User simulations for online adaptation and knowledge-alignment in Troubleshooting dialogue systems

Srinivasan Janarthanam
School of Informatics,
University of Edinburgh,
Edinburgh, EH8 9LW, GB.
s.janarthanam@ed.ac.uk

Oliver Lemon
School of Informatics,
University of Edinburgh,
Edinburgh, EH8 9LW, GB.
olemon@inf.ed.ac.uk

Abstract

We study the problem of alignment between dialogue participants, using the practical example of "troubleshooting" dialogue systems. Recent work on troubleshooting concerns automated spoken dialogue systems which support users who need to repair their internet connection. We address the problem that different users have different types of knowledge of problem domains, so that automated dialogue systems need to adapt online to the different knowledge of these users as it encounters them. We approach this problem using policy learning in a Markov Decision Process (MDP). In contrast to related work we propose a new user model which incorporates the different conceptual knowledge of different users, together with an environment simulation. We show that this model allows us to learn dialogue policies that automatically adapt online to new users, and that these policies are significantly better than threshold-based adaptive hand-coded policies for this problem.

1 Introduction

Adapting between conversation partners was first studied by Issacs & Clark (1987), where the partners identify each other's domain knowledge levels during conversation and share their knowledge, leading to task success. More recently, Larsson (2007) gives a formal account of how the meanings of NL expressions are adapted during conversations. These important aspects of dialogue are not addressed in current automated systems. Here, and in Lemon (2008), we propose a model that allows such decisions to be automatically optimized.

There has been much recent interest in automated dialogue systems for "troubleshooting" or "self-help". These cooperative systems are particularly interesting because they contain aspects of knowledge alignment and tutorial dialogue – for instance, some users may not know certain technical terms, so that systems must be able to "align" with their users, at least at a knowledge/concept-level. For example, some users may be happy with the term "ADSL filter", while others may need this explained to them before their problem can be solved. On the other hand, some users may be frustrated by unnecessary explanations. There is therefore an important tradeoff to be explored regarding how much additional explanation to provide to a particular user. Note that in general we will not know the knowledge profile of a user when they call the system, so our dialogue policies must be able to estimate the user type online, and continuously adapt their behavior based on the estimation.

All of this places additional requirements on our user models and user simulations for training these more complex systems. We provide a new user model for such purposes and show that it allows us to learn these types of adaptive dialogue policies.

2 Related work

Several user simulation models to support MDP -learning of dialogue management policies have been developed over recent years (Eckert et al 1997, Scheffler & Young 2001, Pietquin et al 2004, Georgila et al 2005, Cuayahuitl et al 2005, Schatzmann et al 2007). These simulations simulate users in travel planning and town-information domains.

They produce user responses based on various factors such as the system's action, user's goal/agenda, user's record of the dialogue, etc. Crucially, they only simulate a homogenous group of users, who always understand the system's actions completely and never ask for clarifications. The models also do not simulate the user's environment. In contrast, in this work we have focused on the user's domain knowledge and an environment simulation concerning troubleshooting systems.

Troubleshooting dialogue systems have been designed by Boye et al (2007), who presents a hand-coded system and Williams (2007) who uses machine learning. In Boye (2007) the task of fixing a broadband connection is hierarchically decomposed in to simpler tasks. When the user fails to respond, the system chooses alternative ways to solve the tasks. However, it always tries the standard procedure before choosing the alternatives. The dynamic tree structure that drives the conversation is interleaved with an "adapting-to-user" feature and is likely to become more complex to manually author in the case of realistic systems.

Williams (2007) presents a POMDP-based dialogue system for troubleshooting broadband connections. The system is trained to handle the uncertainty in user's observations and responses and provide the next appropriate instruction. Here the system learns *what* to ask or present rather than *how* to ask for information from the user. The system assumes a homogenous user population in terms of domain knowledge. However, in this work, we present an MDP system that learns to adapt to a user population where different users have different conceptual knowledge of the troubleshooting environment.

3 The Troubleshooting Domain & Dialogue Management

In this setup, the dialogue manager always directs the conversation, because (besides fixing their problem) the user does not have any other goal or agenda in order to direct the conversation. During the course of conversation, the dialogue manager asks the user to describe their troubleshooting environment (e.g. Modem lights etc). The information to be asked is handled using a hand-coded decision tree. The tree encodes what information to ask next based on the user's report on the environment so far. Previous work has shown that such trees can be learned from data (Williams 2007). While the decision tree decides what to ask next, the dialogue manager still has to decide *how* to ask for information and present instructions to users with different domain knowledge. Table 3.1 lists the dialogue manager actions related to the task at hand.

Table 3.1. Dialogue manager action set

1. Greet
2. Request_info
3. Extended_request_info
4. Request_action
5. Extended_request_action
6. Close_dialogue

In general, the dialogue manager must decide between a simple "request" act and an "extended request" act in order to request information from the user or to ask that they perform a manipulate action. An "extended request", although presented as a single turn in this dialogue act representation, is actually a sequence of system utterances that the system uses to educate the user about the concept that the system is querying/instructing about. For instance, a novice user may not know where the ADSL light is. In this case the system spends some time to inform the user where to look for this light. These extra utterances make an extended request more costly than a simple request (and this is later reflected in the reward function for learning this task).

The dialogue manager is also equipped with a *user estimation feature* that allows it to dynamically estimate the expertise of a user based on their responses and frustration. At the start of the session, it is assigned a default intermediate value of 5. This is later incremented or decremented based on evidence (e.g. whether the user answers the asked question). For an expert user, it increases from 5 and can reach 9 by the end of the dialogue. For a novice user, it can decrease to 0. However there is also uncertainty in user responses, since sometimes a novice user may be able to answer a question without extra help and hence may be misjudged as an expert. Similarly an expert user may fail to answer a question and be judged temporarily as a novice. But in the course of the conversation, the estimation function will usually correct itself as evidence about the user accumulates.

The dialogue manager's information state is composed of the following fields (table 3.2). There

are 8 binary slots and one 10-value integer slot in the dialogue information state, giving rise to a state space of 2560 (=2^8 * 10) states.

Table 3.2. Dialogue state features	
More_slots_to_ask	(binary)
Solution_found	(binary)
User_expertise_index	(0-9)
User_said_dont_know	(binary)
Modem_power_light_filled	(binary)
Adsl_light_filled	(binary)
Modem_filter_filled	(binary)
Phone_filter_filled	(binary)
Phone_line_working_filled	(binary)

4 Environment Simulation

This model simulates the troubleshooting environment around the user. The environment simulation represents all the objects connected to the troubleshooting problem. This simulation consists of a *modem, computers, telephone, fax, adsl filters,* etc. In addition, parts of these objects that will be of interest in the troubleshooting process have also been simulated. For instance, the *powerlight, adsl light on the modem, the usb ports on the computer, modem software,* etc are represented. Since connections between these objects cause problems in the real world, they are also simulated. In addition to representing the objects in the environment, the simulation allows the user to access the objects. Just like in the real world, users are able to manipulate the objects and therefore change the state of the environment as a whole. For instance, rebooting the modem might set the internet connection correctly. In the current model, the user will be able to observe and manipulate the objects in the environment in a principled way. In real world troubleshooting practice, the experts will be able to ping the user's modem remotely. This calls for a manipulative interface between the expert and the environment. This feature is not available in the current environment model.

The environment simulation is represented using Prolog facts and rules. The state of the environment S_e is represented using dynamic facts and is set initially using Environment-setting rules. The state of the environment can be observed using observation rules and the environment can be manipulated using manipulation rules. Example rules are shown in table 4.1. The environment-setting rule shown sets the environment initially to one of the faulty scenarios (faulty phone filter) that the current model is able to simulate. The *adsl filter a2* that connects the *phone_socket* to the *telephone t1* is specified as faulty in the definition. This causes interference and is the source of the problem. Using observation actions, the user is able to observe that the *adsl filter* is present. As per the system's instructions, the user can use manipulative actions to fix this problem. In this case, he replaces the *phone adsl filter* and sets the connection correctly.

Table 4.1 Environment Simulation

Environment Setting rule

```
set_env1(faulty_phone_filter) :-
        assert(equipment(comp1, desktop, working)),
        assert(equipment(t1, phone, working)),
        assert(equipment(m1, modem, working)),
        assert(equipment(a1, adsl_filter, working)),
        assert(equipment(a2, adsl_filter, not_working)),
        assert(connected(phone_socket, a1, rj11, firm)),
        assert(connected(phone_socket, a2, rj11, firm)),
        assert(connected(a1, m1, rj11, firm)),
        assert(connected(a2, t1, rj11, firm)),
        assert(connected(m1, comp1, usb, working)),
        assert(phone_line(live)),
        assert(modem_software(installed, working)),
        assert(authentication(correct)),
        !.
```

Observe Environment : Is there a phone filter?

```
pact(adsl_filter_for_phone, present) :-
        equipment(P, phone, _),
        equipment(A, adsl_filter, _),
        connected(A, P, rj11, firm),
        !.
```

Manipulate Environment : Replace the phone filter

```
mact(replace_phone_filter) :-
        equipment(a2, adsl_filter, not_working),
        !,
        retract(connected(phone_socket, a2, rj11, firm)),
        retract(connected(a2, t1, rj11, _)),
        assert(equipment(a4, adsl_filter, working)),
        assert(connected(phone_socket, a4, rj11, firm)),
        assert(connected(a4, t1, rj11, firm)),
        update_env,
        !.
```

The current environment simulation is capable of simulating 6 error configurations - *faulty modem, faulty phone/modem filter, missing phone/modem filter, faulty phone line, authentication failure, faulty modem USB port*.

5 User Simulation

The user simulation model stochastically simulates environment-sensitive and concept-knowledge-

sensitive user behavior, as shown schematically in figure 5.1 and described below.

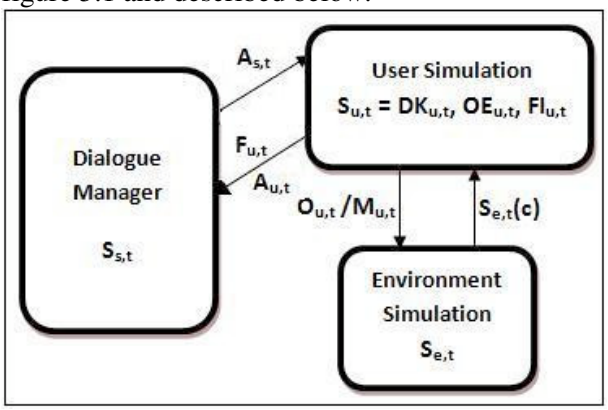

Figure 5.1. The Model & Experimental setup

Besides being faithful to the environment, it also simulates the *domain concept knowledge* of the user DK_u. Therefore, when the user is confronted with concepts he barely knows, he is more likely to request clarification. The model is capable of updating its domain knowledge, when the system provides clarification. By initially selecting different knowledge profiles, the current simulation can simulate a continuum of users from novices to experts. We attach a probability to every concept that the users must know in order to co-operate with the system to fix the problem. This probability determines whether the user follows the system's instruction or requests clarification. The knowledge profiles of three types of users are given in table 5.1. These values would be set by an analysis of data collected for the specific troubleshooting domain – for this proof of concept we select illustrative values.

Table 5.1. User knowledge profiles: P(DK)

Concept\User	Expert	Intermediate	Novice
Phone_line	0.99	0.85	0.5
ADSL filter	0.8	0.6	0.2
ADSL light	0.9	0.6	0.2
USB slot	0.8	0.55	0.3
Powerlight	0.95	0.85	0.5

An expert user has the highest probability to answer queries and perform manipulative actions without requesting clarification. However these values are not static profile thresholds but initial values during the start of a dialogue session. These values increase when the system clarifies the concept by issuing an extended request. In this case, for the particular concept *c* under discussion:

$$P(DK_{u,c,t+1}) = P(DK_{u,c,t} + boost\_c)$$

For these experiments we set *boost_c* = 0.5 for each c. Again, this average probability boost could be determined from real data for each c. In an extended request the system spends a few utterances to educate the user and therefore update the user's domain knowledge. As their knowledge profile gets boosted, their chances of answering the query increase. For instance, in response to the DM's action *'request_info: adsl_light'*, a novice user is more likely to say *'request_clarification: adsl_light',* initially. But if the system clarifies the concept, they are more likely to say *'provide_info: on'*. However an expert user is more likely to return *'provide_info: on'* without requesting clarification (given that the ADSL light is on as per the environment state).

Table 5.2 User's Action set A_u

1. Provide_info
2. Acknowledge
3. Manipulate_and_acknowledge
4. Request_clarification
5. Hang_up

When the system requests information, the user simulation observes the environment by issuing an observation action O_u, updates its observations OE_u, and reports back to the system. Similarly, when the system requests that the user manipulate the environment, the user manipulates the environment using a manipulate action M_u, observes its effects, updates the observations OE_u, and reports it back to the system. A manipulate action also changes the state of the environment S_e. The interaction between the user, environment and the system is shown in fig 5.1.

In addition to the user's verbal response, we also simulate the user's frustration. A user gets frustrated when the dialogue manager does not explain unknown terms or unnecessarily explains well- known terms[1]. We use a frustration index FI_u to capture this behavior. This index affects the probability of the user hanging up the call before the dialogue ends. The probability of hang-up is twice their frustration index, as a probability. The

[1] However, weighing of under-informativeness and over-informativeness on the same scale may be revised in our future work.

user's frustration is also conveyed to the system on a turn by turn basis $F_{u,t}$ (as a boolean value) along with the user's action $A_{u,t}$. We assume that the dialogue manager can detect the user's frustration from the user's utterance. It has been shown that frustration can be determined from prosodic (Lee et al 2002) features in the user's utterance.

The user state S_u, contributing to the action selection process contains the user's domain knowledge DK_u, his observations of the environment OE_u, frustration index FI_u and the number of dialogue turns T. Each of these components is updated at different times during the action selection process.

$$S_u = <DK_u , OE_u , FI_u , T>$$

Based on the updated user and environment states and the system action, the user action A_u is selected as described in the following algorithm, where *return_action P(A|X)* returns a user action A with probability P(A|X):

Step 1: If turns T > 6,
return_action P(hang_up | FI_u)
Step 2: If $A_{s,t}$ = extended_request(c),
Update $DK_{u,c}$, FI_u
Step 3: If $A_{s,t}$ = request or extended_request(c),
Do P(O_u(c) | DK(c)) or P(M_u(c) | DK(c))[2]
Update S_e, $OE_{u,c}$
return_action P($A_{u,t}$ | OE_u)

Figure 5.3 User simulation algorithm

Thus, by conditioning the user's action A_u on various factors like the user's domain knowledge DK_u, user's observations OE_u, frustration FI_u and the environment state S_e, the simulation provides a context-consistent and diverse user behavior. In future, we also plan to validate the simulation against real user datasets using well established metrics (Schatzmann et al 2005).

6 Reward Function

Every dialogue session is rewarded at its completion. A task completion reward (TCR) of 500 is given for successful completion and -500 for unsuccessful dialogues. A dialogue is considered to be successful if the dialogue partners are able to fix the problem and close the dialogue. On the other hand, it is considered unsuccessful if the user gets frustrated and hangs up before the dialogue ends. The following costs are associated with the number of dialogue turns (T) and extended turns (ET).

Turn cost per turn: TC = 10.0
Extended cost per turn: EC = 30.0

This extended turn cost encodes that idea that extended turns cost 3 times as much as normal turns, on average. Again, this parameter would be set by a PARADISE-style (Walker et al 1997) regression analysis on real user data.

The final reward for a dialogue session is calculated based on:

Total Turn cost: TTC = T * TC
Total Extension cost: TEC = ET * EC
Final reward = TCR - TTC – TNC

The reward function is designed to penalize longer dialogues and unnecessary extensions. The task of the learning agent (dialogue manager) is to learn an optimal policy to minimize the costs and increase the chances of successful dialogue of for all kinds of users.

7 Training

The system was trained for 15000 cycles producing approximately 1500 dialogues using the SARSA reinforcement learning algorithm (Sutton & Barto 1998). Our objective was to learn a single policy that can adapt online to any type of user (novice, expert, or intermediate). Hence the user simulation was calibrated to produce an equal number of novice and expert users. Recall that these types of user behave stochastically, so that no two expert users are guaranteed to behave in the same way, for example. The users were allowed to hang up only after the sixth turn. This is manually chosen to avoid early hang ups. After the sixth turn, the probability of the user hanging up is directly proportional to the user's frustration index (as described above). The main learning task here is to decide between using *extended- request* and *request* acts.

After the training runs, the system learned to adapt online to both expert and novice users and in each case to maximize the final reward. It learned to effectively make use of the user expertise index, which is a part of the dialogue state, in order to tune its behavior towards the users. It learned to use *extended-request* acts when the expertise index indicates that the user is a novice and to use *request* acts when it indicates an expert user. It also learns not to use *extended-request* acts for information

[2] Depending whether the system has requested an observe or manipulate action.

that does not depend on the user's conceptual knowledge. For instance, both novice and expert users are equally able to answer if their internet connection is working. Hence an *extended-request* act simply adds to the cost without reducing any risks or costs and so is avoided for such slots. The system learned to reduce the dialogue length during training (shown in fig 7.1) by choosing the appropriate action the very first time an instruction is given to a user. This behavior avoids repeating the instruction and therefore the costly repairs.

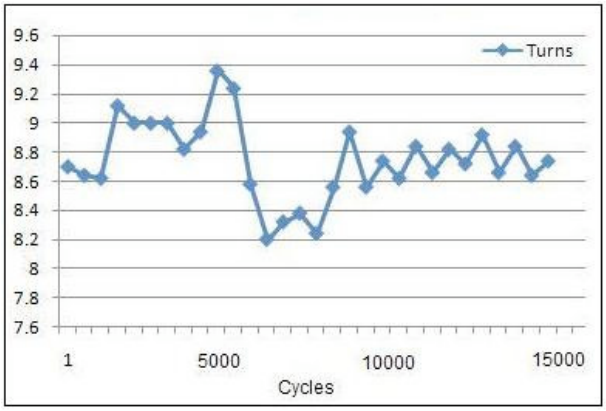

Figure 7.1. Optimization of dialogue length

Similarly, during training, the dialogue manager learned to optimize the user frustration index (shown in fig 7.2).

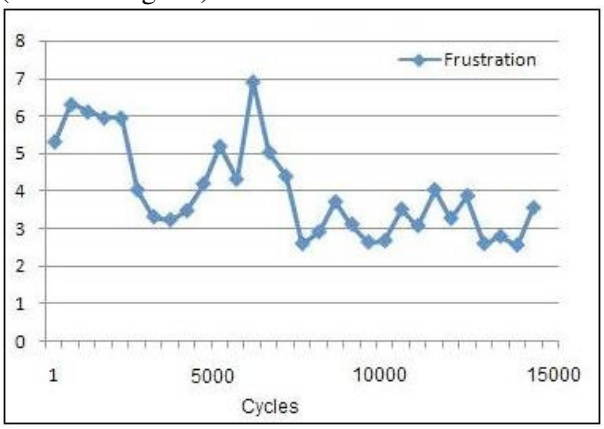

Figure 7.2. Optimization of frustration index

8 Evaluation

We tested the learned policy by comparing its performance with three hand-coded policies - Expert Only, Novice Only, and Adaptive.

1. The *Hand-coded Expert Only* policy treats all users as experts. Hence it always uses request acts.
2. The *Hand-coded Novice Only* policy treats all users as novices and always uses extended-request acts.
3. The *Hand-coded Adaptive* policy adapts to the user based on the estimated expertise index. It uses *extended-request* acts when the index is less than 5 and *request* acts otherwise. This therefore encodes a classic threshold-based approach to adaptation. (see e.g. Varges 2003)

All four policies were run against three groups of users – experts, novices and intermediate users. Each such run produced approximately 800 test dialogues each. Task success rate and average final reward for each policy run on different user groups were calculated. The results for an equal mix of experts, novices, and intermediate users are presented under "Mixed". Statistical significance was calculated using a Wilcoxon signed rank test[3].

Table 8.1 compares the task success rates of different policies on the user populations. Note that the most important column in the following tables is "Mixed", since this indicates the performance of the policies when they encounter a mixed population of users (i.e. as would happen in a real deployed system).

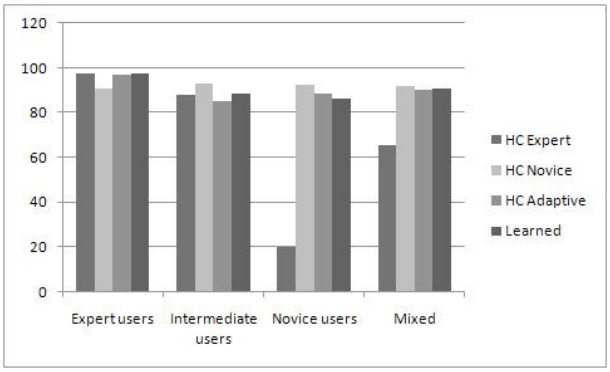

Figure 8.1 Task success rate

Table 8.1. Task success rate

Policy\Users	Expert	Inter	Novice	**Mixed**
HC Expert	**97.7**	87.9	19.5	65.1
HC Novice	90.8	**92.8**	**92.6**	**92.1**
HC Adaptive	96.9	85.2	88.3	90.1
Learned	**97.6**	**88.2**	**86.2**	**90.6**

[3] Difference in average final rewards between the policies were not normally distributed as per Kolmogonrov-Smirnov test.

Here we can see that the learned policy has good task completion across the range of user types. However, the task success metric does not take into account the important cost of different types of system turns. Table 8.2 compares the average final reward (combing task completion and turn cost) of the different policies on three different user populations. All improvements made by the learned policy are statistically significant at $p<0.001$.

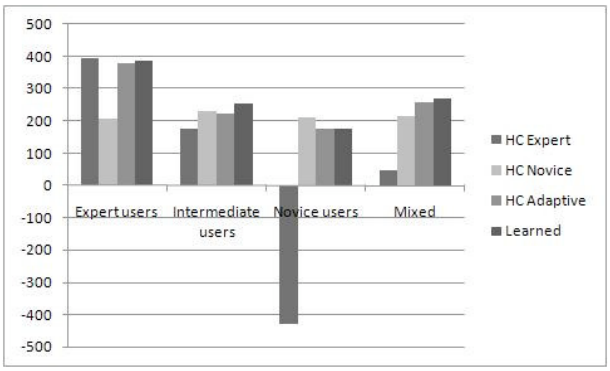

Figure 8.2 Average final reward

Table 8.2: Average final reward				
Policy\Users	Expert	Inter	Novice	**Mixed**
HC Expert	**393.5**	177.5	-430.2	46.9
HC Novice	206.6	230.7	**210.1**	215.8
HC Adaptive	379.6	221.7	177.2	259.4
Learned	**385.9**	**252.6**	**175.2**	**271.2**

The HC Expert Only and HC Novice Only policies were best for their respective populations, but they did not perform well against other populations. The HC Expert Only policy beats all the other policies with the highest average final reward 393.5 for the expert users. It also has the highest task success rate at 97.7, but it performs badly on novice and intermediate users, scoring the lowest average final reward among other policies. The HC Novice policy performed well with both novice and intermediate users, but with expert users it scored the lowest average final reward among all the policies. It also has the highest task success rate for all users combined. This is because it always gives the user an extended request, thereby reducing the number of turns. But this also results in a reduction in average final reward on all user types combined (Mixed). The HC Adaptive policy performs consistently among all the user groups. It scores a very good average final reward and task success rate. But it scores lower than the learned policy in both average final reward and task completion scores in expert and intermediate user groups. However, in the novice user group, its scores are slightly better than the learned policy. The learned policy also performs consistently on all the groups. It scores the best average final reward for intermediate users, although the policy was not trained on intermediate users. Its average final rewards on novice and expert groups are not very far behind the best rewards. However, the key point here is that for the "mixed" user group, the learned policy beats all the other policies with the highest average final reward of 271.2. This result shows that when we do not know the user population in advance, as is the case in real applications, the learned policy is able to handle the range of users encountered by adapting online. This is consistent with the results of Lemon & Liu (2007), who considered dialogue policies for different noise conditions.

The above results are promising because the learned policy has been able to perform better than carefully handcrafted adaptive policy for the same task. While it was easy to hand-code a policy for this task, it would not be so when more parameters are added to the dialogue manager's information state. The policy learning paradigm allows us to learn optimal policies for these types of trade-off without an expensive "implement, test, deploy, refine, redeploy,..." iterative development cycle. The parameters for the model presented above can all be estimated from data, for example collected in a small Wizard-of-Oz experiment (Rieser and Lemon 2008).

9 Conclusion

We addressed the general problem that different dialogue participants have different types of conceptual knowledge of the domain under discussion, so that work must be done to align or coordinate their understanding (see e.g. Issacs & Clark 1987). We studied this problem in the practical case of "troubleshooting", where automated dialogue systems need to adapt online to the different knowledge of different users as it encounters them. We proposed a new user model which incorporates different conceptual knowledge of different users, together with an environment simulation. We show that this model allows us to learn dialogue policies that automatically adapt online to new users, and that these policies are significantly better than threshold-based adaptive hand-coded policies for this problem. Future work in this area would be to ex-

tend the approach, and in particular increase the complexity of the user simulations, to handle other aspects of alignment in dialogue, such as semantic plasticity (Larsson 2007) and lexical and syntactic alignment (Pickering and Garrod 2006) in task-based dialogues.

10 Acknowledgements

The research leading to these results has received funding from the European Community's Seventh Framework (FP7/2007-2013) under grant agreement no. 216594. (CLASSIC Project www.classic-project.org), EPSRC project no. EP/E019501/1 and the British Council (UKIERI Ph.D Scholarships 2007-08).

References

H. Cuayáhuitl, S. Renals, O. Lemon, and H. Shimodaira. 2005. Human-Computer Dialogue Simulation Using Hidden Markov Models. *Proc. of ASRU* '05.

W. Eckert, E. Levin and R. Pieraccini. 1997. User Modelling for Spoken Dialogue System Evaluation. *Proc. of ASRU 1997.*

K. Georgila, J. Henderson, and O. Lemon. 2005. Learning user simulations for Information State Update Dialogue Systems. *Proc. Eurospeech-Interspeech '05*.

J. Boye. 2007. Dialogue management for automatic troubleshooting and other problem-solving applications. *Proc. 8th SIGDial workshop on discourse and dialogue*, 2007.

E. A. Isaacs & H. H. Clark. 1987. References in conversations between experts and novices. *Journal of Experimental Psychology: General, 116, 26-37.*

S. Larsson. 2007. A general framework for semantic plasticity and negotiation. *Proceedings of the 7th Intl Workshop on Computational Semantics.*

C. M. Lee, S. Narayanan, R. Pieraccini. 2002. Classifying emotions in human-machine spoken dialogs. *Proc. of ICME*, 2002.

O. Lemon, X. Liu. 2007. Dialogue Policy Learning for combinations of Noise and User Simulations: transfer results. *Proc. 8th SIGdial Workshop, 2007*.

O. Lemon. 2008. Adaptive Natural Language Generation in Dialogue using Reinforcement Learning. SEMDial (LONDial) 2008.

M. J. Pickering and S. Garrod. 2006. Toward a mechanistic psychology of dialogue. Behavioral and Brain Sciences, 27, 169-225.

O. Pietquin and T. Dutoit. 2006. A probabilistic framework for dialog simulation and optimal strategy learning. *In IEEE Transactions on Audio, Speech and Language Processing*, vol. 14, no. 2, pp. 589–599, March 2006.

V. Rieser and O. Lemon. 2008. Learning Effective Multimodal Dialogue Strategies from Wizard-of-Oz data: Bootstrapping and Evaluation, *In ACL 2008* (to appear).

J. Schatzmann, B. Thomson, K. Weilhammer, H. Ye, and S. Young. 2007. Agenda-based User Simulation for Bootstrapping a POMDP Dialogue System, *Proc. HLT/NAACL, 2007.*

J. Schatzmann, K. Georgila, and S. Young. 2005. Quantitative Evaluation of User Simulation Techniques for Spoken Dialogue Systems. *Proc. 6th SIGDial Workshop on Discourse and Dialogue, Lisbon, 2005.*

K. Scheffler & S. Young. 2001. Corpus-based dialogue simulation for automatic strategy learning and evaluation. *Proc. NAACL Workshop on Adaptation in Dialogue Systems, 2001.*

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

S. Varges. 2003. Instance-based Natural Language Generation. *Ph. D. Thesis. Institute for Communicating and Collaborative Systems, School of Informatics, University of Edinburgh.*

M. Walker, D. Litman, C. Kamm, and A. Abella. 1997. PARADISE: A Framework for evaluating Spoken Dialogue Agents. *Proc 35th Annual meeting of ACL.*

J. Williams. 2007. Applying POMDPs to Dialog Systems in the Troubleshooting Domain. *Proc HLT/NAACL Workshop on Bridging the Gap: Academic and Industrial Research in Dialog Technology* 2007.