Dialogue Strategy Optimization with Reinforcement Learning in an AT&T Call Routing Application

Charles Lewis and Giuseppe Di Fabbrizio

AT&T – Labs Research
180 Park Avenue
Florham Park, NJ 07932 - USA
{clewis,pino}@research.att.com

Abstract

Reinforcement Learning (RL) algorithms are particularly well suited to some of the challenges of spoken dialogue systems (SDS) design. RL provides an approach for automated learning that can adapt to new environments without supervision. SDS are constantly subjected to new environments in the form of new groups of users, and developer intervention is costly. In this paper, I will describe some results from experiments that used RL to select the optimal prompts to maximize the success rate in a call routing application. A simulation of the dialogue outcomes were used to experiment with different scenarios and demonstrate how RL can make these systems more robust.

1. INTRODUCTION

Reinforcement Learning (RL) systems simultaneously learn and perform without supervision. The nature of the RL problem makes it suitable for shifting environments that do not lend themselves to one-pass analysis. Instead of learning a single, static solution, RL implementations learn and adapt continuously over time. This makes RL techniques more applicable to some real-world problems than supervised machine learning (ML).

The goal of this experiment was to explore the potential for an RL solution to reduce the need for re-deployment by assuming responsibility for optimal prompt selection. When the prompts for a system are created, the designer has a number of ways of phrasing each one. For example, the system may take the initiative and tell the user what their options are (closed prompt), or the system may invite the user to express their needs in their own words (open prompt). Some prompts use a fictional personality, and some prompts may try to reassure the user with friendly instructions. The effectiveness of each kind of prompt depends on the situation and how well they are received by the user.

The unsupervised learning of an RL system can have great benefits here, where human supervision of changes to the system is costly and time-consuming. Rather than force the User Experience (UE) expert to decide which level of system initiative to use, this approach defers that decision. Potentially, this can make dialogue designs more robust by allowing the decision to be made at run-time, based on feedback from the environment.

SDS operate in an environment where there is often change. Many environmental factors (such as the volume of calls received, the hours that the application is in operation, and the geographical region where the application is deployed) can affect the performance of the service, and remain unknown at the time that the application is designed.

The design of SDS is a task that currently depends on human experts. UE professionals decide on the dialogue flow, the prompt wording, the confidence threshold required for parsing, and many other facets of these applications. Often, even these experts cannot agree on the best type of introduction prompt, for example, or the best compromise between hand-holding for and empowerment of the user.

After the application is deployed and its performance analyzed, the UE expert can use this data to adjust the application and deploy it a second time. This is repeated as often as necessary. Each re-deployment requires human intervention for analysis, re-authoring, and quality assurance beforehand, engineering resources for the re-deployment itself, and monitoring after to determine the effects of the re-deployment and to determine if further tweaking is necessary.

2. THE APPLICATION

We applied this approach to AT&T's internal call routing application called OMCE. OMCE is used by AT&T's small business customers to report and track service problems. It's a mixed-initiative dialogue that tries to elicit enough information from the caller to route the call to a call center that specializes in their request.

Call centers are sensitive to call volume, so the application tries to route calls as specifically as possible. By accomplishing initial request type identification, the application can reduce the amount of time that human operators spend speaking to customers, and increase the number of customers that the call center can handle. Conversely, the more unclassified calls received by a call center, the fewer customers that they can handle.

The call routing process tries to elicit the user's intent in the initial dialogue turns. If the caller tries to go directly to a customer service representative (CSR), or if the attempt to elicit routing information otherwise fails, a special prompt is played to help the user to interact with the automated system. In this application, the UE expert created four potential prompts for this situation. They run from completely open to completely closed, and utilize different levels of hand-holding and reassurance. In our experiments, RL controls which of these prompts the system plays in this case..

3. RL, AND RL FOR VOICE APPLICATIONS

RL-type techniques have been used in voice applications before. In (Levin, et al.,2000 and Singh, et al., 2002), RL systems are used to plan high-level dialogue strategies. These systems evaluated a design problem and created an optimal solution for deployment. This application of RL does not derive a single, optimal solution, rather, it implements an adaptive one. The approach used here will demonstrate a way for the UE expert to author a range of possible prompts and then leave it to the system to determine which one is best at any given point.

RL systems are guided by reinforcement from their environment. At the completion of a task (or at some intermediate step in a task), the RL system receives feedback which it uses to refine its behavior in future episodes. Unlike supervised learning, which requires a large set of labeled data, RL can be put into motion without any pre-compiled model of its environment. There are many variations of the RL problem and possible solution implementations (as described in Kaelbling and Littman, 1996 and Sutton and Barto, 1998).

Reinforcement Learning is a category of problems with many possible solutions and implementations. The components that most RL systems have in common are a *policy* to guide its decisions, a *value function* to describe the value that the system puts on a state (or a state/action pair) within the course of problem-solving, and a *reward function* to describe the environment's reinforcement of a course of action. These three components operate in a feedback loop, with the value function informing the policy, the policy making a decision, the decision informing the reward function, and the value function changing to more closely match the results of the reward function.

The policy is simply the algorithm that makes the decisions, and any data structures that support it. We denote the current policy as π, and the optimal policy as π^*. π is the subject of continuous refinement in RL, with the goal, of course, of achieving π^*, or to come within a specified tolerance of it.

The value function describes the value that the system puts on a particular state, *s,* or state/action pair, *s,a*. This function relies on the experience of the system to ascribe a long-term utility to available actions, and informs the choices made by the policy. Value functions as they are used in this paper are based on the action taken in a particular state. These are called the *action-value functions* in the literature and denoted as $Q(s,a)$. I will refer to these too as simply value functions. The most accurate action-value function, $Q^*(s,a)$, will result in the best policy, π^*. In fact, we can think of the value functions as being the primary input to the decision-making policy, as they inform all decisions.

The reward function describes the feedback from the environment. It can be deterministic, as when a particular action always creates a particular reward value, it can be stochastic, as when the specific value of the reward is generated from a distribution of values, or it can be the result of a real, un-modeled environment. In all of these cases, the reward is what the value functions are all trying to predict. Part of the RL problem can be thought of as a refinement of the value functions to more closely approximate the reward function. It is possible for there to be both intermediate rewards and final rewards in the decision making process. Only final rewards will be provided in this application.

Learning occurs with these components by a process called Iterative Policy Generation (IPG). In this process, there is a feedback loop between the policy, π, the value function, $Q(s,a)$, and the reward function:

1) The system makes a decision using policy π.
2) The reward function provides feedback on the decision.
3) Relevant value functions are refined to better reflect the reward.
4) A new policy, π', is created, based on the new value functions. This is the policy used in the next dialogue

Because the new policy is based on improved value functions, it will, on average, achieve improved rewards. As the number of iterations increases, and the value function improves, the policy approaches π^*. In order for this system to work, the policy must take advantage of the accuracy of the value functions to try to maximize the expected reward. How this is done depends on the RL implementation that is used.

The so-called Monte Carlo approach (Sutton and Barto,1998, Chapter 5) relies entirely on experience with the environment to improve policy. Under this implementation of RL, the system interacts with the environment in well-defined episodes during which it enacts decisions and gathers data. Between episodes, the data is processed to improve the policy. The two critical operations are refining the value functions (performed between episodes), and deciding which action to take from a given state s (performed during episodes).

When the system receives reinforcement from an episode, it integrates this new data into relevant value functions. To reflect the history of the system's

interactions with the environment, this integration must compromise between allowing a long history to overwhelm new data entirely, and allowing new data to completely supplant the past experience of the system. If the reward function never changes, a straightforward average of all returns from a state-action pair will provide a continuously improving approximation of the true value of the pair as more data becomes available, by the law of large numbers. In this scenario, there is no need to weigh new data any differently from old data. If we cannot make this assumption, however, we must discount the old values as new data becomes available.

As mentioned, the value function can be thought of as an approximation of the reward function. Therefore, we can begin to understand the solution to the problem of integrating new data into the existing value function by thinking of the difference between the result of the reward function, r, and the value of the value function, Q, as the error in the estimation. The question then becomes, to what degree do we wish to adjust the existing value function to reflect this error? This multiple of the error is called the step-size, or α[1].

$$Q(s,a)' \leftarrow Q(s,a) + \alpha\,(\,r - Q(s,a)\,) \qquad (1)$$

A step size of *1/n* for example, where *n* is the number of episodes, results in a simple average of the rewards. This is one possibility, but it gives equal weight to all dialogue episodes without regard to how recent they are. When α is a constant, and $0 < \alpha \leq 1$, it creates an exponential, recency-weighted average, where the weight given to old rewards decays exponentially as new rewards are added. This is the approach that we used for these experiments. This approach makes the new value a compressed history of the rewards experienced by the system, while maintaining the flexibility to adapt to changes in the environment. Practically speaking, this also has the advantage that it does not require us to explicitly maintain a history of the episode rewards. Most importantly, this distinguishes RL from systems that use statistical optimization without consideration for the order in which the data was received. This makes an RL system effective in a dynamic environment where straightforward optimization is not a perfect fit.

The reward update formula is half of the technique. The other half is deciding between competing options at each decision point. To maximize the reward, the most obvious solution is the greedy one: simply pick the action with the highest value. This, however, does not take into account two factors associated with RL problems: 1) that the system may not have sufficient experience in the environment to have value functions that accurately predict

the reward their associated actions will receive, and 2) the environment on which the value functions are based may have changed over time. The risk in 1) is that the system can be fooled into accepting the first action that provides a positive reward as the best one, even if the action does not provide the greatest reward over time. The risk in 2) is that the system may continue to choose an action that was once effective long after it has ceased to be so.

The tension between greedily maximizing the predicted reward and the concerns raised above is often referred to as *explore vs. exploit*. Exploration is necessary to improve the value functions. Exploitation is necessary to see any benefit from the value functions. One solution to balancing these needs is to have the RL system choose the most highly valued action some percentage of the time, described as ε, and choose a random action the rest of the time. This approach is called ε-*greedy*, and systems that use it enjoy the benefit that over time, in a stable system, they will form an accurate measurement of each action's value by the law of large numbers (Sutton and Barto,1998, 2.2). When the approach is used, the probability of the system choosing the action with the highest value function is:

$$P(a_{best}) = \varepsilon + (1\text{-}\varepsilon)/\,|A| \qquad (2)$$

where |A| is the total number of actions possible.
This is the approach that we used in these experiments.

4. DATA COLLECTION

The OMCE application, including the application logic and the prompts used, was designed by an AT&T User Experience expert. This design was then implemented using Florence, the dialogue manager in AT&T's VoiceTone® system (Di Fabbrizio and Lewis, 2004). The application was deployed under the VoiceTone® system, which provided the logging used for data collection. RL was not applied in the data collection. Instead, the system selected a prompt randomly at each decision point. .

Over the three months that the application was in use, data was collected on 9,786 dialogues. The average number of calls per day was about 50, with 473 dialogues logged on the highest volume day. Of these calls, 845 used a CSR prompt. The data collected on each call included the dialogue states entered, prompts used, instructions to increment counters, and the routing of the call. We created probability distributions of outcomes for each prompt, and based the simulated calls in this paper on those models.

The general idea of the reward is to give successful dialogues a higher score than non-successful ones. The question of what constitutes a "successful" dialogue has been examined before, for example, in the context of predicting problematic dialogues (Walker, et al., 2000). In our case, we are not limited to simply successful or not-successful: we can also assign varying scores to varying levels of success after the dialogue is complete.

---

[1] Sutton and Barto 1998 provide a derivation of this equation from a system that uses a simple running average to successfully converge the *reward function* to the *value function*.

The most obvious criterion to use is how the routing was handled. If the call was routed to an operator, it was successful to some degree. If the user hung up before the call could be routed, it was not. The reason for the hang-up is unknown to us, and could be unrelated to the system, but for our purposes this type of dialogue was considered not-successful.

If the caller did not hang up but the system could not gather enough information to determine the type of call, there is one routing destination, the general CSR line, that is used. Whenever this default routing occurs, it is because the system has given up on eliciting more specific information from the caller. Although this is not as serious a failure as a complete disconnect, it is not as preferable as a well-specified routing. This outcome was assigned a score higher than the score for a hang-up, but not as high as the score for more specifically routed calls.

After a call was scored, the reward value was used to update the appropriate value function, as per Equation 1. The rewards used in these experiments for user disconnect, default routing, and specific routing were 0, .5, and 1 respectively, for action values between 0 and 1.

For comparison to the experimental data, Figure 1 shows the number of calls disconnected by the user before routing and the number of calls that were routed to the default CSR line (default routing) during data collection with random prompt selection. The graphs in this paper show one data point for every 100 dialogues that used a CSR prompt. This value is presented as a percentage of calls of each type.
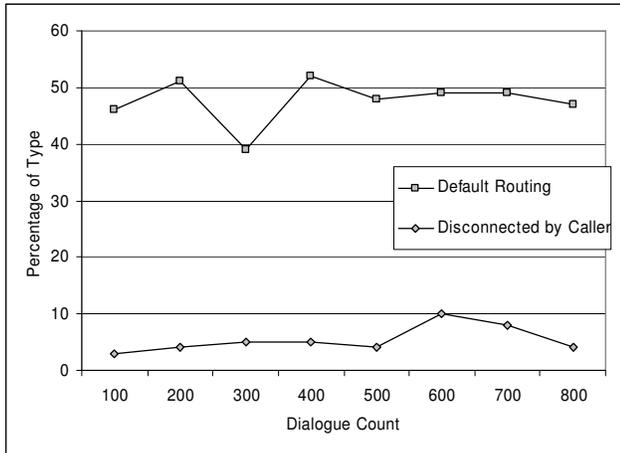

Figure 1 - Number of non-successful dialogues from gathered data

5. SIMULATED RL IN A STABLE ENVIRONMENT

Figure 2 is a typical outcome of the first set of experiments with this simulation, where a data point was computed every 100 dialogues, for a percentage of each type. The number of dialogues with default routing drops sharply

after around 2500 dialogues. The number of dialogues where the caller hangs up remains fairly steady, despite significant differences in the hang-up rates between the prompts. Because the number of calls with default routing is much higher than the number of calls where the caller hangs up, prompt selection is dominated by consideration for the former.
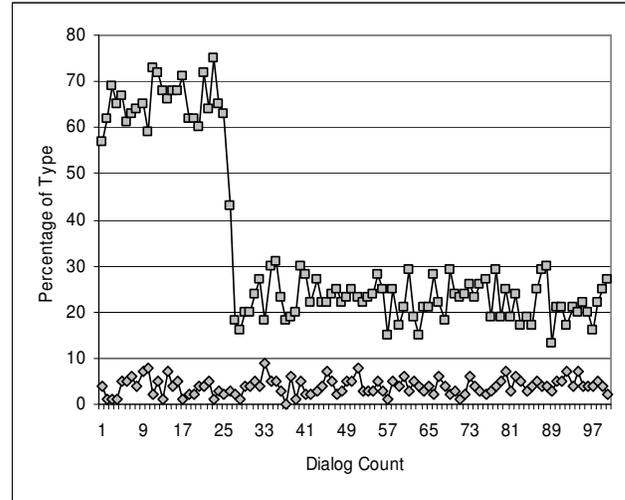

Figure 2 - Number of non-successful dialogues in simulation of a stable environment (legend as per fig. 1)

This demonstrates the potential for the Monte Carlo approach to improve the routing of calls in an application like OMCE. In a real deployment, over time, fewer calls would have been routed to the default operator.

6. SIMULATED RL IN A DYNAMIC ENVIRONMENT

The 'environment' in this simulation is produced by the dialogue outcome models, which are based on the collected data. To simulate a change in the environment, the models were changed halfway through each run in this experiment. At the halfway point, after the 5,000th dialogue, the outcome model used for the top performing prompt is switched with the model used for the worst performing one. As shown in Figure 3, this had an immediate affect on the simulated percentage of default routing. Immediately after the models were switched, the number of calls with default routing returned to its initial level, before RL took effect. Within 100 dialogues, the percentage of default routing dropped again but much more quickly than it did at the beginning of the process. This rapid re-adjustment lowered the number of default routing calls almost down to the level seen before the model change. This pattern was typical of the runs in this experiment.
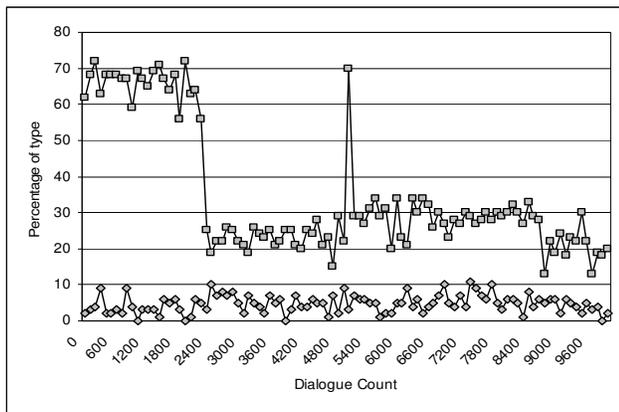
Figure 3 - Number of non-successful dialogues in dynamic environment simulation (legend as per fig. 1)

In Figure 4, we can see what happened behind the scenes in one of these runs. This chart shows the value for each of the prompts as new data is received. In the first half of the run, the value for prompt 031 gradually increases to become the highest, and then the number of calls routed to the default destination (Figure 3) declines drastically. This is the same behavior that we saw in the previous experiment.

Halfway through the run, the model that generated the results for prompt 031 dialogues, the most valued prompt, was switched with the model that generated results for prompt 032, the least valued prompt. This is where Figure 3 shows a large up-tick in the percentage of calls routed to the default call center, and where the value for prompt 031 drops precipitously in Figure 4.
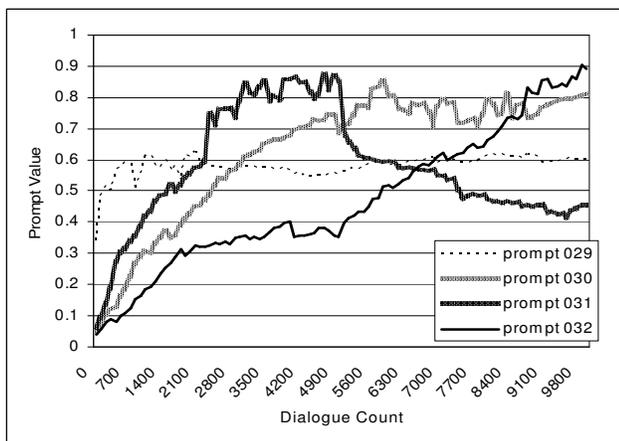

Figure 4 - Prompt scores in dynamic environment

Within a hundred dialogues of the up-tick, however, the number of calls routed to the default destination is down to a much lower level. This adjustment happened very quickly because the second-most valued prompt, prompt 030, takes over as soon as the value of prompt 031 declines. This immediately lowers the percentage of default routings. Eventually the model for prompt 032 (originally the model for prompt 031) takes the lead again,

and the percentage of calls routed to the default call center returns to the same level seen before the model switch.

7. SPEEDING UP THE LEARNING PROCESS

From these simulations a pattern is clear: after some initial period of adjustment, the values of the prompts become ordered to minimize the percentage of calls that are routed to the default center, and the number of these calls suddenly drops. It would obviously be beneficial to shorten this period of initial exploration. One way is to increase the value of α to increase the amount of adjustment that occurs for each learning experience but, as Figure 5 illustrates, if α is set too high ($\alpha = 0.1$) the values will not stabilize (in the rest of the experiments, $\alpha = 0.01$). Another way shorten the learning period is to adjust the value of the explore vs. exploit variable, ε, which was kept at 0.8 in the experiments so far.
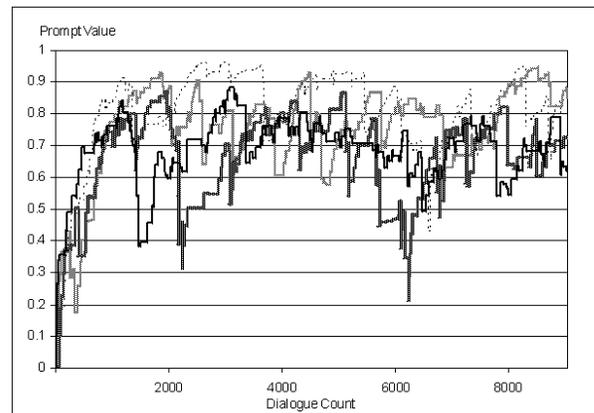

Figure 5 - Change in CSR prompt values $\alpha = 0.1$ (legend as per fig. 4)

Experiments with this variable revealed a well-understood trade-off. With a low value for ε (where the system leans towards exploration), the ordering of actions occurs more quickly but higher values of ε (where the system leans towards exploitation) have better performance. To get the best of both worlds, the system was configured to begin by stressing exploration ($\varepsilon = 0.0$), and to change to exploitation when performance improved ($\varepsilon = 1.0$). In most cases, this was very successful: the system achieved the short start-up time of the low ε value (on average, it took 260 dialogues to reach the breakpoint where there was a dramatic performance improvement) with the performance of the high ε value (on average, after the breakpoint, the percentage of calls routed to the default call center per 100 calls was 21.6).

There are two downsides to this approach, however. Obviously, this makes the system more brittle. With no exploration after the breakpoint, only the most-valued action is used, so the values of the other actions remain the same. This makes it is more difficult to recover from a

change in environment because the unused actions are not accurately assessed. The second problem is demonstrated in Figure 6. In this case, a badly performing action did very well initially, and was able to achieve the performance goal. After this initial luck, the action did not perform well and averaged 69% default routings. Figure 6 illustrates the values of all of the actions throughout this run. When the performance goal was reached, prompt 029 was the most highly valued prompt and, despite fluctuations in its value, never dropped from that rank. Meanwhile, the other prompts did not have the opportunity to outperform this prompt because no further exploration took place. This is an unusual case, but it demonstrates the risk of prematurely eliminating exploration.
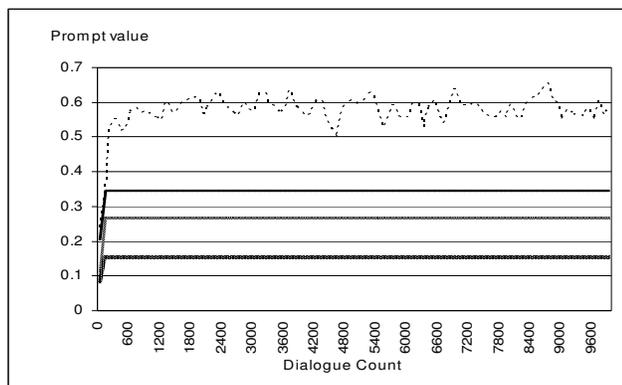

Figure 6 – A case where a non-optimal prompt ranked highest when ε value changed after initial performance improvement. (legend as per fig. 4)

8. CONCLUSIONS

These experiments brought to light an aspect of exploration that is not typically mentioned: beyond initial exploration to arrive at the best action, beyond testing the waters to make sure that the most highly-valued action is still the most highly-rewarded, exploration in RL also maintains the accuracy of the value functions for all of the state/action pairs throughout the lifetime of the application. In this experiment, when the top-performing action stopped performing, the second-best action was valued highly enough to take its place almost immediately. This controlled degradation of performance provided a safety net for the system until it was able to readjust the values of the effected prompts.

As for the use of RL to optimize the dialogue strategy, these experiments demonstrate that it is possible to frame prompt selection as an RL problem, and that it is possible for the Monte Carlo RL technique to provide continuous, unsupervised learning for this task. In the first set of experiments, it was demonstrated how this approach works in a new environment, with no assumptions made about the relative values of each action. In the second set of experiments, the environment was changed and the system was tested to see how well it would adapt. This provided

an interesting example of how the exploration inherent in RL systems created a robust system that could recover quickly.

We believe that the same techniques that were successfully used here could also be applicable to other facets of dialogue strategy. In future work, we hope to experiment with dynamically tuning voice application parameters such as such as required confidence scores and barge-in levels, and to apply RL to high-level strategy decisions about the experience of the user and their expected behavior.

References

G. Di Fabbrizio and C. Lewis, "Florence: A Dialogue Manager Framework for Spoken Dialogue Systems*," ICSLP 2004, 8th International Conference on Spoken Language Processing, Jeju, Jeju Island, Korea*, October 4-8, 2004.

L.P. Kaelbling, M.L. Littman, "Reinforcement Learning: A Survey," *Journal of Artificial Intelligence Research* 4, May 1996, 237-285

E. Levin, R. Pieraccini, and W. Eckert, "A Stochastic Model of Human-Machine Interaction for Learning Dialog Strategies," *IEEE Transactions on Speech and Audio Processing*, Vol. 8 No. 1, January 2000

D. Litman, M. Walker, and M. Kearns, "Automatic Detection of Poor Speech Recognition at the Dialogue Level," *ACL-1999*

Mitchell, T., *Machine Learning*, McGraw-Hill, 1997.

S. Singh, M. Kearns, D. Litman, and M. Walker, "Reinforcement Learning for Spoken Dialogue Systems," *NIPS-1999*

S. Singh, D. Litman, M. Kearns, and M. Walker, "Optimizing Dialogue Management with Reinforcement Learning: Experiments with the NJFun System," Journal of Artificial Intelligence Research1 16 (2002), 105-133

Learning: Experiments with the NJFun System," *Journal of Artificial Intelligence Research* 16 (2002), 105-133

Sutton, R. S. and Barto, A. G., *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA, 1998

M. Walker, I. Langkilde, J. Wright, A. Gorin, and D. Litman, "Learning to Predict Problematic Situations in a Spoken Dialogue System: Experiments with How May I Help You*?" In Proc. 1st Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2000.