User and Noise Adaptive Dialogue Management Using Hybrid System Actions

Senthilkumar Chandramohan, Olivier Pietquin

SUPELEC - IMS Research Group, Metz - France
{senthilkumar.chandramohan, olivier.pietquin}@supelec.fr

Abstract. In recent years reinforcement-learning-based approaches have been widely used for management policy optimization in spoken dialogue systems (SDS). A dialogue management policy is a mapping from dialogue states to system actions, *i.e.* given the state of the dialogue the dialogue policy determines the next action to be performed by the dialogue manager. So-far policy optimization primarily focused on mapping the dialogue state to simple system actions (such as confirm or ask one piece of information) and the possibility of using complex system actions (such as confirm or ask several slots at the same time) has not been well investigated. In this paper we explore the possibilities of using complex (or hybrid) system actions for dialogue management and then discuss the impact of user experience and channel noise on complex action selection. Our experimental results obtained using simulated users reveal that user and noise adaptive hybrid action selection can perform better than dialogue policies which can only perform simple actions.

1 Introduction

Spoken Dialog Systems (SDS) are systems which have the ability to interact with human beings using speech as the medium of interaction. The dialogue policy plays a crucial role in dialogue management and informs the dialogue manager what action to perform next given the state of the dialogue. Thus building an optimal dialogue management policy is an important step when developing any spoken dialogue system. Using an hand-coded dialogue policy is one of the simplest ways for building dialogue systems, but as the complexity of the dialogue task grows it becomes increasingly difficult to code a dialogue policy manually. Over the years various statistical approaches such as [9, 3, 21] have been proposed for dialogue management problems with reasonably large state spaces.

Most of the literature on spoken dialog systems optimization focuses on the optimal selection of elementary dialog acts at each dialog turn. In this paper, we investigate the possibility to learn how to combine these simple dialog acts into complex actions to obtain more efficient dialogue policies. Since complex system acts combine several system acts together it can lead to shorter dialogue episodes. Also by using complex system acts the system designer can introduce some degrees of flexibility to the human-computer interaction by allowing the users with prior knowledge about the system to furnish and receive as much as information as they wish in one act. The use of complex system actions for dialogue management has been studied only to a little extent. Works related to the use of open-ended questions are such studies [11].

First the focus is to learn a *hybrid* policy which can choose to perform simple system acts as well as more complex and flexible system acts. The challenge in learning such a hybrid policy is the unavailability of dialogue corpora to explore complex system acts. Secondly, the impact of noise and user simulation on complex system acts is analyzed and means to learn a noise and user adaptive dialogue policy is discussed.

This paper is organized as follows: In Section 2 a formal description of Markov Decision Process (MDP) is presented first, following which casting and solving the dialogue problem in the framework of an MDP. In Section 3 complex system actions are formally defined and then the impact of channel noise and user experience is discussed. Section 4 outlines how channel noise can be simulated using user simulation and how noise adaptive hybrid action policy can be learned. Section 5 describes how a user-adaptive hybrid-action policy can be learned. Section 6 outlines our evaluation set-up and analyzes the performance of different policies learned. Eventually Section 7 concludes.

2 MDP for dialogue management

The MDP [1] framework comes from the optimal control community. It is originally used to describe and solve sequential decision making problems in stochastic dynamic environments. An MDP is formally a tuple $\{S, A, P, R, \gamma\}$ where S is the (finite) state space, A the (finite) action space, $P \in \mathcal{P}(S)^{S\times A}$ the family of Markovian transition probabilities[1], $R \in \mathbb{R}^{S\times A\times S}$ the reward function and γ the discounting factor ($0 \leq \gamma \leq 1$). According to this formalism, during the interaction with a controlling agent, an environment steps from state to state ($s \in S$) according to transition probabilities P as a consequence of the controller's actions ($a \in A$). After each transition, the system produces an immediate reward (r) according to its reward function R. A so-called *policy* $\pi \in A^S$ mapping states to actions models the way the agent controls its environment. The quality of a policy is quantified by the so-called *value function* $V^\pi(s)$ which maps each state to the expected discounted cumulative reward given that the agent starts in this state and follows the policy π:

$$V^\pi(s) = E[\sum_{i=0}^{\infty} \gamma^i r_i | s_0 = s, \pi] \tag{1}$$

An optimal policy π^* maximizes this function for each state:

$$\pi^* = \operatorname*{argmax}_\pi V^\pi \tag{2}$$

Suppose that we are given the optimal value function V^* (that is the value function associated to an optimal policy), deriving the associated policy would require to know

[1] Notation $f \in A^B$ is equivalent to $f : B \to A$

the transition probabilities P. Yet, this is usually unknown and the optimal control policy should be learned only by interactions. This is why the state-action value (or Q-) function is introduced. It adds a degree of freedom on the choice of the first action:

$$Q^{\pi}(s,a) = E[\sum_{i=0}^{\infty} \gamma^i r_i | s_0 = s, a_0 = a, \pi] \tag{3}$$

$$Q^{*}(s,a) = E[\sum_{i=0}^{\infty} \gamma^i r_i | s_0 = s, a_0 = a, \pi^*] \tag{4}$$

where $Q^*(s,a)$ is the optimal state-action value function. An action-selection strategy that is greedy according to this function ($\pi(s) = \text{argmax}_a\, Q^*(s,a)$) provides an optimal policy. There are many algorithms that solve this optimization problem. When this optimization is done without any information about the transition probabilities and the reward function but only transition and immediate rewards are observed, the solving algorithms belong to the Reinforcement Learning (RL) family [20].

2.1 Dialogue as an MDP

The spoken dialogue management problem can be seen as a sequential decision making problem. It can thus be cast into an MDP and the optimal policy can be found by applying a RL algorithm. Indeed, the role of the dialogue manager (or the decision maker) is to select and perform dialogue acts (actions in the MDP paradigm) when it reaches a given dialogue turn (state in the MDP paradigm) while interacting with a human user (its environment in the MDP paradigm). There can be several types of system dialogue acts. For example, in the case of a restaurant information system, possible acts are request(cuisine_type), provide(address), confirm(price_range), close *etc.* The dialogue state is usually represented efficiently by the Information State paradigm [3]. In this paradigm, the dialogue state contains a compact representation of the history of the dialogue in terms of system acts and its subsequent user responses (user acts). It summarizes the information exchanged between the user and the system until the considered state is reached.

A dialogue management strategy is thus a mapping between dialogue states and dialogue acts. Still following the MDP paradigm, the optimal strategy is the one that maximizes some cumulative function of rewards collected all along the interaction. A common choice for the immediate reward is the contribution of each action to the user's satisfaction [17]. This subjective reward is usually approximated by a linear combination of objective measures (dialogue duration, number of ASR errors, task completion *etc.*). Weights of this linear combination can be computed from empirical data [10]. Yet, most of the time, more simple reward functions are used, taking into account that the most important objective measures to take into account are

2.2 Restaurant information MDP-SDS

The dialogue problem studied in the rest of this paper is a slot filling restaurant information system. The dialogue manager has 3 slots to be filled and confirmed by the user *(1) Cuisine (Italian-French-Thai), (2) Location (City center-East-West) and (3) Price-range (Cheap-Moderate-Expensive)*. Here the goal of the dialogue system is to fill these slots with user preferences and also to confirm the slot values, if the confidence in the retrieved information is low, before proceeding to seek relevant information from the database. The initial list of all possible system actions is, *(1) Ask cuisine, (2) Ask location, (3) Ask restaurant type, (4) Explicit confirm cuisine, (5) Explicit confirm location, (6) Explicit confirm type, (7) Greet the user and (8) Implicitly confirm* $\{$*cuisine/location/type*$\}$ *and Ask* $\{$*cuisine/location/type*$\}$.

The state representation of the restaurant information MDP-SDS is the information state (which is a succinct representation of dialogue progress) as discussed in [3]. The reward function is defined as follows, the system will receive a completion reward of 300 if the task is successfully completed and will receive a time step penalty of -20 for every transition.

2.3 Dialogue policy optimization

Once the dialogue management problem is cast into an MDP, Dynamic Programming or RL methods [20] can be applied to find the optimal dialogue policy [9]. The goal of the policy optimization task is to find the dialogue policy which maximizes the expected discounted sum of rewards that can obtained by the agent over an infinite time period. Most of the recent works done in this direction [6] focuses on using online reinforcement learning algorithms such as SARSA for policy optimization. Online RL algorithms like SARSA are data intensive and so it is customary to simulate or model the user behavior based on the available dialogue corpus [8, 18, 13] and to artificially generate simulated dialogues. The RL policy learner will then interact with the simulated user to find the optimal dialogue policy. DIPPER dialogue management framework [4] along with REALL, a hierarchical reinforcement learning policy learner [7] was used to learn and test the dialogue policies discussed in this paper (the exploration rate of the RL policy learner was set as 0.2). The user simulation used in the experiments was trained using town information corpus discussed in [4]. The policy learned using the reward function, action and state space described in 2.2 will be the baseline and will be referred to as *simple action policy*.

3 Complex system actions

Simple actions are commonly used. These are system acts which are related to to one slot such as asking a slot value or explicitly confirming the slot value etc. Actions listed in the subsection 2.2 are all examples of simple system acts (except implicit confirmation). Complex actions are system actions which are formed by combining two or more simple systems actions. Complex actions deal with multiple slots such as confirming two slot values or asking for three slot values. Thus for the restaurant information dialogue system there can be several possible complex actions that can be performed. Some of the complex actions in this case are *(1) Ask two slot values, (2) Ask three slot values, (3) Explicitly confirm two slot values, (4) Explicitly confirm three slot values and (5) Implicitly confirm two and ask third slot.*

3.1 Hybrid action policy

In this section is described how to learn an hybrid action policy which can choose to perform a simple system acts as well as a complex system acts. Firstly the action set of the restaurant information MDP-SDS described in 2.2 is updated with the following complex system action, *(1) Ask values for two slots, (2) Explicitly confirm two slot values, (3) Implicitly confirm two slot values and ask value for the third slot*. Since the action set is updated with the simple and complex actions the RL policy learner will explore both types of actions. But the user simulation learned using the dialogue corpora (which only had simple actions) has an ability to respond only for simple system actions.

Thus the user behavior for the complex system acts was hand-coded and combined with the learned user simulation. The hand-coded behavior for complex actions is as follows *(1) ask 2 slots* $\{$*ProvideTwoSlotValue 0.9, ProvideOneSlotValue 0.1, SayNothing 0*$\}$*, (2) explicitConfirm 2 slots* $\{$*SayYes 1.0, SayNo 0, SayNothing*$\}$*, (3) implicitConfirm 2 slots and ask value for slot* $\{$*ProvideOneSlotValue 1.0, SayYes 0.0, SayNo 0*$\}$. The updated user simulation can thus respond to both simple (behavior learned from corpora) and complex system actions (hand-coded behavior). As explained in 2.3 a dialogue policy was learned using the updated user simulation and MDP-SDS, this policy will be referred to as *hybrid action policy*.

System: Hello
User: Silence
System: What type of restaurant are you looking for and in which location? (complex)
User: Italian restaurant in city center
System: In what price-range are you looking for a Italian restaurant?
User: A cheap one
System: Did you say you are looking for a cheap restaurant in city-center?
(complex) User: Yes

The dialogue episode presented here is the interaction between the RL policy learner (exploration rate set as zero) and user simulation using the hybrid action policy. One can observe that the policy can now choose complex system actions and simple actions when required. It can be observed that given the action set of the restaurant information dialogue system the sample dialogue presented here is an optimal behavior for grounding the three slots.

3.2 Effect of noise and user on complex action

The hand-coded user behavior for the complex action in Section 3.1 simulates the zero channel noise scenario *i.e.,* when the user says something it is assumed that the system will capture it correctly and there is no chance for error. This is not always true and there may be some noise in the transmission channel. Thus ideally the probability for *SayNo* user act is not zero (the fact that the system doesn't understand what the user said is modeled as the user saying nothing). But if the user response is *SayNo* for the complex system act *ImplicitConfirm2slotsAndAskASlot* it would be difficult to identify which of the two slots is wrong. Based on this our *1st assumption* is : when there is noise in the automatic speech recognition (ASR) channel it is advisable to perform simple system acts and not complex actions.

System: Can you let me know your preferences for restaurant selection are?
User 1: Nothing (Novice user)
User 2: Italian restaurant (Novice user)
User 3: Cheap Italian restaurant in City Center (Experienced user)

The users who intend to use the restaurant information SDS may range from novice (new) users to experienced (frequent) users. Now let us consider the above mentioned example. Here the user is under providing information in the first two cases but provides all necessary information in the third. Based on this our *2nd assumption* is: it is ideal to perform simple systems actions while interacting with novice users and perform hybrid actions while interacting with experienced users.

4 Noise adaptive policy

Based on the 1st assumption action selection has to be performed depending on the noise level in the ASR channel. First step to learn a noise dependent policy is to have a noise simulation module. Several works have be done in the recent past such as [14, 12, 5, 16, 19] to simulate channel noise for dialogue modeling. A simple approach to simulate the channel noise is to tune the probabilities of user responses for confirmation system actions [15]. By increasing the probability for negation we can simulate the high noise scenario and by reducing the probability for negation we can simulate the low noise scenario. The user behaviors for complex confirmation actions presented in section 3.1 are modified as shown below in Table 1 to simulate low noise scenario and high noise scenario. Let us refer to these user simulations as *low noise user model* and *high noise user model*.

Table 1. Handcrafted user behavior for noise simulation

Channel noise	Action	Provide 1Value	SayYes	SayNo
Low	ImplicitConfirm2AndAskASlot	0.9	0	0.1
Low	ExplicitConfirm2	0	0.9	0.1
High	ImplicitConfirm2AndAskASlot	0.6	0	0.4
High	ExplicitConfirm2	0	0.6	0.4

The goal now is to learn a hybrid dialogue policy which is sensitive to noise in the ASR channel. A new variable is introduced in the Information State to measure the negation index (*i.e.* to measure the frequency of negations made by the user). This approach is similar to the one suggested in [2]. The reward function defined is 2.2 is modified to include the negation index as shown below

$$\textit{Reward} = \textit{Completion reward} - \textit{Time step penalty} - \textit{Negation count} * 20$$

Using this updated reward function and the user simulation with channel noise simulations, noise sensitive policies can be learned. One policy is learned using the low noise user model and the second using the high noise user model. Let us refer to these dialogue policies as *low noise policy* and *high noise policy*. An interaction between the policy learner (learning rate set as zero) using the low noise policy and user simulation with simulated low channel noise is presented below. It can be observed that the system performs complex system actions as well as simple actions.

System: greet
User: silence
System: ask2Slot [askCuisine, askLocation]
User: provide 2 Values [cuisine, central Location]
System: implConfAskASlot [confirmCuisine, askPrice]
User: provide 1 value [Moderate]
System: explicitConfirm2 [confirm centralLocation,confirm Moderate]
User: yesAnswer

An interaction between the policy learner (learning rate set as zero) using the high noise policy and user simulation with simulated high channel noise is presented below. It can be observed that the system performs only simple (confirmation) actions.

System: greet
User: silence
System: ask2Slot [askLocation, askPrice]
User: provide 2 values [CentralLocation, Moderate]
System: askASlot [askCuisine]
User: provide 1 value [Italian]
System: explicitConfirm [CentralLocation]
User: yesAnswer
System: explicitConfirm [Moderate]
User: yesAnswer
System: explicitConfirm [Italian] User: yesAnswer

In real life situations it is practically impossible to say when the channel noise will be low and when it will be high, thus one may not know when to switch between the high noise and low noise policies. One possible solution is to learn a dialogue policy which can adapt itself to different ASR channel noise level. A noise adaptive dialogue policy is learned by using the high noise user model and low noise user model in parallel. For every dialogue cycle the policy learner randomly choose one of the two user simulations for interaction. This way one can learn a policy that can adapt to different channel noise levels. So let the policy learned by randomly switching between the user models during the policy optimization process be called as *noise adaptive hybrid (action) policy*.

5 User adaptive policy

The goal now is to first simulate the user experience in the user simulation and use it to learn a user experience dependent policy. To perform this task novice users are assumed as shown in section 3.2 example to under provide information for complex actions whereas the experienced users will provide the necessary slot values. In order to simplify the problem the novice users are assumed to say nothing for complex (information seeking) actions whereas the experienced users will provide the necessary slot values in most cases. Tuning the probabilities for user behavior in this way results in two user behaviors and let us term them as *novice user simulation* and *experienced user simulation*. In addition to simulating the user experience the user behavior also simulated the low noise level scenario. Novice and experienced user behaviors with low channel noise is outlined in Table 2

Table 2. Handcrafted user behavior for user experience simulation

User	Noise	Action	Give 2 values	Give 1 value	Yes	No	Nothing
Novice	Low	ImplicitConfirm2AndAskASlot	0	0.9	0	0.1	0
Novice	Low	ExplicitConfirm2	0	0	0.9	0.1	0
Novice	Low	Ask2Slots	0	0	0	0	1.0
Experienced	Low	ImplicitConfirm2AndAskASlot	0	0.9	0	0.1	0
Experienced	Low	ExplicitConfirm2	0	0	0.9	0.1	0
Experienced	Low	Ask2Slots	0.9	0	0	0	0.1

Similar to the negation index we introduce a term called experience index in the state representation of the restaurant information MDP-SDS. The reward function updated in section 4 is again updated as follows

$$Reward = Completion\ reward + TimePenalty - NegCount * 5 - ExpIndex * 10$$

By using the novice user and experienced user behaviors one can learn two different dialogue policies. Let us term these policies as *novice user policy* and *experienced user policy*. Also as explained in the previous section by using these user simulations simultaneously *i.e.* by randomly switching them during the policy optimization one can learn a user adaptive policy. Let us term this policy as *adaptive hybrid (action) policy*.

6 Policy evaluation and analysis

Table 3, presents the results of comparisons between simple action policy and hybrid action policy derived in section 2.3 and 3.1. The results are based on 300 dialogue cycles between the policy learner using the two policies (learning rate set as zero) with user simulation (which was used to learn hybrid action policy). One can observe that

Table 3. Simple action vs Hybrid action policy

Policy Name	Average Reward	Completion Average	Length
Simple	160	300	7.0
Hybrid	214	300	4.2

by using complex actions along with simple actions we can considerably reduce the dialogue length and hence the overall reward of the dialogue manager can be improved.

Table 4, presents the result of comparison between low noise policy and adaptive noise hybrid action policy derived in section 4. The results are based on 300 dialogue cycles between the policy learner using the two policies (learning rate set as zero) with user simulation (also) simulating low channel noise. It can be observed that the adaptive noise policy performs equally as good as the low noise policy in the low channel noise scenario.

Table 4. Low noise policy vs Adaptive noise policy in low noise scenario

Policy Name	Average Reward	Completion Average	NegationPenalty	Length
Low noise	216.51	300	4.12	
Adaptive noise	214.06	300	4.20	

Table 5, presents the result of comparison between high noise policy and adaptive noise hybrid action policy derived in section 4. The results are based on 300 dialogue cycles between the policy learner using the two policies (exploration rate set as zero) with user simulation (also) simulating high channel noise. It can be observed that the adaptive noise policy performs equally as good as the high noise policy in the high channel noise scenario, but there is a small degradation with regard to the task completion average.

Table 5. High noise policy vs Adaptive noise policy in high noise scenario

Policy Name	Average Reward	Completion Average	Length
High noise	160.52	300	6.65
Adaptive noise	175.99	295.84	5.30

Table 6, presents the result of comparison between low noise, high noise and adaptive noise hybrid action policy derived in section 4. The results are based on 300 dialogue cycles between the policy learner using the three policies (exploration rate set as zero) with two user simulations, simulating mixed channel noise. It can be observed that the adaptive noise policy performs better than the high noise policy and low noise policy in the mixed channel noise scenario. This shows that the adaptive policy learns a trade off to switch between complex and simple actions with regard to changing noise levels (where us low noise policy tries to perform complex actions always and high noise policy performs simple actions always). It actually takes advantage of the extended state representation to perform this adaptation.

Table 6. Low noise policy Vs High noise policy Vs Adaptive noise policy in mixed noise scenario

Policy Name	Average Reward	Completion Average	Length
Low noise	140.19	297.56	7.38
High noise	170.06	300	6.33
Adaptive noise	191.38	298.07	4.87

Table 7, presents the result of comparison between novice user policy, experienced user and adaptive user hybrid action policy derived in section 5. The results are based on 250 dialogue cycles between the policy learner using the three policies (exploration rate set as zero) with both novice and experienced user simulations, (randomly switched to simulating mixed user experience). It can be observed that the user adaptive policy performs better than the novice user policy and experienced user policy in the mixed user scenario. This shows that the user adaptive policy learns a trade off to switch between complex and simple actions with regard to changing user experience levels (where the novice user policy tries to perform simple actions always and experienced user policy performs complex actions always).

Table 7. Novice user policy Vs Experienced user policy Vs Adaptive user policy in mixed user scenario

Policy	Avg. Reward	Completion Avg.	SimpleAct	ComplexAct	Dialogue Length Avg.
Novice	145.9	250.0	7.66	0	7.66
Experience	-197.7	228.6	2.9	16.0	18.9
Adaptive	151.9	250.0	4.69	1.0	5.69

7 Conclusion

So far the possibilities of using complex system actions along with simple actions for spoken dialogue management has not been much investigated. Based on the experimental results presented in this contribution one can conclude that complex action selection can considerably reduce the dialogue length but in the mean time it is important to consider the channel noise and user experience factors before choosing complex actions. Since it is not possible to predict the channel noise level or the experience level of the user in real life scenario one can learn an adaptive hybrid action policy that can adapt to the channel noise and user experience. Yet, it requires extending the state representation to take into account the behaviour of the user (*SayNo* or overinformative users for example).

All the tasks (learning and testing) presented in this paper are carried out using simulated users partially learned from corpus and partially hand tuned, thus it will be ideal to test these policies with real users in future. Hybrid action selection may move human - machine interaction a step closer towards human - human communication. One other interesting direction of future work will be to explore the possibilities of automatically generate new complex actions from given list simple actions and use online policy learning approaches to learn an hybrid dialogue policy. This way we may come across potentially new and interesting system actions which may not available in the dialogue corpus.

References

[1] Richard Bellman. A markovian decision process. *Journal of Mathematics and Mechanics*, 1957.

[2] Srinivasan Janarthanam and Oliver Lemon. User simulations for online adaptation and knowledge-alignment in troubleshooting dialogue systems. In *In proceedings of LONDial, 2008*, London, UK, 2008.

[3] Staffan Larsson and David R. Traum. Information state and dialogue management in the TRINDI dialogue move engine toolkit. *Natural Language Engineering*, 2000.

[4] Oliver Lemon, Kallirroi Georgila, James Henderson, and Matthew Stuttle. An ISU dialogue system exhibiting reinforcement learning of dialogue policies: generic slot-filling in the TALK in-car system. In *Proceedings of the meeting of the European chapter of the Associaton for Computational Linguistics (EACL'06)*, Morristown, NJ, USA, 2006.

[5] Oliver Lemon and Xingkun Liu. Dialogue Policy Learning for combinations of Noise and User Simulation: transfer results. In *In Proceedings of SIGdial 2007, Antwerp, Belgium*, 2007.

[6] Oliver Lemon and Olivier Pietquin. Machine learning for spoken dialogue systems. In *Proceedings of the International Conference on Speech Communication and Technologies (InterSpeech'07)*, Antwerpen (Belgium), 2007.

[7] Oliver Lemon, Xingkun Xingkun Liu, Daniel Shapiro, and Carl Tollander. Hierarchical Reinforcement Learning of Dialogue Policies in a development environment for dialogue systems: REALL-DUDE. In *Proceedings of the 10th SemDial Workshop, BRANDIAL 2006*, 2006.

[8] Ester Levin, Roberto Pieraccini, and Wieland Eckert. A Stochastic Model of Human-Machine Interaction for learning dialog Strategies. *IEEE Transactions on Speech and Audio Processing*, 2000.

[9] Esther Levin, Roberto Pieraccini, and Wieland Eckert. Using markov decision process for learning dialogue strategies. In *Proceedings of ICASSP*, 1998.

[10] Candace A. Kamm Marilyn A. Walker, Diane J. Litman and Alicia Abella. PARADISE: A framework for evaluating spoken dialogue agents. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL'97)*, pages 271–280, Madrid (Spain), 1997.

[11] Olivier Pietquin. *A Framework for Unsupervised Learning of Dialogue Strategies*. PhD thesis, Faculté Polytechnique de Mons, TCTS Lab (Belgique), apr 2004.

[12] Olivier Pietquin and Thierry Dutoit. A Probabilistic Framework for Dialog Simulation and Optimal Strategy Learning. *IEEE Transactions on Audio, Speech and Language Processing*, 14(2):589–599, 2006.

[13] Olivier Pietquin and Thierry Dutoit. A probabilistic framework for dialog simulation and optimal strategy learning. *IEEE Transactions on Audio, Speech & Language Processing*, 2006.

[14] Olivier Pietquin and Steve Renals. ASR System Modeling For Automatic Evaluation And Optimization of Dialogue Systems. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2002)*, Orlando, (USA, FL), May 2002.

[15] Verena Rieser. *Bootstrapping Reinforcement Learning-based Dialogue Strategies from Wizard-of-Oz data*. PhD thesis, Saarland University, Dpt of Computational Linguistics, July 2008.

[16] Verena Rieser and Oliver Lemon. Learning effective multimodal dialogue strategies from wizard-of-oz data: bootstrapping and evaluation. In *Proceedings of the ssociation for Computational Linguistics (ACL) 2008*, Columbus, USA, 2008.

[17] Satinder Singh and Michael Kearns and Diane Litman and Marilyn Walker. Reinforcement learning for spoken dialogue systems. In *Proceedings of the Annual meeting of the Neural Iniformation Processing Society (NIPS'99), Denver, USA*. Springer, 1999.

[18] Jost Schatzmann, Karl Weilhammer, Matt Stuttle, and Steve Young. A survey of statistical user simulation techniques for reinforcement-learning of dialogue management strategies. *Knowledge Engineering Review*, 2006.

[19] Jost Schatzmann and Steve Young. Error simulation for training statistical dialogue systems. In *Proceedings of the ASRU 2007*, Kyoto, Japan, 2007.

[20] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press, 3rd edition, March 1998.

[21] Jason D. Williams and Steve Young. Partially observable markov decision processes for spoken dialog systems. *Computer Speech Language*, 2007.