 Strategy**	**DA2 Strategy**
System:	askSlot1	askSlot1	askSlot1
User:	slot1,slot2	slot1,slot2	slot1,slot2
System:	impConfSlot1AskSlot2	impConfSlot1AskSlot2	impConfSlot1AskSlot2
User:	slot3,slot2	slot3,slot2	slot3,slot2
System:	impConfSlot3AskSlot2	impConfSlot2AskSlot3	impConfSlot2AskSlot3
User:	slot3	slot3	slot3
System:	expConfSlot3	expConfSlot3	expConfSlot3
User:	yes(slot3)	yes(slot3)	yes(slot3)

Table 5.12: Mixed-initiative dialogues in which each user turn fills/confirms slots values.

	Slot-Status Strategy	**DA1 Strategy**	**DA2 Strategy**
System:	askSlot1	askSlot1	askSlot1
User:	slot1,slot2	slot1,slot2	slot1,slot2
System:	impConfSlot1AskSlot2	impConfSlot1AskSlot2	impConfSlot1AskSlot2
User:	slot2,slot3	slot2,slot3	slot2,slot3
System:	impConfSlot3AskSlot2	expConfSlot3	impConfSlot3AskSlot2
User:	slot3	yes(slot3)	slot3
System:	expConfSlot3	expConfSlot2	expConfSlot3
User:	yes(slot3)	yes(slot2)	yes(slot3)

Table 5.13: Mixed-initiative dialogues in which each user turn fills/confirms slots values.

Let us now consider the behaviour of the different strategies when the user turns are filling/confirming slot-values in order to assess whether the DA1 and DA2 strategies might have improved over the Slot-Status strategy here as well. Table 5.11 shows example dialogues for each of the three strategies in which the system always maintains the initiative (i.e. the user never over-answers - see Section 1.1.1 for a definition of initiative), and every user turn fills or confirms a slot value. The reader can see that the behaviour of the three strategies is identical - the DA1 and DA2 strategies are not making any performance gains over the Slot-Status strategy here. Table 5.12 shows example mixed-initiative dialogues for each of the three strategies where again no user turn fails to fill or confirm a slot value. Again, the behaviour of the three strategies is identical. In mixed-initiative dialogues where no user turn fails to fill or confirm a slot value, there will be occasional slight differences in the behaviour of the strategies. Table 5.13 shows an example where the DA1 strategy explicitly confirms slot 3 when the Slot-Status and DA2 strategies implicitly confirm slot 3 and ask slot 2. Unlike with the repair strategies in Slot-Status Features Unchanged (SSFU) states, when observing the test dialogues, these slight differences did not appear to affect the length of the dialogue. Therefore we hypothesise that they are primarily due to the random element in the action selection used by the reinforcement learner during training.

We now summarise the most important findings of our analysis of the three-slot learned strategies.

5.3.7.6 Summary

In this section we provided analysis to show that the Dialogue Acts (DAs) of the last system and user turns improve the learned strategy by producing more effective repair strategies for Slot-Status Features Unchanged (SSFU) states. The four types of repair strategy for SSFU states observed in the learned strategies were repeating, switching focus (ask/confirm a different unfilled/unconfirmed slot), backtracking (re-ask a filled slot) and giving help. The principal causes of SSFU states are non-understanding errors, namely user utterances that are recognised as out-of-domain or ASR rejections. The distinction between these two different types of non-understanding error seems to be important because the DA2 Strategy, (the best learned strategy), used the repeat repair strategy most often following ASR rejections, but the switch focus repair strategy most often following out-of-domain user utterances. We did not find compelling evidence to suggest that the DAs were making significant improvements to the learned

strategy with respect to dealing with user indications of misunderstanding errors, nor in portions of dialogue in which there is smooth progress towards the goal of filling and confirming all of the slots. However we did observe a general trend for the reinforcement learner to learn to focus on the problem slot following misunderstanding errors.

In the next section here, we now describe a four-slot version of this section's experiment - the introduction gives the reasons for conducting such an experiment.

5.4 Four-slot experiment to investigate the usefulness of recent Dialogue Acts

5.4.1 Introduction

This section describes a second Reinforcement Learning (RL) experiment with the 4 and 5-gram user simulations in which strategies are now learned for a four rather than a three-slot system, (see also Frampton and Lemon 2006). The main hypothesis is the same as in the three-slot experiment i.e. adding the DAs of the last user turn, and then last system turn to the slot-status features already in the state will produce significant incremental improvements in the performance of the learned strategy when tested with the n-gram simulations. The basic methodology is also the same: strategies are trained with the 4-gram simulation and tested with the 5-gram and vice versa, the training reward function is the same (see Section 5.3.4), the system action set is as described in Section 5.3.3 except that *i* should now obviously be substituted with 4 rather than 3, and strategies are learned with the same three different state representations i.e. Slot-Status, DA1 and DA2 (see Section 5.3.2). The reasons then for conducting this equivalent four-slot experiment are the following:

1. It gives the opportunity to gather more evidence to show that representing recent DAs in the state can produce improved learned strategies, and if so why.

2. We can investigate whether the state-action space remains tractable when the number of slots is increased to a more commercially-realistic 4.

3. We can compare the performance of learned 4-slot strategies with the COMMUNICATOR systems: There are a number of dialogues in the COMMUNICATOR

data where the user is only interested in a single-leg flight (i.e. they do not want a return flight, hotel or car), and hence there are just four slots to fill (departure city, destination city, departure date, and departure time). We can therefore compare the performance of the COMMUNICATOR systems in these dialogues with learned 4-slot strategies. However, no strong conclusions can be drawn based on such a comparison because while the learned strategies are tested in simulation, the COMMUNICATOR dialogues were conducted with real users.

4. We can compare the performance of learned 4-slot strategies with the Hybrid strategy of Henderson et al. (2008) (see Section 3.10.4): A meaningful comparison is possible here because the Hybrid strategy is a four-slot strategy and it was tested with the linear function approximation simulation which has been shown to simulate very similar dialogues to the n-gram user simulations (Georgila et al., 2006). Indeed, although they do not report the numbers, Henderson et al. (2008) state that the Hybrid Strategy was tested with the n-gram simulations and the results were very similar.

We now move on to describe the results.

5.4.2 Results

As for the three-slot experiment, we first present graphs which track the improvement of the learned strategies during training. Figure 5.8 tracks the improvement of the three learned strategies for 50000 training dialogues with the 4-gram user simulation, and Figure 5.9 for 50000 training dialogues with the 5-gram. These graphs show the average score per dialogue over intervals of 1000 training dialogues according to the training reward function described in section 5.3.4. After 50000 training dialogues with the 4-gram, 636 state-action pairs had been visited for the Slot-Status strategy, 7056 for DA1, and 18180 for DA2, and after 50000 training dialogues with the 5-gram, 636 for the Slot-Status strategy, 6701 for DA1, and 17731 for DA2.

Table 5.14 displays the test results for the strategies learned after 50000 training dialogues (the Slot-Status, DA1 and DA2 strategies). As for the three-slot experiment, the ‘a’ strategies are those trained with the 4-gram user simulation and tested with the 5-gram, and the ‘b’ strategies are those trained with the 5-gram user simulation and tested with the 4-gram. The table also shows the average of the ‘a’ and ‘b’ test scores. As in

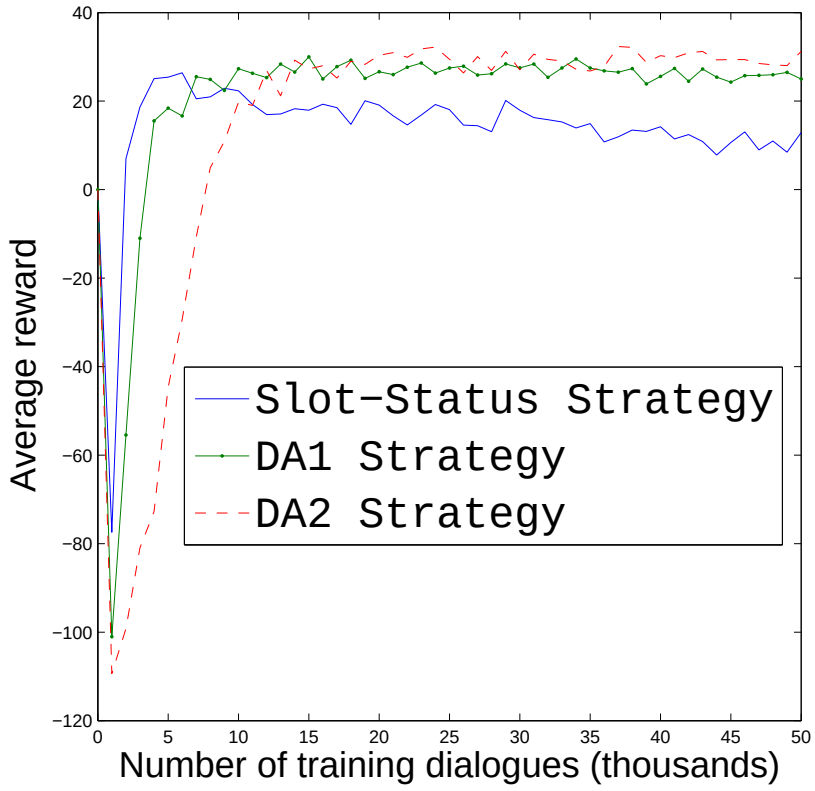

Figure 5.8: Training the four-slot dialogue strategies with the 4-gram simulation

Table 5.1, the third column shows the average scores according to the training reward function obtained per dialogue by each strategy over 1000 test dialogues[2]. Recall from Section 5.4.1 that a big part of the reason for conducting this four-slot experiment was so that we would be able to compare the performance of our learned strategies with the Hybrid Strategy of Henderson et al. (2008), and the original hand-crafted COMMUNICATOR systems. This is done using the HLG05 reward function, for which the composite parts are +25 per filled slot, +25 per confirmed slot and -1 per system turn (see Section 5.2.3). Hence there are additional rows here for the Hybrid Strategy of Henderson et al. 2008 and the COMMUNICATOR systems, and there are columns for the HLG05 reward function and its composite parts. Again, the columns for the

[2]These scores cannot be derived for the Hybrid Strategy and COMMUNICATOR systems from the HLG05 scores because we need to know how often all of the slots were confirmed. If we checked this in the data and calculated the scores, then they would have large negative values. This is because these strategies often fail to confirm slots and have a high average number of system turns.

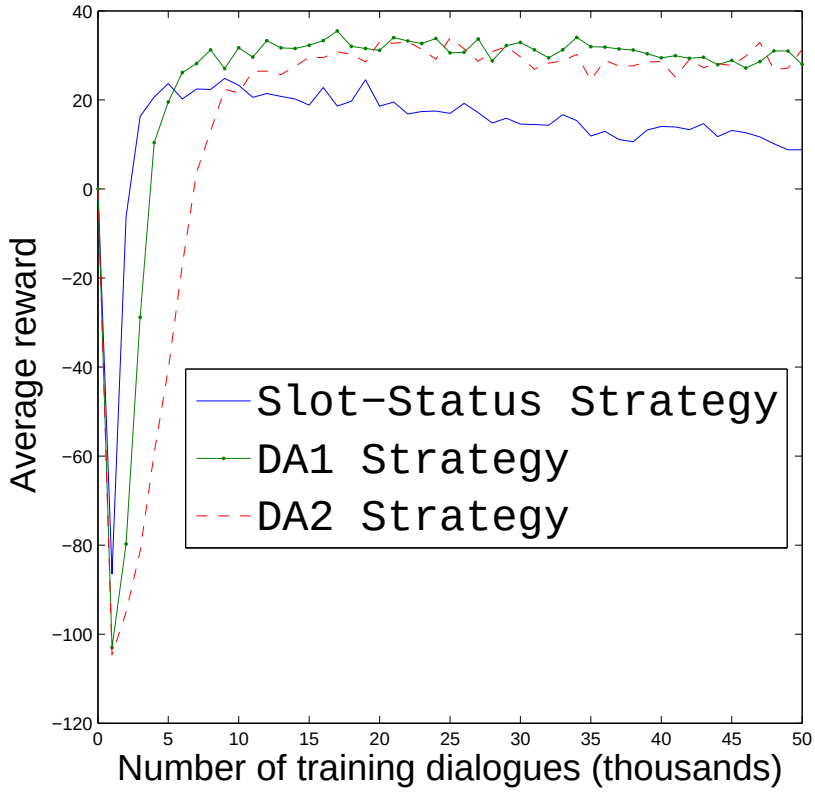

Figure 5.9: Training the four-slot dialogue strategies with the 5-gram simulation

HLG05 reward function and its composite parts display average scores per dialogue.

Let us first concentrate only on our RL strategies and the average scores according to our training reward function i.e. column 3. As in the three-slot experiment, the 1000 test dialogues for each strategy were divided into 10 sets of 100, and one-tailed independent samples t-tests showed that for the 'a', 'b' and average cases, the DA1 strategy performs significantly better than the Slot-Status strategy, and the DA2 strategy also performs significantly better than DA1. Hence the basic result is the same as in the three-slot experiment i.e. adding the Dialogue Acts (DAs) of the last user and then last system turns into the state produces incremental significant improvements in the learned strategy. The significance level here is $p < 0.005$, except in the case of DA1 Strategy(b)'s improvement over the Slot-Status strategy, where it is $p < 0.025$. The DA1 strategy improves over the Slot-Status strategy by 4.9% in terms of average dialogue reward, and the DA2 strategy by 7.8%.

	Features	**Av. Score**	**HLG05**	**Filled Slots**	**Conf. Slots**	**Length**
$4 \rightarrow 5$ **gram = (a)**						
Slot-Status (a)	Slot-status	51.60	190.32	100	100	-9.68
DA1 Strategy(a)	+ Last User DA(s)	53.35**	190.67	100	100	-9.33
DA2 Strategy(a)	+ Last System DA	54.9**	190.98	100	100	-9.02
$5 \rightarrow 4$ **gram = (b)**						
Slot-Status (b)	Slot-status	51.4	190.28	100	100	-9.72
DA1 Strategy(b)	+ Last User DA(s)	54.15*	190.83	100	100	-9.17
DA2 Strategy(b)	+ Last System DA	**56.25****	**191.25**	100	100	-8.75
Slot-Status (av)	Slot-status	51.5	190.3	100	100	-9.7
DA1 Strategy(av)	+ Last User DA(s)	53.75**	190.75	100	100	-9.25
DA2 Strategy(av)	+ Last System DA	**55.80****	**191.16**	100	100	-8.84
COMM Systems		***	103.6	85.0	63.0	-44.4
Hybrid RL/SL	Info States	***	140.3	88.0	70.0	-17.7

Table 5.14: Testing the learned strategies after 50000 training dialogues, average reward achieved per dialogue over 1000 test dialogues. a = strategy trained using 4-gram and tested with 5-gram; b = strategy trained with 5-gram and tested with 4-gram; av = average; * significance level $p < 0.025$; ** significance level $p < 0.005$. *** see below for why these are not calculated.

Let us now consider how the performance of our learned strategies compares to the Hybrid Strategy of Henderson et al. (2008), and the original hand-crafted COMMUNICATOR systems. Looking at column 4, we see that according to the HLG05 reward function, the DA2 strategy improves over the Hybrid Strategy by 36.25%, and over the average COMMUNICATOR system by 84.52%. Recall that the comparison with the COMMUNICATOR systems should be taken with-a-pinch-of-salt because these systems were evaluated with real users. However, the comparison with the Hybrid Strategy is much fairer because the Hybrid strategy was evaluated with the linear function approximation simulation, (see Section 3.10.4), which has been shown to simulate very similar dialogues to the n-gram simulations (Georgila et al., 2006). We had access to logs for the Hybrid Strategy's real user tests, but not to logs for its simulated user tests. Hence we will provide analysis to explain why the Hybrid Strategy performs

worse in Chapter 6 where we describe our own real user tests

As we did for the three-slot experiment, we now move on to provide analysis in order to explain the relative performance of our learned strategies.

5.4.3 Analysis

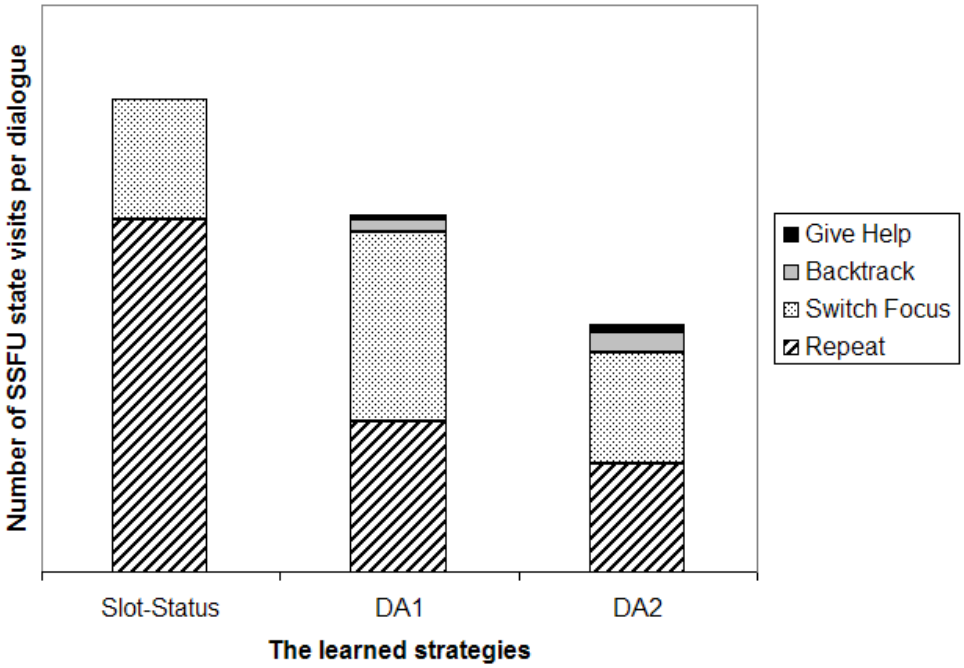

Figure 5.10: The relative number of total visits to SSFU states per dialogue, and the proportion of times which each of the four possible repair strategies is applied.

Analysis of our learned 4-slot strategies produced the same basic findings as the analysis of our learned 3-slot strategies (see Section 5.3.7). The important states again seemed to be Slot-Status Features Unchanged (SSFU) states, which in the vast majority of cases are caused by non-understanding errors: user utterances that are recognised as out-of-domain or for which there is no Automatic Speech Recognition (ASR) hypothesis i.e. an ASR rejection. In these SSFU states, the recent Dialogue Acts (DAs) enable the reinforcement learner to learn more effective repair strategies i.e. repair strategies which are better at getting the dialogue "back-on-track". As for the 3-slot strategies, the number of first visits to an SSFU state per dialogue is approximately the same for each of the learned strategies. However, the total number of visits to an SSFU state differs - the DA2 Strategy visits the least, the DA1 Strategy the second least, and the Slot-Status Strategy the most. Of the four possible kinds of repair strategy i.e.

Slot-Status Strategy		DA2 Strategy	
System:	ask1	System:	ask1
User:	out-of-domain	User:	out-of-domain
System:	ask1	System:	ask3
User:	out-of-domain	User:	out-of-domain
System:	ask1	System:	give-help
User:	asr_rejection	User:	asr_rejection
System:	ask1	System:	ask3
User:	out-of-domain	User:	slot3
System:	ask1	System:	impConfSlot3_askSlot4
User:	asr_rejection	User:	slot4
System:	ask1	System:	impConfSlot4_askSlot1
User:	slot1	User:	slot1
System:	impConfSlot1_askSlot2	System:	impConfSlot1_askSlot2
User:	slot2	User:	slot2
System:	impConfSlot2_askSlot3	System:	expConfSlot2
User:	slot3	User:	yes
System:	impConfSlot3_askSlot4	System:	dbQuery
User:	slot4		
System:	expConfSlot4		
User:	yes		
System:	dbQuery		

Table 5.15: Example test dialogues for the Slot-Status and DA2 strategies.

repeating, giving help, switching focus or backtracking, (see Section 5.3.7 for definitions), DA2 uses a greater mix than DA1, but DA1 a greater mix than the Slot-Status Strategy. These last two points are well illustrated by figure 5.10.

As in the analysis of the three-slot experiment, we now provide example dialogues in order to show the repair strategies in action. Table 5.15 contains transcripts of two test dialogues - one conducted using the Slot-Status strategy, and the other using the DA2 strategy. The reader can see that in both cases, the system has difficulty at the start of the dialogue - it cannot obtain a value for slot 1. While the Slot-Status strategy repeatedly asks for slot 1 until it is eventually able to obtain a value, the DA2 strategy

switches focus to slot 3 and makes use of the "give help" function. It returns to slot 1 later in the dialogue. Employing more sophisticated repair strategies such as this, the DA2 Strategy fills and confirms all of the slots in fewer turns on average.

DA2 state representation	**DA2 Strategy**	**No. visits**	**Slot-Status**
[conf,emp,conf,conf,ec2,user(no(slot2))]	ask2	1366	ask2
[emp,emp,conf,conf,ic1ask2,user(no(slot1))]	ask1	167	ask1
[conf,emp,conf,conf,ec2,user(no)]	ask2	148	ask2
[emp,emp,fill,emp,ec3,user(no,slot3)]	ic3ask4	91	ic3ask4
[emp,emp,fill,conf,ec3,user(no,slot3)]	ic3ask2	62	ic3ask2
[fill,emp,conf,conf,ec1,user(no(slot2))]	ic1ask2	502	ic1ask2
[emp,conf,conf,conf,ec1,user(no(slot1))]	ask1	4489	ask1
[emp,conf,fill,emp,ec3,user(no,slot3)]	ic3ask4	431	ic3ask4
[emp,emp,conf,conf,ec1,user(no(slot1))]	ask1	1230	ask1
[emp,fill,conf,conf,ec2,user(no(slot1))]	ic2ask1	32	ic2ask1
[conf,fill,fill,conf,ec3,user(no,slot3)]	ic3ask2	27	ic3ask2
[fill,emp,conf,conf,ask1,user(no)]	ic1ask2	53	ic1ask2
[fill,conf,conf,conf,ec1,user(no,slot1)]	ec1	29	ec1
[emp,emp,emp,emp,ec1,user(no(slot1))]	ask3	64	ask1
[emp,conf,conf,conf,give_help,user(no(slot1))]	ask1	26	ask1
[emp,conf,conf,emp,ec1,user(no(slot1))]	ask4	26	ask4
[emp,conf,conf,conf,ec1,user(no)]	ask1	42	ask1
[emp,emp,emp,emp,ask3,user(no(slot2))]	ask3	31	ask1
[emp,fill,conf,conf,ec1,user(no(slot1))]	ic2ask1	113	ic2ask1
[conf,conf,fill,conf,ec3,user(no,slot3)]	ec3	45	ec3

Table 5.16: Example states for misunderstandings where the user indicates the misunderstanding directly after the system tries to confirm the incorrect value.

Also as in the three-slot experiment, we could not find compelling evidence to suggest that the recent Dialogue Acts (DAs) were making significant improvements to the learned strategy in other ways as well. There were apparently no significant effects on the learned strategy's behaviour in dealing with misunderstandings, nor in portions of dialogue in which there is smooth progress towards the goal of filling and confirming all of the slots. Table 5.16 shows all of the DA2 states where the user has just indicated a misunderstanding error, and which were visited 25 times or more during the 50000 training dialogues, (the third column shows the number of visits). For each

	Slot-Status Strategy	**DA1 Strategy**	**DA2 Strategy**
System:	askSlot3	askSlot3	askSlot3
User:	slot3	slot3	slot3
System:	impConfSlot3AskSlot4	impConfSlot3AskSlot4	impConfSlot3AskSlot4
User:	slot4	slot4	slot4
System:	impConfSlot4AskSlot1	impConfSlot4AskSlot1	impConfSlot4AskSlot1
User:	slot1	slot1	slot1
System:	impConfSlot1AskSlot2	impConfSlot1AskSlot2	impConfSlot1AskSlot2
User:	slot2	slot2	slot2
System:	expConfSlot2	expConfSlot2	expConfSlot2
User:	yes(slot2)	yes(slot2)	yes(slot2)
System:	dbQuery	dbQuery	dbQuery

Table 5.17: System-initiative dialogues in which each user turn fills/confirms a slot value.

	Slot-Status Strategy	**DA1 Strategy**	**DA2 Strategy**
System:	askSlot3	askSlot3	askSlot3
User:	slot3,slot4	slot3,slot4	slot3,slot4
System:	impConf3AskSlot2	impConfSlot3AskSlot2	impConfSlot3AskSlot4
User:	slot2,slot4	slot2,slot4	slot2,slot4
System:	impConf4AskSlot1	impConfSlot2AskSlot1	impConfSlot4AskSlot1
User:	slot1	slot1	slot1
System:	impConfSlot1AskSlot2	impConfSlot4AskSlot1	impConfSlot1AskSlot2
User:	slot2	slot1	slot2
System:	expConfSlot2	expConfSlot1	expConfSlot2
User:	yes(slot2)	yes(slot1)	yes(slot2)
System:	dbQuery	dbQuery	dbQuery

Table 5.18: Mixed-initiative dialogues in which each user turn fills/confirms slots values.

state, the second and fourth columns show the learned actions for the DA2 and Slot-Status strategies respectively. As in the three slot experiment, almost all are the same, and the general trend seems to be for the reinforcement learner to learn to maintain focus on the problem slot rather than move on as if the user had not indicated the misunderstanding.

For each of the three learned strategies, Tables 5.17 and 5.18 then shows dialogues in which there is smooth progress throughout i.e. no user turn fails to fill or confirm a slot value. Table 5.17 shows system-initiative dialogues, and Table 5.18, mixed-initiative dialogues. There are no behavioural differences between the strategies in Table 5.17, and in observing test dialogues, the couple of differences in Table 5.18 did not seem to be important. Table 5.18 shows example mixed-initiative dialogues for each of the three strategies where again no user turn fails to fill or confirm a slot value. Again, the behaviour of the three strategies is identical. Note, in Chapter 7, we will investigate the hypothesis that the recent DAs are only making significant improvements to the learned strategies with respect to better strategies in SSFU states.

We now move on to draw conclusions based on the findings of the three and four slot experiments which have been described in this chapter.

5.5 Conclusion: Adding recent Dialogue Acts to the state improves the learned strategy

The results of the three and four-slot experiments described in this chapter showed the main hypothesis to be correct: with the slot-status features already in the state, adding the Dialogue Acts (DAs) of the last user and then system turns produces significant incremental improvements in the performance of the learned strategy when tested with the n-gram simulations. Subsection 5.2.2.2 explained why adding more previous DAs could not have improved the learned strategy further - when training with an n-gram simulation, if the slot-status features and the DAs of the last $n-2$ turns are already in the state, then DAs from turns further back than $n-2$ cannot improve the strategy. In testing, we found that all of the learned strategies could always fill and confirm all of the slots, but adding DAs to the state produced learned strategies which were able to achieve this in fewer turns on average. Our analysis strongly suggests that this is solely due to differences in behaviour in particular states which involve communi-

cation problems. These are states in which the slot-status features are unchanged by the last user turn, (usually due to a non-understanding error - an out-of-domain user response or an ASR rejection). In these states then, the recent DAs seem to enable the reinforcement learner to learn repair strategies which are more effective at getting the dialogue "back-on-track" i.e. inducing the user simulation to provide recognisable slot values/confirmations. They produce a greater variety of repair strategies, and the four different types are repeating the previous action, switching focus to ask/confirm a different unfilled/unconfirmed slot, using the give-help function and backtracking (re-asking a filled slot). There seems to be an important distinction between the two different kinds of non-understanding error which cause SSFU states, because the best learned strategy most often used the repeat repair strategy following ASR rejections, but the switch focus repair strategy following user utterances that are recognised as out-of-domain. This perhaps motivates future work that investigates the usefulness of making more fine-grained distinctions between different kinds of non-understandings, a point we return to in Chapter 8. Qualitative analysis of the learned strategies suggested that the DAs were not producing improvements with respect to dealing with misunderstanding errors, nor with respect to portions of dialogue in which there is smooth progress towards the goal of filling and confirming all of the slots. However, we did observe a general trend for the reinforcement learner to learn to maintain focus on the problem slot following a misunderstanding error.

In addition to testing the main hypothesis, we compared the performance of our learned four-slot strategies with the four-slot Hybrid strategy of Henderson et al. (2008). This was a meaningful comparison because the Hybrid strategy was also tested with the linear function approximation simulation which has been shown to simulate very simular dialogues to the n-gram simulations (Georgila et al., 2006). This comparison showed our strategies to perform better in terms of dialogue length, and the number of filled / confirmed slots. We will provide analysis to explain this in the next chapter. We also compared the performance of our learned four-slot strategies to the hand-crafted COMMUNICATOR systems, and again found our strategies to perform better in terms of dialogue length and the number of filled/confirmed slots. However, this comparison should be taken with a pinch-of-salt because the scores for the hand-crafted COMMUNICATOR systems were obtained from tests conducted with real users, not with the n-gram simulations.

There is no reason not to believe that for use with the n-gram simulations, our best

learned strategy is very near-optimal. However we should note that the n-gram simulations do not simulate real users perfectly and so our learned strategies are very likely to work less well with real users. With a more realistic simulation, it may become useful to include additional features in the state e.g. more DAs, a feature for dialogue length. Given a simulation which simulated unobtainable slot values, a dialogue length feature could enable the reinforcement learner to learn a strategy which made partially complete database queries when dialogue length was high and a particular slot value was proving unobtainable. This would obviously require a reward function that gave positive reward to partially complete database queries, but as the final Bayesian Network user simulation experiment demonstrated, (see Section 4.5), using such a "partial" reward function will produce slower learning and potentially inferior strategies than an "all-or-nothing" reward function. Perhaps a good compromise then would be to use a reward function that only gives positive reward when the Dialogue Manager (DM) has done as well as it can. In some dialogues this will mean filling and confirming all of the slots, but in others it may mean only filling and confirming less. Hence at the end of each dialogue, the user simulation will have to communicate to the DM whether any particular slot value(s) were unobtainable so that the DM can assess whether it has done as well as it can, and so compute the appropriate reward.

The findings of the three and four slot experiments described in this chapter are encouraging, but they leave questions which we address in experiments described in the next two chapters. These questions include whether the DA strategies also work better with real users, whether the DAs really only improve the learned strategy in certain contexts as seems to be the case in the experiments of this chapter, and whether the DAs are useful for choosing which repair strategy to apply or only for recognising that one is required. In the next chapter, we address the question of whether the DA strategies also work better with real users. Here, we describe an experiment in which we test a state-of-the-art hand-crafted strategy, and the three-slot Slot-Status and DA2 strategies on real users with a full Spoken Dialogue System (SDS). Since the Hybrid strategy of Henderson et al. (2008) was tested on real users using the same methodology, we are able to make another direct performance comparison, and as stated above, this time we provide analysis to explain the differences in results.

Chapter 6

Testing the learned strategies on real users

6.1 Introduction

This chapter describes experiments which test the hypothesis that when implemented in a full Spoken Dialogue System (SDS) and tested on real users, a strategy learned with recent Dialogue Acts (DAs) will outperform a state-of-the-art hand-crafted strategy or strategy learned with only slot-status information. We test this hypothesis because although we have already demonstrated in Chapter 5 that adding DAs improves the learned strategy's performance in testing in simulation, this still leaves the question of whether the user simulation is sufficiently realistic, and whether the new strategy is also better for real users. To test this hypothesis then, we use a state-of-the-art hand-crafted strategy, and two of the strategies which were learned in the three-slot experiment of Chapter 5 - the Slot-Status strategy i.e. the strategy learned with only the slot-status features represented in the state, and the DA2 strategy i.e. the strategy learned with the slot-status features and the DAs of the last two turns. These strategies are tested on 11 real users using the "TownInfo" SDS (Lemon et al., 2006b), which operates in the tourist information domain. Hence setting up these experiments involved porting the learned strategies from the COMMUNICATOR (travel-booking) domain into the tourist-domain. It also involved creating an "Action Retrieval" agent in Java which was responsible firstly, for maintaining an internal representation of the dialogue state, and secondly, for using this representation to retrieve learned actions from a database. The DIPPER Dialogue Manager's (DM's) (Bos et al., 2003) update

rules were modified so that the Action Retrieval agent was called to perform its functions at the appropriate times. Both subjective and evaluation measures are collected from the logs created by the Dialogue Manager (DM), and from user questionnaires. We provide analysis to explain the relative performance of DA2 versus the Slot-Status and hand-crafted strategies, and also DA2 versus the Hybrid Reinforcement Learning (RL) / Supervised Learning (SL) strategy of Henderson et al. (2008) - this is possible because the Hybrid Strategy has previously been tested on real users using the same experimental methodology (see Lemon et al. 2006a).

The remainder of this chapter will proceed as follows. Section 6.2 will describe the related work of Lemon et al. (2006a) and Singh et al. (2002) which also both concern testing learned strategies on real users with full-working SDSs. Next Section 6.3 will detail the experimental methodology including an overview of the TownInfo system (Section 6.3.1), the Action Retrieval agent (Section 6.3.2), the process of porting the learned strategies from the COMMUNICATOR to TownInfo domain (Section 6.3.3), and how the dialogues were evaluated (Section 6.3.4). Section 6.4 gives the results, Section 6.5 provides analysis and explanation for the relative performance of the various strategies, and Section 6.6 summarises and draws conclusions.

6.2 Related work

We begin in this section by focusing on the closely-related work of Lemon et al. (2006a). Lemon et al. describe an experiment in which the Hybrid Strategy of Henderson et al. (2008) is tested on real users using the same methodology as here, and hence the results are directly comparable. Like our learned strategies, although the Hybrid Strategy is learned in the COMMUNICATOR domain (Walker et al., 2001a), in Lemon et al. it is ported to the tourist information domain and tested on real users with the TownInfo system. Lemon et al. also have a hand-crafted strategy for performance comparison, but their hand-crafted strategy is a little different to ours. Both hand-crafted strategies attempt to fill and confirm the slots in the order $1-3$ and allow the user to take the initiative, but whereas our hand-crafted strategy won't query the database until it has confirmed every slot, the hand-crafted strategy of Lemon et al. does not bother to confirm slots that are filled with a acoustic confidence value above a certain threshold. Lemon et al. report that the users of the (ported) learned strategy had an average gain in Perceived Task Completion of 14.2% (from 67.6% to 81.8%

at $p < 0.03$), that the hand-crafted strategy dialogues had on average 3.3 more system turns ($p < 0.01$), and that the user satisfaction results were comparable, even though the strategy was learned for a different domain. Combining the PTC and dialogue length measures in a dialogue reward score, the authors found a 14.4% increase for the learnt policy (a 23.8% relative increase $p < 0.03$). They state that their results are important because they show:

1. Results for real users are consistent with results for automatic evaluation (Henderson et al., 2005) of learned strategies using simulated users (Georgila et al., 2005a, 2006).

2. A strategy learned using linear function approximation over a very large strategy space (Henderson et al., 2005) is effective for real users.

3. A strategy learned using data for one domain can be used successfully in other domains.

The results which we obtain in the real user experiment of this chapter also show 1 and 3. Since our experimental results are directly comparable with those of Lemon et al. (2006a), we provide analysis in order to explain the relative performance of our learned strategies and the Hybrid Strategy of Henderson et al. (2008).

A less recent experiment in which a learned strategy is tested on real users is described in Singh et al. (2002), (see also Section 3.3 for more details). Recall that 21 subjects perform 6 tasks with the NJFUN system, and that Singh et al. only learned which action to take in certain "choice states" rather than all states i.e. they learned partial rather than full strategies. In some of the choices states, the reinforcement learner learned between taking the initiative in the dialogue or giving the initiative to the user, and in others, it learned whether to use confirmation. Unlike in Lemon et al. (2006a) and the work described in this chapter, the baseline strategy for comparison here was not a fully hand-crafted strategy - it differed from the learned strategy in that it chose randomly between the possible actions in the "choice states". Hence this baseline could be referred to as an "Exploratory for Initiative and Confirmation" strategy. In these conditions, task completion for the learned strategy rose from 52% to 64% with $p < 0.059$. Data was then divided into the first 2 tasks ("novice users") and tasks $3-6$ ("expert users") to control for learning effects. The learned strategy led to significant improvement in task completion for experts but a non-significant degradation for novices.

The reader should also be aware that like Pietquin and Renals (2002) (see Section 3.5, we have learned strategies with only slot-status variables in the state, (we call these Slot-Status Strategies). Pietquin and Renals did not test this strategy on real users, nor compare its performance to a hand-crafted baseline with a user simulation. However, in the previous chapter we tested Slot-Status strategies in simulation, and in this chapter, we test one on real users and compare its performance with other strategies i.e. the DA2 strategy, a state-of-the-art hand-crafted strategy, and the Hybrid Strategy of Henderson et al. (2008).

We now move on to describe our experimental methodology, and begin with an overview of the "TownInfo" SDS (Lemon et al., 2006b) which we use to test strategies on real users.

6.3 Methodology

6.3.1 Overview of the TownInfo system

In order to test them on real users, the learned and hand-crafted strategies are implemented in the TALK project's "TownInfo" tourist information system (Lemon et al., 2006b). TownInfo is designed to help a user to find a particular hotel, bar or restaurant. Any TownInfo dialogue has 3 identifiable phases which can be referred to as:

1. Goal-selection phase,
2. Information slot filling/confirming phase,
3. Presentation of options phase.

In the Goal-selection phase, TownInfo tries to establish whether the user is looking for a hotel, bar or restaurant. Having achieved this, it moves onto the Information slot filling/confirming phase where it tries to fill and confirm information slots, these being:

1. Location,
2. Type,
3. Price.

When the system has finished filling and confirming information slots, it queries its database in order to find suitable hotels/bars/restaurants, and at this point it enters the Presentation of options phase, where it presents the possible options to the user.

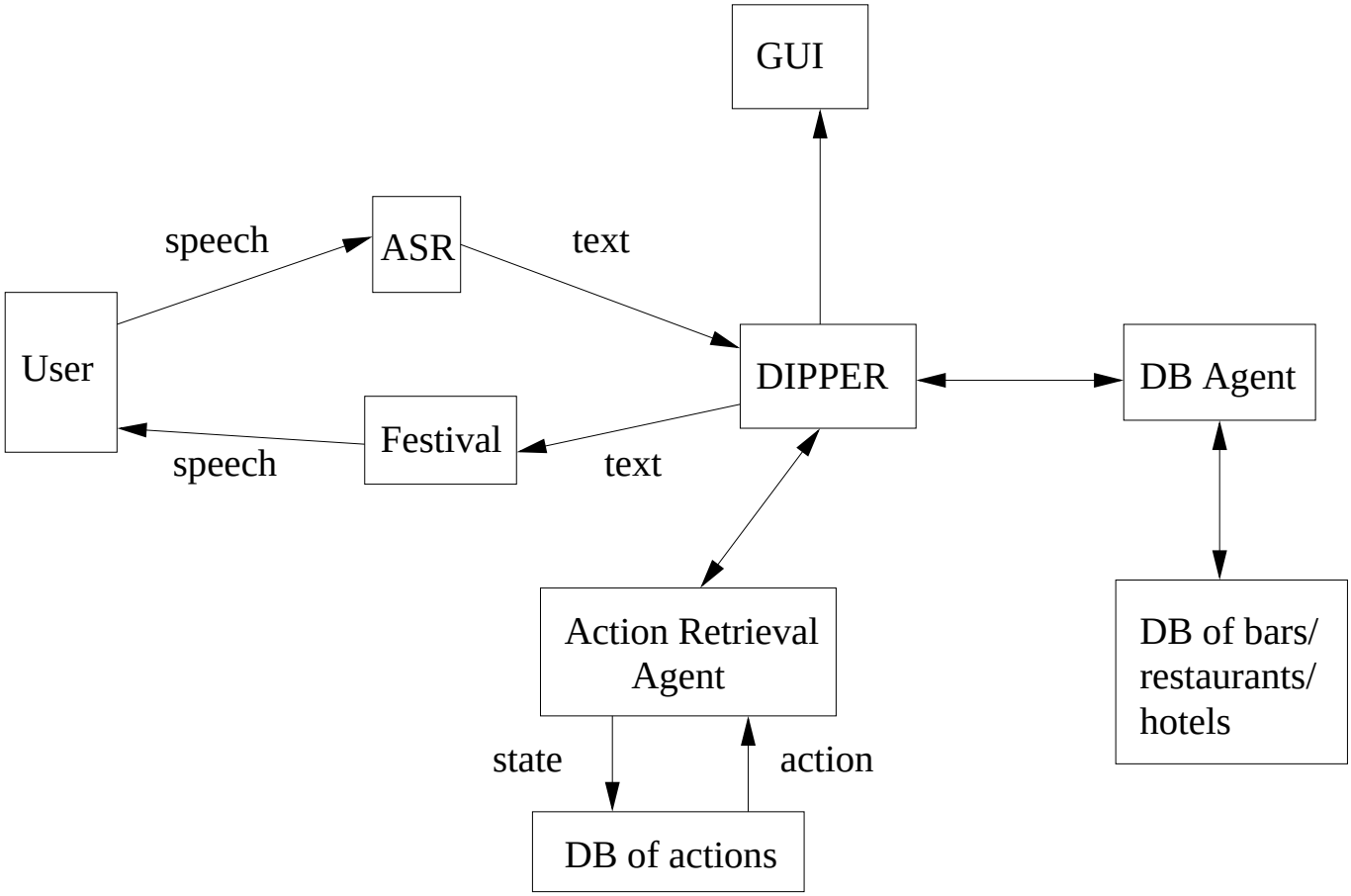

Figure 6.1: TownInfo System. The database of actions contains the state-action pairs for whichever strategy is currently being used.

The TownInfo system is composed of a number of different components which communicate with one another via the Open Agent Architecture (OAA) (Cheyer and Martin, 2001), and this is represented in Figure 6.1. On the input side, the choice of speech recogniser is the open speech recogniser, HTK, (Young, 2004). The user is able to take the initiative in the conversation i.e. supply one or more slot values not asked for by the system, and to give start over, quit, help and "show" commands. An example of a "show" command would be "Show me the hotels". Assuming the command is recognised correctly, this would prompt the system to make an immediate database query and present the hotels which fit the criteria provided by the user up to this point in the dialogue. The system's presentation of suitable hotels/bars/restaurants is multi-modal - as the system generates speech to describe a possible option, it also

highlights it on the map displayed by the Graphical User Interface (GUI) agent (see Figure 6.2). The Natural Language Generation (NLG) for the presentation of database results is template-based [1]. The logging agent produces log files of the conversation in the TALK Information State Update (ISU) format (Georgila et al., 2005b). One of the ways in which the dialogues are evaluated is via dialogue length, and these log files can be used to count the number of turns in each dialogue.

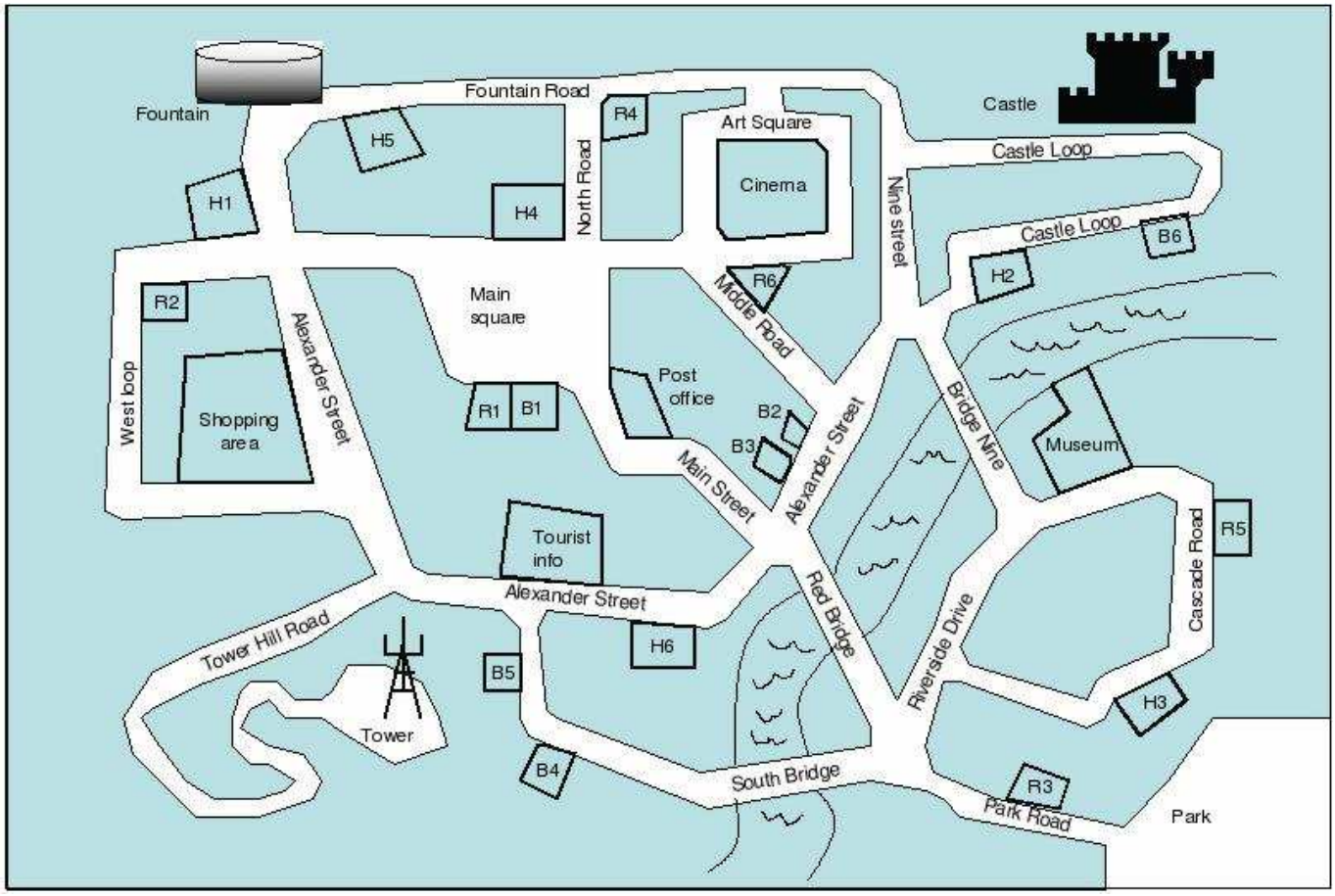

Figure 6.2: TownInfo System GUI

The Action Retrieval agent, which I built in Java, is used to implement the learned and hand-crafted strategies. Given one of the strategies, and a representation of the current dialogue context, the Action Retrieval agent returns the corresponding action to DIPPER. The Action Retrieval agent is only called in the Information slot filling/confirming phase of a TownInfo dialogue because our learned and hand-crafted strategies apply only to this phase. In the first and third phases i.e. the Goal-selection

[1]Template-based NLG systems are usually defined as NLG systems that map their non-linguistic input directly, (i.e. without intermediate representations), to the linguistic surface structure, (see page 83-84 of Reiter and Dale 1997).

and Presentation of options phases, the system uses hand-crafted dialogue plans. For example, in the Goal-selection phase, the system will repeatedly ask the open-initiative initial question "How can I help you?", and then try to confirm the user's response until it has successfully confirmed whether the user is interested in a bar, restaurant or hotel. The next section will now provide a detailed description of the Action Retrieval agent.

6.3.2 Action retrieval agent

I created an Action Retrieval OAA agent in Java which has four functions - *updateTheState*/2, *resetSlotVariables*/1, *getLearnedActionForTownInfo1*/4 and *getLearnedActionForTownInfo2*/4. I modified the TownInfo DIPPER (Bos et al., 2003) update rules so that they called these functions at the appropriate times. The first function, *updateTheState*/2, took the last system-user exchange, represented in terms of speech act-task pairs from DIPPER's information state, and then updated the agent's internal representation of the current dialogue state accordingly. This internal state representation would necessarily be of the same form as was used to learn the particular strategy being tested e.g. in the case of the DA2 Strategy, the state would be represented in terms of the following 5 variables:

1. status of slot 1: empty/filled/confirmed,
2. status of slot 2: empty/filled/confirmed,
3. status of slot 3: empty/filled/confirmed,
4. Dialogue Act (DA) of the last system turn,
5. DA(s) of the last user turn.

The second function, *resetSlotVariables*/1 is called at the end of each dialogue in order to reset each of the slot status variables to "empty".

The third function, *getLearnedActionForTownInfo1*/4 is responsible for using this state representation to query the database table containing the strategy and so retrieve the learned action. Before being returned, this action then has to be mapped into a form that can be used by TownInfo, which means that each action has to be represented in terms of the three DATE (Walker and Passonneau, 2001) dimensions. Recall from Section 3.10.1 that these are:

1. conversational domain,
2. speech Act,
3. task.

The TownInfo system next translates the DATE triplet into text and then speech. In the case that there is no learned action for the DA2 Strategy in a particular state i.e. that state had not been visited during training, then the Action Retrieval agent would "back-off" in the order:

1. DA1 Strategy from Chapter 5,
2. Slot-Status Strategy from Chapter 5.

Backing-off to another strategy would obviously mean dropping the appropriate turn variable from the state representation e.g. in the case of backing-off from the DA2 Strategy to the DA1 Strategy, the last system turn but one variable would be dropped.

The fourth function, *getLearnedActionForTownInfo2*/4, is called when the learned action is to implicitly confirm one slot and ask another, and so cannot be expressed as a single DATE triple. For example, the learned action "impConfSlot1-askSlot2" must be mapped to two DATE triples - <about-communication,implicit-confirm,dest-city>, <about-task,request-info,depart-arrive-date> . In this case, *getLearnedAction-ForTownInfo1*/4 would be called first and would use the internal state representation to retrieve the learned action "impConfSlot1-askSlot2". It would then map this action to the two date triples, but would only return the first to DIPPER, whilst storing the second internally. Once this first triple has been translated into text and then speech, DIPPER next calls *getLearnedActionForTownInfo2*/4, which returns the second triple. This second triple is then also translated into text and then speech, and the system turn is released.

Why was a function not written which combined the functionalities of *updateTheS-tate*/2 and *getLearnedActionForTownInfo1*/4 i.e. one which updated the internal state representation and returned the learned action as one or more DATE triplets? The explanation is that in the first phase of the dialogue (Goal-selection phase - see Section 6.3.1), we only want the function which updates the internal state representation. Although our strategies and hence the *getLearnedActionForTownInfo*/4 functions are

redundant during this initial phase, it is still necessary to keep track of the status of each of the slots. This is because having been asked "How may I help you?", the user may take the initiative and fill one or more slot values e.g. "I want an expensive Italian restaurant." rather than just "I want a restaurant." Hence, *updateTheState*/2 must be called during the first phase in order to update the status of each of the slots.

6.3.3 Porting strategies from the COMMUNICATOR to the TownInfo domain

Our learned strategies were learned in the COMMUNICATOR (travel-booking) domain - Chapter 5 describes how strategies were learned as the DIPPER Dialogue Manager (DM) conducted training dialogues with an n-gram user simulation, the probabilities for which were learned from COMMUNICATOR data (Walker et al., 2001a). We are now porting these strategies into a different domain - the tourist information domain of the TownInfo Spoken Dialogue System (SDS). This is straightforward to do because if we abstract away from what the information slots actually represent, the COMMUNICATOR and TownInfo systems have the same aim i.e. to fill up and confirm information slots, then access a database and present results to the user. We simply need to provide a mapping between the COMMUNICATOR and TownInfo information slots/sub-tasks. Table 6.1 shows the mappings that are used.

Sub-task	
COMMUNICATOR	**TownInfo**
dest_city	food_type
depart_date	food_price
depart_time	food_location
dest_city	hotel_location
depart_date	room_type
depart_time	hotel_price
dest_city	bar_type
depart_date	bar_price
depart_time	bar_location

Table 6.1: Mappings between COMMUNICATOR and TownInfo tasks.

These mappings allows us to:

1. map TownInfo actions to COMMUNICATOR actions so that they can be included in the Action Retrieval agent's internal state representation which it uses to retrieve the next action,
2. map the returned COMMUNICATOR actions to TownInfo actions so that they can be implemented by the TownInfo system.

In fact it was only necessary for me to do 1, because the work of Lemon et al. (2006a) had already enabled DIPPER to do 2.

If strategies learned in the COMMUNICATOR domain also work well in the TownInfo domain, then this suggests that we have learned effective generic slot-filling strategies. It is our hypothesis that this will be the case - that certain types of behaviour exhibited by the strategies such as switching focus away from problem slots, making use of the give-help function and backtracking (see Section 5.3.7.2 for definitions) will prove effective in other domains as well.

6.3.4 Evaluation methodology

As stated previously, we test the following three-slot strategies on real users by implementing them in the TownInfo Spoken Dialogue System (SDS):

1. a state-of-the-art hand-crafted strategy,
2. a Slot-Status strategy,
3. a DA2-strategy.

The two learned strategies were produced in the three-slot experiment of Chapter 5. The hand-crafted strategy is the state-of-the-art mixed-initiative hand-crafted strategy described in Section 6.2. The dialogue strategy is the only part of the TownInfo SDS which is varied. Everything else always remains the same, i.e. the information presentation routines, Automatic Speech Recogniser (ASR), Graphical User Interface (GUI), Text-To-Speech (TTS) synthesiser, and hotel/bar/restaurant database. There are 11 subjects, and following the methodology of Walker et al. (2000) and Lemon et al.

(2006a), each subject is given a questionnaire containing 15 tasks, (5 in each condition), to give a total of 165 test dialogues. In order to evaluate the system in the three different conditions i.e. with the three different strategies, we collect Perceived Task Completion (PTC), dialogue length and subjective evaluations. Section 6.3.1 stated that as the dialogue progresses, the logging agent produces log files containing each DIPPER information state, and these files can be used to find dialogue length in terms of the number of system turns. PTC and the subjective evaluations are obtained from the user via the questionnaire. Learning and temporal ordering effects are controlled for by rotating the order in which the strategies are applied for each user.

To see the questionnaire, see Appendix B. The presentation of the tasks on the questionnaire is such that the subjects cannot simply read them out to the system e.g.

```
Task 1: You are on a business trip on your own. You need to find a
hotel room in the middle of town. Price is no problem.
```

The user's PTC is collected like so:

```
Write the name of the result that the system presented to you (e.g.
FOG BAR) here:

Does this item match your search? Yes/No
```

For subjective evaluation of the system in each task, the user is asked to answer the following questions using a 5-point Likert scale (1 = strongly disagree and 5 = strongly agree):

```
1. In this conversation, it was easy to get the information that I
wanted.
2. The system worked the way I expected it to, in this conversation.
3. Based on my experience in this conversation, I would like to use
   this system regularly.
```

Below, for shorthand, I will refer to question 1 as "Task ease", question 2 as "System behaviour as expected", and question 3 as "Reuse the system".

A total reward is calculated for each dialogue, using the following reward function:

1. $+100$ for PTC,
2. -1 for each system turn.

This reward function is chosen because it is the same as that used in Lemon et al. (2006a) and so allows us to directly compare results. The data for reward, PTC, dialogue length and subjective measures for all 165 test dialogues is logged. The mean for each measure in each of the three conditions is calculated, and dependent samples t-tests are calculated for each of the following in order to test whether the DA2 strategy shows any significant improvement over the Slot-Status and the hand-crafted strategies:

1. reward for hand-crafted versus DA2,
2. reward for Slot-Status versus DA2,
3. Task ease for hand-crafted versus DA2,
4. Task ease for Slot-Status versus DA2,
5. System behaviour as expected for hand-crafted versus DA2,
6. System behaviour as expected for Slot-Status versus DA2,
7. Reuse the system for hand-crafted versus DA2,
8. Reuse the system for Slot-Status versus DA2.

Dependent samples t-tests are the appropriate kind of t-test to use because the same (rather than different) subjects use the system in the three different conditions.

We now move on to describe our results.

6.4 Results

Table 6.2 shows Perceived Task Completion (PTC), average number of system turns and reward per dialogue for each of the three strategies. The reward is calculated using the reward function described in Section 6.3.4. The PTC for the DA2 strategy (90.91%) is significantly better than the PTC for both the hand-crafted and Slot-Status strategies

Strategy	**PTC (Av. %)**	**System turns (Av.)**	**Reward (Av.)**
Hand-crafted	81.82%	8.46	73.36
Slot-Status	81.82%	8.98	72.84
DA2	90.91%**	7.95*	82.96**

Table 6.2: ** = significantly better than hand-crafted and Slot-Status, * = significantly better than Slot-Status

($p < 0.48$). Average number of system turns per dialogue for the DA2 strategy (7.95) is significantly fewer than for the Slot-Status strategy (8.98) ($p < 0.0085$) The average reward per dialogue for the DA2 Strategy is significantly better than for both the hand-crafted and Slot-Status strategies - 82.96 versus 73.36 ($p < 0.0425$) and 72.84 ($p < 0.034$) respectively.

Subjective	**Strategies**		
Measures	**Hand-crafted**	**Slot-Status**	**DA2**
Task Ease	3.33	3.13	3.45
Expected Behaviour	3.29	3.15	3.36
Re-use	3.11	3.04	3.22

Table 6.3: The user preference scores for the hand-crafted, Slot-Status and DA2 Strategies.

Table 6.3 shows the user preference scores for each of the three strategies (see Section 6.3.4 for a description of how these were collected). Although the DA2 Strategy is the best according to each of the measures, it is not significantly better.

Strategy	**PTC (Av. %)**	**User pref. (Av.)**	**System turns (Av.)**	**Reward (Av.)**
DA2	90.91	3.34	9.65	82.96
Hybrid	81.8	2.67	11.6	74.9

Table 6.4: Comparison between the Hybrid Strategy of Henderson et al. (2008) and the DA2-strategy

Table 6.4 compares the performance of the DA2 Strategy and the Hybrid Strategy of Henderson et al. (2008). No significance tests are possible because I only had the average scores for the Hybrid Strategy, rather than the scores for all of the trial runs. The PTC is greater for the DA2 Strategy than the Hybrid Strategy (90.91 versus 81.8), and the average number of system turns fewer (9.65 versus 11.6). Here all of the system turns are counted, rather than only those in the Information slot filling/confirming phase of the dialogue. Table 6.4 shows that the DA2 Strategy achieves greater PTC in fewer turns on average, and hence gains a greater average reward (82.96 versus 74.9).

We now move on to provide analysis in order to explain the relative performance of the various strategies in the real user tests.

6.5 Analysis

In this section we provide analysis in order to explain the relative performance of the various strategies. We begin in the next section by focusing on why the DA2 Strategy outperformed the Slot-Status and hand-crafted strategies.

6.5.1 The DA2 Strategy versus the Slot-Status and hand-crafted strategies

As in testing in simulation, in testing with real users, the DA2 Strategy outperforms the Slot-Status Strategy. It also outperforms the state-of-the-art hand-crafted strategy. It outperforms both in terms of Perceived Task Completion (PTC) and dialogue length, and hence also in terms of the reward function described in Section 6.3.4. Recall that in Sections 5.3.7 and 5.4.3, qualitative analysis was provided to show that the learned strategies apparently only differed significantly when the slot-status features are left unchanged by the previous user turn. In these situations, whereas the Slot-Status Strategy, (and hand-crafted strategy), repeat the last question/confirmation, the DA2 Strategy sometimes uses alternative repair strategies: switch focus to a different unfilled/unconfirmed slot, give help or backtrack (re-ask an already filled slot). Hence it seems that these repair strategies must also be the cause of DA2's better performance in the real user tests here.

It is one thing to know that the DA2 repair strategies are more effective than always

using repetition, but it would be nice to know why, and this is something which these real user experiments help to explain. In supervising the experiments, I witnessed users become frustrated when the Automatic Speech Recogniser (ASR) was failing to recognise a slot value and the Slot-Status or hand-crafted strategy dictated that the system persist in asking the same question/attempting the same confirmation. As a result, the user would often begin to speak overly-clearly i.e. *hyper-articulate* their words and/or adopt an angry tone, and this made it less likely that the ASR would correctly recognise what they were saying. Users would get stuck on particular slots in this way, producing longer dialogues. Occasionally they would completely lose patience and quit, or alternatively, as a strategy to induce the system to move on, they might deliberately supply a slot value which was different to the one that they wanted, but which they thought might be easier for the speech recogniser to correctly recognise. This then was responsible for the lower rate of PTC. Below is an example from the experiments, (task 2 using the hand-crafted strategy), in which the system could not recognise a particular slot value, and this caused the user to eventually lose patience and quit.

Hyper-articulate speech is often associated with elevated rates of recognition failure (Shriberg et al., 1992). This is because hyper-articulated speech departs from the original training data upon which a recogniser was developed. The basic principle of Automatic Speech Recognition (ASR) is pattern matching of human speech with relatively static stored representations of sub-word units. Although recognition algorithms typically model phonemes and co-articulation effects, they do not tend to model dynamic stylistic changes in the speech signal that are elicited by environmental factors, such as hyper-articulate speech during miscommunication or Lombard speech during noise. Lombard speech refers to how a user's speech changes in a noisy environment. Oviatt et al. (1998) provides a technical description of how hyper-articulate speech is different from normal speech. It is also worth noting that the design of recognition technology means that one recognition error can lead to a clustering of errors. For example, a statistical model called a Hidden Markov Model (HMM), (see Rabiner (1989) for an introduction), is often used in speech recognition, and since an HMM's prediction for a word is based in part on its prediction for the previous word, if one word is misrecognised, then this may cause subsequent words to be misrecognised as well.

To summarise then, always repeating the same question/confirmation when a slot value is not recognised is likely to make a user frustrated. The user will then react by

Speaker	Transcript (ASR transcript for user)
System	how can I help you?
User	i want no I want to eat
System	how can I help you?
. User	er said eat.
System	did you say eat?
User	yes I would.
System	would you like something cheap, expensive or reasonably priced?
User	er i dont mind
System	you do not mind what kind of food would you like?
User	you er chinese food
System	a chinese restaurant do you want something central?
User	in the town
System	do you want something central?
User	er in my hotel
System	do you want something central?
User	can you repeat near the hotel the primus
System	do you want something central?
User	i would [User then quits]

Table 6.5: A hand-crafted strategy dialogue in which Automatic Speech Recognition (ASR) errors and over-use of repetition as a repair strategy cause the user to lose patience and quit.

hyper-articulating and/or adopting an angry tone, and this will further degrade speech recognition, producing longer dialogues and a lower rate of task completion. With the DAs of the last two turns represented in the state (the DA2 state), the reinforcement learner has learned to sometimes use alternative repair strategies, (switch-focus, give-help, backtrack), and hence it has learned a strategy that better avoids eliciting hyper-articulate / irritated speech from the user. This results in shorter dialogues and a higher rate of task completion on average. It also raises the question of whether any "sensible" avoidance of repetition is an optimal repair strategy whenever the slot-status features are unchanged by the previous user turn e.g. any of switching focus/ giving help or backtracking are always optimal. We would think not given that the DA2 strat-

egy uses different repair strategies in different DA contexts, and indeed sometimes uses repetition, but we will investigate this further in the experiment described in Section 7.4.

We now move on to explain the relative performance of the DA2 Strategy and the Hybrid Strategy of Henderson et al. (2008).

6.5.2 The DA2 Strategy versus the Hybrid Strategy of Henderson et al. (2008)

In the real user tests here, like the Slot-Status and Hand-crafted strategies, the Hybrid Strategy of Henderson et al. (2008) is outperformed by the DA2 Strategy both in terms of Perceived Task Completion (PTC) and dialogue length, and hence also in terms of the reward function described in Section 6.3.4. The next section provides qualitative analysis of the Hybrid Strategy in order to account for this performance difference. Since I did not have access to the Lemon et al. (2006a) test logs, but instead to the "Cambridge" real user test logs, the Hybrid Strategy dialogues which we use for qualitative analysis come from these Cambridge tests. The Cambridge tests used a four-slot system and hence the full four-slot Hybrid Strategy. The four-slot Hybrid strategy was implemented in the three-slot TownInfo Spoken Dialogue System (SDS) used in Lemon et al. (2006a) by setting one of the slots as confirmed before the start of each dialogue. Note that the slot which was confirmed before the start of each dialogue was always the same. In the example dialogues of the next section then, in addition to the slots for type, price and location, the reader will notice the system attempting to fill a fourth slot. We will refer to this fourth slot as the "near to" slot because it specifies what the bar/hotel/restuarant should be near to. Following on from the qualitative analysis of the next section, Section 6.5.2.2 will then go on to provide an explanation for why the learning approach used by Henderson et al. (2008) produced the strategy it did.

6.5.2.1 Qualitative Analysis of the Hybrid Strategy

In Sections 5.3.7 and 5.4.3, we argued that the Slot-Status and Hand-crafted strategies were outperformed by DA2 in the real user experiments here because in many cases they use less effective repair strategies for when the slot-status features are unchanged

by the last user turn i.e. they over-use repetition. Hence let us consider whether this is also a factor in explaining why the results for the Hybrid Strategy in Lemon et al. (2006a) are worse than those of the DA2 Strategy here.

System transcript	**User (ASR transcript)**
how can i help you?	id like to find a hotel
did you say hotel?	yes
what area would you like, for example north, east or riverside?	east
an in the east hotel where would you like the hotel to be near to?	
where would you like the hotel to be near to?	
where would you like the hotel to be near to?	oh
where would you like the hotel to be near to?	like
where would you like the hotel to be near to?	like
where would you like the hotel to be near to?	
where would you like the hotel to be near to	to the park
near the park, would you like cheap expensive or reasonably priced	
would you like something cheap, expensive, or reasonably priced?	
would you like something cheap, expensive, or reasonably priced?	
would you like something cheap, expensive, or reasonably priced?	oh
would you like something cheap, expensive, or reasonably priced?	something stop some
would you like something cheap, expensive, or reasonably priced?	i dont mind
you do not mind, what hotel category would you like? for example 5*	
i can respond to simple requests, for example, a cheap indian	
restaurant, or you can say quit or restart. How can i help you?	a five star hotel
now i am going to search for a hotel five star near the park in the east please wait a moment while i search for suitable hotel options.	

Table 6.6: The Hybrid Strategy of Henderson et al. (2008) apparently over-using repetition as a repair strategy.

Table 6.6 shows an example Hybrid Strategy dialogue which involves repair strategies. There are two points in this dialogue at which progress stalls for a number of turns. The first is when the system tries to obtain a value for location, and the second is when the system tries to obtain a value for price. The user responses are blank for these turns because this is how they appeared in the logs, and we are therefore unsure whether they are ASR rejections or recognised as out-of-domain. In any case, at both these points at which progress stalls, the system keeps repeating its original question until it

is finally able to recognise a slot value in the user response - it needs 7 attempts for the "near to" slot, and 6 for the price slot. As was said in Section 6.5.1, always repeating the same question/confirmation when a slot value is not recognised is likely to make a user frustrated. The user will then react by hyper-articulating and/or adopting an angry tone, and this will further degrade speech recognition, producing longer dialogues and a lower rate of task completion. It seems then that less effective repair strategies for when the slot-status features are unchanged by the last user turn are at least part of the explanation for why the Hybrid Strategy performed worse in real user experiments than the DA2 Strategy.

What other possible factors could there be to account for performance differences between DA2 and the Hybrid Strategy? Are there other contexts in which the Hybrid Strategy seems to use sub-optimal actions? Notice that in the example dialogue in Table 6.6, the Hybrid Strategy queries the database without first trying to confirm the slot value for hotel "type". In the example dialogue in Table 6.7, we see the Hybrid Strategy query the database without first confirming the "near-to" slot value. We could speculate then that since the Hybrid Strategy does not always confirm all of the slots, it might at least sometimes be submitting database queries that are incorrect with respect to unconfirmed slots, and failing tasks as a result. It could be however that the Hybrid Strategy is actually doing something very sensible - it may not be confirming in the above examples because it has somehow inferred based on certain contextual features e.g. Word Error Rate (WER) that the probability of the slot value being incorrect is very small, (see Section 3.10.2 for a list of the different types of state features used by the Hybrid Strategy). If such slot-values are invariably correct, then querying the database without confirming them would obviously enable the Hybrid Strategy to successfully complete the task in fewer turns. Note that the example dialogue in Table 6.6 seems to suggest that dialogue history is not a decisive factor for the Hybrid Strategy in deciding whether to confirm the last slot - despite all of the problems on previous turns, it still chooses not to confirm.

It is unclear then whether these choices not to confirm are sub-optimal, but it seems that making minimal effort to fill certain slots and then querying the database with unfilled slots must be sub-optimal. This is because the information slots for a COMMUNICATOR flight-booking system e.g. the destination city, have a very large number of possible values, and as a result, querying the database with unfilled slot values would mean that an infeasibly large number of possible flight options would have to be pre-

System	User (ASR transcript)
hello, welcome to the talk city information dialogue system. how can i help you?	i would like to find a quick snack please
did you say snack ?	yes
what kind of food would you like, for example italian or chinese?	i dont mind
you do not mind would you like cheap, expensive, or reasonably priced?	dont care
you do not mind what area would you like, north, east or riverside?	south
an in the south restaurant where would you like it to be near to?	close to the train station
now i am going to search for a restaurant in the south. please wait a moment while i search for suitable restaurant options.	

Table 6.7: Here the Hybrid Strategy does not attempt to confirm the “near to” slot.

sented to the user. Such action choices would surely therefore have a disastrous effect on average task completion. Despite this, we see the Hybrid Strategy making such action choices in the example dialogue in Table 6.8. Here the Hybrid Strategy only asks for the “type” and “near to” slots once, and when it fails to obtain values, it nevertheless goes ahead and queries the database. Such behaviour will not have anything like as dramatic a negative effect on task completion for the TownInfo system because here the number of possible values for each slot is small enough so that the system can often get away with presenting all of the possible options when the database query is not fully constrained. However, even if task completion is achieved in such cases, having to listen to a few different options will likely annoy the user and so reduce their satisfaction with the system. Given then that the Hybrid RL/SL approach has produced sub-optimal actions for both filling slots and repair strategies, it seems likely then that at least some non-confirming actions of the type shown in Tables 6.6 and 6.7 are also sub-optimal.

Note that apart potentially from failing to confirm the last slot, in the example dialogues

System	User (ASR Transcript)
hello. Welcome to the talk city information dialogue system. how can i help you?	
how can i help you?	id like to find out plays red can get a do you have to
how can i help you?	id like to find a bar
did you say bar?	yes
what type of bar are you looking for? Maybe a cocktail bar or a wine bar or a beer?	oh
would you like something cheap, expensive, or reasonably priced?	reasonably priced please
a reasonably,priced bar what area of the town for example north,east or riverside area?	oh
what area of the town, for example north, east or riverside area?	i dont mind
you do not mind where would you like the bar to be near to?.	
now i am going to search for a bar moderate please wait a moment while i search for suitable bar options. options there are four options matching your query.	

Table 6.8: Here the Hybrid Strategy makes minimal effort to fill the “type” and “near to” slots and queries the database with them unfilled.

which we have seen here, so long as the dialogue is progressing smoothly, the Hybrid Strategy seems to behave sensibly - it implicitly confirms filled slots whilst asking empty slots. Hence in conclusion, it seems that less effective repair strategies for when the slot-status features are unchanged i.e. over-use of repetition is one cause for the Hybrid Strategy performing worse than the DA2-strategy. A second cause is making minimal effort to fill slots in certain contexts and so querying the database with unfilled slots, and another potential cause is failure to confirm certain slots, especially the last slot. However these causes alone do not seems to be sufficient to account for the Hybrid Strategy’s much worse performance in testing in simulation - in Chapter 5,

when tested with the linear function approximation simulation, we reported that the Hybrid Strategy had obtained an average HLG05 score of 140.3 (88 for slots filled, 70 for slots confirmed, -17.7 for number of system turns). Notice especially that the dialogue length is very high. I was unable to view any logs for these simulation tests but have found out via personal communication that in some contexts, the Hybrid Strategy will attempt to confirm an empty slot, (there are no examples of this in the "Cambridge" tests). Such behaviour is clearly sub-optimal and had a detrimental effect on the Hybrid Strategy's performance in testing in simulation, particularly with respect to dialogue length.

We now move on to provide an explanation for why the Hybrid RL/SL approach produced the kind of strategy which we have described here.

6.5.2.2 Explanation for sub-optimal action choices in Hybrid Strategy

Why then did the Hybrid Reinforcement Learning (RL)/Supervised Learning (SL) approach of Henderson et al. (2008) produce a strategy which makes the sub-optimal action choices described in the previous section, and hence does not perform as well as DA2 in the real and simulated user tests? The main explanation seems to be that the Hybrid Strategy is almost entirely derived from SL, (the RL component only leads to an approximately 1% improvement over the purely supervised policy), and thus does not fully exploit the power of exploratory RL as the DA2 strategy does. Recall from Section 3.10 that the Hybrid RL/SL approach had an enormous policy space to search, with relatively little data to work with, and so it is clear that the optimal strategy has not yet been found. The main advantage of the Hybrid RL/SL approach in fact comes from its supervised component: a "multi-version" system is being learned which blends the best aspects of the original COMMUNICATOR systems. In addition, it was not even possible for our reinforcement learner to learn one of the types of sub-optimal action choice described in the previous section i.e. confirming an empty slot. Recall from Section 5.3.3 that unlike Henderson et al., we used a rule during RL which stopped the Dialogue Manager (DM) from choosing to confirm an empty slot.

It should be noted that apart from a different learning approach, Henderson et al. also used a different training reward function - the Hybrid Strategy was trained with PARADISE scores i.e. a measure of user satisfaction (see Section 3.4.2 for the list of features), whereas our training reward function was based only on filling/confirming

slots and dialogue length. Recall that the filling/confirming slots element was present because the number of filled/confirmed slots is strongly correlated to task completion, and task completion is obviously something which we want a dialogue strategy to be optimised for. Task completion and dialogue length are clearly very important in determining user satisfaction, but other features are also required in order to best predict PARADISE user satisfaction scores, (again refer to Section 3.4.2 for the list of features). Hence, all other things being equal, a strategy optimised for the PARADISE reward function, and a strategy optimised for our reward function can be expected to be different to some degree. We should note then that the main evaluation measures used in the real user tests were Perceived Task Completion (PTC) and dialogue length, and in the simulated user tests, they were the number of filled/confirmed slots and dialogue length, (see Section 5.2.3 for the HLG05 reward function). These evaluation measures favour our strategies over one optimised for PARADISE user satisfaction scores, and so may have been another factor in the Hybrid Strategy's worse performance as compared to DA2.

6.6 Summary

This chapter described a real user experiment which successfully tested the hypothesis that when implemented in a full Spoken Dialogue System (SDS) and tested on real users, a state-of-the-art hand-crafted strategy, and a strategy learned with only slot-status features are both outperformed by a strategy learned with additional recent Dialogue Acts (DAs). The DA2 strategy outperformed both the Slot-Status and Hand-crafted strategies in terms of Perceived Task Completion (PTC) ($p < 0.05$ versus both) and dialogue length ($p < 0.05$ versus Slot-Status). This was due to its more effective repair strategies for when the slot-status features were unchanged by the previous user turn, (usually due to one or more speech recognition errors). The Slot-Status and Hand-crafted strategies would simply keep asking for the same slot value until eventually something was recognised, while the DA2-strategy would sometimes use an alternative repair strategy i.e. switch focus/give help/backtrack. Repeatedly asking the same question was likely to frustrate the user, eliciting *hyper-articulate speech* which increased the likelihood of speech recognition errors. Users could get stuck on particular slots in this way, resulting in longer dialogues. Occasionally they would completely lose patience and quit, or alternatively, as a strategy to induce the system to move on,

they might deliberately supply a slot value which was different to the one that they wanted, but which they thought might be easier for the speech recogniser to correctly recognise. This then was responsible for a lower rate of perceived task completion.

The Spoken Dialogue System (SDS) used in this experiment is called "TownInfo" (Lemon et al., 2006b), and it operates in the tourist information domain. As a result, in order to run the experiment, the learned strategies had to first be ported from the COMMUNICATOR domain into this new domain. Like Lemon et al. (2006a) then, the results here also demonstrate that it is possible to learn a slot-filling strategy in one domain which then works well in another i.e. it is possible to learn generic slot-filling strategies.

We also compared our results to those of Lemon et al. (2006a), which used the same methodology to test the Hybrid Strategy of Henderson et al. (2008). Although no significance tests were possible, the scores for Perceived Task Completion (PTC) and dialogue length were better for DA2. Qualitative analysis of the Hybrid Strategy showed apparently sub-optimal action choices in a number of contexts e.g. overuse of repetition as a repair strategy, making minimal effort to fill slots and querying the database with unfilled slots, confirming empty slots. We provided an explanation for why the Hybrid RL/SL approach may have produced sub-optimal behaviour - the main reason is that it is almost entirely a supervised strategy, and the enormous policy space means that the optimal policy has not yet been found. We also noted that the Hybrid Strategy was trained with a different training reward function - it was trained on PARADISE user satisfaction scores. A strategy optimised for this reward function would not have been as favoured by the main evaluation measures used in the real user tests, nor those used in the simulated user tests reported in Chapter 5.

Chapter 7

Investigating the role of Dialogue Acts in learning repair Strategies

7.1 Introduction

This chapter seeks to add to the insights provided by the experiments of the previous two chapters. The experiments of the previous two chapters showed that when training with stochastic user simulations whose probabilities are derived from real user data, adding the Dialogue Acts (DAs) of the last system and user turns into the state produces better learned strategies. In Chapter 5 we showed an improvement in testing with these user simulations, and in Chapter 6, an improvement in testing with real users. Analysis of these experiments enabled us to form hypotheses as to how the DAs were improving the learned strategy. In Chapter 5 our analysis suggested that the DAs were making significant improvements to the learned strategy only with respect to repair strategies for states in which the slot-status features are unchanged by the previous user turn i.e. Slot-Status Features Unchanged (SSFU) states. They did not appear to be making significant improvements with respect to dealing with user indications of misunderstanding errors, nor in portions of dialogue in which there was smooth progress towards the goal of filling and confirming all of the slots. With regard to learning repair strategies in SSFU states, the DAs seemed to be important both for identifying SSFU states, and then also for choosing which repair strategy was best to apply. In Chapter 6, our analysis suggested that the strategy learned with only slot-status features over-used repetition in SSFU states - this often frustrated the users, eliciting irritated/hyperarticulated speech which caused more Automatic Speech

Recognition (ASR) errors and hence longer dialogues and lower task completion. A possible hypothesis then is that any kind of "sensible" avoidance of repetition in SSFU states is optimal. Here we define "sensible" as any of the repair strategies for SSFU states which emerged from the Reinforcement Learning (RL) i.e. *repeating*, *switching focus*, *backtracking* and *giving help* (see Section 5.3.7.2 for definitions).

In this chapter then, (see also Frampton and Lemon 2008b), we conduct new experiments in order to further investigate the hypotheses described above. We conduct these experiments using the n-gram simulations - we justify this with the fact that Chapter 6's real user experiment for strategies learned with these simulations produced a positive result. In addition to the hypotheses described above, we also investigate which is most important in learning a dialogue strategy - the DA of the last system turn or the DA(s) of the last user turn. We also compare our findings to those of relevant previous research, Bohus and Rudnicky (2005) and Skantze (2003)/Chapter 4 of Skantze (2007), which investigates repair strategies for non-understanding errors without using RL or user simulations, (see Section 1.1.1 for a definition of non-understanding errors).

The remainder of this chapter proceeds as follows. Section 7.2 describes the first of the four new experiments. This experiment further investigates whether the DAs only significantly improve the learned strategy through more effective repair strategies in SSFU states. Section 7.3 then describes the second experiment, which investigates whether DAs are useful for choosing *which* repair strategy to apply, rather than just for identifying SSFU states and hence states in which a repair strategy is required. Next, Section 7.4 describes the third experiment which investigates the possibility that any sensible repair strategy which avoids repetition will always be optimal. Section 7.5 then describes the fourth experiment which investigates whether the DA of the last system turn or the DA(s) of the last user turn are more important in learning a dialogue strategy. Following this, Section 7.6 reviews the relevant previous research of Bohus and Rudnicky (2005) and Skantze (2003)/Chapter 4 of Skantze (2007) and makes comparisons to the experimental work of this thesis. Finally, Section 7.7 is a summary of the contents of the chapter.

7.2 Are Dialogue Acts only useful for repair strategies?

Here we investigate whether the Dialogue Acts (DAs) of the last system and user turns are only significantly improving the learned strategy by producing better repair strate-

gies in Slot-Status Features Unchanged (SSFU) states. This would mean that they are not significantly improving the learned strategy with respect to dealing with user indications of misunderstanding errors (e.g. "no [a slot-value]" following an attempted confirmation of a slot's value), nor in portions of dialogue in which there is smooth progress towards the goal of filling and confirming all of the slots. If this were the case, then this would be a useful finding because it would mean that these recent DAs could potentially be sacrificed in other states, so reducing the size of the state space and hence speeding up learning.

7.2.1 Hypothesis: Dialogue Acts improve the learned strategy only through better repair strategies for SSFU states

The hypothesis tested here is that representing the Dialogue Acts (DAs) of the last system and user turns in the Reinforcement Learning (RL) state only produces improvements in the learned strategy with respect to better repair strategies in Slot-Status Features Unchanged (SSFU) states.

7.2.2 Methodology

In all of the experiments described in this chapter, we train three-slot strategies with the 4-gram simulation and test with the 5-gram and vice versa, and use the action set from Section 5.3.3, and the all-or-nothing reward function from Section 5.3.4. We test for significant differences in performance between strategies using one-tailed independent samples t-tests. Here, to test the above hypothesis, we use 2 new state features:

1. "User DA(s) if Slot-Status Features Unchanged (UDAsifSSFU) feature" : a feature which takes the value of the DA(s) of the last user turn only in an SSFU state, (otherwise it has the value "changed"),

2. "System DA if Slot-status Features Unchanged (SDAifSSFU) feature": a feature which takes the value of the DA of the last system turn only in an SSFU state, (otherwise it has the value "changed").

We learn a new strategy, which we call the "DA2ifSSFU Strategy", using the following state features:

1. a slot-status feature for each slot with the values “empty”, “filled” and “confirmed”,
2. SDAifSSFU,
3. UDAsifSSFU.

We call this strategy the DA2ifSSFU Strategy, because it is learned with a state space that represents the DAs of the last 2 turns, only if the state is an SSFU state. Since the DA information is now only available in SSFU states, if our hypothesis is correct, then the DA2ifSSFU Strategy will perform no better or worse than the DA2 Strategy. Note that DA information is not available in states in which there is smooth progress towards the goal of filling and confirming all of the slots i.e. states in which one or more slots are now filled/confirmed as compared to the last state. Neither is the DA information available in dealing with user indications of misunderstanding errors.

We now provide results for the performance of the DA2ifSSFU Strategy in testing, and compare them to those of the Slot-Status, DA1 and DA2 Strategies.

7.2.3 Results and Analysis

The results in Table 7.1 show that the DA2ifSSFU Strategy did not perform significantly better or worse than the DA2 Strategy. This supports the hypothesis that the recent DAs are only improving the learned strategy with respect to better repair strategies in Slot-Status Features Unchanged (SSFU) states.

7.2.4 Conclusion: Dialogue Acts improve the learned strategy only through better repair strategies for SSFU states

The results of this experiment support the hypothesis that the recent DAs are only improving the learned strategy through repair strategies. Strategies learned with DA information only when the dialogue was not progressing performed as well as strategies for which the learner used DA information in all contexts.

We now move on to describe an experiment which investigates whether the recent DAs are useful for choosing *which* repair strategy to use in SSFU states, rather than only for identifying SSFU states.

	Features	Av. Score	Conf. Slots	Length
$4 \rightarrow 5$ **gram = (a)**				
Slot-Status (a)	Slot-Status	47.57	100	10.49
DA1 Strategy(a)	+ Last User DA(s)	58.00**	100	8.40
DA2 Strategy(a)	+ Last System & User DAs	59.14*	100	8.17
DA2ifSSFU (a)	+ UDAsifSSFU & SDAifSSFU	59.07	100	8.19
$5 \rightarrow 4$ **gram = (b)**				
Slot-Status (b)	Slot-Status	55.59	100	8.88
DA1 Strategy(b)	+ Last User DA(s)	59.12**	100	8.18
DA2 Strategy(b)	+ Last System & User DAs	59.77*	100	8.05
DA2ifSSFU (b)	+ UDAsifSSFU & SDAifSSFU	59.82	100	8.04
Slot-Status (av)	Slot-Status	51.58	100	9.68
DA1 Strategy(av)	+ Last User DA(s)	58.56**	100	8.29
DA2ifSSFU (av)	+ UDAsifSSFU & SDAifSSFU	59.45*	100	8.11
DA2 Strategy(av)	+ Last System & User DAs	59.46	100	8.11

Table 7.1: Testing the learned strategies after 50000 training dialogues, average reward achieved per dialogue over 1000 test dialogues. a = strategy trained using 4-gram and tested with 5-gram; b = strategy trained with 5-gram and tested with 4-gram; av = average; * Improvement over the strategy in above row with significance level $p < 0.05$; ** Significance level $p < 0.005$.

7.3 Are Dialogue Acts useful for choosing which repair strategy to use?

We have shown that the Dialogue Act (DAs) of the last system and user turns improve the learned strategy by producing better repair strategies in Slot-Status Features Unchanged (SSFU) states, but it would be useful to know more about how exactly they do this. Certainly the DAs enable the reinforcement learner to recognise SSFU states, and hence that a repair strategy is required. However, assuming that different repair strategies are more appropriate in different contexts and given that the slot-status features are already represented in the state, are the DAs also helpful in deciding *which* repair strategy to apply? Based on the analysis in Section 5.3.7, we expect to find that they are e.g. recall from Section 5.3.7.2 that a DA Strategy was more likely to use the re-

peat repair strategy following an Automatic Speech Recognition (ASR) rejection, but to switch focus following a user utterance that is recognised as out-of-domain. However, if we were to find that the DAs are not helpful in deciding *which* repair strategy to apply, then we would be better off replacing them with a binary feature that explicitly records whether the slot-status features are unchanged. This would capture the only useful information in the DAs and at a smaller cost in terms of increasing the size of the state space.

7.3.1 Hypothesis: Dialogue Acts are useful for choosing which repair strategy to use

Given that slot-status features are already in the state, the hypothesis is that the Dialogue Acts (DAs) of the last system and user turns are useful for choosing *which* repair strategy to use.

7.3.2 Methodology

To test the above hypothesis, we learn two new strategies with two new different state representations. The first new strategy, which we call the "Slot-Status Features Unchanged (SSFU) Strategy" is learned using the following state features:

1. a slot-status feature for each slot with the values "empty", "filled" and "confirmed",
2. a Slot-Status Features Unchanged (SSFU) feature i.e. a binary feature that records whether or not the state is an SSFU state.

Evidently, we call this strategy an SSFU Strategy because it is learned using an SSFU feature instead of recent Dialogue Act(s) (DAs). With such a state representation, the learner is able to recognise all of the contexts in which the slot-status features are unchanged, and hence where a repair strategy is required. However, it has no DA information to help it choose which repair strategy to apply - it can only make these choices based on the values of the slot-status features. Hence if such a strategy performs worse than the DA2 Strategy, then the hypothesis is correct, while if it performs as well as or better than the DA2 Strategy, then the hypothesis is incorrect.

The second new strategy, which we call the "DA1ifSSFU Strategy" is learned using the following state features:

1. a slot-status feature for each slot with the values "empty", "filled" and "confirmed",
2. the UDAsifSSFU feature already defined in Section 7.2.2.

Slot-Status features	**Next sys DA**	**Next user DA(s)**	**New DA1 state**
[conf,fill,emp]	icSlot2AskSlot3	no	[conf,emp,emp,user(no)]
[conf,emp,emp]	askSlot3	no	[conf,emp,emp,user(no)]

Table 7.2: In the first row, the slot-status features change, and so the new state is a non-SSFU state, but in the second they do not change, and so the new state is a SSFU state. A DA1ifSSFU state representation represents distinguishes between these two contexts, but a DA1 state representation does not.

We call this strategy a DA1ifSSFU Strategy because it is learned with a state space that represents the DA(s) of the last 1 turn, (the last user turn), only if the state is an SSFU state. Comparing the performance of the DA1ifSSFU Strategy with the DA1 and DA2 Strategies can tell us about the role of the last system DA in learning repair strategies in SSFU states. The reason for this centres on contexts in which the DA(s) of the last user turn are insufficient to clearly identify whether or not a state is an SSFU state. These contexts occur when the user response is not an ASR rejection or recognised as out-of-domain, but instead "yes" or "no" in response to a straightforward request for a slot value. Table 7.2 shows an example - the first row contains a dialogue exchange between the system and user which leads to a non-SSFU state, and the second, an exchange which leads to a SSFU state, but the DA1 state representation for both of these contexts is the same. Recall that SSFU states of the kind in the second row are fairly uncommon, (see Section 5.3.7 and Figure 5.6), so it is perhaps unlikely that the system's behaviour in such states will have significant effects on performance. In any case, for the DA1ifSSFU Strategy, there is never any ambiguity as to whether or not a state is an SSFU state because the UDAsifSSFU feature explicitly represents this - if it takes a DA value, then the state is an SSFU state, otherwise it is not. Hence if the role of the last system DA in learning repair strategies in SSFU states is only to resolve such ambiguities and hence help to clearly identify more SSFU states, then we would

expect the DA1ifSSFU Strategy to perform as well as the DA2 Strategy. However, if the last system DA plays a role in choosing *which* repair strategy to apply, then the DA1ifSSFU Strategy will not perform as well as the DA2 Strategy - it does not have explicit knowledge of the last system DA to inform its choice.

We now provide results for the performance of the SSFU and DA1ifSSFU Strategies in testing, and compare these results to those of the DA1 and DA2 Strategies.

7.3.3 Results and Analysis

	Features	**Av. Score**	**Conf. Slots**	**Length**
$4 \rightarrow 5$ **gram = (a)**				
Slot-Status (a)	Slot-Status	47.57	100	10.49
SSFU (a)	+ SSFU	49.32*	100	10.14
DA1 Strategy(a)	+ Last User DA(s)	58.00*	100	8.40
DA1ifSSFU (a)	+ UDAsifSSFU	58.84	100	7.94
DA2 Strategy(a)	+ Last Sys & User DAs	59.14*	100	8.17
$5 \rightarrow 4$ **gram = (b)**				
Slot-Status (b)	Slot-Status	55.59	100	8.88
SSFU (b)	+ SSFU	57.61*	100	8.48
DA1ifSSFU (b)	+ UDAsifSSFU	58.87*	100	8.23
DA1 Strategy(b)	+ Last User DA(s)	59.12	100	8.18
DA2 Strategy(b)	+ Last Sys & User DAs	59.77*	100	8.05
Slot-Status (av)	Slot-Status	51.58	100	9.68
SSFU (av)	+ SSFU	53.47*	100	9.31
DA1 Strategy(av)	+ Last User DAs	58.56*	100	8.29
DA1ifSSFU (av)	+ UDAsifSSFU	58.86	100	8.23
DA2 Strategy(av)	+ Last Sys & User DAs	59.46*	100	8.11

Table 7.3: Testing the strategies, average reward achieved per dialogue over 1000 test dialogues. a = strategy tested with 4-gram; b = strategy tested with 5-gram; av = average; * improvement over strategy in above row with significance level $p < 0.005$

Table 7.3 shows that although both the Slot-Status Features Unchanged (SSFU) and DA1ifSSFU Strategies outperformed the Slot-Status Strategy ($p < 0.05$), they performed worse than the DA2 Strategy ($p < 0.05$). This supports the hypothesis - given

that slot-status features are already in the state, additional Dialogue Act (DA) information is useful for choosing *which* repair strategy to use.

We now provide qualitative analysis of both of the new strategies in order to explain their performance as compared to the Slot-Status, DA1 and DA2 Strategies. We start with the SSFU Strategy.

7.3.3.1 The Slot-Status Features Unchanged Strategy

We first consider why the Slot-Status Features Unchanged (SSFU) Strategy outperforms the Slot-Status Strategy ($p < 0.05$). As usual, we only detected apparently important behavioural differences in SSFU states. Unlike for the Slot-Status Strategy, the SSFU Strategy always knows when the state is an SSFU state because the SSFU feature provides this information. Hence for first-visits to SSFU states, the value of the SSFU feature changes and the state is now different to the previous state. This means that the reinforcement learner could learn to do something other than *repeat* e.g. it could *switch focus*, and previous experiments have already demonstrated that this is often preferable, (see Sections 5.3.7.2, 5.4.3 and 6.5). Figure 7.4 contains an example which shows the SSFU Strategy *switching focus* when a user turn fails to fill a slot value and so the resulting state is an SSFU state. By contrast, the Slot-Status Strategy keeps repeating its previous question.

Speaker	**SSFU dialogue**	**Slot-Status dialogue**
System:	impConfSlot1_askSlot2	impConfSlot1_askSlot2
User:	[out-of-domain]	[out-of-domain]
System:	askSlot3	impConfSlot1_askSlot2
User:	slot3	[out-of-domain]

Table 7.4: Here, slot 1 is initially filled, while slots 2 and 3 are empty, and following an out-of-domain user response, the Slot-Status Features Unchanged (SSFU) Strategy switches focus from slot 2 to 3, while the Slot-Status Strategy uses the *repeat* repair strategy.

Note that if after the first-visit to an SSFU state, the state remains an SSFU state, the state for the SSFU Strategy will not have changed, and so the reinforcement learner could not have learned a different repair strategy to the one applied in the first-visit

SSFU state i.e. it could not learn combinations of different repair strategies over consecutive turns.

We now consider why the SSFU Strategy was outperformed by both the DA1 and DA2 Strategies ($p < 0.05$). Since for the SSFU Strategy the reinforcement learner is now able to recognise all SSFU states, and hence where a repair strategy is required, this means that Dialogue Acts (DAs) must have some additional importance with respect to learning repair strategies, and this can only be that they are useful for choosing *which* repair strategy to apply. States for which the slot-status features are unchanged, and corresponding learned action pairs can be found to support this hypothesis, and those in Table 7.5 are examples. For these SSFU states, the slot-status features are the same, but the last user DA and learned actions are not. If the distinctions between the different user DAs in the table were unimportant, then we would expect the DA1 Strategy to use the same action in each case, and for the SSFU Strategy to perform as well as the DA1 Strategy, but this is not the case. Note that in the table, for the SSFU Strategy, the state representation is always "[confirmed,filled,empty,user,unchanged]", and so the learned action is also always the same - to ask slot 3.

State Representation (DA1 Strategy)	**DA1 Strategy**	**SSFU Strategy**
[confirmed,filled,empty,user(out-of-domain)]	askSlot2	askSlot3
[confirmed,filled,empty,user(asr_rejection)]	askSlot3	askSlot3
[confirmed,filled,empty,user(no)]	askSlot2	askSlot3
[confirmed,filled,empty,user(yes)]	askSlot3	askSlot3

Table 7.5: The table shows learned actions in different states for the DA1 Strategy and the SSFU strategy (trained with the 5-gram simulation). For the SSFU strategy, the state representation is always the same i.e. "[confirmed,filled,empty,user,unchanged]", and hence the learned action is always the same.

We now provide qualitative analysis of the DA1ifSSFU Strategy.

7.3.3.2 The DA1ifSSFU Strategy

We consider here why the DA1ifSSFU Strategy failed to perform as well as the DA2 Strategy, and no better than the DA1 Strategy. As stated in Section 7.3.2, with a DA1 state representation, there are some cases where it is impossible to be sure whether or not the slot-status features have changed. With a UDAsifSSFU feature, there are no

such ambiguities, and as a result, it was possible that the DA1ifSSFU Strategy would improve over the DA1 Strategy. However, as already stated, the DA1ifSSFU Strategy did not improve over the DA1 Strategy. This is unsurprising because ambiguous DA1 states of the type already shown in Table 7.2 are rare, (see Figure 5.6). In any case, when we inspected such ambiguous DA1 Strategy states and equivalent DA1ifSSFU states, we often found the learned actions to be the same. Table 7.6 shows an example of this.

Last Sys DA	**Last User DA**	**DA1ifSSFU state**	**DA1ifSSFU action**
impConfSlot2AskSlot3	no	[conf,emp,emp,statechanged]	askSlot3
askSlot3	no	[conf,emp,emp,no]	askSlot3

Table 7.6: For both of the DA1ifSSFU states, the equivalent state representation for the DA1 Strategy would be the same as the second DA1ifSSFU state. Despite the alternative state representations, the learned action is still the same.

The fact that the DA1ifSSFU Strategy performs significantly worse than the DA2 Strategy suggests two things regarding the last system turn:

1. The last system DA is important for choosing *which* repair strategy to apply.
2. The system DA cannot always be inferred from the UDAsifSSFU and slot-status features.

The first point is supported by examples where the last system DA is not required to identify a state as an SSFU state, but nevertheless, the DA2 Strategy behaves differently to the DA1 and DA1ifSSFU Strategies i.e. it chooses a different repair strategy. Table 7.7 shows such examples where the DA2 state-learned action pairs were visited between 51 and 659 times during the 50000 training dialogues.

We now provide our conclusion for this experiment.

7.3.4 Conclusion: Dialogue Acts are useful for choosing which repair strategy to use

The hypothesis is correct - given that slot-status features are already in the state, additional Dialogue Act (DA) information is useful for choosing *which* repair strategy to

DA2 state representation	DA2 Strategy	DA1 Strategy
[confirmed,empty,empty,giveHelp,out-of-domain]	askSlot3	askSlot2
[confirmed,empty,empty,askSlot2,out-of-domain]	askSlot3	askSlot2
[empty,filled,empty,askSlot1,asr_reject]	askSlot2	askSlot1
[confirmed,filled,empty,expConfSlot2,out-of-domain]	askSlot3	askSlot2
[confirmed,filled,empty,giveHelp,out-of-domain]	askSlot3	askSlot2
[empty,empty,empty,askSlot3,asr_reject]	giveHelp	askSlot1
[empty,empty,empty,askSlot2,asr_reject]	giveHelp	askSlot1

Table 7.7: Examples where the last system DA is not required to identify a state as an SSFU state, but nevertheless, the DA2 and DA1 Strategies, (both trained with the 5-gram), use a different repair strategy - such examples support the claim that the last System DA is important in choosing *which* repair strategy is best to apply.

use. We had already established that the recent DAs only improve the learned strategy through repair strategies in Slot-Status Features Unchanged (SSFU) states, and hence that the reinforcement learner can do without it otherwise. However, our result here shows that it should be retained in SSFU states - we will learn better repair strategies as a result.

Note that although the recent DAs affect which repair strategy should be used in an SSFU state, finding the optimal repair strategy could still be very simple if it is true that any "sensible" avoidance of repetition will always be optimal. This hypothesis became a possibility following the real user experiment of Chapter 6, and we now move on to describe an experiment which further investigates.

7.4 Repair strategies that avoid repetition

We have established now that the Dialogue Acts (DAs) of the last system and user turns are useful for choosing *which* repair strategy to apply in Slot-Status Features Unchanged (SSFU) states. However, we should recall our analysis of the relatively poor performance of the Slot-Status Strategy in the real user experiment in Chapter 6, (see Section 6.5). Here we found that the Slot-Status Strategy over-used repetition in SSFU states, and that this often frustrated the users, eliciting irritated / hyperarticulate speech

which caused more Automatic Speech Recognition (ASR) errors and hence longer dialogues and lower task completion. This raises the possibility, that finding the optimal repair strategy in an SSFU state is very simple because any "sensible" avoidance of repetition will always be optimal. Here we define any of the repair strategies that emerged from the Reinforcement Learning (RL) to be "sensible" i.e. *repeat*, *switch focus, give help* or *backtrack* (see Section 5.3.7.2 for definitions). Hence if this were true, as regards choosing which repair strategy to apply in an SSFU state, the recent DAs are useful, simply because they enable the learned strategy to avoid repetition. Using machine learning to optimise repair strategies in SSFU states would then seem unnecessary. We suspect that the proposition which we are describing here is false because despite often avoiding repetition, the DA1 and DA2 Strategies still use repetition in various SSFU states. However, in the experiment here, we investigate further.

7.4.1 Hypothesis: Any kind of sensible avoidance of repetition is an optimal repair strategy

For any Slot-Status Features Unchanged (SSFU) state, any "sensible" repair strategy which avoids repeating the last system action will always be optimal. Here we define a "sensible" repair strategy as any of those which emerged from the Reinforcement Learning (RL) experiments of Chapter 5 i.e. *repeat*[1], *switch focus, give help* or *backtrack.*

7.4.2 Methodology

In order to test the hypothesis, we tested new strategies with the n-gram simulations. Each of these strategies follows either the Slot-Status or DA2 Strategies until the state is a Slot-Status Features Unchanged (SSFU) state, and then they always use a "sensible" repair strategy which avoids repeating the last system action. We define the new strategies here in terms of these repair strategies:

1. Switch Focus (SF): In SSFU states, this strategy always switches focus to a different unfilled/unconfirmed slot.

[1]Recall that the *repeat* strategy repeats the original question / attempted confirmation which led to the first SSFU state in a sequence of consecutive SSFU states. Hence if the current SSFU state is not the first, then this repair strategy does not necessarily mean repeating the previous system action e.g. see the Give Help and Repeat (GHR) Strategy described above.

2. Give Help and Repeat (GHR): In a first-visit SSFU state, this strategy always gives help, and then if the state remains an SSFU state, on the next system turn, it repeats the original question/confirmation. So long as the state remains an SSFU state, the strategy will continue to alternate in this way between first giving help and then repeating the original question/ confirmation on consecutive system turns.

3. Give Help and Switch Focus (GHSF): This is like the GHR Strategy except that instead of repeating the original question/confirmation, this strategy switches focus.

Note that if there are no unfilled/unconfirmed slots to switch focus to, the SF and GHSF Strategies continue to follow the learned strategy which they are following in non-SSFU states i.e. the Slot-Status or DA2 Strategy. We originally tested versions of these strategies that used the DA2 Strategy learned actions in non-SSFU states. If the hypothesis was correct, then we expected these new strategies to perform at least as well as the DA2 Strategy, otherwise the hypothesis would be incorrect. It was possible however that this was not a totally controlled test, because the new hand-coded repair strategies might have sometimes taken the DA2 Strategy into a state which had not been sufficiently explored during training and hence in which the learned action was sub-optimal. As a result the SF, GHR and GHSF Strategies might show worse performance than the DA2 Strategy despite the hypothesis being correct. For this reason, we also tested versions of the SF, GHR, and GHSF Strategies which used the Slot-Status Strategy learned actions in non-SSFU states. The state space for the Slot-Status Strategy was small and so we could be confident that all non-SSFU states had been fully explored. It was still possible to test the hypothesis in the same way as before by comparing the performance of the SF, GHR and GHSF Strategies with the DA2 Strategy. This is because in previous experiments, (see Section 7.2.1), we have already established that the DA2 Strategy does not perform better than the Slot-Status Strategy in non-SSFU states.

We now describe the results.

7.4.3 Results and Analysis

We found no significant difference in performance depending on whether the new strategies followed the DA2 or Slot-Status Strategy in non-SSFU states. Table 7.8

	Av. Score	**Conf. Slots**	**Length**
$4 \rightarrow 5$ **gram = (a)**			
RL Baseline (a)	47.57	100	10.49
SwitchFocus (a)	51.98	100	9.60
HelpSwitchFocus (a)	53.18	100	9.36
HelpRepeat (a)	53.93	100	9.21
DA2 Strategy (a)	59.15*	100	8.17*
$5 \rightarrow 4$ **gram = (b)**			
SwitchFocus (b)	52.04	100	9.59
HelpSwitchFocus (b)	53.26	100	9.35
HelpRepeat (b)	54.36	100	9.13
RL Baseline (b)	55.59	100	8.88
DA2 Strategy (b)	59.75*	100	8.05*
RL Baseline (av)	51.58	100	9.68
SwitchFocus (av)	52.01	100	9.60
HelpSwitchFocus (av)	53.22	100	9.36
HelpRepeat (av)	54.15	100	9.17
DA2 Strategy (av)	59.45*	100	8.11*

Table 7.8: Testing the strategies, average reward achieved per dialogue over 1000 test dialogues. a = strategy tested with 4-gram; b = strategy tested with 5-gram; av = average; * improvement on strategy in above row with significance level $p < 0.005$

shows the results obtained when the new strategies followed the DA2 Strategy in non-SSFU states. The three new strategies were all found to perform significantly worse than the DA2 Strategy. This indicates that the hypothesis is incorrect - whenever the state is an SSFU state, it is not the case that regardless of the values of the slot-status and Dialogue Act (DA) features, any "sensible" repair strategy which avoids repetition will always be optimal.

7.4.4 Conclusion: Any kind of sensible avoidance of repetition is not guaranteed to be an optimal repair strategy

Prior to this experiment we had established that the Dialogue Acts (DAs) of the last system and user turns were useful to the learner in choosing *which* repair strategy to apply in different Slot-Status Features Unchanged (SSFU) states - their usefulness was not limited to simply identifying SSFU states. The finding of this experiment relates to this problem of *which* repair strategy to apply in different SSFU states. Here we have established that it is not the case that any "sensible" repair strategy which avoids repeating the last system action is guaranteed to be optimal. If it were then this would mean finding the optimal repair strategy in any SSFU state was a simple task, and would make the kind of machine learning approach used in this thesis unnecessary. The problem is more complex than this, but the experimental work of this thesis has shown a method for tackling it effectively i.e. using Reinforcement Learning (RL) with a realistic user simulation, and a state which represents recent DAs.

We now move on to an experiment which investigates the relative importance of the DAs of the last system and user turns.

7.5 Experiment to investigate the relative importance of the last user and system turns

We have previously demonstrated that with slot-status features already in the state, adding the Dialogue Acts (DAs) of both the last system and user turns produces a better learned strategy than if only the DA(s) of the last user turn are added. Here we now address the question of which is more important - the DAs of the last user turn, or the DA of the last system turn. This is an interesting question, and it is potentially also an important one for future usages of Reinforcement Learning (RL) in dialogue strategy design. For example, if a strategy-designer used a more fine-grained DA schema to that used here, in order to maintain tractability, it may become necessary to choose between including the DAs of the last system or user turns in the state. The decision of which to include in the state would then depend on which was likely to improve the learned strategy the most. Indeed, although it seems unlikely based on our previous results, we are yet to rule out the possibility that when the DA of the last system turn

is included in the state, the DA(s) of the last user turn become redundant. If this were the case, then we could sacrifice the DAs of the last user turn and so decrease the size of the state space, thus making learning faster/more tractable.

7.5.1 Hypothesis: The last user turn is more important than the last system turn

Given that slot-status features are already represented in the state, adding the DAs of the last user turn will produce a greater improvement in the learned strategy than adding the DA of the last system turn.

7.5.2 Methodology

In order to test the above hypothesis, we learn a new strategy, the "SysDA Strategy", for which the following features are represented in the state:

1. a slot-status feature for each slot with the values "empty", "filled" and "confirmed",
2. the DA of the last system turn.

Clearly, if the SysDA Strategy fails to outperform the DA1 Strategy, then this suggests that the hypothesis is correct. It will also be interesting to see how the SysDA Strategy's performance compares to that of the DA2 Strategy.

We now describe the results.

7.5.3 Results

The results in Table 7.9 show that the SysDA strategy significantly outperforms the Slot-Status Strategy, but is significantly outperformed by both the DA1 and DA2 Strategies. The fact that it fails to perform as well as the DA1 Strategy means that the hypothesis is true.

We now provide analysis in order to explain the performance of the SysDA Strategy as compared to the Slot-Status, DA1 and DA2 Strategies.

	Features	Av. Score	Conf. Slots	Length
$4 \rightarrow 5$ **gram = (a)**				
Slot-Status (a)	Slots-Status	47.57	100	10.49
SysDA (a)	+ Last Sys DA	56.36**	100	8.73
DA1 (a)	+ Last User DA(s)	58.00	100	8.40
DA2 (a)	+ Last Sys & User DAs	59.14	100	8.17
$5 \rightarrow 4$ **gram = (b)**				
Slot-Status (b)	Slots-Status	55.59	100	8.88
SysDA (b)	+ Last Sys DA	56.92*	100	8.62
DA1 (b)	+ Last User DA(s)	59.12	100	8.18
DA2 (b)	+ Last Sys & User DAs	59.77	100	8.05
Slot-Status (av)	Slots-Status	51.58	100	9.68
SysDA (av)	+ Last Sys DA	56.64**	100	8.67
DA1 (av)	+ Last User DAs	58.56	100	8.29
DA2 (av)	+ Last Sys & User DAs	59.46	100	8.11

Table 7.9: Testing the learned strategies after 50000 training dialogues, average reward achieved per dialogue over 1000 test dialogues. a = strategy trained using 4-gram and tested with 5-gram; b = strategy trained with 5-gram and tested with 4-gram; av = average; * Better than Slot-Status $(p < 0.05)$ but worse than DA1 $(p < 0.05)$; ** Better than Slot-Status $(p < 0.005)$ but worse than DA1 $(p < 0.05)$.

7.5.3.1 Analysis

Let us first consider why it is that the SysDA Strategy outperforms the Slot-Status Strategy. As per-usual, we find significant behavioural differences in Slot-Status Features Unchanged (SSFU) states. Whereas the Slot-Status Strategy almost always uses the *repeat* repair strategy in SSFU states, (see Figure 5.5), the SysDA Strategy is more likely to use alternatives such as *switch focus* and *give help*. As we know, this can prove more effective at getting the dialogue "back-on-track", and so reduce the number of turns required to fill and confirm all of the slots. Below are examples of the SysDA Strategy using these alternative repair strategies. In Table 7.10, the SysDA Strategy switches focus between slots 2 and 1, and in Table 7.11, it gives help and then

repeats its previous action.

SysDA state representation	**System DA**	**User DA(s)**
[empty,empty,filled,askSlot1]	icSlot3_askSlot2	yes(slot3)
[empty,empty,confirmed,icSlot3_askSlot2]	askSlot1	[out-of-domain]
[empty,empty,confirmed,askSlot1]	askSlot2	[asr_rejection]
[empty,empty,confirmed,askSlot2]	askSlot1	slot1

Table 7.10: An example dialogue snippet where the SysDA Strategy switches focus between slots 2 and 1; ic = "implicit confirm".

SysDA state representation	**System DA**	**User DA(s)**
[confirmed,empty,empty,icSlot1_askSlot2]	askSlot3	[out-of-domain]
[confirmed,empty,empty,askSlot3]	giveHelp	[asr_ rejection]
[confirmed,empty,empty,giveHelp]	askSlot3	[out-of-domain]
[confirmed,empty,empty,askSlot3]	giveHelp	[asr_rejection]
[confirmed,empty,empty,giveHelp]	askSlot3	slot3

Table 7.11: An example dialogue snippet where the SysDA Strategy uses the give help function and then repeats the original question; ic = "implicit confirm".

That the SysDA Strategy is significantly outperformed by the DA1 and DA2 Strategies can again be explained by behavioural differences in SSFU states. We now provide examples of behavioural differences in SSFU states, and start by referring again to Table 7.10, and comparing the DA2 Strategy's action choices in the contexts represented by the SysDA state in the third row, (an SSFU state). Whereas the SysDA Strategy obviously always switches focus from slot 1 to 2, if the last user response is recognised as out-of-domain, then the DA2 Strategy will also *switch focus*, but if it is an ASR rejection, then it will instead use the *repeat* repair strategy by re-asking slot 1. Table 7.12 shows the relevant state-action pairs for a second example of behavioural differences between the SysDA and DA2 Strategies in SSFU states. Here we see that at the start of a dialogue in which the first user utterance(s) fails/fail to fill a slot-value, the SysDA Strategy will always use the *repeat* repair strategy. The DA2 Strategy will most often use the *repeat* repair strategy as well, but on occasion, when the last user DA is "yes", it will *give help*.

Finally, Table 7.13 shows example dialogue snippets conducted using the SysDA Strategy and the DA2 Strategy. Slots 1 and 3 have already been filled and confirmed, and

DA2 state representation	**DA2 Strategy**	**SysDA Strategy**
[empty,empty,empty,askSlot1,user(out-of-domain)]	askSlot1	askSlot1
[empty,empty,empty,askSlot1,user(asr_rejection)]	askSlot1	askSlot1
[empty,empty,empty,askSlot1,user(no)]	askSlot1	askSlot1
[empty,empty,empty,askSlot1,user(yes)]	giveHelp	askSlot1

Table 7.12: The table shows learned actions in different states for the DA2 and SysDA strategies (trained with the 5-gram simulation). For the SysDA strategy, the state representation is always the same i.e. "[empty,empty,empty,askSlot1]", and hence the learned action is always the same.

all that remains is to confirm slot 2. The SysDA Strategy struggles because the user is supplying DAs other than "user(slot2)" in response to the action "askSlot2". Ideally the Dialogue Manager (DM) is trying to produce the following sequence of DAs " askSlot2,user(slot2),expConfSlot2", because this is the sequence which maximises the probability of the next user DA being "yes(slot2)". This would of course change the Slot 2 status feature to "confirmed", meaning that all of the slots were "confirmed", and the DM could query the database and obtain $+100$ reward. However, without knowledge of the last user DA(s), the SysDA Strategy does not know whether the last user response was "user(slot2)" or something else e.g. "user(out-of-domain)", which would make the current state an SSFU state. Since the most common user response in training in these situations was "user(slot2)", the reinforcement learner has learned to assume that the last user response was "user(slot2)", and so the strategy alternates between "askSlot2", and "expConfSlot2" on consecutive turns. The DA2 Strategy is sensitive to the last user DA(s), and so behaves differently and is more successful in these situations. In the example dialogue snippet, the DA2 Strategy only goes on to attempt to explicitly confirm slot 2 when the previous user response is "slot2", and this allows it to confirm the final slot in fewer turns on average, producing higher reward.

To summarise then, the examples presented here seem to show that the last user DAs play a crucial role in identifying SSFU states, and also in determining which repair strategy is likely to be most effective, In addition, they show that the slot-status features and last system DA cannot be used to reliably infer the last user DAs.

Speaker	SysDA Strategy	DA2 Strategy
System:	expConfSlot2	expConfSlot2
User:	out-of-domain	out-of-domain
System:	askSlot2	askSlot2
User:	out-of-domain	out-of-domain
System:	expConfSlot2	askSlot2
User:	out-of-domain	slot2
System:	askSlot2	expConfSlot2
User:	slot2	yes(slot2)
System:	expConfSlot2	dbQuery
User:	yes(slot2)	
System:	dbQuery	

Table 7.13: At the start of this dialogue snippet, the status of the first slot is “confirmed”, the second, “filled”, and the third “confirmed”. Without an explicit representation of the last user DA(s), the SysDA Strategy struggles to confirm the second slot.

7.5.4 Conclusion: The last user turn is more important than the last system turn

The hypothesis was correct - assuming that the slot-status features are already present in the state, adding the last system DA produces a significant improvement in the learned strategy, but not as great as the improvement produced by adding the DA(s) of the last user turn. As usual, test performance differences between strategies are caused by different repair strategies in SSFU states - the SysDA Strategy’s SSFU repair strategies are more effective in general than those of the Slot-Status Strategy, but less so than those of the DA1 and DA2 Strategies. Hence if we do away with the DAs of the last user turn, then we will obtain inferior repair strategies in SSFU states. The DAs of the last user turn are important for identifying SSFU states and for choosing a repair strategy, and they cannot always be reliably inferred from the last system DA and the slot-status features.

This was the last experiment which we describe here, and so we now move on to the review of relevant previous research on repair strategies for Spoken Dialogue Systems

(SDSs). Apart from describing this previous research, we also compare its findings with our own.

7.6 Previous research on repair strategies for non-understanding errors

7.6.1 Introduction

In this section, we review recent previous research on repair strategies for non- understanding errors in human-machine dialogue. This includes Skantze (2003)/Chapter 4 of Skantze (2007) and Bohus and Rudnicky (2005). Recall that a *non-understanding error* is defined as a user utterance for which the system has failed to obtain any interpretation, as opposed to a *misunderstanding error*, which is where the system obtains an incorrect interpretation. We review this work because of its relevance to this thesis - we have found that adding recent Dialogue Acts (DAs) to the state representation produces better repair strategies for states in which the Slot-Status Features are Unchanged, (SSFU states), and non-understanding errors are a major cause of SSFU states, (see Figure 5.6). Since neither Skantze nor Bohus and Rudnicky use Reinforcement Learning (RL) or user simulations, their experimental methodology is quite different to ours. We describe the experimental methodology and findings of Skantze and Bohus and Rudnicky in Sections 7.6.2 and 7.6.3 respectively. In Section 7.6.4, we then draw comparisons between the findings of Skantze and Bohus and Rudnicky, and those of the experimental work of this thesis.

7.6.2 Exploring human non-understanding error repair strategies: Skantze 2003

Skantze (2003) describes an experiment which explores non-understanding recovery in human-human dialogues with a view to drawing lessons for Spoken Dialogue Systems (SDSs). Hence a speech recogniser is used to introduce errors into the dialogues and the general experimental setup is designed to make the results more transferable to human-computer dialogue.

In the next section, we describe the methodology, and in the section after that, the

results.

7.6.2.1 Methodology

In the experiment described in Skantze (2003), pairs of humans are given the task of speaking to each other, where one takes on the role of the system (referred to as the "operator"), and the other, the role of the "user". Since they are sitting in different rooms, the operator and user cannot directly hear one another's utterances. The user speaks into a speech recogniser, and the operator's only knowledge about the user's utterances comes from the speech recognition results as displayed on a screen. The user hears distorted versions of the operator's utterances - a vocoder does the distorting. The user and operator do not meet before or during the experiment, and both are fully informed about the experimental setup.

Why then did Skantze use this basic experimental setup? First of all, the use of a speech recogniser produces the kinds of errors which occur in human-computer dialogues, and so enables Skantze to investigate how humans try to recover from them. Since the intention was for the results to be relevant to SDSs, it was necessary to make the subjects' dialogues like possible human-computer dialogues, and for this, the speech recogniser is very important. This is because the different properties of human-computer and human-human dialogues (Fraser and Gilbert, 1991) arise from the limitations that real SDSs impose on the dialogue, and speech recognition is regarded as the biggest of these limitations in most complex SDSs. The reason for the subjects not meeting, and for the distortion of the operator's utterances is to make it harder for the subjects to form assumptions about one another and so establish common ground. This is desirable because lack of common ground in human-computer dialogues is another of the limitations which makes them different from human-human dialogues.

Skantze compares his experimental setup to the "Wizard-Of-Oz" method. To design SDSs that can handle the various situations that occur in human-computer dialogue e.g. miscommunication, data of such interactions must first be collected, and the Wizard-Of-Oz method is commonly used to collect such data before an SDS is actually built. Here the operator (the "wizard") simulates parts of the system, but the user is told that they are speaking to a computer - it is assumed that users behave differently if they think they are interacting with a machine (Dahlback et al., 1993). Skantze preferred not to tell the user that they were talking to a machine, and hence use the "Wizard-Of-Oz"

method. He was concerned that the operator's behaviour might be affected so that the collected data was based on a priori assumptions about the users' behaviour, and how a system is supposed to react to them. Hence it might not cover other unanticipated interaction patterns. To further support his decision not to use Wizard-of-Oz, he refers to Amalberti et al. (1993). Amalberti et al. obtained results which suggest that at least after an initial stage, a user's linguistic behaviour is less affected by their conceptions about the speaker e.g. whether they are speaking with a machine, than it is by their experience of the interaction.

We move on now from describing the justification for Skantze's basic experimental setup and give other details about the methodology. Firstly, the domain selected for the experiment is pedestrian navigation on a simulated campus, and the user converses with the system, (the human operator), in order to get directions and find their way to a specific location. To know where the user is, the system (operator) relies on the user's description of their surroundings. There are 16 subjects - 8 users and 8 operators, ranging in age from 16 to 42. Five different scenarios are given to each pair of subjects, and so there are 40 dialogues in total. For the speech recognition results displayed on the screen, a colouring scale is used in order to efficiently provide the operator with information about the confidence scores of the words. Words that have higher confidence scores are coloured in darker tones, while words that have lower confidence scores are coloured in lighter tones. In order to facilitate turn-taking, an indicator on the screen tells the operator whether the user is speaking.

After each scenario, the subjects filled out a questionnaire about the interaction. The questionnaires consisted of a number of statements, and for each there was a choice of seven levels of agreement ranging from "strongly disagree" to "strongly agree". One of the statements on the users' questionnaire was "we did well in solving the task". Skantze conducted a PARADISE-style multiple regression (see Section 3.4) in order to see whether various objective measures of the dialogue were good predictors for this. After the whole experiment, both the user and operator were interviewed. For the users, the questions mainly concerned how well they thought that they had been understood, and if they had understood the vocoder.

In order to investigate the kinds of recovery strategies used by the operators, and how successful they were, each user utterance was also annotated with regard to how well it was immediately "understood" by the operator. "Understood" here means that the operator continued the dialogue with one interpretation, knowing that it may turn out to

be incorrect. To estimate the level of understanding, the speech recognition result and the operator's reaction to the utterance were considered. The degree of understanding was classified into the four categories presented in Table 7.14.

Label	Meaning
FULL	Full understanding: The full intention of the utterance was understood.
PARTIAL	Partial understanding: Only a part of the full intention was understood.
NON	Non-understanding: No part or fragment of the intended message (with the possible exception of a single vague word) was understood.
MIS	Misunderstanding: The operator continued with an interpretation that was not in line with the user's intention.

Table 7.14: Skantze (2003)'s annotation scheme for degrees of understanding for each user utterance.

We now move on to describe the results.

7.6.2.2 Results

The 40 dialogues contained 736 user utterances (18.4 per dialogue on average), and the mean utterance length was 6.7 words. There were a lot of errors in the recognition results - the Word Error Rate (WER) was about 40%. This was expected due to the users' unrestricted speech, and the fact that the bigram language model used was limited: 250 training utterances with a vocabulary of 350 words and 19 classes. 7.3% of the words were out of vocabulary. There was a large variation in the different operators' understanding i.e. the proportion of user utterances that were *Full understandings* versus *Partial understandings* versus *Non-understandings* versus *Misunderstandings*. The average breakdown was approximately 55% *Full understandings*, 20% *Partial understandings*, 23% *Non-understandings*, and 2% *Misunderstandings*. Hence, very few of the user utterances were *Misunderstandings*. This means that when misrecognitions occurred, the operators were very good at deciding which words were correct, and which were not. When there were a lot of misrecognitions, this resulted in *Partial understanding* or *Non-understanding*, instead of *Misunderstanding*. Thus the operators were very good at error detection. The proportion of full understandings might seem high given the high WER, but note that all words do not have to be correctly recognised for full understanding, and moreover, the WER was not equally distributed

between utterances - some had very low WER and some very high. Despite the numerous non-understandings, users reported in the post-dialogue interview that they were almost always understood. Skantze (2003) also found that the subjects seemed to improve at the task over the five sessions - the proportion of *Non-understandings* and the number of user utterances, (a measure of the length of the dialogue), both decreased after subsequent sessions (one-way repeated measures Analysis Of Variance ANOVA [2]; $p < 0.05$).

Interestingly, the analysis of the operators' error recovery strategies revealed that the operators did not routinely signal non-understanding when faced with incomprehensible speech recognition results i.e. instead of the *SignalNonUnderstanding* repair strategy, they often preferred to use *AssertRoute*, (state the route which the user should be following), or *RequestPosition*, (ask the user for their current position). It was also found that there were significantly less *Non-understandings* and significantly more *Partial Understandings* following *RequestPositions* than there were following *SignalNonUnderstandings* (χ^2 Goodness-of-fit test[3]: degrees of freedom [4] $= 3$; $\chi^2 = 12.52$; $p < 0.01$). i.e. in general, *RequestPosition* was better at getting the dialogue "back-on-track", and hence was a more effective repair strategy than *SignalNonUnderstanding*. No deviation from the general distribution was found after *AssertRoute* and *SignalNonUnderstanding*. Skantze states that the better understanding of utterances following *RequestPosition* was probably explained by the fact that they constrain the vocabulary and syntax of the response to the domain and the language models, thus increasing speech recognition performance. However, it is also noted that non-understanding may often lead to error spirals, where the user just repeats the non-understood utterance and perhaps starts to hyperarticulate, which is likely to worsen the speech recognition performance, (see Section 6.5). If, as we would expect, *SignalNonUnderstanding* is more likely to produce such error spirals, this could be another reason for its worse performance.

Skantze then goes on to report the results for the PARADISE-style multiple regression analysis which investigates which objective dialogue measures were good pre-

[2]One-way repeated measures ANOVA, (see page 127 of Greene and D'Oliveira 2001), compares how a within-subjects experimental group performs in three or more conditions. It assumes that the differences between matched values are Gaussian and compares whether the mean in any of the conditions differs significantly from the aggregate mean.

[3]The χ^2 Goodness-Of-Fit Test, (see page 257 of Sheskin 2007), uses the χ^2 distribution to test the null hypothesis that the frequency distribution of certain events observed in a sample is consistent with a particular theoretical distribution.

[4]The number of degrees of freedom is equal to the number of possible outcomes minus 1.

dictors of task success. It was found that the only factors that contributed were *time for task completion* and the *number of non-understandings that the operator had signalled* (both had a negative effect). It is interesting then that neither the *number of non-understandings* nor the *WER* per se had any effect on the user's experience, but only cases where the user was made aware of the non-understanding.

Here now, we summarise the experiment's main findings. Firstly, the results showed that the operators did not routinely signal non-understanding when faced with incomprehensible speech recognition results. Instead they tried to ask task-related questions that confirmed their hypothesis about the user's position. This strategy led to fewer non-understandings of the subsequent user utterance, and thus to a faster recovery from the problem. When they did signal non-understanding, this had a negative effect on the user's experience of task success. Despite the numerous non-understandings, users reported that they were almost always understood.

We now move on to describe Bohus and Rudnicky (2005). There is common ground with Skantze (2003) in terms of results, but differences in the basic experimental setup and the scope of the experimental aims.

7.6.3 Using the Wizard-of-Oz method to study non-understanding error repair strategies: Bohus and Rudnicky 2005

This section describes Bohus and Rudnicky (2005), which presents an empirical analysis of non-understanding errors and ten non-understanding repair strategies. There is common ground with Skantze (2003) because both examine the performance of different non-understanding repair strategies, and as we will see, their basic findings with respect to this are very similar. However Bohus and Rudnicky also consider other issues relevant to non-understanding error repair strategies, and uses a different kind of experimental setup. In contexts in which there has been no non-understanding error, Bohus and Rudnicky's system employs a fixed dialogue strategy, and then in contexts in which there has been, the Wizard-Of-Oz method is used in order to investigate whether the performance of non-understanding error repair strategies can be improved by engaging them at certain times. Given results which suggest that this is possible, Bohus and Rudnicky apply supervised learning to the experimental data in order to try to learn such a policy. Prior to all this, Bohus and Rudnicky also examine the source of non-understanding errors and their impact on dialogue performance.

We start now by describing Bohus and Rudnicky's experimental methodology in greater detail.

7.6.3.1 Methodology

Bohus and Rudnicky (2005) collected data based on interactions between 46 real users and an information slot-filling Spoken Dialogue System (SDS) called "RoomLine" (Bohus, 2003), which is an SDS for making conference room reservations. RoomLine is a phone-based mixed-initiative system which has access to live information about schedules and characteristics (e.g. size, location, Audio/Visual equipment) of 13 conference rooms in two different buildings. To make a room reservation, the system finds the list of available rooms that satisfy an initial set of user-specified constraints/slot-values, and engages in a follow-up negotiation dialogue to present this information to the user and identify which room best matches their needs. The system uses two parallel SPHINX-II Automatic Speech Recognition (ASR) engines, configured with telephone-based acoustic models[5] and a trigram statistical language model[6]. The resulting top hypothesis from each engine is parsed using the Phoenix robust parser (Ward 1994). Subsequently, semantic confidence scores are computed for each hypothesis and the winning hypothesis is forwarded to the RavenClaw-based Dialogue Manager (DM) (Bohus and Rudnicky, 2003). For output, the system uses a *template-based* Natural Language Generation (NLG) module and the Theta Text-To-Speech (TTS) synthesizer (Cepstral 2004).

In the experiment described in Bohus and Rudnicky (2005), the real users interacted with RoomLine under two different conditions - the *wizard condition*, and the *control condition*. Under both of these conditions, all aspects of the system, (including a fixed dialogue strategy), were identical apart from when there was a non-understanding error, at which point there was a choice of ten possible non-understanding error repair strategies. Under the wizard condition, the Wizard-Of-Oz method was used to determine which repair strategy to engage i.e. although the user is under the impression that there is no human involvement, it is in fact a human operator (the "wizard", in this case the first author), who makes the decision. By contrast, under the control con-

[5] An acoustic model is used by a speech recogniser to recognise speech. It is created by taking audio recordings of speech, and their text transcriptions, and using software to create statistical representations of the sounds that make up each word.

[6] A statistical language model assigns a probability to a sequence of words by means of a probability distribution.

dition, the choice of which repair strategy to engage was random. The ten possible non-understanding error repair strategies were:

- AskRepeat(AREP) “Can you please repeat that?”,
- AskRephrase(ARPH) “Can you please try to rephrase that?”,
- Reprompt(RP) “Would you like a small room or a large room?”,
- DetailedRePrompt(DRP) “I’m sorry, I don’t think I understood you correctly. Right now I’m trying to find out if you would prefer a small room or a large one.”,
- Notify(NTFY) “Sorry, I’m not sure I understood what you said...”,
- Yield(YLD) [the system remains silent, and thus implicitly notifies the user that a non-understanding has occurred],
- MoveOn(MOVE),
- YouCanSay(YCS) [the system tell the user what he or she can say at this point in the dialogue],
- TerseYouCanSay [a terser version of the YouCanSay strategy],
- FullHelp(HELP) [the system provides a longer help message which includes an explanation of the current state of the dialogue].

In making his choice, the wizard had live access to the user’s speech and several other system state variables via a Graphical User Interface (GUI) (e.g. ASR result, confidence score, semantic parse). The wizard informs the system of his choice via the GUI. Based on the collected data, Bohus and Rudnicky then derived results for the *source of understanding errors*, the *impact of non-understandings on dialogue performance*, *performance of non-understanding repair strategies*, *user responses to repair strategies*, *control versus wizard*, and *learning a policy*.

We now move on to describe Bohus and Rudnicky’s results.

7.6.3.2 Results

In this section, we describe Bohus and Rudnicky (2005)'s results, starting with the source of non-understanding errors and their impact on dialogue performance.

- **The source of non-understanding errors and their impact on dialogue performance:** Bohus and Rudnicky first report that the majority of errors are seen to originate at the Signal (i.e. speech recognition level) but at the same time, a large number of non-understandings, and a smaller but still significant number of misunderstandings are caused by either out-of-application utterances (outside of the application's functionality), or out-of-grammar utterances (within the domain and scope of the application, but outside of the system's grammar). The next results concern the impact of non-understanding errors on overall dialogue performance. These results were obtained by using a logistic regression model [7] to assess how well the frequency of non-understandings and misunderstandings predicted task success - hence the independent variables were the frequency of non-understandings in a session, and the frequency of misunderstandings, and the dependent variable was a binary task success indicator. Each data-point corresponded to an entire dialogue session. Adding the frequency of non-understandings to the model increased the average data log-likelihood from the majority baseline of -0.5200 to -0.4306 ($p < 10^{-4}$ in a likelihood ratio test [8]). Then, adding the frequency of misunderstandings to the model further increased the average log-likelihood of the data to -0.2795 ($p < 10^{-4}$).

  We now move on to describe Bohus and Rudnicky's results which relate to the performance of the different non-understanding error repair strategies.

- **The performance of non-understanding error repair strategies:**
  Bohus and Rudnicky report that according to logistic ANOVA, the mean recovery rates of the 10 strategies can be divided into three tiers, where there are statistically significant differences between tiers, but not within. The top tier contains the *MoveOn*, *Help* and *TerseYouCanSay* strategies, the second tier contains *Reprompt*, *YouCanSay* and *AskRephrase*, and the bottom tier contains *Detaile-*

[7] Logistic regression, (see page 1581 of Sheskin 2007), is a model used for prediction of the probability of an event. It is therefore a form of regression that is used when the dependent variable is dichotomous (or binary). It can use several predictor variables (either numerical or categorical).

[8] A likelihood-ratio test, (see Sheskin 2007), is a statistical test for making a decision between two hypotheses based on the ratio of the maximum probability of a result under these two hypotheses.

dReprompt, Notify, AskRepeat and *Yield*. The *AskRepeat* recovery strategy was seen to induce the largest number of *rephrase responses* (64%), the *MoveOn* strategy, the largest number of *change responses* (52%), and the *AskRephrase* and *Notify* strategies, the largest number of *rephrase responses* (64%). Further analysis showed that the best recovery performances were attained by *change responses*, which is what we would expect given that the *MoveOn* and *Help* strategies performed the best, and that these both induced a large number of *change responses*.

We now move on to describe Bohus and Rudnicky's results which relate to the comparison of the system's performance under the wizard versus the control condition. Here Bohus and Rudnicky were investigating the hypothesis that the performance of the non-understanding error repair strategies can be improved by engaging them at more appropriate times.

- **Wizard versus control:** The impact of the wizard policy was assessed both on overall dialogue performance and on local non-understanding recovery performance metrics. The wizard policy does lead to statistically significant performance improvements on a number of metrics, but the improvements appear mostly within the non-native population i.e. in the group of users that had more difficulties using the system. For example, while no task success improvement can be detected for native users, there is a large task success improvement for non-native users (31.6 to 57.4%). Despite this increase in task success rate, no statistically significant differences can be detected with respect to user satisfaction - Bohus and Rudnicky state that the small number of samples and the large variance of this metric lead to wide confidence bounds on the mean estimates and preclude reliable comparison. Statistically significant improvements for the non-native users were observed again on the local recovery performance metrics - *recovery Word Error Rate (WER), recovery concept utility* and *recovery efficiency*. *Recovery WER* is the average WER for the user turns following non-understanding recovery attempts, *recovery concept utility* measures the average number of concepts correctly acquired by the system from the user's response, and *recovery efficiency* extends *recovery concept utility* by normalising for the amount of time spent during recovery. Before moving on, we should note here that while Bohus and Rudnicky seem to have shown a policy that improves the performance of non-understanding error repair strategies by engaging them at

more appropriate times, this policy cannot be used by a system. This is because the wizard chose which repair strategy to engage based on the full audio signal (i.e. human-level speech recognition), and hence used information which the system would not have access to at runtime.

Looking at the individual repair strategies, a statistically significant improvement was detected only for *AskRepeat*. Bohus and Rudnicky state that not gaining statistically significant improvements for the other nine repair strategies can potentially be explained by the fact that the wizard's decision task was quite difficult, even with access to the full audio signal. To maintain the illusion that the users were interacting with an autonomous system the wizard had to choose one of ten repair strategies in a very short time interval: 1 or 2 seconds. This selection task is easier for some of the strategies than for others, and as a result, a number of repair strategies such as *YouCanSay*, *Reprompt*, and *DetailedReprompt* were very rarely engaged by the wizard and so the confidence intervals on their performance estimates are very wide and preclude accurate comparisons. We will return to this issue in Section 7.6.4 where we compare the findings of Skantze (2003) and Bohus and Rudnicky (2005) to those of this thesis.

We now move on to describe Bohus and Rudnicky's results which relate to trying to use the collected data to learn a policy for engaging the non-understanding repair strategies.

- **Learning a policy from data:** Having established that the performance of the non-understanding repair strategies can be improved by a policy which engages them at the right time, Bohus and Rudnicky attempt to learn such a policy from data. For each repair strategy, a logistic regression model is fitted to training data in order to predict whether that repair strategy will be successful based on various contextual features. Here, successful means that "the next user turn is correctly understood by the system." The training data for each model is derived from the turns in the uninformed dialogues in which a non-understanding occurred and the strategy of interest was engaged. The contextual features used include:

  - features from the speech recognition level - the number of words, the signal and noise levels, the number of and proportion of words tagged as unconfident by the speech recogniser,
  - features from the language understanding level - features reflecting the

quality of the parse,

– features from the dialogue management level - information about the dialogue state, history of the dialogue up to that point (e.g. how many previous consecutive non-understandings have been encountered, the average confidence score so far).

Two policies are learned, the first of which, *max-recovery-rate*, aims to maximize the recovery rate by choosing the repair strategy with the maximum likelihood of success, while the second, *max-recovery-efficiency*, aims to maximize the recovery efficiency. A preliminary estimate of the performance of these policies is obtained by looking at what happened in the data for the wizard condition when the wizard happened to make the same decisions as the learned policy. While the wizard's overall recovery rate was 50.1%, within the subset of instances where the wizard made the same decision as the *max-recovery-rate policy*, it was significantly higher at 69.8%. Similarly, on the instances where the wizard agreed with the *max-recovery-efficiency policy*, the recovery efficiency performance was 2.02, significantly higher than the overall wizard *recovery efficiency* (0.81), and the uninformed policy *recovery efficiency* (0.00).

Bohus and Rudnicky acknowledge a number of problems with the learned policies. First of all, the performance of the individual predictors is not very good due to the small number of training instances. Indeed, no information is given as to which contextual features were good predictors, and which not, and there is no qualitative analysis of the learned policies. Bohus and Rudnicky also state that there are problems with the evaluation method:

> "Given that both the wizard and the learned policy strive to maximize performance, the distribution of the subset of non-understandings where they agree might not be representative for the true distribution of non-understandings - these might be the cases where it's easier to tell which strategy should be used to recover. Ultimately, a new user study where the system runs with the learned policy is required in order to robustly evaluate its performance."

We now move on to compare the findings of Skantze (2003) and Bohus and Rudnicky (2005) with those of the experimental work of this thesis.

7.6.4 Comparison with the findings of the experimental work of this thesis

Let us start here by considering the results concerning the relative performance of different repair strategies. Recall that following non-understanding errors, Skantze (2003) and Bohus and Rudnicky (2005) found that their better performing repair strategies did something other than signal the non-understanding to the user / ask the user to repeat / re-prompt. Skantze's *RequestPosition* repair strategy significantly outperformed *SignalNonUnderstandings*, and Bohus and Rudnicky's *MoveOn* and *Help* repair strategies performed the best, significantly outperforming *AskRepeat* and *Reprompt*. Indeed Bohus and Rudnicky found *AskRepeat* to be one of the worst performing repair strategies. Bohus and Rudnicky also showed that the best performing repair strategies induced the highest number of *change responses*, and the worst performing, the highest number of *repeats*. All of this coincides with the results that we have obtained in learning dialogue strategies using stochastic user simulations whose probabilities are derived from real user data. We have found that in states in which the Slot-Status Features were Unchanged (SSFU states), (often due to non-understanding errors), rather than to *repeat*, (the equivalent of Bohus and Rudnicky's *Reprompt* repair strategy), representing recent DAs in the state causes the reinforcement learner to often learn to *switch focus*, (the equivalent of Bohus and Rudnicky's *MoveOn*), occasionally to *give help*, (the equivalent of Bohus and Rudnicky's *Help*), and sometimes to *bactrack*, (no equivalent in Skantze 2003 or Bohus and Rudnicky 2005). Engaging these new repair strategies in the SSFU states in which they had been learned enabled the Dialogue Manager (DM) to get the dialogue "back-on-track" more quickly on average, and so achieve task completion in fewer turns. Additionally, in the real user experiment of Chapter 6, overuse / misuse of our *repeat* repair strategy was shown to be potentially very destructive. Engaging this repair strategy usually caused the user to repeat themselves and as a result, they often became frustrated. They were then more likely to adopt an irritated / hyperarticulated tone which caused more ASR errors, and hence longer dialogues and less chance of task completion.

It is also worth mentioning here that the case of *Backtracking* highlights a key advantage of the Reinforcement Learning (RL) approach. The methodology used by Bohus and Rudnicky for example meant that all of the repair strategies had to be designed and pre-programmed in advance - no new effective repair strategies emerged from this work. *Backtracking* is a repair strategy that we did not initially think of, and hence

explicitly program in advance, but instead which "emerged" from the RL. It is an example of how given the freedom to do so, the reinforcement learner can take its basic action set and maybe sometimes apply actions in novel unanticipated ways. This might be achieved by combining actions. For example, we did not experiment with this, but it could have proved advantageous in certain contexts to allow combinations of our four different repair strategies in a single system turn e.g. "give help + repeat".

Let us now move on to consider Bohus and Rudnicky's results concerning the performance of the wizard versus the control policy. Like Bohus and Rudnicky here, we have obtained results to suggest that overall dialogue performance can be improved by engaging repair strategies at more appropriate times. However, recall that unlike our learned strategies, Bohus and Rudnicky's wizard policy cannot be used by a real SDS because the wizard had access to the full audio signal i.e. it assumes human-level speech recognition. As regards the individual repair strategies, recall that Bohus and Rudnicky found the wizard policy to produce a significant improvement in the performance of *AskRepeat*, (the repair strategy which was most likely to make the user repeat themselves) . This certainly makes sense given what we said in the last paragraph about the destructive effect of overuse / misuse of *repeat* in the real user experiment of Chapter 6. We should comment here on the fact that Bohus and Rudnicky were unable to detect statistically significant improvements for the other nine repair strategies. Recall that Bohus and Rudnicky suggested this was because the wizard's decision task was difficult, particularly for certain repair strategies, and as a result these were rarely engaged. This kind of problem does not occur in the RL approach used in this thesis. Use of a user simulation also makes it much easier to generate a large number of example dialogues and so obtain statistically significant results. Of course, as has been stated before, the user simulation must be sufficiently realistic if any useful insights are to be drawn based on the resulting data.

We have seen then that Skantze (2003) and Bohus and Rudnicky (2005) made useful findings with regard to the performance of different non-understanding repair strategies. However, unlike in the experimental work of this thesis, their aims and methodology meant that they did not produce a full dialogue strategy which takes advantage of their findings, and which can be implemented in a Spoken Dialogue System (SDS) and shown to work well with real users. Skantze made no attempt to learn a dialogue strategy, while Bohus and Rudnicky used Supervised Learning (SL) in order to learn a partial dialogue strategy i.e. a policy for which repair strategy to use following

non-understanding errors, but there are problems with this. These have already been mentioned in Section 7.6.3.2 and include poor performance by the individual predictors due to the small number of training instances, and problems with the evaluation method.

Apart from producing full dialogue strategies which have been shown to work well both in simulation and with real users, in this thesis, we have also made additional findings with respect to recent Dialogue Acts (DAs) and repair strategies. In our RL experiments, we have found the DAs of the last system and user turns to only be useful in Slot-Status Features Unchanged (SSFU) states - they enable the learner to identify SSFU states and are also important in choosing *which* repair strategy to apply. Any sensible repair strategy which avoids repetition will not necessarily be optimal, and so the role of the DAs in determining which repair strategy to use is more complicated than simply telling the learner what the previous system action was. We have also shown the DA(s) of the last user turn to make a greater contribution to improving the learned strategy in SSFU states than the DA of the last system turn.

We now move on to provide a summary of this chapter.

7.7 Chapter summary

In this chapter we tested a number of hypotheses which were formed based on the results and analysis of the experiments of Chapters 5 and 6. The first experiment described in this chapter successfully tested the hypothesis that the Dialogue Acts (DAs) of the last system and user turns are only improving the learned strategy with respect to better repair strategies in Slot-Status Features Unchanged (SSFU) states. The second experiment then showed that as regards these repair strategies, the DAs are useful not only for identifying SSFU states, but also in choosing *which* repair strategy to apply. There was the possibility that despite this finding, the task of choosing the appropriate repair strategy in any SSFU state is very simple because any repair strategy which avoids repeating the last system action is guaranteed to be optimal. This hypothesis was formed based on the analysis of the real user experiment in Chapter 6, where we found the Slot-Status Strategy's poor performance to be caused by overuse of repetition. However the third experiment showed this hypothesis to be false - evidently, finding the optimal repair strategy in an SSFU state is more complicated than this. The final experiment of this chapter then found that given the slot-status features are already

present in the state, adding the last system DA produces a significant improvement in the learned strategy, but not as great as the improvement produced by adding the DA(s) of the last user turn. The results and analysis showed that the SSFU repair strategies should ideally be sensitive to the DAs of the last user turn, and that these user DAs cannot always be reliably inferred from the last system DA and the slot-status features.

Having described the final experiment, we then provided a review of recent related research on repair strategies for Spoken Dialogue Systems i.e. Skantze (2003) and Bohus and Rudnicky (2005). Both Skantze and Bohus and Rudnicky investigated non-understanding error repair strategies and used a different experimental methodology to here - neither used Reinforcement Learning (RL) or user simulations, We found common ground with this previous work in regard to results concerning the performance of different repair strategies - it seems that following a non-understanding error, it is often preferable to do something other than signal the non-understanding / repeat the previous system action. Unlike this thesis, this previous work did not produce full dialogue strategies which take advantage of these findings, and which have been shown to work well with both simulated and real users. The insights provided by this thesis with respect to recent Dialogue Acts (DAs) and their role in learning better repair strategies are also novel.

Chapter 8

Summary and conclusions

8.1 Thesis summary

Designing dialogue strategies for the Dialogue Manager (DM) component of a Spoken Dialogue System (SDS) is a potentially very complicated task, and so if it is done by hand, it may involve a time-consuming test-and-refine process. A particularly important issue in dialogue strategy design, is how best to limit the number of understanding errors made by the SDS, and how best to deal with them when they occur. This is because the limitations of the input components i.e. the Automatic Speech Recognition (ASR) and Natural Language Understanding (NLU) components mean that in general, SDSs may frequently make understanding errors, more so than humans. Due to the complicated nature of designing dialogue strategies, over the past 10 years, researchers have begun investigating whether a machine learning technique called Reinforcement Learning (RL) can be successfully applied to the problem. RL algorithms are used to learn a series of *actions* to take in different *states* so as to achieve some goal specified by a *reward function*, and this makes them appropriate for dialogue strategy design - within any dialogue, our ideal dialogue strategy should choose a series of system actions in different dialogue contexts so as to maximise the chances of a successful conclusion e.g. task completion, high user satisfaction. Substantial progress has been made in the previous research, but none involved training full dialogue strategies with accurate stochastic user simulations whose probabilities are derived from real user data, and then going on to show that the resulting learned strategy works well with real users e.g. better than a hand-crafted strategy. In general, the previous researchers also included very limited contextual information in the RL state, and

little insight was gained as to which contextual features were important and why. For example, in learning dialogue strategies for information-seeking/slot-filling SDSs, previous researchers e.g. Levin and Pieraccini (1997), Scheffler and Young (2001), Singh et al. (2002), Pietquin and Renals (2002), Singh et al. (2000) have generally used only slot-based features i.e. whether or not a slot has been filled and the confidence score associated with any supplied value. At the opposite extreme, by using a Hybrid RL / Supervised Learning (SL) approach, and augmenting the RL component with a generalisation technique, Henderson et al. (2005) was able to include a very large amount of contextual information in the RL state. However again, this work gave little insight as to which contextual features are important and why, and there is the question of how the Hybrid strategy performs relative to one learned with a much smaller state space and standard RL.

This thesis has presented experimental work on these issues, and here we will make a brief summary of their contributions.

8.1.1 Preliminary proof-of-concept Reinforcement Learning experiments

Chapter 4 discussed issues related to how best to set the reinforcement learner's parameters. The points which we raised regarding which kind of Eligibility Traces (ETs) to use, and how high to set the initial Q-values parameter apply especially when relatively little contextual information is represented in the state e.g. just the slot-status features. This is because, at least during the early stages of training, the reinforcement learner may frequently take dead-end actions i.e. action which do not lead to a change in state. Assuming that only the slot-status features are represented in the state, an example occurs if the system asks for a slot value for which it already has the correct value, and then the simulation re-supplies this correct value. If the system goes on to successfully complete the dialogue and so receives a high reward, accumulating ETs will propagate back reward to the dead-end state-actions, and the greater the number of times which a dead-end state-action was visited, the greater the amount of reward that will be propagated back. As a result, we may well find the reinforcement learner learns dead-end state actions. To guard against this, we prefer to use replacing ETs which whenever the state changes, set the trace for any dead-end actions which were taken in the previous state to zero. However, given that we are using these replacing

ETs, we must be careful not to set the initial Q-values too high else we are still likely to learn dead-end actions. This is because a large proportion of early training dialogues end in low reward, and such ETs propagate this low reward back only to the non-dead-end actions. Hence if the initial Q-values are set too high, then the dead-end action's Q-value will remain high while that of the non-dead end action will be greatly reduced. Then, due to it's relatively high Q-value, assuming we are using ε-greedy or softmax action selection, the reinforcement learner will be very likely to continue selecting the dead-end action when it re-visits this state. This may be corrected in time, but learning will certainly be significantly slowed as a result.

Moving on to the experiments of Chapter 4 themselves, in the analysis of our first experiment, we described possible circumstances under which representing the user's last Dialogue Act (DA) in the state would be important for learning to deal with user indications of misunderstanding errors. If users prefer their indications of misunderstanding errors to be immediately addressed i.e. for the system to focus on the problem slot, (which seems like a reasonable assumption), then clearly if the problem slot cannot be inferred from the slot-status features alone, the reinforcement learner requires the user's last DA. Such ambiguous states could occur if the system has switched focus between slots earlier in the dialogue (e.g. as a repair strategy following a non-understanding error), or if the user does not indicate the misunderstanding error immediately following the incorrect confirmation, but waits until a later turn.

The second experiment demonstrated a first simple case in which adding another feature to the slot-status variables in the state enables the reinforcement learner to learn a more effective strategy, in this case for a 2-slot system. Previous researchers who have used RL to learn full strategies have included only slot-status information in the state e.g. Levin and Pieraccini (1997), Levin et al. (1998), Pietquin and Renals (2002). Training and testing is conducted using a Bayesian Network user simulation, the probabilities for which are supplied using intuition, and the feature added to the state is the DA of the last user turn. Adding this feature enables the learner to learn to give help when asked. The user simulation is impatient and if it asks for help and does not receive help in the next system turn, there is a 90% chance that it will hang-up (end the dialogue). According to a reward function that gives $+100$ for a correct database query, (0 otherwise) and -1 for each system turn, adding the Dialogue Act (DA) of the last user turn produces a 52% ($p < 0.05$) increase in average reward in testing

In the third and final experiment, we went on to find that if the user's last DA is rep-

resented in the state, the RL problem remains tractable if we scale up from 2 to a more commercially realistic 4 slots. We also found that an "all-or-nothing" reward function i.e. a function that gives a large positive reward if all of the slot-values are correct, else nothing, can produce faster learning than a "partial" reward function i.e a function that gives a small positive reward for each correct slot-value. A "mixed" reward function which gave very small positive reward for each correct slot-value, and a large positive reward if all of the slot-values were correct i.e. a mix of the other two reward functions produced faster learning than the "partial" reward function, but slower than the "all-or-nothing" reward function. These findings have important implications for how best to design reward functions for learning dialogue strategies. The "all-or-nothing" reward function worked well here because the system did not have to deal with unobtainable slot-values - this might happen because the system cannot recognise the user's accent when trying to communicate the slot-value. If unobtainable slot values is a problem which real SDSs have to deal with, then we should attempt to learn strategies that can cope. This would mean that we should train with a user simulation that is capable of simulating unobtainable slot-values, Assuming we train with a user simulation that does simulate unobtainable slot-values, then an all-or-nothing reward function is clearly not ideal. Thinking about this scenario led us to propose an "as-good-as-possible" reward function which provides a large positive reward as long as all of the obtainable slots are filled, otherwise nothing. This reward function is an all-or-nothing reward function and so will help to speed up learning, but it will not penalise the reinforcement learner for failing to fill and confirm an unobtainable slot. To use such a reward function, in each dialogue, the user simulation would have to tell the reinforcement learner whether any of the slots were unobtainable.

8.1.2 Learning with real user data: n-gram user simulation experiments

In Chapter 5, we found that when training with accurate stochastic simulations whose probabilities are derived from real user data, including in the state the DAs of both the last system and user turns in addition to slot-status features produces strategies which perform better in testing in simulation. Note that we used a simple "all-or-nothing" training reward function based only on dialogue length and whether or not all of the slots are confirmed when the system queries the database i.e. task completion. We found that the DAs of the last system and user turns produced improvements for both

3 and 4-slot learned strategies. As stated in Section 8.1.1, past research has in general included only slot-status information in the state when using RL to learn full strategies e.g. Levin and Pieraccini (1997), Levin et al. (1998), Pietquin and Renals (2002). We were able to make a meaningful performance comparison with the 4-slot Hybrid RL/Supervised Learning (SL) Strategy of Henderson et al. (2008). This is because the Hybrid Strategy was tested with a linear function approximation user simulation derived from COMMUNICATOR data, (see Section 3.10.4), which has been shown to simulate very similar dialogues to the n-gram simulations (Georgila et al., 2006). Our learned strategies were found to achieve much higher scores in testing for dialogue length and the number of slots that are filled/confirmed when the system queries the database.

Chapter 5 described analysis of our learned strategies and found that the DAs were enabling the reinforcement learner to learn more effective repair strategies in Slot-Status Features Unchanged (SSFU) states, (most often caused by non-understanding errors i.e. ASR rejections or user utterances that are recognised as out-of-domain). These repair strategies were more effective in the sense that they were more likely to ensure that the next state was not also an SSFU state - they were better at getting the dialogue back-on-track. The four different types of repair strategy which we observed in the learned strategies in SSFU states were *repeat*, *switch focus*, *give help* and *backtrack*. Whereas the Slot-Status Strategy apparently over-used the *repeat* repair strategy, adding the recent DAs to the state caused the learned strategy to make greater use of the other three types. Qualitative analysis of the learned strategies suggested that the DAs of the last system and user turns were important for identifying SSFU states and hence that some kind of repair strategy was required, and then also in making the choice of *which* repair strategy to apply. For example, our best learned strategy was more likely to use the *repeat* repair strategy following an ASR rejection, but the *switch focus* repair strategy following a user utterance that is recognised as out-of-domain. We were unable to find what looked like any significant improvements in the learned strategy outside of SSFU states e.g. in dealing with user indications of misunderstandings, and in portions of dialogue in which there was smooth progress towards the goal of filling and confirming all of the slots. However, a general trend was noticed for the learned strategies to focus on the problem slot following a user indication of a misunderstanding error i.e. they would re-ask / attempt to re-confirm the problem slot rather than shift focus to a new slot.

We also noted that if the intention is to use the learned strategies with real users in an SDS, then the user simulation for training the strategies should be as realistic as possible. Hence, we also pointed out the limitations of the n-gram simulations. We noted that no dataset is perfect, and hence there may be problems with the COMMUNICATOR data which mean the n-gram simulations are not as realistic as they might be. In addition, we highlighted the key weakness of the n-gram model itself i.e. the fact that it is not sensitive to dialogue context from earlier than $n-1$ turns ago, (3 or 4 turns in our case depending on whether we were using the 4 or 5 gram simulation). It seems that real users must sometimes take account of contextual information from turns earlier than the last 3 or 4, and so this assumption is false at least some of the time. For example, it is responsible for the n-gram simulations' failure to simulate "impossible-to-fill" slots, - a phenomenon which did occasionally occur in the COMMUNICATOR data. It also means that if DAs are added to the state in order starting with the most recent, only those of the last $n-2$ turns can potentially produce improvements in the learned strategy, e.g. the last 3 turns in the case of a 5-gram simulation. Furthermore, the n-gram models are only sensitive to previous DAs - they are not sensitive to other features of the dialogue context. Taking account of other features of the dialogue context may produce more realistic user simulations with which we could then potentially train better dialogue strategies.

8.1.3 Testing the learned strategies on real users

Chapter 6 described how we tested two of our 3-slot learned strategies and a state-of-the-art hand-crafted strategy on real users. The first of the learned strategies which we tested was learned with only the slot-status features (the Slot-Status Strategy), while the second was learned with the slot-status features and the Dialogue Acts (DAs) of both the last system and user turns (the DA2 Strategy). The Spoken Dialogue System (SDS) which was used, the TownInfo SDS, operates in the tourist information domain and so we were required to port our learned strategies into this new domain by treating them as generic slot-filling strategies.

We found the DA2 Strategy to outperform the Slot-Status Strategy. The DA2 Strategy outperformed the Slot-Status Strategy in terms of Perceived Task Completion (PTC) (90.91% versus 80.82%), and dialogue length (7.95 turns on average for the slot-filling phase of the dialogue versus 8.98). Although the improvements were not significant,

the DA2 Strategy also performed better according to three different subjective measures - “task ease”, “expected behaviour” and “re-use”. We also found the DA2 Strategy to outperform the state-of-the-art hand-crafted strategy. The DA2 Strategy outperformed the state-of-the-art hand-crafted strategy in terms of perceived task completion (90.91% versus 80.82%), and dialogue length (7.95 turns on average for the slot-filling phase of the dialogue versus 8.46). Again, although the improvements were not significant, the DA-strategy also performed better according to three different subjective measures - “task ease”, “expected behaviour” and “re-use”.

Analysis of this real user experiment showed that the Slot-Status and state-of-the-art hand-crafted strategies over-used the repeat repair strategy in Slot-Status Features Unchanged (SSFU) states. Repeatedly asking the same question often annoyed the user, and so was very likely to elicit hyper-articulate/irritated speech. This in turn is likely to cause more speech recognition errors, and so longer dialogues and lower task completion. Hence this is an explanation for the inferior performance of the state-of-the-art hand-crafted and Slot-Status Strategies as compared to the DA2 Strategy.

Although statistical tests were impossible, direct performance comparisons were possible with the Hybrid RL/SL Strategy of Henderson et al. (2008) because this too was tested using the TownInfo Spoken Dialogue System (SDS). The DA2 Strategy achieved higher average perceived task completion (90.91% versus 80.8%) and lower dialogue length (9.65 turns on average for the whole dialogue versus 11.6). The DA2 Strategy also did better according to an overall score for the three different subjective measures: “task ease”, “expected behaviour” and “re-use”. Qualitative analysis of the Hybrid Strategy seemed to show that it was sub-optimal e.g. it made little effort to fill slots in certain circumstance and would query the database with unfilled slots, and it seemed to over-use the repeat repair strategy, the negative impact of which has just been described above. It seems the main reason for the Hybrid Strategy being sub-optimal is that it was trained with an enormous state-action space, and the optimal strategy has not yet been found - it only improves over the purely supervised strategy by 1.4%.

8.1.4 Investigating the role of Dialogue Acts in learning repair strategies

Having tested the strategies learned with the n-gram simulations both in simulation and with real users, the results and analysis led us to form a number of hypotheses

regarding how and why the recent Dialogue Acts (DAs) improved the learned strategy. Since training with the n-gram simulations had produced learned strategies which gave a positive result in the real user tests, we felt it was justified to use these simulations to train and test new strategies for testing these hypotheses.

The first hypothesis which we tested was that the DAs were only improving the learned strategy with respect to more effective repair strategies in Slot-Status Features Unchanged (SSFU) states. We obtained further supporting evidence for this hypothesis by showing that a strategy learned with only the additional DA information in SSFU states performed no better or worse than one learned with DA information in all states. Next we learned strategies with states that explicitly represented whether the current state was an SSFU state - one such strategy was learned with no DA information in the state, and another with only the DAs of the last user turn. Neither of these strategies performed as well as the DA2 Strategy and so this indicates that the DAs of the last system and user turns are important not only in identifying SSFU states and hence that a repair strategy is required, but also in then choosing *which* repair strategy to apply. Despite this, it was still possible that the task of finding the best repair strategy to apply in any state was very simple because any "sensible" avoidance of repetition was guaranteed to be optimal. Here we defined a "sensible" repair strategy to be any of the four types that emerged from our original RL experiments with the n-gram simulations i.e. *repeat*, *give help*, *switch focus* and *backtrack*. This hypothesis had been formed based on analysis of the real user experiment which showed that the Slot-Status and hand-crafted strategies performed relatively poorly because they over-used repetition in SSFU states. In order to test the hypothesis, we tested different strategies which followed one of our learned strategies except in SSFU states. Here they applied a "sensible" repair strategy which did not involve repeating the previous system action. These strategies were found to perform worse than the DA2 Strategy and so this result contradicts the hypothesis that any kind of "sensible" avoidance of repetition in an SSFU state is guaranteed to be optimal.

In a final experiment we found that a strategy learned with the slot-status features and the DA of the last system turn did not perform as well as the DA2 Strategy. Hence the DAs of the last user turn cannot be reliably inferred from the DA of the last system action and the slot-status features, and given that they are important in learning effective repair strategies in SSFU states, not representing them caused the learned strategy to deteriorate. This experiment also showed the DAs of the last user turn to be more

important than the DA of the last system action - removing the DAs of the last user turn from the state caused a greater deterioration in the learned strategy than removing the last system DA.

Having conducted these additional experiments with the n-gram simulations, we then went on to compare our findings on repair strategies with relevant previous research. This relevant previous research included Skantze (2003)/Chapter 4 of Skantze (2007) and Bohus and Rudnicky (2005) who both investigated repair strategies for non- understanding errors in human-machine dialogue. Neither Skantze nor Bohus and Rudnicky used RL or user simulations and so their experimental methodology is quite different to ours. Skantze created an environment in which human-human dialogues became much more like human-machine dialogues and then studied how his human subjects tried to recover from non-understanding errors. Bohus and Rudnicky preferred to use the Wizard-of-Oz methodology and where non-understanding errors occurred, compared the performance of choosing a repair strategy at random versus the human wizard's choice. Both Skantze and Bohus and Rudnicky's results concerning the performance of different repair strategies coincide with ours - they both found that when non-understanding errors occur, it is often better to do something other than encourage the user to repeat themselves. However, in contrast to here, they did not produce dialogue strategies which exploit these findings and which can be shown to work effectively both in simulation and with real users. The conclusions which we drew with respect to the role of recent DAs in learning better repair strategies in SSFU states are also original contributions.

8.2 Conclusions and future work

Here we consider the findings of this thesis, their limitations and the state of the research field as a whole. Based on this, we draw conclusions and suggest areas for future research according to different themes:

- **Recent Dialogue Acts can improve performance in testing in simulation and with real users through better repair strategies:** In the experimental work of this thesis, we have succeeded in producing full slot-filling dialogue strategies which work well both in simulation and with real users, and which outperform alternative approaches i.e. full dialogue strategies learned with a state

containing only slot-status features (e.g. Pietquin and Renals 2002), the Hybrid RL/Supervised Learning (SL) Strategy of Henderson et al. (2008), and a state-of-the-art hand-crafted strategy. We have found that by including recent Dialogue Acts (DAs) in the state in addition to the slot-status features, we are able to learn better repair strategies for Slot-Status Features Unchanged (SSFU) states - dialogue contexts in which progress has stalled due to communication problems e.g. non-understanding errors. This is an encouraging and useful finding because the system's behaviour in such dialogue contexts can be critical to the dialogue's overall success or failure. Once the dialogue enters such a context, if the system fails to quickly get the dialogue back-on-track, then the user can soon become very frustrated. The user is likely to adopt an irritated tone / hyperarticulate, which will then cause more Automatic Speech Recognition (ASR) errors, less chance of task completion and lower user satisfaction. Since the kind of problem contexts which we are discussing here are a feature of human-machine dialogue in general, this finding that recent DAs can be used to learn better repair strategies should translate to other domains. Indeed, like Lemon et al. (2006a), our real user experiment showed that a strategy learned in one slot-filling domain (flight-booking) can work well in another (tourist information). Here we mapped between slots in the two domains based on their number (see Section 6.3.3), but in the future, dialogue strategy designers may find it more profitable to map between slots based on how likely it is that a user's value will be correctly recognised - recall that Pietquin and Renals (2002) learned a strategy which asks the slots in an order that is sensitive to Word Error Rate (WER) (see Section 3.5).

- **The time and effort involved in developing accurate user simulations:** Our learned strategies may have worked well with simulated and real users, but we should now also say something about the amount of time and effort which is required to apply the RL approach advocated in this thesis. This very much depends on whether data collection is required. In order to apply the approach of this thesis, we first need data of dialogues between real users and Spoken Dialogue Systems (SDSs) which use different dialogue strategies. From this data, we can then derive probabilities for accurate stochastic user simulations, which in turn are used to interact with a Dialogue Manager (DM) and reinforcement learner over a number of training dialogues in order to learn dialogue strategies.

The probabilities for the user simulations used in Chapters 5 to 7 were derived from an existing data source, (the COMMUNICATOR data Walker et al. 2001a), but if the design team of a commercial system decided that this was inadequate for their needs, then collecting their own new data could obviously be very time-consuming. Depending on time and cost constraints, and the complexity of the domain, the designers may decide that this is not justifiable and hand-craft their dialogue strategies instead. We should mention again though, the finding made in Lemon et al. (2006a) and repeated here in this thesis, that a strategy learned in one slot-filling domain e.g. flight-booking, can work well in another e.g. tourist information. This finding suggests that it should be unnecessary to collect new training data for every different slot-filling domain, and indeed that data from different slot-filling domains can be pooled. Even if system designers in industry do not use the RL approach for producing dialogue strategies, the research community can use it to gain insights into which contextual features are important in dialogue management and why e.g. as we have done for recent DAs in this thesis. System designers in industry can use these insights in order to hand-craft better strategies than they would have otherwise.

- **Possibilities for improving different aspects of the RL approach used in this thesis:** Although our learned strategies performed well in testing in simulation and with real users, there is obviously still room for further improvements. Representing the recent Dialogue Acts (DAs) in the state produced better repair strategies for SSFU states, but these repair strategies can surely be improved, and furthermore, we did not detect significant improvements in any other aspect of the learned strategy e.g. not in dealing with user indications of misunderstanding errors, nor in portions of dialogue in which there is smooth progress towards the goal of filling and confirming all of the slots. Therefore we must think about how the RL approach used in this thesis can be modified to find further improvements.

  Firstly, since our learned DA-strategies often employ different repair strategies following ASR rejections and user utterances that are recognised as out-of-domain, it might be useful to conduct a study to better understand why, and to consider making more fine-grained distinctions between different kinds of non-understandings e.g. distinguishing between user utterances that are within-domain but recognised as out-of-domain due to ASR errors, versus user utterances that

are really out-of-domain. If making such a distinction manually led to better learned repair strategies, then it would motivate looking at how a real-time system can try to make the same distinction automatically, e.g. by using Supervised Learning (SL) to produce a model that makes predictions based on features in the SDS's representation of the dialogue context. A possible second thing to investigate is changing our training reward function, which only takes account of task completion and dialogue length. Perhaps we could obtain subtle improvements in the learned strategy in non-SSFU states with a reward function that better models overall user satisfaction.

Probably the most important factor in gaining further improvements is the user simulation. The 4 and 5 gram simulations used in this thesis output actions based on the DA tags of the last 3 and 4 turns respectively i.e. they are only sensitive to the the DA tags of the last 3 and 4 turns. This is a useful approximation, but surely real users are sensitive to additional features of the context, including sometimes events that occurred earlier than 3 or 4 turns ago. If we can produce user simulations which are sensitive to additional contextual features in a realistic manner, it becomes worthwhile to represent additional context in the state because a reinforcement learner can then potentially learn further-improved repair strategies. For example, recall that with respect to the limitations of the n-gram simulations, we stated in Chapter 5 that although certain slot-values in COMMUNICATOR dialogues were "impossible-to-fill" (e.g. because the speech recogniser could not recognise the user), the n-gram simulations failed to simulate this. If we trained with a simulation that did simulate impossible-to-fill slot values, then we could add a dialogue length feature to the state and potentially learn dialogue strategies that give up asking for a particular slot-value after a certain number of attempts and query the database anyway. Recall also that the reward function ought to ideally award maximum reward for task completion in such cases in order to speed up learning. On the point of increasing the amount of dialogue history to which the n-gram simulations are sensitive, we could just increase the value of n, but in the case of the COMMUNICATOR data, the accuracy of the n-gram simulations seemed to be deteriorating for $n > 4$ due to sparsity of data (see Georgila et al. 2006).

- **The size of the state space and the problem of intractability:** Note that if we did have a user simulation which was sensitive to a greater number of con-

textual features in a realistic manner, and as a result we included more features in the RL state, it is likely that we would soon run up against serious problems with intractability. One way to deal with the intractability problem is to apply some kind of generalisation method as was used in producing the Hybrid RL/SL Strategy of Henderson et al. (2008). However, as was shown in Chapter 6, this strategy seems to be sub-optimal, and this is presumably because of its enormous state-action space. We should note though that much better performance could surely have been obtained if rather than all of the available contextual features, only the more important ones were represented in the state. It would be very useful then if we could take the kind of dataset used by Henderson et al. (2008), which contains a very large number of contextual features, and apply feature selection techniques e.g. CFS Hall (1999) in order to automatically identify those which are important. However the best way to do this is not obvious. One idea is to approximate the dialogue management problem to a categorisation problem in which we want to predict the current system action based on the contextual features. We could take only dialogues in which the system is apparently behaving "more optimally" e.g. dialogues with task completion, and then apply feature selection to identify a subset of contextual features which are predictive of these "more optimal" system actions.

Hierarchical RL is another method for tackling larger state-action spaces, and has already been used by Cuayáhuitl et al. (2007) to learn a simple slot-filling dialogue strategy. It reduces the state-action space by exploiting prior knowledge. Firstly, where it is thought appropriate, sub-problems are cast as different examples of the same sub-problem, so allowing re-use of learned solutions e.g. filling slot 1 and filling slot 2 could be considered as two different examples of filling a slot. Additionally, for each sub-problem, only those features which are considered relevant are included in the state representation. The findings of this thesis then potentially provide prior knowledge which can be exploited by hierarchical RL - a full dialogue strategy could be divided into the sub-problems of filling / confirming slots when dialogue progress is smooth versus when it has stalled, and additional recent DA state features could be used in learning a solution for the latter.

- **Using POMDPs as an alternative to MDPs:** Another interesting avenue for future research is to use Partially Observable Markov Decision Processes (POMDPs)

rather than MDPs for RL of dialogue strategies - some work has already been done in this area (Williams and Young, 2007). A POMDP is an extension of an MDP, and is used for choosing actions when the entire world, or state-space is not always directly observable. Since the true state of the world cannot be uniquely identified, a POMDP reasoner must maintain a probability distribution, called the *belief state*, which describes the probabilities for each true state of the world. Maintenance of the belief state is Markovian in that it only requires knowledge of the previous belief state and the action taken. POMDPs are therefore able to handle uncertainty in a principled way, and since Automatic Speech Recognition (ASR) and Natural Language Understanding (NLU) are error prone, this makes them theoretically appealing for dialogue management. However, at present POMDPs are computationally intractable to solve for optimal behaviour for dialogue problems of realistic size. Assuming this intractability problem can be overcome, it will be interesting to compare the performance of dialogue strategies learned with POMDPs as opposed to MDPs, and in particular, their confirmation and repair strategies.

To round off this thesis, we now provide a brief summary of its main contributions. Firstly, we have shown that if we generate training dialogues with an accurate probabilistic user simulation whose probabilities are derived from real user data, then we can use the MDP-RL approach to learn full dialogue strategies which work well with real users. Secondly, we have also shown that adding recent DAs to the MDP produces dialogue strategies which work better both in simulation and with real users. The final main contribution concerns how the recent DAs improve the learned strategy - we have shown how they produce better repair strategies for contexts in which the slot-status features are unchanged and hence dialogue progress has stalled.

Appendix A

Bayesian Network user simulation probability table

SP	SC	H	quiet	slot1	slot2	slot3	slot4	s1234	no rep	no	yes	ask help	hang up
open question	< two	null	0	0.16	0.16	0.16	0.16	0.16	0	0	0	0.2	0
ask slot 1	< two	null	0	0.8	0	0	0	0	0	0	0	0.2	0
ask slot 2	< two	null	0	0	0.8	0	0	0	0	0	0	0.2	0
ask slot 3	< two	null	0	0	0	0.8	0	0	0	0	0	0.2	0
ask slot 4	< two	null	0	0	0	0	0.8	0	0	0	0	0.2	0
exp conf 1	< two	null	0	0.1	0	0	0	0	0	0	0.9	0	0
exp conf 2	< two	null	0	0	0.1	0	0	0	0	0	0.9	0	0
exp conf 3	< two	null	0	0	0	0.1	0	0	0	0	0.9	0	0
exp conf 4	< two	null	0	0	0	0	0.1	0	0	0	0.9	0	0
ask 1 ic 4	< two	null	0	0.8	0	0	0	0	0	0	0	0.2	0
ask 2 ic 1	< two	null	0	0.8	0	0	0	0	0	0	0	0.2	0
ask 3 ic 2	< two	null	0	0.8	0	0	0	0	0	0	0	0.2	0
ask 4 ic 3	< two	null	0	0	0	0.1	0.9	0	0	0	0	0	0
give help	< two	null	0.1	0	0	0	0	0	0	0	0.9	0	0

Table A.1: The probability table for the Bayesian Net User Simulation. SP = System Prompt, SC = Slot Counter, H = History, ic = implicitly confirm

SP	SC	H	quiet	slot1	slot2	slot3	slot4	s1234	no rep	no	yes	ask help	h u
open question	< two	slot 1 wrong	0	0.16	0.16	0.16	0.16	0.16	0	0	0	0.2	0
ask slot 1	< two	slot 1 wrong	0	0.9	0	0	0	0	0	0.1	0	0	0
ask slot 2	< two	slot 1 wrong	0	0	0.1	0	0	0	0	0	0	0	0.9
ask slot 3	< two	slot 1 wrong	0	0	0	0.1	0	0	0	0	0	0	0.9
ask slot 4	< two	slot 1 wrong	0	0	0	0	0.1	0	0	0	0	0	0.9
exp conf 1	< two	slot 1 wrong	0	0	0	0	0	0	0.9	0.1	0	0	0
exp conf 2	< two	slot 1 wrong	0	0	0.1	0	0	0	0	0	0	0	0.9
exp conf 3	< two	slot 1 wrong	0	0	0	0.1	0	0	0	0	0	0	0.9
exp conf 4	< two	slot 1 wrong	0	0	0	0	0.1	0	0	0	0	0	0.9
ask 1 ic 4	< two	slot 1 wrong	0	0.5	0	0	0	0	0	0	0	0	0.5
ask 2 ic 1	< two	slot 1 wrong	0.2	0	0	0	0	0.6	0.2	0	0	0	0
ask 3 ic 2	< two	slot 1 wrong	0	0.1	0	0	0	0	0	0	0	0	0.9
ask 4 ic 3	< two	slot 1 wrong	0	0.1	0	0	0	0	0	0	0	0	0.9
give help	< two	slot 1 wrong	0	0.1	0	0	0	0	0	0	0.9	0	0
open question	< two	slot 2 wrong	0	0.16	0.16	0.16	0.16	0.16	0	0	0	0.2	0
ask slot 1	< two	slot 2 wrong	0	0	0.1	0	0	0	0	0	0	0	0.9
ask slot 2	< two	slot 2 wrong	0	0	0.9	0	0	0	0	0.1	0	0	0
ask slot 3	< two	slot 2 wrong	0	0	0	0.1	0	0	0	0	0	0	0.9
ask slot 4	< two	slot 2 wrong	0	0	0	0	0.1	0	0	0	0	0	0.9
exp conf 1	< two	slot 2 wrong	0.1	0	0	0	0	0	0	0	0	0	0.9
exp conf 2	< two	slot 2 wrong	0	0.1	0	0	0	0	0.9	0	0	0	0
exp conf 3	< two	slot 2 wrong	0	0	0.1	0	0	0	0	0	0	0	0.9
exp conf 4	< two	slot 2 wrong	0	0	0	0.1	0	0	0	0	0	0	0.9
ask 1 ic 4	< two	slot 2 wrong	0	0.1	0	0	0	0	0	0	0	0	0.9
ask 2 ic 1	< two	slot 2 wrong	0	0	0.5	0	0	0	0	0	0	0	0.5
ask 3 ic 2	< two	slot 2 wrong	0.2	0	0	0	0	0	0.6	0.2	0	0	0
ask 4 ic 3	< two	slot 2 wrong	0	0	0	0.1	0	0	0	0	0	0	0.9
give help	< two	slot 2 wrong	0.1	0	0	0	0	0	0	0	0.9	0	0
open question	< two	slot 3 wrong	0	0.16	0.16	0.16	0.16	0.16	0	0	0	0.2	0
ask slot 1	< two	slot 3 wrong	0	0.1	0	0	0	0	0	0	0	0	0.9
ask slot 2	< two	slot 3 wrong	0	0	0.1	0	0	0	0	0	0	0	0.9
ask slot 3	< two	slot 3 wrong	0	0	0	0.9	0	0	0	0.1	0	0	0
ask slot 4	< two	slot 3 wrong	0	0	0	0	0.1	0	0	0	0	0	0.9
exp conf 1	< two	slot 3 wrong	0	0.1	0	0	0	0	0	0	0	0	0.9
exp conf 2	< two	slot 3 wrong	0	0	0.1	0	0	0	0	0	0	0	0.9
exp conf 3	< two	slot 3 wrong	0	0	0	0.1	0	0	0.9	0	0	0	0
exp conf 4	< two	slot 3 wrong	0	0	0	0	0.1	0	0	0	0	0	0.9
ask 1 ic 4	< two	slot 3 wrong	0	0.1	0	0	0	0	0	0	0	0	0.9
ask 2 ic 1	< two	slot 3 wrong	0.2	0	0	0	0	0	0.6	0.2	0	0	0
ask 3 ic 2	< two	slot 3 wrong	0	0	0.5	0	0	0	0	0	0	0	0.5
ask 4 ic 3	< two	slot 3 wrong	0.2	0	0	0	0	0	0.6	0.2	0	0	0
give help	< two	slot 3 wrong	0.1	0	0	0	0	0	0	0	0.9	0	0

Table A.2: The probability table for the Bayesian Net User Simulation. SP = System Prompt, SC = Slot Counter, H = History, ic = implicitly confirm

SP	SC	H	quiet	slot1	slot2	slot3	slot4	s1234	no rep	no	yes	ask help	h u
open question	< two	slot 4 wrong	0	0.16	0.16	0.16	0.16	0.16	0	0	0	0.2	0
ask slot 1	< two	slot 4 wrong	0	0.1	0	0	0	0	0	0	0	0	0.9
ask slot 2	< two	slot 4 wrong	0	0	0.1	0	0	0	0	0	0	0	0.9
ask slot 3	< two	slot 4 wrong	0	0	0	0.1	0	0	0	0	0	0	0.9
ask slot 4	< two	slot 4 wrong	0	0	0	0	0.9	0	0	0.1	0	0	0
exp conf 1	< two	slot 4 wrong	0	0.1	0	0	0	0	0	0	0	0	0.9
exp conf 2	< two	slot 4 wrong	0	0	0.1	0	0	0	0	0	0	0	0.9
exp conf 3	< two	slot 4 wrong	0	0	0	0.1	0	0	0	0	0	0	0.9
exp conf 4	< two	slot 4 wrong	0	0	0	0	0.1	0	0.9	0	0	0	0
ask 1 ic 4	< two	slot 4 wrong	0	0	0.5	0	0	0	0	0	0	0	0.5
ask 2 ic 1	< two	slot 4 wrong	0	0	0.1	0	0	0	0	0	0	0	0.9
ask 3 ic 2	< two	slot 4 wrong	0	0	0	0.1	0	0	0	0	0	0	0.9
ask 4 ic 3	< two	slot 4 wrong	0	0	0	0	0.5	0	0	0	0	0	0.5
give help	< two	slot 4 wrong	0.1	0	0	0	0	0	0	0	0.9	0	0
open question	< two	help	0.1	0	0	0	0	0	0	0	0	0	0.9
ask slot 1	< two	help	0	0.1	0	0	0	0	0	0	0	0	0.9
ask slot 2	< two	help	0	0	0.1	0	0	0	0	0	0	0	0.9
ask slot 3	< two	help	0	0	0	0.1	0	0	0	0	0	0	0.9
ask slot 4	< two	help	0	0	0	0	0.1	0	0	0	0	0	0.9
exp conf 1	< two	help	0	0.1	0	0	0	0	0	0	0	0	0.9
exp conf 2	< two	help	0	0	0.1	0	0	0	0	0	0	0	0.9
exp conf 3	< two	help	0	0	0	0.1	0	0	0	0	0	0	0.9
exp conf 4	< two	help	0	0	0	0	0.1	0	0	0	0	0	0.9
ask 1 ic 4	< two	help	0	0.1	0	0	0	0	0	0	0	0	0.9
ask 2 ic 1	< two	help	0	0	0.1	0	0	0	0	0	0	0	0.9
ask 3 ic 2	< two	help	0	0	0	0.1	0	0	0	0	0	0	0.9
ask 4 ic 3	< two	help	0	0	0	0	0.1	0	0	0	0	0	0.9
give help	< two	help	0.1	0	0	0	0	0	0	0	0.9	0	0
open question	two plus	null	0	0.16	0.16	0.16	0.16	0.16	0	0	0	0.2	0
ask slot 1	two plus	null	0	0.4	0	0	0	0	0	0	0	0	0.6
ask slot 2	two plus	null	0	0	0.4	0	0	0	0	0	0	0	0.6
ask slot 3	two plus	null	0	0	0	0.4	0	0	0	0	0	0	0.6
ask slot 4	two plus	null	0	0	0	0	0.4	0	0	0	0	0	0.6
exp conf 1	two plus	null	0	0.1	0	0	0	0	0	0	0.9	0	0
exp conf 2	two plus	null	0	0	0.1	0	0	0	0	0	0.9	0	0
exp conf 3	two plus	null	0	0	0	0.1	0	0	0	0	0.9	0	0
exp conf 4	two plus	null	0	0	0	0	0.1	0	0	0	0.9	0	0
ask 1 ic 4	two plus	null	0	0.4	0	0	0	0	0	0	0	0	0.6
ask 2 ic 1	two plus	null	0	0	0.4	0	0	0	0	0	0	0	0.6
ask 3 ic 2	two plus	null	0	0	0	0.4	0	0	0	0	0	0	0.6
ask 4 ic 3	two plus	null	0	0	0	0	0.4	0	0	0	0	0	0.6
give help	two plus	null	0.1	0	0	0	0	0	0	0	0.9	0	0

Table A.3: The probability table for the Bayesian Net User Simulation. SP = System Prompt, SC = Slot Counter, H = History, ic = implicitly confirm

SP	SC	H	quiet	slot1	slot2	slot3	slot4	s1234	no rep	no	yes	ask help	h u
open question	two plus	slot 1 wrong	0	0.16	0.16	0.16	0.16	0.16	0	0	0	0.2	0
ask slot 1	two plus	slot 1 wrong	0	0.4	0	0	0	0	0	0	0	0	0.6
ask slot 2	two plus	slot 1 wrong	0	0	0.1	0	0	0	0	0	0	0	0.9
ask slot 3	two plus	slot 1 wrong	0	0	0	0.1	0	0	0	0	0	0	0.9
ask slot 4	two plus	slot 1 wrong	0	0	0	0	0.1	0	0	0	0	0	0.9
exp conf 1	two plus	slot 1 wrong	0	0.1	0	0	0	0	0.9	0	0	0	0
exp conf 2	two plus	slot 1 wrong	0	0	0.1	0	0	0	0	0	0	0	0.9
exp conf 3	two plus	slot 1 wrong	0	0	0	0.1	0	0	0	0	0	0	0.9
exp conf 4	two plus	slot 1 wrong	0	0	0	0	0.1	0	0	0	0	0	0.9
ask 1 ic 4	two plus	slot 1 wrong	0	0.4	0	0	0	0	0	0	0	0	0.6
ask 2 ic 1	two plus	slot 1 wrong	0.3	0	0	0	0	0.3	0.3	0	0	0	0.1
ask 3 ic 2	two plus	slot 1 wrong	0	0	0	0.1	0	0	0	0	0	0	0.9
ask 4 ic 3	two plus	slot 1 wrong	0	0	0	0	0.1	0	0	0	0	0	0.9
give help	two plus	slot 1 wrong	0.1	0	0	0	0	0	0	0	0.9	0	0
open question	two plus	slot 2 wrong	0	0.16	0.16	0.16	0.16	0.16	0	0	0	0.2	0
ask slot 1	two plus	slot 2 wrong	0	0.1	0	0	0	0	0	0	0	0	0.9
ask slot 2	two plus	slot 2 wrong	0	0	0.4	0	0	0	0	0	0	0	0.6
ask slot 3	two plus	slot 2 wrong	0	0	0	0.1	0	0	0	0	0	0	0.9
ask slot 4	two plus	slot 2 wrong	0	0	0	0	0.1	0	0	0	0	0	0.9
exp conf 1	two plus	slot 2 wrong	0	0.1	0	0	0	0	0	0	0	0	0.9
exp conf 2	two plus	slot 2 wrong	0	0	0.1	0	0	0	0.9	0	0	0	0
exp conf 3	two plus	slot 2 wrong	0	0	0	0.1	0	0	0	0	0	0	0.9
exp conf 4	two plus	slot 2 wrong	0	0	0	0	0.1	0	0	0	0	0	0.9
ask 1 ic 4	two plus	slot 2 wrong	0	0.1	0	0	0	0	0	0	0	0	0.9
ask 2 ic 1	two plus	slot 2 wrong	0.4	0	0	0	0	0	0	0	0	0	0.6
ask 3 ic 2	two plus	slot 2 wrong	0.3	0	0	0	0	0	0.3	0.3	0	0	0.1
ask 4 ic 3	two plus	slot 2 wrong	0	0	0	0	0.1	0	0	0	0	0	0.9
give help	two plus	slot 2 wrong	0.1	0	0	0	0	0	0	0	0.9	0	0
open question	two plus	slot 3 wrong	0	0.18	0.18	0.18	0.18	0.18	0	0	0	0.1	0
ask slot 1	two plus	slot 3 wrong	0	0.1	0	0	0	0	0	0	0	0	0.9
ask slot 2	two plus	slot 3 wrong	0	0	0.1	0	0	0	0	0	0	0	0.9
ask slot 3	two plus	slot 3 wrong	0	0	0	0.4	0	0	0	0	0	0	0.6
ask slot 4	two plus	slot 3 wrong	0	0	0	0	0.1	0	0	0	0	0	0.9
exp conf 1	two plus	slot 3 wrong	0	0.1	0	0	0	0	0	0	0	0	0.9
exp conf 2	two plus	slot 3 wrong	0	0	0.1	0	0	0	0	0	0	0	0.9
exp conf 3	two plus	slot 3 wrong	0	0	0	0.1	0	0	0.9	0	0	0	0
exp conf 4	two plus	slot 3 wrong	0	0	0	0	0.1	0	0	0	0	0	0.9
ask 1 ic 4	two plus	slot 3 wrong	0	0.1	0	0	0	0	0	0	0	0	0.9
ask 2 ic 1	two plus	slot 3 wrong	0.3	0	0	0	0	0	0.3	0.3	0	0	0.1
ask 3 ic 2	two plus	slot 3 wrong	0	0	0.4	0	0	0	0	0	0	0	0.6
ask 4 ic 3	two plus	slot 3 wrong	0.3	0	0	0	0	0	0.3	0.3	0	0	0.1
give help	two plus	slot 3 wrong	0.1	0	0	0	0	0	0	0	0.9	0	0

Table A.4: Probability table continued. SP = System Prompt, SC = Slot Counter, H = History, ic = implicitly confirm

SP	SC	H	quiet	slot1	slot2	slot3	slot4	s1234	no rep	no	yes	ask help	h u
open question	two plus	slot 4 wrong	0	0.18	0.18	0.18	0.18	0.18	0	0	0	0.1	0
ask slot 1	two plus	slot 4 wrong	0	0.1	0	0	0	0	0	0	0	0	0.9
ask slot 2	two plus	slot 4 wrong	0	0	0.1	0	0	0	0	0	0	0	0.9
ask slot 3	two plus	slot 4 wrong	0	0	0	0.1	0	0	0	0	0	0	0.9
ask slot 4	two plus	slot 4 wrong	0	0	0	0	0.9	0	0	0	0	0	0.1
exp conf 1	two plus	slot 4 wrong	0	0.1	0	0	0	0	0	0	0	0	0.9
exp conf 2	two plus	slot 4 wrong	0	0	0.1	0	0	0	0	0	0	0	0.9
exp conf 3	two plus	slot 4 wrong	0	0	0	0.1	0	0	0	0	0	0	0.9
exp conf 4	two plus	slot 4 wrong	0	0	0	0	0	0	0.9	0.1	0	0	0
ask 1 ic 4	two plus	slot 4 wrong	0	0	0.4	0	0	0	0	0	0	0	0.6
ask 2 ic 1	two plus	slot 4 wrong	0	0	0.1	0	0	0	0	0	0	0	0.9
ask 3 ic 2	two plus	slot 4 wrong	0	0	0	0.1	0	0	0	0	0	0	0.9
ask 4 ic 3	two plus	slot 4 wrong	0	0	0	0	0.4	0	0	0	0	0	0.6
give help	two plus	slot 4 wrong	0.1	0	0	0	0	0	0	0	0.9	0	0
open question	two plus	help	0.1	0	0	0	0	0	0	0	0	0	0.9
ask slot 1	two plus	help	0.1	0	0	0	0	0	0	0	0	0	0.9
ask slot 2	two plus	help	0.1	0	0	0	0	0	0	0	0	0	0.9
ask slot 3	two plus	help	0.1	0	0	0	0	0	0	0	0	0	0.9
ask slot 4	two plus	help	0.1	0	0	0	0.9	0	0	0	0	0	0.9
exp conf 1	two plus	help	0.1	0	0	0	0	0	0	0	0	0	0.9
exp conf 2	two plus	help	0.1	0	0	0	0	0	0	0	0	0	0.9
exp conf 3	two plus	help	0.1	0	0	0	0	0	0	0	0	0	0.9
exp conf 4	two plus	help	0.1	0	0	0	0	0	0.9	0	0	0	0.9
ask 1 ic 4	two plus	help	0.1	0	0	0	0	0	0	0	0	0	0.9
ask 2 ic 1	two plus	help	0.1	0	0	0	0	0	0	0	0	0	0.9
ask 3 ic 2	two plus	help	0.1	0	0	0	0	0	0	0	0	0	0.9
ask 4 ic 3	two plus	help	0.1	0	0	0	0	0	0	0	0	0	0.9
give help	two plus	help	0.1	0	0	0	0	0	0	0	0.9	0	0

Table A.5: Probability table continued. SP = System Prompt, SC = Slot Counter, H = History, ic = implicitly confirm

Appendix B

Spoken Dialogue System user evaluation questionnaire

OVERALL INSTRUCTIONS:

You are driving towards a town that you don't know well. You can speak to your in-car information system to hear information about the town. The system will also show you items on the map display.

Please REMEMBER to wait for the 'BEEP' before speaking!!

Try to complete the following tasks, using normal conversation as you would expect to with a human. You will go through the 5 tasks three times, each time with a slightly different version of the system. You can press the 'stop' button at the end of each task when you have chosen a particular bar/hotel/restaurant to allow a pause for answering the questions on the form. Use a cross to choose a number the first time through the five tasks, a circle, the second time, and a square the third time. Press 'go' to continue the dialogue.

If you have no preference about something that you are asked, you can say things like 'I don't mind' or 'don't know' , and so on. You can ask the system for help but the experimenter will not assist you with the tasks. If you are really struggling with a task you can ask the system to restart or quit. If the system is failing to recognise these commands, press the stop button and tell the experimenter. Please tell the experimenter

each time you complete the five tasks.

TASK 1:

You are on a business trip on your own. You need to find a hotel room in the middle of the town. Price is no problem.
Write the name of result that the system presented to you (e.g. FOG BAR) here:

_____-

Was this the item that matched your search? Yes / No

Please indicate your degree of agreement with each statement below.

1:Strongly Disagree 2:Disagree 3:Neutral 4:Agree 5:Strongly agree

In this conversation, it was easy to get the information that I wanted.

1 2 3 4 5

The system worked the way I expected it to, in this conversation.

1 2 3 4 5

Based on my experience in this conversation, I would like to use this system regularly.

1 2 3 4 5

TASK 2:

You are hungry and in the mood for some Chinese egg-fried-rice. You want to eat somewhere near your hotel, which is in the centre of the town. You don't want anything too expensive, nor too cheap.

Write the name of result that the system presented to you (e.g. FOG BAR) here:

_____-

Was this the item that matched your search? Yes / No

Please indicate your degree of agreement with each statement below.

1:Strongly Disagree 2:Disagree 3:Neutral 4:Agree 5:Strongly agree

In this conversation, it was easy to get the information that I wanted.

1 2 3 4 5

The system worked the way I expected it to, in this conversation.

1 2 3 4 5

Based on my experience in this conversation, I would like to use this system regularly.

1 2 3 4 5

TASK 3:

You just want to find somewhere for a beer, again near your hotel.

Write the name of result that the system presented to you (e.g. FOG BAR) here:

_____-

Was this the item that matched your search? Yes / No

Please indicate your degree of agreement with each statement below.

1:Strongly Disagree 2:Disagree 3:Neutral 4:Agree 5:Strongly agree

In this conversation, it was easy to get the information that I wanted.

1 2 3 4 5

The system worked the way I expected it to, in this conversation.

1 2 3 4 5

Based on my experience in this conversation, I would like to use this system regularly.

1 2 3 4 5

TASK 4:

It is 6 months later and you are driving towards the same town. You want to take a friend out for a luxurious meal, at a French place. Since you are driving it doesn't matter where the restaurant is located.

Write the name of result that the system presented to you (e.g. FOG BAR) here:

_____-

Was this the item that matched your search? Yes / No

Please indicate your degree of agreement with each statement below.

1:Strongly Disagree 2:Disagree 3:Neutral 4:Agree 5:Strongly agree

In this conversation, it was easy to get the information that I wanted.

1 2 3 4 5

The system worked the way I expected it to, in this conversation.

1 2 3 4 5

Based on my experience in this conversation, I would like to use this system regularly.

1 2 3 4 5

TASK 5:

It is a year later, and you are on holiday. You need a single room somewhere for the night but you are not fussy about where because you are driving.

Write the name of result that the system presented to you (e.g. FOG BAR) here:

_____-

Was this the item that matched your search? Yes / No

Please indicate your degree of agreement with each statement below.

1:Strongly Disagree 2:Disagree 3:Neutral 4:Agree 5:Strongly agree

In this conversation, it was easy to get the information that I wanted.

1 2 3 4 5

The system worked the way I expected it to, in this conversation.

1 2 3 4 5

Based on my experience in this conversation, I would like to use this system regularly.

1 2 3 4 5

Bibliography

R. Amalberti, N. Carbonell, and P. Falzon. User representations of computer systems in human-computer speech interaction. *International Journal of Man-Machine Studies*, 38(4):547–566, 1993.

R. Bellman. Dynamic programming. Princeton University Press, Princeton, NJ, USA, 1957.

D. Bersekas and J. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

D. Bohus. http://www.cs.cmu.edu/ dbohus/roomline, 2003.

D. Bohus and A. Rudnicky. Sorry, I didn't catch that! - an investigation of non-understanding errors and recovery strategies. In *Proceedings of SIGdial*, 2005.

D. Bohus and A. Rudnicky. Ravenclaw: dialogue management using hierarchical task decomposition and an expectation agenda. In *Proceedings of Eurospeech*, 2003.

J. Bos, E. Klein, O. Lemon, and T. Oka. DIPPER: description and formalisation of an information-state update dialogue system architecture. In *Proceedings of the 4th SIGdial Workshop on Discourse and Dialogue*, pages 115–124, 2003.

Cepstral. Theta: a small footprint text-to-speech synthesizer. Technical report, Cepstral LLC, Pittsburgh, PA, USA, 2004.

A. Cheyer and D. Martin. The Open Agent Architecture. *Journal of Autonomous Agents and Multi-Agent Systems*, 4(1/2):143–148, 2001.

D. Chickering and T. Paek. Online adaptation of influence diagrams. Technical Report MSR-TR-2005-55, Microsoft Corporation, 2005.

M. Core and J. Allen. Coding dialogues with the DAMSL annotation scheme. In *Proceedings of the AAAI Fall Symposium on Communicative Action in Humans and Machines*, 1997.

H. Cuayáhuitl, S. Renals, O. Lemon, and H. Shimodaira. Hierarchical dialogue optimization using Semi-Markov Decision Processes. In *Proceedings of Interspeech*, 2007.

N. Dahlback, A. Jonsson, and L Ahrenberg. Wizard of Oz studies - why and how. In *Proceedings of the International Workshop on Intelligent User Interfaces*, pages 193–200, 1993.

K. Doya. Temporal difference learning in continuous time and space. In *Advances in Neural Information Processing Systems: Proceedings of the 1995 Conference*, pages 1073–1079. MIT Press, 1996.

M. English and P. Heeman. Learning mixed-initiative dialog strategies by using reinforcement learning on both conversants. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 1011–1018, 2005.

K. Forbes-Riley and D. Litman. Using bigrams to identify relationships between student certainness states and tutor responses in a spoken dialogue corpus. In *Proceedings of SIGdial*, 2005.

M. Frampton and O. Lemon. Reinforcement learning of dialogue strategies using the user's last dialogue act. In *Proceedings of the IJCAI workshop on Knowledge and Reasoning in Practical Dialogue Systems*, 2005.

M. Frampton and O. Lemon. Learning more effective dialogue strategies using limited dialogue move features. In *Proceedings of the 44th Annual Meeting of the Association for Computational Linguistics*, 2006.

M. Frampton and O. Lemon. Using dialogue acts to learn better repair strategies. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2008a.

M. Frampton and O. Lemon. Recent advances in reinforcement learning in spoken dialogue systems. To appear in Knowledge Engineering Review, 2008b.

N. Fraser and G. Gilbert. Simulating speech systems. *Computer Speech and Language*, 5(1), 1991.

K. Georgila, J. Henderson, and O. Lemon. Learning user simulations for information state update dialogue systems. In *Proceedings of Eurospeech*, 2005a.

K. Georgila, O. Lemon, and J. Henderson. Automatic annotation of COMMUNICATOR dialogue data for learning dialogue strategies and user simulations. In *Proceedings of the 9th Workshop on the Semantics and Pragmatics of Dialogue (SEMDIAL: DIALOR)*, 2005b.

K. Georgila, J. Henderson, and O. Lemon. User simulation for spoken dialogue systems: learning and evaluation. In *Proceedings of the 9th International Conference on Spoken Language Processing*, 2006.

J. Greene and M. D'Oliveira. *Learning to use statistical tests in psychology*. Open University press, United Kingdom, 2001.

M. Hall. *Correlation-based feature selection for machine learning*. PhD thesis, University Of Waikato, New Zealand, 1999.

D. Heckerman. A Bayesian approach to learning causal networks. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence*, pages 285–295, 1995.

J. Henderson, O. Lemon, and K. Georgila. Hybrid reinforcement/supervised learning for dialogue policies from COMMUNICATOR data. In *Proceedings of the IJCAI workshop on Knowledge and Reasoning in Practical Dialogue Systems*, 2005.

J. Henderson, O. Lemon, and K. Georgila. Hybrid reinforcement/supervised learning of dialogue policies in information state update systems. To appear in Computational Linguistics, 2008.

M. Kearns, Y. Mansour, and A. Ng. A sparse sampling algorithm for near-optimal planning in large Markov Decision Processes. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 1324–1231, 1999.

O. Lemon and O. Pietquin. Machine learning for spoken dialogue systems. In *Proceedings of Interspeech*, 2007.

O. Lemon, K. Georgila, and J. Henderson. Evaluating effectiveness and portability of reinforcement learned strategies. In *Proceedings of the IEEE/ACL Workshop on Spoken Language Technology*, 2006a.

O. Lemon, K. Georgila, J. Henderson, and M. Stuttle. An ISU dialogue system exhibiting reinforcement learning of dialogue policies: generic slot-filling in the TALK in-car system. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics*, 2006b.

E. Levin and R. Pieraccini. A stochastic model of computer-human interaction for learning dialogue strategies. In *Proceedings of Eurospeech*, pages 1883–1886, 1997.

E. Levin, R. Pieraccini, and W. Eckert. Using Markov Decision Processes for learning dialogue strategies. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, 1998.

E. Levin, R. Pieraccini, and W. Eckert. A stochastic model of computer-human interaction for learning dialogue strategies. *IEEE Transactions On Speech And Audio Processing*, 8(1), 2000.

D. Litman and S. Silliman. ITSPOKE: An Intelligent Tutoring Spoken dialogue system. In *Companion Proceedings of the Human Language Technology Conference: 4th Meeting of the North American Chapter of the Association for Computational Linguistics*, 2004.

Norsys. *Netica-J Manual*. Norsys Software Corporation, 2315 Dunbar Street, Vancouver, Canada, June 2002.

S. Oviatt, M. MacEachern, and G. Levow. Predicting hyperarticulate speech during human-computer error resolution. *Speech Communication*, 24:87–110, 1998.

T. Paek and D. Chickering. The Markov assumption in spoken dialogue management. In *Proceedings of the 6th SIGdial Workshop on Discourse and Dialogue*, 2005.

O. Pietquin. *A framework for unsupervised learning of dialogue strategies*. PhD thesis, Faculté Polytechnique de Mons, TCTS Lab, Belgique, 2004.

O. Pietquin and S. Renals. ASR system modeling for automatic evaluation and optimization of dialogue systems. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2002.

L. Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

E. Reiter and R. Dale. Building applied natural language generation systems. *Natural Language Engineering*, 3:57–87, 1997.

J. Schatzmann, K. Weilhammer, M. N. Stuttle, and S. Young. A survey of statistical user simulation techniques for reinforcement-learning of dialogue management strategies. *Knowledge Engineering Review*, 21(2):97–126, 2006.

K. Scheffler. *Automatic design of spoken dialogue systems*. PhD thesis, University of Cambridge, United Kingdom, 2002.

K. Scheffler and S. Young. Probabilistic simulation of human-machine dialogues. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 1217–1220, 2000.

K. Scheffler and S. Young. Corpus-based dialogue simulation for automatic strategy learning and evaluation. In *Proceedings of the NAACL Workshop on Adaptation in Dialogue Systems*, pages 64–70, 2001.

K. Scheffler and S. Young. Automatic learning of dialogue strategy using dialogue simulation and reinforcement learning. In *Proceedings of the Human Language Technology conference*, 2002.

J. Searle. *Speech Acts*. Cambridge University Press, United Kingdom, 1969.

D. Sheskin. *Handbook of parametric and nonparametric statistical procedures*. Taylor and Francis Group, 4th edition, 2007.

E. Shriberg, E. Wade, and P. Price. Human-machine problem solving using spoken language systems (SLS): factors affecting performance and user satisfaction. In *Proceedings of the DARPA Speech and Natural Language Workshop*, pages 49–54, 1992.

S. Singh, M. Kearns, D. Litman, and M. Walker. Reinforcement learning for spoken dialogue systems. In *Proceedings of the Annual Conference on Neural Information Processing Systems*, 1999.

S. Singh, M. Kearns, D. Litman, and M. Walker. Reinforcement learning for spoken dialogue systems. In *Advances in Neural Information Processing Systems*, volume 12, 2000.

S. Singh, D. Litman, M. Kearns, and M. Walker. Optimizing dialogue management with reinforcement learning: experiments with the NJFun system. *Journal of Artificial Intelligence Research*, 16:105–133, 2002.

G. Skantze. Exploring human error handling strategies: implications for spoken dialogue systems. In *Proceedings of the ISCA Workshop on Error Handling in Spoken Dialogue Systems*, 2003.

G. Skantze. *Error handling in Spoken Dialogue Systems*. PhD thesis, KTH Computer Science and Communication, Stockholm, Sweden, 2007.

R. Sutton and A. Barto. *Reinforcement learning: an introduction*. The MIT Press. Cambridge, Massachusetts, USA, 1998.

J. Tetreault and D. Litman. Comparing the utility of state features in spoken dialogue using reinforcement learning. In *Proceedings of the Human Language Technology Conference/North American chapter of the Association for Computational Linguistics annual meeting*, 2006.

M. Walker. An application of reinforcement learning to dialogue strategy selection in a spoken dialogue system for email. *Journal of Artificial Intelligence Research*, 12: 387–416, 2000.

M. Walker and R. Passonneau. DATE: a Dialogue Act Tagging scheme for Evaluation of spoken dialogue systems. In *Proceedings of the Human Language Technology Conference*, 2001.

M. Walker, D. Litman, C. Kamm, and A. Abella. PARADISE: a framework for evaluating spoken dialogue agents. In *Proceedings of the 35th Annual Meeting of the Association of Computational Linguistics*, pages 271–280, 1997.

M. Walker, J. Fromer, and S. Narayanan. Learning optimal dialogue strategies: a case study of a spoken dialogue agent for email. In *Proceedings of the 36th Annual Meeting of the Association of Computational Linguistics*, pages 1345–1352, 1998.

M. Walker, C. Kamm, and D. Litman. Towards developing general models of usability with PARADISE. *Natural Language Engineering*, 6(3), 2000.

M. Walker, J. Aberdeen, J. Boland, E. Bratt, J. Garofolo, L. Hirschman, A. Le, S. Lee, S. Narayanan, K. Papineni, B. Pellom, B. Polifroni, A. Potamianos, P. Prabhu, A. Rudnicky, G. Sanders, S. Seneff, D. Stallard, and S. Whittaker. DARPA Communicator dialog travel planning systems: the June 2000 data collection. In *Proceedings of Eurospeech*, 2001a.

M. Walker, R. Passonneau, and J. Boland. Quantitative and qualitative evaluation of Darpa Communicator spoken dialogue systems. In *Proceedings of the 39th Annual Meeting of the Association for Compuational Linguistics*, pages 515–522, 2001b.

M. Walker, A. Rudnicky, R. Prasad, J. Aberdeen, E. Bratt, J. Garofolo, H. Hastie, A. Le, B. Pellom, A. Potamianos, R. Passonneau, S. Roukos, G. Sanders, S. Seneff, and D. Stallard. DARPA Communicator: cross-system results for the 2001 evaluation. In *Proceedings of the International Conference on Spoken Language Processing*, 2002.

W. Ward. Integrating semantic constraints into the SPHINX-II recognition search. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 17–20, 1994.

P. Werbos. Approximate dynamic programming for real-time control and neural modeling. *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, pages 493–525, 1992.

J. Williams and S. Young. The SACTI-1 corpus: guide for research users. Technical report, University Of Cambridge, United Kingdom, May 2004.

J. Williams and S. Young. Partially Observable Markov Decision Processes for spoken dialog systems. *Computer Speech and Language*, 21(2):231–422, 2007.

S. Young. ATK: an application toolkit for HTK, version 1.4. Technical report, University of Cambridge, United Kingdom, 2004.