The Knowledge Engineering Review, Vol. 24:4, 375–408.
doi:10.1017/S0269888909990166

Recent research advances in Reinforcement Learning in Spoken Dialogue Systems

MATTHEW FRAMPTON[1] and OLIVER LEMON[2]

[1]*Center for the Study of Language and Information, Stanford University, Stanford, CA 94305-4101, USA; e-mail: frampton@stanford.edu*
[2]*School of Mathematical and Computer Sciences, Heriot Watt University, Edinburgh EH14 4AS, UK; e-mail: o.lemon@hw.ac.uk*

Abstract

This paper will summarize and analyze the work of the different research groups who have recently made significant contributions in using Reinforcement Learning techniques to learn dialogue strategies for Spoken Dialogue Systems (SDSs). This use of stochastic planning and learning has become an important research area in the past 10 years, since it promises automatic data-driven optimization of the behavior of SDSs that were previously hand-coded by expert developers. We survey the most important developments in the field, compare and contrast the different approaches, and describe current open problems.

1 Introduction

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). One 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 could be true for human–computer interactions. Another motivation is the use of speech for interaction in scenarios where a person's hands and eyes are busy or otherwise unavailable (e.g. while driving). However, there are obviously many serious challenges in developing a SDS, and one of these challenges concerns how to design the *dialogue strategy*. The dialogue strategy specifies which dialogue action (e.g. ask a question or confirm information) the system should take next based on its knowledge of the current dialogue context, and it is implemented by a component called the *Dialogue Manager* (*DM*). Designing a dialogue strategy by hand, even for a relatively simple SDS, can be a difficult and time-consuming process. The limitations of a SDS (e.g. unreliable speech recognition and understanding) mean that human–machine dialogue is quite different in nature to human–human dialogue (Fraser & Gilbert, 1991), and that action choices must be made based on sometimes highly imperfect representations of the dialogue context—representations that potentially include large amounts of information, varying amounts of which could be either relevant or irrelevant, and either accurate or inaccurate. Thus, designers of a dialogue strategy may need to spend a great deal of time anticipating how potential users will interact with the system, and/or on repeated testing and refining. In addition, when designing a dialogue strategy by hand, there is no way of guaranteeing that the result is the best design possible.

1.1 The Reinforcement Learning approach

Due to the difficult nature of designing dialogue strategies, researchers have begun to investigate how automatic *machine learning* can be applied to the problem. Machine learning is a broad

sub-field 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 data sets. In dialogue strategy design, the goal is to choose actions that maximize the chances of a dialogue reaching a successful conclusion, and this has led previous researchers to use a machine-learning approach that involves modeling a dialogue strategy as a sequential decision problem called a Markov Decision Process (MDP).

In a MDP, a stochastic system interacting with its environment through actions is described by a number of states $\{s_i\}$ in which a given number of actions $\{a_j\}$ can be performed. For SDSs, the states represent the possible dialogue contexts (e.g. how much information we have so far obtained from the user) and the actions are system dialogue actions (such as 'greet' or 'present options'). Each state–action pair is associated with a transition probability $\mathcal{T}^a_{ss'}$: the probability of moving from state s at time t to state s' at time $t+1$, after having performed action a when in state s. This transition is also associated with a reinforcement signal (or reward) r_{t+1}, which describes how good the result of action a was when performed in state s. For SDSs, these reward signals are most often associated with task completion and dialogue length. If we denote the expected immediate reward by $\mathcal{R}^a_{ss'}$, then $\{\mathcal{T}, \mathcal{R}\}$ defines the dynamics of the system.

To control a system described in this way, one then needs a strategy or policy π mapping all states to actions: $\pi(s) = P(a|s)$ (or $\pi(s) = a$ if the strategy is deterministic). In this framework, a Reinforcement Learning (RL) agent is a system, which aims at optimally mapping states to actions, that is, finding the best strategy π^* so as to maximize an overall reward R which is a function (most often a weighted sum) of all the immediate rewards. Trial-and-error search and learning from delayed reward are two key characteristics of RL.

We can distinguish two basic learning approaches for solving such a system, and these have been referred to in the dialogue strategy learning literature as the *model-based* approach, and the *simulation-based* approach (Schatzmann *et al*., 2006). The model-based approach involves estimating the state transition probabilities from a corpus of dialogues and then computing an analytical solution by Dynamic Programming (DP) (Sutton & Barto, 1998). In practice, this approach is only used to learn the optimal action in certain specific states. We call this learning of 'partial strategies', since decisions in some states are still specified by hand. If the corpus is to be suitable for model-based learning of 'full strategies' (i.e. an action for every possible state), then it should ideally contain exploratory data for all states, but unless the state representation is very simple, it is highly unlikely to do so. This is because collecting dialogues with real as opposed to simulated users is costly in terms of time and possibly money, and in any case, real users cannot be expected to interact with a system that is exploring different actions in every state, many or most of which will be unreasonable. Even if the corpus does contain exploratory data for every state, there is then the issue of DP's high computational expense to contend with.

The alternative *simulation-based* approach involves using an RL algorithm, which is able to learn from sample returns[1]. Such an algorithm can learn a dialogue strategy as the DM interacts with a stochastic *user simulation*. A user simulation is a predictive user model for simulating user responses, and can be produced by training on a dialogue corpus using Supervised Learning (SL)[2]. For this approach to be effective, it is necessary for the user simulation to be accurate and to generalize to unseen dialogue situations (e.g. using a smoothing technique). Producing such a simulation is a significant challenge, but assuming that this is achieved, the simulation-based approach then has the advantage that a very large number of training dialogues can be generated, meaning that the state space can be explored much more exhaustively (including strategies not in the original data set), and 'full' instead of 'partial strategies' can be learned.

[1] For in-depth discussions of technical details such as temporal difference learning, Monte Carlo learning, eligibility traces, and Q-values, we refer the reader to Sutton and Barto (1998).

[2] Supervised Learning (SL) algorithms are machine-learning algorithms, which generate a function that maps inputs to desired outputs.

1.2 Overview

Several research groups have been working in this area in the past 10 years (see Table 1), and significant progress has been made. Earlier research (1998–2002) mostly applied the model-based approach, while later research has applied the simulation-based approach. Note that Schatzmann *et al.* (2006) has previously surveyed different user simulation approaches in using RL for dialogue strategy design. By contrast, in this article we will survey the different RL systems themselves (including recent advances such as Frampton and Lemon (2008) and Henderson *et al.* (2008)), highlight the main advances, and point out open problems.

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

In the case of Walker (2000) (Section 4), it becomes necessary to introduce the PARADISE (PARADIgm for System Evaluation) framework for evaluating dialogue systems Walker *et al.* (2000) (Section 4.2), because Walker (2000) use PARADISE scores as reward.

Table 1 Timeline of previous research on Reinforcement Learning of dialogue strategies for Spoken Dialogue Systems

Research publications	Summary
Levin and Pieraccini (1997)	**Proof-of-concept** that a DM can be modeled as a MDP and RL applied to learn a dialogue strategy.
Singh *et al.* (1999, 2002)	Learned/tested partial ISF strategies with real users; state features for slot-status and which ASR grammar used last.
Walker (2000)	As for Singh *et al.*, but using **PARADISE reward:** predicts **user satisfaction** from dialogue efficiency/quality and 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 and Renals, but **probabilities for US and ES learned from data**; US and ES evaluated (unconvincing).
English and Heeman (2005)	Learned **both** the US and system strategies simultaneously via RL, hence problem over accuracy of US.
Paek and Chickering (2005)	Learned versus simple strategies with MDPs, **non-Markov models** and 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, 2008) (HLG)	Hybrid RL/SL to learn **full ISF** strategy versus large state-space from fixed dataset; **real user** tests and comparison to hand-coded strategy; no insights for which state features important and why.
Frampton (2008)	RL of **full ISF** strategy with **recent DAs**; real user tests and comparison to hand-coded; DAs produce better repair strategies.

DM = Dialogue Manager; MDP = Markov Decision Process; RL = Reinforcement Learning; ISF = Information-Slot-Filling; ASR = Automatic Speech Recognition; PARADISE = PARADIgm for System Evaluation; U/ES = User/Error Simulation; FS = Feature Selection; SL = Supervised Learning; DA = Dialogue Act.

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.

1.3 Related work: Partially Observable Markov Decision Processes

There is also an important and more recent strand of research (starting with Roy *et al.* (2000) but with significant activity from 2005) which extends the MDP model described above. Roy *et al.* (2000), Thomson *et al.* (2007), Williams (2007), and Williams and Young (2005, 2005a, 2005b, 2006, 2007) use *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. When the true state of the world cannot be uniquely identified (e.g. when using a speech recognizer), a POMDP reasoner must maintain a probability distribution, called the *belief state*, which describes the probabilities of each 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 ASR and NLU are error-prone, this makes them theoretically appealing for dialogue management. However, at present, the use of POMDPs is not widespread, since they are computationally intractable to solve for optimal behavior for dialogue problems of realistic size, and so this article will focus on the range of work done on MDP models and solution methods.

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

For almost all of the systems discussed below, the task of the dialogue system is 'slot-filling': that is, to collect a set of preferences or search constraints from the user (e.g. destination city, preferred food type), and to then retrieve appropriate items from an information source (e.g. database) and present them to the user. In general, dialogue management action decisions such as initiative (should the user or system lead the conversation?) and confirmation strategies are studied by all groups. In addition, all prior research uses 'slot-status' features in dialogue states, that is, for each information slot in the particular domain, whether it is filled or confirmed. 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.

2 Early theory and proof-of-concept: Levin and Pieraccini (1997) and Levin *et al.* (2000)

Levin and Pieraccini (1997) contains the first presentation of the concept of using a MDP and RL to learn a dialogue strategy. Levin *et al.* (1998, 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).

2.1 State features and action set

In learning strategies for the ATIS task, Levin *et al.* (1998, 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 that 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, and so on);
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, for example, 'Do you mind considering other airlines?';
6. close the dialogue.

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—the data is assumed to match the user request, that is, there are no recognition or understanding errors. As we shall see, the majority of previous research described in this survey uses a reward function based on task completion and dialogue length, and the weights are set in a fairly *ad hoc* manner.

2.3 User simulation

Unlike other earlier research between 1998 and 2002, Levin *et al.* (2000) applies the simulation-based approach for using RL to learn a dialogue strategy (see Section 1.1 for definitions of model and simulation-based approaches). Levin *et al.* (2000)'s user simulation is partly stochastic, and is made up of unigram and bigram models, which output 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). 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 ASR and NLU errors. Note that Levin *et al.* (2000) do not use any kind of ASR/NLU error simulation. Levin *et al.* (2000)'s user simulation supplies slot-values rather than abstracting away from them (e.g. 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* behavior, so that within a dialogue, the simulation always supplies the same values for each slot, for example, the simulated user does not change its mind halfway through so that it wants to fly to 'Philadelphia' instead of 'Pittsburgh'.

The simulation used unigram and bigram models for the following, 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 1 and 2 in the above list, 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. 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 behavior;
- consistent/goal-driven behavior;
- some probabilities are derived from appropriate data, some are hand-coded;
- the user simulation and system communicate via abstract representations of utterances, e.g. 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, for example, Scheffler and Young (2002) and Henderson *et al.* (2008).

2.4 Experimental results

Levin *et al.* (2000) reports that by the end of training, the system had explored 111 states, and converged to the optimal strategy. A summary of how the strategy behaves is provided. First, 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, meaning that 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 then retrieves again. If there are too many flights in the data set, then it asks for additional constraints (e.g. the departure time) and 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 handcrafted 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 these 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, evaluation methodologies have become more sophisticated in recent years, for example, Lemon *et al.* (2006a).

2.5 Summary

This early work was very important in pioneering the basic concepts and methods in RL for 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 Reinforcement Learning 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 and Pieraccini (1997) and Levin *et al.* (2000) in that it uses the model-based as opposed to simulation-based

approach for using RL. Recall from Section 1.1 that this means that the exploratory data used by the reinforcement learner is generated by real, not simulated user interactions, and that partial rather than full strategies are learned (action choices are learned only in certain states, not all states). Since the probabilities for Levin *et al.* (2000)'s stochastic user simulation 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. Later in this survey, we will see further examples of previous research in which the model-based approach has been applied (e.g. Walker, 2000; Tetreault & Litman, 2006).

Singh *et al.* (1999) describes six 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 MDP, and then uses a standard 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 versus 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 the related work of Singh *et al.* (2002) (see also Litman *et al.*, 2000), sample dialogues between real users and another slot-filling system called NJFun were collected. 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.

3.1 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, that is, one which filled all 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, for example, *timeouts*, *resets*, *user requests for help*, and other indicators that the dialogue is potentially in trouble. 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 two 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 Levels (CL)[3], 3, 4 for explicitly confirmed and disconfirmed);

[3] 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.

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.

3.2 Reward functions

Singh *et al.* (1999, 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 labeled by a +1 in the case of a completed task, or −1 otherwise.

3.3 Experimental results

The main findings of the six 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;
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 were 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 a two-sample t-test over subject means). 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 handcrafted strategy;
- the strategy was only learned 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.

4 Predicting user satisfaction and defining reward: Walker (2000)

Like Singh *et al.* (1999, 2002), Walker (2000) also describes an experiment which applies the model-based learning approach (see Section 1.1), and hence in which a partial slot-filling strategy is learned from exploratory data generated by real-user interactions. However, Walker (2000) makes an important novel contribution with respect to the reward function by proposing

'PARADISE', which is a method for predicting a final *user satisfaction* score based on metrics that can be easily collected by the system itself. PARADISE will be introduced in detail in Section 4.2. The first advantage of the PARADISE approach is that the reward function is data-driven, and the second is that it goes beyond a reward function that is based only on task completion and dialogue length. Task completion and dialogue length are clearly very important in evaluating a system's performance in a dialogue, but it seems likely that other factors should ideally be considered as well. Possible examples include whether the user enjoyed their experience in conversing with the system, and whether they are keen to use the system again in the future. Such additional factors are not necessarily perfectly correlated with task completion and dialogue length.

The particular 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; Sutton & 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 summarizing folders, and used fixed strategies elsewhere, for example, for requesting and providing information. Hence RL is being used to learn action choices for initiative, reading messages and summarizing folders. The learned strategy is tested in 18 dialogues with six new users. Training and testing dialogues are evaluated by a user satisfaction score, which is computed from the user's answers to questions about how the dialogue went.

4.1 State features and action sets

As stated above, Walker (2000) explores different action choices with regard to initiative, and summarizing 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 fixed 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 recognizable response, then the system will take the initiative to repair the situation before switching back to user-initiative actions. ELVIS explores three alternative summarization actions:

1. The Summarize-Both (SB) action uses both the sender and the subject attributes in the summary.
2. The Summarize-System (SS) action summarizes 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 summarize by.

Finally, ELVIS explores two different Read actions for reading multiple messages following a user request, for example, 'Read my messages from Kim.':

1. The Read-First (RF) action involves summarizing all 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 (2000) uses the following state features:

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, while the *InitStrat* (*I*), *SummStrat* (*S*) and *ReadStrat* (*R*) features record whether ELVIS has already employed a particular initiative, summarize, or reading strategy in the current dialogue, and if so, which strategy it was. The *TaskProgress* (*P*) feature then tracks how much progress the user has made in completing the experimental task, and the *CurrentUserGoal* (*G*) feature corresponds to the system's belief about what the user's current goal is. The *WhichSelection* (*W*) feature records whether the system knows what type of selection criteria the user would like to use to read her messages, and 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 records how many messages match the user's selection criteria, and the *Confidence* (*C*) feature is a threshold variable indicating whether the speech recognizer'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. Finally, 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 (2000) reports that these state features produced 110 592 possible states but that not all of these states occur.

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

1. Actual Task Completion (0 or 1);
2. Perceived Task Completion (1 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 modeling technique, *multivariate linear regression*[4], 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;

[4] 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.

2. Dialogue Quality Metrics—mean recognition score, number of timeouts, ASR rejections, user requests for help, user requests to restart the dialogue, barge-ins.

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

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

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^2 [5]. How well the model generalized to unseen data was tested with a ten-fold cross-validation[6]—90% of the training dialogues were randomly sampled and the goodness of fit was tested on the remaining 10%. 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 generalize 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 1.1).

5 Learning with simulated users and Automatic Speech Recognition errors: Pietquin and Renals (2002)

This section describes the work of Pietquin and Renals (2002) (see also Pietquin, 2004), which uses a 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 ASR system is introduced into the RL environment. This ASR simulation simulates both speech recognition errors and CLs. However, the probabilities for the user and ASR simulations are not learned from data.

5.1 State features and action sets

For the computer-dealing application, each state is represented with a confidence feature for each of the seven slots, and the possible values for each of these features are: 0 if the slot is unfilled, 1 if the confidence is low or 2 if it is high. This means that there are 3^7 possible states.

The action set contains six generic actions:

1. Greeting, e.g. 'How may I help you?'.
2. Ask: ask to constrain the value of a slot.

[5] 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.

[6] 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.

3. Conf: ask to confirm the value of a slot.
4. Relax: ask to relax the value of a slot.
5. Database query: perform a database query.
6. Close: present data and close the dialogue session.

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

5.2 *Reward functions*

After each turn, the reward is a weighted sum of the following:

1. a negative reward if the final state has not yet been reached;
2. number of database accesses;
3. number of presented records;
4. confidence level of the current user's utterance;
5. a 'function of the modeled user's satisfaction'.

Pietquin and Renals (2002) does not provide any details about the 'function of the modeled user's satisfaction'.

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 behavior. 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 (2002) simulation, none of the probabilities are learned from data—they are all supplied via intuition.

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

1. probabilities associated with responses to greeting, e.g. $P(n|Greeting, g)$, $P(u^\alpha|s^\beta, k_t, g)$;
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 this probability is set so that as the number of times the simulation has to supply a particular slot-value increases, the chances of it hanging up also increase.

The novel feature of this work is that it uses a stochastic ASR simulation that simulates ASR errors and outputs CL scores. Just as for the user simulation, the probabilities 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 1 represents a CL distribution output from a real ASR system obtained using

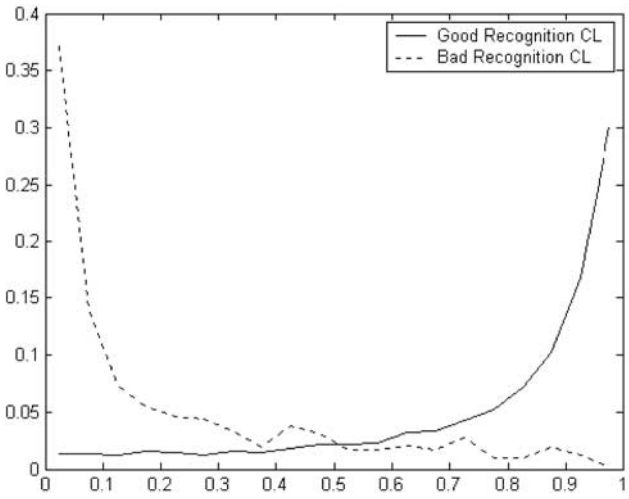

Figure 1 A Confidence Level (CL) distribution for good and bad recognitions (taken from Pietquin and Renals, 2002)

some of its training data (isolated words). As the two curves cover each other, it is unavoidable to reject some well-recognized 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 attribute–value pairs from the user simulation, which it then splits into individual attribute–value elements. The probability of it simulating an ASR error for a particular attribute–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 attribute-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.

5.4 Experimental results

Pietquin and Renals (2002) reports that after several thousand simulated dialogues, the learned strategy stabilizes 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 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 (2002) reports 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 such as 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 and Pieraccini (see Section 2.4)—for example, there is no quantitative evaluation of the learned strategy or testing with real users. We now present similar work.

6 A goal-directed user simulation and error model with probabilities learned from real data

Scheffler and Young (2002) use Q-learning (an off-policy TDL algorithm; Sutton and Barto; 1998), Eligibility Traces (ETs) (Sutton & Barto, 1998), a goal-directed user simulation, and a

system error model to learn a dialogue strategy for a slot-filling cinema information SDS. The error model simulates both ASR and 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 handcrafted baselines.

6.1 State features and action sets

Scheffler and Young (2002) used five 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 five 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 that 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. 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).

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.

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 behavior. 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. Nodes are probabilistic or deterministic choice points relating to user behavior. The deterministic choices are based on user state and so ensure consistent goal-directed behavior. The parameters for the probabilistic user behavior choice points and error model are estimated based on training data collected with a prototype cinema information SDS. For the probabilistic user behavior 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, while for the error model, an event would be whether or not a recognition/understanding error occurred. In all

cases, probabilities are estimated from the data using *Maximum Likelihood Estimation* (*MLE*)[7]. 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 this evaluation is not very convincing, since it is only based on the gross metric of dialogue duration.

6.4 Experimental results

In evaluating the learned strategies, Scheffler and Young (2002) used two handcrafted strategies as baselines. The first used a small state-space—slot-status features and feature 1 from Section 6.1, while the second used a large state-space, which included additional features such as a confidence feature, and a feature that counts how many times each slot has been asked. The performance of the learned and handcrafted strategies was evaluated in test dialogues with the user simulation according to the reward function of Section 6.2. Of the learned strategies, those learned with larger state-spaces tended to outperform those learned with smaller ones, but the improvement was not great. The learned strategies outperformed the small-state-space-handcrafted strategy by a large margin, and performed roughly as well as with the large-state-space-handcrafted 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:

- testing and training are performed with the same user simulation;
- no tests with real users were performed;
- the quality of the user simulation 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 that treated both the user simulation and DM as RL systems.

7 Reinforcement Learning for both user and system: English and Heeman (2005)

English and Heeman (2005) use RL to learn the system dialogue strategy for a collaborative task, which requires the system and user to agree on five 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, for example, '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 and system strategies simultaneously. The same RL algorithm is used for both strategies—an on-policy 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 behavior 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 Section 7.4 will explain how problems arise if it is to be used for learning dialogue strategies for interacting with real users.

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

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, and *I-Proposed* 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, while *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 which 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*.

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:

$$o(S, I) = w_1 S - w_2 L \tag{1}$$

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

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 & Barto, 1998);
2. the Q-values (see Sutton & Barto, 1998) for the different actions in the current state.

7.4 Experimental results

English and Heeman (2005) report that they succeeded in learning system and user dialogue strategies that achieved comparable performance with handcrafted system and user strategy pairs. The authors also claim that the learned system strategies are robust—when the learned system strategies 'conversed' with the handcrafted user strategies, the resulting dialogues had comparable solution quality to what the handcrafted 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 RL problem becomes more complex with two interacting learning agents.

However, if the goal is to produce system strategies for interacting with real users, the approach of using RL to simultaneously learn both the system and user strategies seems to be problematic. To learn a strategy that works well with real users, we need to train with a simulation that accurately simulates real users. For example, we will see in the discussion of Frampton and Lemon (2006, 2008) in Section 11 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 (2005) cannot be relied upon to produce optimal strategies for real users.

We now discuss about the work which presents a method for automatically selecting features to include in the state, and which investigates alternatives to MDPs for modeling the dialogue management problem.

8 Alternative learning approaches and feature selection: Paek and Chickering (2005)

Alternatives to MDPs for modeling 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. Paek and Chickering (2005) are interested in whether it is possible to learn better strategies with these alternative models, that is, strategies that obtain higher reward. Paek and Chickering (2005) also present a data-driven method for identifying which features should be represented in the MDP state. This data-driven method generalizes 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 modeling that facilitates decision-theoretic optimization. There are techniques for learning the parameters and structure of a Bayesian Network that have been extended for influence diagrams (Heckerman, 1995; Chickering & 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 three system turns due to the typically low tolerance users have in command-and-control settings for extended repairs. The authors state that using DP for a state space that includes more than a handful of variables can be computationally expensive. Hence, they use *forward sampling* to approximate the DP solution for the MDP (Kearns *et al.*, 1999).

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 scores, and so on.
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, and so on.
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, and so on.

Four different DM models were constructed:

1. a zero-order Markov model;
2. a first-order Markov model, i.e. an MDP;

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

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 scores;
2. the mean confidence score in the *n*-best list;
3. sum of all confidence scores from *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 confidence scores, which makes sense given that most handcrafted dialogue management strategies use some kind of confidence threshold for taking actions, for example, 'Do the top recognized command if its confidence is greater than 95%'. When applied to the other models, Paek and Chickering (2005)'s method learned different feature sets, for example, feature sets including between-utterance 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 8.3 will provide more details about this training simulation environment.

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.

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 recognized 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 8.1.

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 (2005) 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 that 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*) (Hall, 1999) offer an alternative.

The work described here in this section proposed a method for automatically identifying features that should be included in the state. We now present some alternative work that 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.

9 Feature selection in Reinforcement Learning for tutorial dialogue systems: Tetreault and Litman (2006)

Like Singh *et al*. (1999, 2002) and Walker (2000), Tetreault and Litman (2006) also use the model-based learning approach and hence learn a partial strategy from data generated by real user interactions. However, the work of Tetreault and Litman (2006) has two main novel features. First, 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 SDS which acts as a tutor for undergraduate-level physics, that is, the ITSPOKE SDS (Litman & 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, that is, 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 performance thus far on the current problem. Note that these features could potentially be important to any SDS. Tetreault and Litman (2006) focuses 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 generalize, because in any domain, asking users questions of varying complexity is likely to elicit different responses.

The action choices are learned by applying the 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 five different physics problems, so giving a total of 100 dialogues. Before

each session, the student read physics material for 30 min and then took a pre-test 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. a question) is based only on the correctness of the student's last answer, and once the student has successfully completed the dialogue section, 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 *normalized learning gain*:

$$Gain = \frac{posttest - pretest}{1 - pretest} \tag{2}$$

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

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 (2006) 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, for example, a hint or a restatement.

9.2 Reward functions

The 10 students with the highest normalized learning gain scores were labeled high learners and their respective five dialogues were given a final reward of 100. The other ten students were labeled low learners and their respective dialogues were given a final reward of −100.

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 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, then 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 (2006) 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: (i) *Diff's* (ii) *% Policy Change* and (iii) *Expected Cumulative Reward* (*ECR*). The number of *Diff's* is the number of states whose learned action differs from the baseline. *%Policy Change* weight 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 that it affects occur less frequently. However, *% Policy Change* still does not take into account the effect which the state feature has on the V-values (for each state the expected total future reward when starting from that state and following the learned strategy; Sutton & Barto, 1998). The third metric, *ECR*, does take this into account. *ECR* is calculated by normalizing the V-value of each state by the number of times it occurs as a start state in a dialogue and the summing over all states. Tetreault and Litman (2006) 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 handcrafted strategy.

This work then demonstrates a simple but useful approach to performing feature selection for state features. The use of automatic feature selection methods such as CFS subset evaluation (Hall, 1999) is very much an open problem.

10 Learning in large state-spaces via a generalization method: Henderson *et al.* (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', in which learning in large state-spaces becomes intractable. One approach to large state-spaces is to use generalization 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 generalization method called *linear* function *approximation* in order to cope with a very large state-space that includes linguistically motivated features—the entire dialogue history represented as 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 and/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 4.2. Prior to applying RL, an automatic system was used to assign DATE DA tags (Walker *et al.*, 2001b) to the user utterances, and to compute *information states* (representations of the dialogue context), for each point in the system. Section 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 that combines RL with 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 data set, 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.

10.1 Dialogue Act Tagging Scheme

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

1. Speech act;
2. Task/Subtask;
3. Conversational domain.

The *Speech act* dimension characterizes 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 subtasks can be quantified. Below are the different tasks/subtasks:

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 characterizes 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, for example, 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 *Instruction*, utterances directed at shaping the user's behavior and expectations about how to interact with a machine.

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) to track dialogue context and several Open Agent Architecture (OAA) agents (Cheyer & 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. 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.

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 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. Section 11 will describe work by Frampton and Lemon (2006) which also evaluates learned strategies with the HLG05 reward function. This is so as to enable comparison with the Hybrid Strategy and the handcrafted COMMUNICATOR systems.

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. ASR and NLU errors are incorporated since the model is built from the user utterances as they are recognized by the ASR and NLU components of the original COMMUNICATOR systems. Note that the user simulation does not provide instantiated slot-values, for example, 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 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 approximation simulation, and also like the linear function approximation 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[9]. However, user simulations which generate a new action based on some amount of dialogue history (such as *n*-gram simulations) ought to be able to generate actions that bear common traits with previously generated actions. The degree to which *n*-gram simulations are able to achieve this is obviously limited by the size of *n*. Note that the *n*-gram simulations of HLG were also used for training and testing strategies by Frampton and Lemon (2006, 2008), and so will be mentioned again in the discussion of this work in Section 11.

In order to establish their accuracy, the linear function approximation simulation and the *n*-gram simulations were evaluated on the 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

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

SL (with linear function approximation) and 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 6.3). Recall that Scheffler and Young (2002) only showed that their user and error models simulate different scenarios well enough to perform relative predictions of their durations.

10.5 Experimental results

A dialogue strategy was learned from the annotated COMMUNICATOR data, and then tested with a user simulation trained on the same data. The strategy outperformed all the COMMUNICATOR systems—it scores 35.42% higher than the average COMMUNICATOR system. The authors find that when the relative importance of the RL and SL components are adjusted, the best hybrid policy performs 302% better than the standard RL policy, and 1.4% better than the supervised policy. The standard RL technique (i.e. without the SL component to constrain exploration) has an enormous policy space to search, so it is clear that the optimal strategy has not yet been found with this method. The main advantage of the hybrid method in fact comes from its supervised component: a 'multi-version' system is being learned which blends the best aspects of the original COMMUNICATOR systems. RL, in the hybrid method, is able to improve slightly, but significantly, over this SL policy.

Lemon *et al.* (2006a) shows that the Hybrid Strategy also outperforms a state-of-the-art handcrafted system when tested on real users with the 'TownInfo' SDS (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. This 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. Such an outcome is welcome because collecting dialogue data can be very time-consuming.

These results 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 learned strategy is better than what could have been learned using a small state-space and standard RL without a generalization method. In the next section, we go on to describe work that addresses these issues.

11 Investigating the usefulness of Dialogue Acts: Frampton and Lemon (2006, 2008)

With the exception of HLG (Henderson *et al.*, 2005, 2008), we have seen thus far in this survey that previous researchers have given only a very limited amount of contextual information to the reinforcement learner, and have neglected linguistically motivated features—for slot-filling SDSs, Levin and Pieraccini (1997), Pietquin and Renals (2002), Scheffler and Young (2001) and Singh *et al.* (2000, 2002) have used only slot-based features, for example, whether a slot has been filled or confirmed, or the confidence score for a supplied value. Note that this is also true of the more recent POMDP-based approaches (Williams & Young, 2005, 2006, 2007; Williams *et al.*, 2005a, 2005b; Thomson *et al.*, 2007; Williams, 2007).

HLG, because of their use of a generalization technique, were able to give the reinforcement learner a much larger feature set, and this included linguistically motivated features, namely DA features for the whole dialogue history. However, as already stated at the end of Section 10, this

work did not tell us which of the many contextual features were important or why. In this section, we now focus on the work of Frampton (2008) and Frampton and Lemon (2006, 2008) which investigates whether recent DAs are useful in learning slot-filling strategies, and given initial results which show what they are and why they are useful.

In Frampton and Lemon (2006) (see also Frampton, 2008), full strategies are learned with user simulations which are *n*-gram models derived from COMMUNICATOR data (these simulations were introduced in Section 10.4). The RL algorithm used is a TDL algorithm called *Sarsa*(λ), (see Sutton & Barto, 1998). The learned strategies are tested in simulation, and are subjected to quantitative and qualitative analysis in order to understand the effect of the DAs. This analysis suggests that the DAs are improving the learned strategy by producing better *repair strategies*. Repair strategies are the parts of the dialogue strategy, which attempt to get the dialogue back-on-track when progress stalls, usually due to an ASR/NLU error, and hence the slot-status features are unchanged. The experiments of Frampton and Lemon (2008) (see also Frampton, 2008), then lead on from those of Frampton and Lemon (2006). Having learned strategies in the same way as Frampton and Lemon (2006), a state-of-the-art handcrafted strategy, and strategies learned with and without additional DA-state features are all tested on real users with the 'TownInfo' SDS (Lemon *et al.*, 2006b), and their performance is compared. Given positive results, Frampton and Lemon (2008) goes on to describe two more experiments with the *n*-gram simulations which further investigate why the DA strategies are better. The first investigates whether the DAs are only proving useful for learning repair strategies for when the slot-status features are unchanged, while the second experiment investigates whether and how the recent DAs affect *which* repair strategy to apply.

11.1 State features and action sets

In Frampton and Lemon (2006), strategies are learned for a 4-slot system, but in Frampton and Lemon (2008), they are learned for a 3-slot system because the 'TownInfo' SDS (Lemon *et al.*, 2006b) used in the real user experiment has three information slots. Both Frampton and Lemon (2006, 2008) use the following three state representations:

1. Slot-status baseline strategy: a slot-status feature for each of the slots, with the values 'empty', 'filled' and 'confirmed';
2. DA1 strategy: the slot-status features plus the DA(s) of the last user turn;
3. DA2 strategy: the slot-status features plus the DAs of both the last user and system turns.

Below is a list of all of the different actions that the RL DM can take and must learn to choose between based on the context, (i = the total number of slots):

1. an open question e.g. 'How may I help you?';
2. ask the value for any of slots $1 \ldots i$;
3. explicitly confirm any of slots $1 \ldots i$;
4. ask for the ith slot whilst implicitly confirming slot value $i-1$ or $i+1$[10].
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. It is only possible to attempt to confirm non-empty slots.

As stated above, following the real user experiment, Frampton and Lemon (2008) describes two additional simulation experiments that further investigate how the DAs improve the learned

[10] 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, for example, 'So you want to fly from Edinburgh to where?'.

strategies. These experiments involve learning three new strategies using three new state representations, which are listed below. The $DA1^{\beta}$ and $DA2^{\beta}$ strategies are used in the first experiment to investigate whether the recent DAs only improve the learned strategy through repair strategies for when the last user turn fails to change the slot-status features. The Slot Features Unchanged (SFU) strategy is used in the second experiment to investigate whether the recent DAs also affect *which* is the best repair strategy to apply.

1. $DA1^{\beta}$ Strategy: Same as DA1 except that the last user turn DA(s) feature was changed to only take a DA value when the slot-status features are unchanged.
2. $DA2^{\beta}$ Strategy: Same as DA2 except that the last user and system DA features were changed to only take DA values when the slot-status features are unchanged.
3. SFU Strategy: the slot-status features plus a binary feature that records whether the slot-status features have been changed by the last user turn.

In investigating how the recent DAs affect *which* repair strategy is best to apply, the second experiment also involves testing strategies which follow the learned DA2 strategy except when the slot-status features are unchanged, whereupon they use handcrafted repair strategies. More details will be provided in Section 11.4.

11.2 Reward function

Frampton and Lemon (2006) uses 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. dialogue system passes the call to a human operator: −50.
5. each system turn: −5.

This reward function rewards confirmed slots because the slot-values are more likely to be correct if they are confirmed. 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, and the DM asks for confirmation of all of the slots, the user gives it, and the DM submits a database query). The reward function is called 'all-or-nothing' because it only awards positive reward when a database query is made with all four slots confirmed. An alternative kind of reward function would give some positive reward for each confirmed slot. Such a reward function can be referred to as a 'partial' reward function. An 'all-or-nothing' training reward function was preferred here because the *n*-gram simulations do not simulate unobtainable slot-values, and given that this is the case, an 'all-or-nothing' reward function will produce faster learning. Frampton and Lemon (2005) provides an example of this. Frampton and Lemon (2008) also learns strategies using a reward function which gives +100 for a database query with all slots confirmed, and −5 for each system turn.

For evaluating strategies, Frampton and Lemon (2006) used both the reward function described above, and the 'HLG05' reward function, which was used by HLG and was introduced in Section 10.3. Frampton and Lemon (2006) used the 'HLG05' reward function in evaluation in order to enable performance comparisons between their own learned strategies, and the Hybrid Strategy of HLG and the handcrafted COMMUNICATOR systems.

11.3 Stochastic user and error simulations

Both Frampton and Lemon (2006, 2008) trained and tested strategies using the *n*-gram simulations already described in 10.4. Recall that the *n*-gram probabilities are learned from the COMMUNICATOR data, and that ASR and NLU errors are incorporated since the model is built from the user utterances as they are recognized by the ASR and NLU components of the original COMMUNICATOR systems. Frampton and Lemon (2006, 2008) used 4- and 5-gram simulations.

For each state representation, a strategy was learned with the 4-gram and then tested with the 5-gram, and then vice-versa. With regard to the state representation, note that DAs of turns further back than the last system turn were beyond the context window to which the 4-gram user simulation can be sensitive, and so could not have improved the learned strategy.

11.4 Experimental results

Initial simulation experiments. Frampton and Lemon (2006) reports that in testing with the *n*-gram simulations, DA2 performs significantly better than DA1, which performs significantly better than the Slot-Status Strategy. This is as a result of significant improvements in dialogue length (8.84 system turns versus 9.25 versus 9.7), rather than task completion—since the *n*-gram simulations do not simulate unobtainable slot-values, all three of the learned strategies are always able to fill and confirm all of the slot-values. Analysis strongly suggests that the improvement in dialogue length is due to better repair strategies for states in which dialogue progress has stalled and hence the slot-status features are unchanged. Such states are caused by ASR rejections or user utterances that are recognized as out-of-domain[11], and the better repair strategies of DA1 and DA2 are more likely to ensure that the same is not also true in the next user turn. Whereas the Slot-Status Strategy always repeats its last question/confirmation, the DA strategies use additional repair strategies, for example, 'switching focus' (asking/confirming a different unfilled/unconfirmed slot), and giving help. Frampton (2008) and Frampton and Lemon (2008) also mention 'backtracking' (re-asking a filled slot). Frampton (2008) provides more detailed analysis, and points out that the findings make sense given that previous research on repair strategies has shown that in such contexts, repeating the last action is often not the best choice (e.g., Skantze, 2003; Bohus & Rudnicky, 2005).

According to the HLG05 reward function, the learned strategies also outperform the handcrafted COMMUNICATOR systems and the Hybrid Strategy of HLG (191.16 on average per dialogue versus 103.6 and 140.3 respectively). Note that the comparison with the handcrafted COMMUNICATOR systems must be taken with-a-pinch-of-salt, because these systems were tested with real users (who could fail to fill slots) rather than in simulation. On the comparison with the Hybrid Strategy, this performs worse than DA2 because it is almost entirely a supervised strategy (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 that the Hybrid Strategy has an enormous policy space to search, with relatively little data to work with, so it is clear that the optimal strategy has not yet been found with this method. A further potential reason why the Hybrid Strategy performs worse is that it was trained with PARADISE scores rather than with a reward function based only on filling/confirming slots and dialogue length. Recall that PARADISE scores indicate user satisfaction (see Section 4.2 for the list of features). We would expect the HLG and PARADISE reward functions to be directly correlated in general, but this is not necessarily always true. For example, it may well be that in certain circumstances, user satisfaction is higher when fewer slots are filled/confirmed—if there are a large number of speech recognition errors, the process of filling/confirming another slot may cause the user great frustration and so reduce their overall satisfaction.

Real user experiment. In the real user experiment of Frampton and Lemon (2008), DA2 outperforms both the state-of-the-art handcrafted strategy and the Slot-Status Strategy in terms of task completion (+9% average, $p < 0.05$), and dialogue length ($p < 0.05$ versus Slot-Status). It is said that analysis of these experiments emphasized why re-asking or re-attempting a confirmation is often a poor repair strategy. When the last user turn had failed to fill/confirm a slot-value it was usually due to one or more ASR errors. The repetition then employed by the Slot-Status and handcrafted strategies was very likely to frustrate the user, eliciting irritated/hyperarticulate speech which caused further ASR errors, and so longer dialogues and lower task completion. Note that since the strategy was tested here in a different domain from the one in which it was trained, like Lemon *et al.* (2006a), this result suggests that it is possible to learn generic slot-filling strategies.

[11] Utterances that cannot be handled by the system.

How do Dialogue Acts improve the learned strategy? In the first additional simulation experiment in Frampton and Lemon (2008), no significant difference in performance is found between DA1 and $DA1^{\beta}$, or DA2 and $DA2^{\beta}$. Hence, ignoring DAs in contexts where progress is being made in the dialogue has no significant impact on the learned strategy's performance. The value of the DAs here seems to be entirely in learning repair strategies for when the slot-status features are unchanged by the last user turn.

In the second additional simulation experiment, the SFU Strategy is found to perform significantly worse than DA2 ($p < 0.005$), showing that recent DAs not only indicate when a repair strategy is required, but also *which* is best to apply. However, given that previous research, Bohus and Rudnicky (2005) and Skantze (2003) and the real user experiment highlighted that repetition can be a poor repair strategy, it is still possible that choosing the best repair strategy is very simple because any 'sensible' avoidance of repetition will always be optimal. This hypothesis is also investigated by the second simulation experiment. Here any of the repair strategies that emerged from the RL are considered 'sensible', for example, having asked for a slot-value on the last turn, switching focus, giving help or backtracking would be 'sensible'. Hence, in order to investigate this, tests were conducted for three new strategies that follow DA2 until the slot-status features are unchanged, whereupon they always take a sensible action that is different from the last system action. The first switches focus to an unfilled/unconfirmed slot[12], while assuming that the slot-status features remain unchanged, the second alternates between first giving help and then re-attempting the original question/confirmation, and the third between giving help and then switching focus. However all three of these strategies perform significantly worse than DA2 ($p < 0.005$), and so the conclusion is that it is at least not always true that any sensible repair strategy which avoids repetition will be optimal.

This work of Frampton (2008) and Frampton and Lemon (2006, 2008) then shows that recent DAs can be used to learn improved, full, slot-filling dialogue strategies. They improve the learned strategies here by producing more effective repair strategies for when the slot-status features are unchanged by the last user turn. They indicate when a repair strategy is required (when the slot-status features are unchanged), and also which is best to apply. Although repetition is shown to often be a poor repair strategy, the final experiment also shows that it is at least not always true that any sensible repair strategy that avoids repetition will be optimal.

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

12 Summary and open problems

This paper has summarized the previous research in using MDPs, RL, and related methods, to learn dialogue strategies for SDSs.

Most of the research described in this paper involved learning strategies for systems that 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:

- application domains of the SDS: flight-booking, internet-browsing, tutorial, negotiation and so on;
- RL technique: DP versus MC versus TDL, RL with and without a generalization method;
- data set/corpus used;
- state–action space: its size, whether the complete space was explored, i.e. a full strategy, or only part and the rest hand-coded, i.e. a partial strategy, whether there was quantitative and/or qualitative state feature analysis/selection;

[12] If there are no unfilled/unconfirmed slots to switch focus to, the strategy continues to follow DA2.

Table 2 Comparison of work of different research groups

Research group	App	RL tech.	State- actions	Reward	User sims	Error sims	Results
Levin & Pieraccini	S	MC	SF	H	U	–	No
Singh *et al.*	S	DP	SP	H	–	–	No
Walker	S	TDL	SP	P	–	–	No
Pietquin & Renals	S	MC	SF	H	U	S	No
Scheffler & Young	S	TDL	SF	H	U*	S*	No
English & Heeman	N	MC	SF	H	U	–	No
Paek & Chickering	N	DP	SF	H	U	S*	No
Tetreault & Litman	N	DP	SPQ	H	–	–	No
Henderson *et al.*	S	TDL*	LF	P	U*	S*	Yes
Frampton & Lemon	S	TDL	SFQ	H	U*	S*	Yes

'App' column indicates application domain: a (S)lot-filling versus (N)on-slot-filling system.
'RL tech' column indicates Reinforcement Learning technique: Dynamic Programming (DP), Monte Carlo (MC), Temporal Difference Learning (TDL); * = with generalisation method.
'State-actions' column indicates state–action space details: (S)mall versus (L)arge, (F)ull strategy (complete space explored) versus (P)artial strategy (only part explored and rest hand-coded), (Q)uantitative and qualitative state feature analysis/selection.
'Reward' column indicates reward function type: (H)and-coded versus (P)ARADISE (PARADIgm for System Evaluation).
'User sims' column relates to user simulation details: (U)ser sim used; * = data-driven.
'Error sims' column relates to Automatic Speech Recognition/Natural Language Understanding simulation details: (E)rror sim used; * = data-driven.
'Results' column indicates whether the learned strategy was tested with real users and its performance compared to a state-of-the-art handcrafted strategy.

- reward function: data-driven, e.g. PARADISE versus hand-coded;
- user simulations: use of simulations or real user data for training;
- ASR/NLU simulations: whether they are used, parameters set based on data/by intuition;
- experimental results: evaluation with real or simulated users, statistical significance of results.

Table 2 compares the research based on important elements of the experimental methodology.

We now summarize the various options covered in the prior work.

12.1 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 (Walker, 2000; Singh *et al.*, 1999, 2002; Tetreault & Litman, 2006). Due to the smaller number of variables, 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 & Renals, 2002; Scheffler & Young, 2002; English & Heeman, 2005; Paek & Chickering, 2005; Frampton & Lemon, 2006, 2008). A larger number of variables means some approximation to DP must be applied, e.g. MC or TDL.

The exception to this is HLG. Here, a 'hybrid' SL/RL approach is used for learning full strategies from data generated by dialogues with real users. Additionally, a generalization technique is used to cope with the very large state–action space. Like a model-based approach, HLG do not use a user simulation for training, and instead, learn a dialogue strategy from a fixed data set of real user dialogues. However, they do not try to directly model the transition probabilities and apply DP. Instead they use an alternative learning approach that means that they can learn full strategies with a very large state–action space: 'hybrid' SL/RL (TDL) with a generalization method.

One drawback with applying RL to learn dialogue strategies is that it can be very time-consuming to collect the initial data set. However, it should be noted that both Lemon *et al.* (2006a) and Frampton and Lemon (2008) showed that it was possible to learn a strategy in one slot-filling domain (flight-booking) which works well in another (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. 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 slot-value will be correctly recognized—recall that Pietquin and Renals (2002) learned a strategy which asks the slots in an order that is sensitive to WER (see Section 5).

12.2 Comparing the state spaces

In constructing their state space, researchers must try to include as much important contextual information as possible without making it so large that the RL problem becomes intractable. All of the researchers apart from Paek and Chickering (2005) hand-picked their state features. Paek and Chickering (2005) 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) and Scheffler and Young (2002) only used slot-based features, for example, whether a slot is filled, any associated confidence score, and so on. HLG used additional features, for example, features for the complete dialogue history, which produced a very large state-space and necessitated a hybrid SL/RL method with linear function approximation. Frampton and Lemon (2006, 2008) used recent DAs in addition to slot-status features. 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).

12.3 Different reward functions

Most of the researchers use a simple reward function that rewards *task completion* and penalizes *dialogue length*, and the weights for each are set in a fairly *ad hoc* manner. By contrast Walker (2000) and HLG use PARADISE (Walker *et al.*, 2000) user satisfaction scores. Such scores have the advantage that they are data-driven (they were supplied by real users), and secondly, they take account of more than just task completion and dialogue length. PARADISE is a framework for predicting user satisfaction based on metrics, which can be automatically logged by the system, and so future researchers may find it productive to experiment with this framework and perhaps add new and different variables.

12.4 User simulation and error modeling 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 behavior is consistent within a dialogue. The simulations developed by HLG (linear function approximation and *n*-grams derived from COMMUNICATOR data; Walker *et al.*, 2001) 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 (2002) and HLG are learned from data, while the probabilities for the error model of Pietquin and Renals (2002) are supplied using intuition.

Efforts were made to establish the accuracy of the simulations used by Scheffler and Young (2002), HLG (linear function approximation simulation derived from COMMUNICATOR data; Walker *et al.*, 2001a), and Frampton and Lemon (2006, 2008) (*n*-gram simulations derived from COMMUNICATOR). Scheffler and Young (2002) 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, 2005b). 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 SL (using linear function approximation) and 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. The accuracy of the *n*-gram simulations is further established by the positive results in the real user tests of Frampton and Lemon (2008).

12.5 Open problems

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

- Can we develop user simulations that are accurate enough to reliably train RL dialogue managers? What metrics can be used to evaluate user simulations?
- What additional contextual information will it be useful to represent in the state, particularly for non-slot-filling systems which have received less attention so far? Can automatic feature selection methods be applied to dialogue data to provide an automatic method for identifying the important contextual features?
- Is additional contextual information only useful in certain portions of the policy space? If it is, then it could be sacrificed where it is known to be redundant, so helping to reduce the size of the policy search-space.
- Can we develop better reward functions using the PARADISE framework, e.g. are there additional factors that we should consider in evaluating user satisfaction and are these domain-specific or domain-independent?

These questions provide very interesting avenues for future research.

13 Conclusion

We summarized and analyzed the work of the different research groups who have made significant contributions in using RL techniques to learn dialogue strategies for SDSs. We explained the main advantages of the approach, and then surveyed the most important developments in the field. Finally, we described the open research issues in this emerging field.

References

Bohus, D. & Rudnicky, A. 2005. Sorry, I didn't catch that!—an investigation of non-understanding errors and recovery strategies. In *Proceedings of the 6th SIGdial Workshop on Discourse and Dialogue*, Dybkjaer, L. & Minker, W. (eds). Lisbon, Portugal, 128–143.

Bos, J., Klein, E., Lemon, O. & Oka, T. 2003. DIPPER: description and formalisation of an information-state update dialogue system architecture. In *Proceedings of the 4th SIGdial Workshop on Discourse and Dialogue*, 115–124.

Cheyer, A. & Martin, D. 2001. The open agent architecture. *Journal of Autonomous Agents and Multi-Agent Systems* **4**(1/2), 143–148.

Chickering, D. & Paek, T. 2005. *Online Adaptation of Influence Diagrams*. Technical Report MSR-TR-2005-55. Microsoft Corporation.

English, M. & Heeman, P. 2005. 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*, 1011–1018.

Forbes-Riley, K. & Litman, D. 2005. Using bigrams to identify relationships between student certainness states and tutor responses in a spoken dialogue corpus. In *Proceedings of the 6th SIGdial Workshop on Discourse and Dialogue*, Dybkjaer, L. & Minker, W. (eds). Lisbon, Portugal, 87–96.

Frampton, M. 2008. *Using Dialogue Acts in Dialogue Strategy Learning: Optimising Repair Strategies*. PhD thesis, University of Edinburgh, UK.

Frampton, M. & Lemon, O. 2005. Reinforcement learning of dialogue strategies using the user's last dialogue act. In *Proceedings of the 4th IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, Zukerman, I., Alexandersson, J. & Jönsson, A. (eds). Edinburgh, UK, 83–90.

Frampton, M. & Lemon, O. 2006. Learning more effective dialogue strategies using limited dialogue move features. In *Proceedings of the 44th Annual Meeting of the Association for Computational Linguistics*, Sydney, Australia, 185–192.

Frampton, M. & Lemon, O. 2008. Using dialogue acts to learn better repair strategies. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, Las Vegas, USA, 5045–5048.

Fraser, N. & Gilbert, G. 1991. Simulating speech systems. *Computer Speech and Language* **5**(1), 81–99.

Georgila, K., Henderson, J. & Lemon, O. 2005a. Learning user simulations for information state update dialogue systems. In *Proceedings of Eurospeech*, Lisbon, Portugal, 893–896.

Georgila, K., Lemon, O. & Henderson, J. 2005b. 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*), Gardent, C. & Gaiffe, B. (eds). Nancy, France.

Hall, M. 1999. *Correlation-based Feature Selection for Machine Learning*. PhD thesis, University Of Waikato, New Zealand.

Heckerman, D. 1995. A Bayesian approach to learning causal networks. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence*, 285–295.

Henderson, J., Lemon, O. & Georgila, K. 2005. Hybrid reinforcement/supervised learning for dialogue policies from COMMUNICATOR data. In *Proceedings of the 4th IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, Zukerman, I., Alexandersson, J. & Jönsson, A. (eds). Edinburgh, UK, 68–75.

Henderson, J., Lemon, O. & Georgila, K. 2008. Hybrid reinforcement/supervised learning of dialogue policies from fixed datasets. *Computational Linguistics* **34**(4), 487–511.

Kearns, M., Mansour, Y. & Ng, A. 1999. A sparse sampling algorithm for near-optimal planning in large Markov Decision Processes. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, 1324–1331.

Lemon, O., Georgila, K. & Henderson, J. 2006a. Evaluating effectiveness and portability of reinforcement learned strategies. In *Proceedings of the IEEE/ACL Workshop on Spoken Language Technology*, Palm Beach, Aruba, 178–181.

Lemon, O., Georgila, K., Henderson, J. & Stuttle, M. 2006b. 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*, Trento, Italy, 119–122.

Levin, E. & Pieraccini, R. 1997. A stochastic model of computer-human interaction for learning dialogue strategies. In *Proceedings of Eurospeech*, 1883–1886.

Levin, E., Pieraccini, R. & Eckert, W. 1998. Using Markov Decision Processes for learning dialogue strategies. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, Seattle, USA, 201–204.

Levin, E., Pieraccini, R. & Eckert, W. 2000. A stochastic model of computer-human interaction for learning dialogue strategies. *IEEE Transactions On Speech and Audio Processing* **8**(1), 11–23.

Litman, D., Kearns, M., Singh, S. & Walker, M. 2000. Automatic optimization of dialogue management. In *Proceedings of COLING*, Saarbrücken, Germany, 502–508.

Litman, D. & Silliman, S. 2004. 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*, Boston, USA, 5–8.

Paek, T. & Chickering, D. 2005. The Markov assumption in spoken dialogue management. In *Proceedings of the 6th SIGdial Workshop on Discourse and Dialogue*, Dybkjaer, L. & Minker, W. (eds). Lisbon, Portugal, 35–44.

Pietquin, O. 2004. *A Framework for Unsupervised Learning of Dialogue Strategies*. PhD thesis, Faculté Polytechnique de Mons, TCTS Lab, Belgique.

Pietquin, O. & Renals, S. 2002. ASR system modeling for automatic evaluation and optimization of dialogue systems. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, Orlando, USA, 45–48.

Roy, N., Pineau, J. & Thrun, S. 2000. Spoken dialogue management using probabilistic reasoning. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, Hong Kong, 93–100.

Schatzmann, J., Weilhammer, K., Stuttle, M. N. & Young, S. 2006. A survey of statistical user simulation techniques for reinforcement-learning of dialogue management strategies. *Knowledge Engineering Review* **21**(2), 97–126.

Scheffler, K. & Young, S. 2000. Probabilistic simulation of human-machine dialogues. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 1217–1220.

Scheffler, K. & Young, S. 2001. Corpus-based dialogue simulation for automatic strategy learning and evaluation. In *Proceedings of the NAACL Workshop on Adaptation in Dialogue Systems*, 64–70.

Scheffler, K. & Young, S. 2002. Automatic learning of dialogue strategy using dialogue simulation and reinforcement learning. In *Proceedings of the Human Language Technology conference*, Marcus, M. (ed.). San Diego, USA, 12–19.

Sheskin, D. 2007. *Handbook of Parametric and Nonparametric Statistical Procedures*, 4th edn Taylor and Francis Group.

Singh, S., Kearns, M., Litman, D. & Walker, M. 1999. Reinforcement learning for spoken dialogue systems. In *Proceedings of the Annual Conference on Neural Information Processing Systems*, Denver, USA, 956–962.

Singh, S., Kearns, M., Litman, D. & Walker, M. 2000. Reinforcement learning for spoken dialogue systems. In *Advances in Neural Information Processing Systems*, Solla, S.A., Leen, T.K. & Müller, K.-R. (eds). **12**, 956–962. MIT Press.

Singh, S., Litman, D., Kearns, M. & Walker, M. 2002. Optimizing dialogue management with reinforcement learning: experiments with the NJFun system. *Journal of Artificial Intelligence Research* **16**, 105–133.

Skantze, G. 2003. Exploring human error handling strategies: implications for spoken dialogue systems. In *Proceedings of the ISCA Workshop on Error Handling in Spoken Dialogue Systems*, Vaud, Switzerland, 71–76.

Sutton, R. & Barto, A. 1998. *Reinforcement Learning: An Introduction*. The MIT Press.

Tetreault, J. & Litman, D. 2006. 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*, Moore, R.C., Bilmes, J.A. & Chu-Carroll, J. (eds). New York, USA, 272–279.

Thomson, B., Schatzmann, J., Weilhammer, K., Ye, H. & Young, S. 2007. Training a real-world POMDP-based dialog system. In *Proceedings of Bridging the Gap: Academic and Industrial Research in Dialog Technologies*, 9–16. ACL.

Walker, M. 2000. An application of reinforcement learning to dialogue strategy selection in a spoken dialogue system for email. *Journal of Artificial Intelligence Research* **12**, 387–416.

Walker, M., Aberdeen, J., Boland, J., Bratt, E., Garofolo, J., Hirschman, L., Le, A., Lee, S., Narayanan, S., Papineni, K., Pellom, B., Polifroni, B., Potamianos, A., Prabhu, P., Rudnicky, A., Sanders, G., Seneff, S., Stallard, D. & Whittaker, S. 2001a. DARPA Communicator dialog travel planning systems: the June 2000 data collection. In *Proceedings of Eurospeech*, Aalborg, Denmark, 1371–1374.

Walker, M., Fromer, J. & Narayanan, S. 1998. 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*, 1345–1352.

Walker, M., Kamm, C. & Litman, D. 2000. Towards developing general models of usability with PARADISE. *Natural Language Engineering* **6**(3), 363–377.

Walker, M., Litman, D., Kamm, C. & Abella, A. 1997. PARADISE: a framework for evaluating spoken dialogue agents. In *Proceedings of the 35th Annual Meeting of the Association of Computational Linguistics*, 271–280.

Walker, M. & Passonneau, R. 2001. DATE: a Dialogue Act Tagging scheme for Evaluation of spoken dialogue systems. In *Proceedings of the Human Language Technology Conference*, San Diego, USA.

Walker, M., Passonneau, R. & Boland, J. 2001b. Quantitative and qualitative evaluation of Darpa Communicator spoken dialogue systems. In *Proceedings of the 39th Annual Meeting of the Association for Compuational Linguistics*, 515–522.

Walker, M., Rudnicky, A., Prasad, R., Aberdeen, J., Bratt, E., Garofolo, J., Hastie, H., Le, A., Pellom, B., Potamianos, A., Passonneau, R., Roukos, S., Sanders, G., Seneff, S. & Stallard, D. 2002. DARPA Communicator: cross-system results for the 2001 evaluation. In *Proceedings of the International Conference on Spoken Language Processing*, Denver, USA, 269–272.

Williams, J. 2007. Applying POMDPs to dialog systems in the troubleshooting domain. In *Proceedings of the Workshop on Bridging the Gap: Academic and Industrial Research in Dialog Technologies*, 1–8. ACL.

Williams, J., Poupart, P. & Young, S. 2005a. Factored Partially Observable Markov Decision Processes for dialogue management. In *4th IJCAI Workshop on Knowledge and Reasoning in Practical Dialog Systems*, Zukerman, I., Alexandersson, J. & Jönsson, A. (eds). Edinburgh, UK, 76–82.

Williams, J., Poupart, P. & Young, S. 2005b. Partially Observable Markov Decision Processes with continuous observations for dialogue management. In *Proceedings of the 6th SIGdial Workshop on Discourse and Dialogue*, Dybkjaer, L. & Minker, W. (eds). Lisbon, Portugal, 87–96.

Williams, J. & Young, S. 2005. Scaling up POMDPs for dialog management: the 'Summary POMDP' method. In *Automatic Speech Recognition and Understanding Workshop*, Puerto Rico, USA, 250–255.

Williams, J. & Young, S. 2006. Scaling POMDPs for dialog management with composite summary point-based value iteration (CSPBVI). In *AAAI Workshop on Statistical and Empirical Approaches for Spoken Dialogue Systems*, Boston, USA, 37–42.

Williams, J. & Young, S. 2007. Partially Observable Markov Decision Processes for spoken dialog systems. *Computer Speech and Language* **21**(2), 231–422.