Learning Lexical Alignment Policies for Generating Referring Expressions in Spoken Dialogue Systems

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

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

Abstract

We address the problem that different users have different lexical knowledge about problem domains, so that automated dialogue systems need to adapt their generation choices online to the users' domain knowledge as it encounters them. We approach this problem using policy learning in Markov Decision Processes (MDP). In contrast to related work we propose a new statistical user model which incorporates the lexical knowledge of different users. We evaluate this user model by showing that it allows us to learn dialogue policies that automatically adapt their choice of referring expressions online to different users, and that these policies are significantly better than adaptive hand-coded policies for this problem. The learned policies are consistently between 2 and 8 turns shorter than a range of different hand-coded but adaptive baseline lexical alignment policies.

1 Introduction

In current "troubleshooting" spoken dialogue systems (SDS)(Williams, 2007), the major part of the conversation is directed by the system, while the user follows the system's instructions. Once the system decides what instruction to give the user (at the dialogue management level), it faces several decisions to be made at the natural language generation (NLG) level. These include, deciding which concepts to include in the utterance, deciding the referring expressions (RE) to use in the utterance and so on. A little-studied problem is to what extent a system could automatically align to the user's lexical knowledge by adapting its RE choices, in particular based on his domain expertise, and how this can be modelled and optimised computationally.

(Issacs and Clark, 1987) show how two interlocutors adapt their language in a conversation by assessing each other's domain expertise during dialogue, by observing how they react to each other's RE choices. This is called alignment through *Audience Design* (Clark, 1996; Bell, 1984). Using inappropriate REs in instructions has been identified as a serious problem affecting a system's usability (Molich and Nielsen, 1990).

In this paper, we treat NLG within a computational learning paradigm (Lemon, 2008; Rieser and Lemon, 2009; Janarthanam and Lemon, 2008). We examine whether a SDS can automatically learn a lexical alignment policy for audience design, which enables it to choose appropriate REs by predicting the user's lexical knowledge dynamically during the course of the dialogue. This can avoid clarification requests from the users and keep the dialogues short. The example given below describes the kind of lexical alignment behaviour that we want the system to learn. The system chooses "small white box" instead of "ADSL filter" and "monitor symbol" instead of "network icon", because it learnt that the user is a novice based on their clarification requests. However, it switches to using technical terms like "browser", when it learns that the user is not a complete novice (as he verifies the description for the network icon in Usr 4).

```
Example 1.
Sys 1>> Is your router connected
to the computer?
Usr 1>> Uh.  What's a router?
Sys 2>> It's the big black box.
Usr 2>> Ok..  yes.
Sys 3>> Do you see a small white
box connected to the router?
Usr 3>> Yes.
Sys 4>> Ok.  Is there a flashing
monitor symbol at the bottom
```

Proceedings of the 12th European Workshop on Natural Language Generation, pages 74–81,
Athens, Greece, 30 – 31 March 2009.

```
right of the screen?
Usr 4>> the network icon?
Sys 5>> Yes.  Is it flashing?
Usr 5>> Yes.  It is flashing.
Sys 6>> Ok.  Please open your
browser.
```

We study lexical alignment in the context of troubleshooting dialogues – where users wish to mend their broken internet connection. This task demands a new type of User Simulation.

In this paper, after a review of related work (section 2), we first present (section 3) a statistical User Simulation which supports different domain knowledge profiles and reacts accordingly to questions or instructions from an SDS. We then present a reinforcement learning model of lexical alignment due to audience design (in sections 4 & 5). We then evaluate the User Simulation (section 6), testing whether a simulation that is sensitive to a system's RE choices can be used to learn good lexical alignment policies. Finally, we compare policies learned in interaction with the User Simulation with hand-coded policies, and present the results in section 7.

2 Related work

Several statistical user simulation models that model a user's behaviour in a conversation have been proposed (Georgila et al., 2005; Schatzmann et al., 2006; Schatzmann et al., 2007). These models issue task specific dialogue acts like informing their search constraints, confirming values, rejecting misrecognised values, etc. However, they do not model a user population with varying domain expertise. Also, none of these models seek clarification at conceptual or lexical levels that occur naturally in conversations between real users. (Komatani et al., 2003) proposed using user models with features like skills, domain knowledge and hastiness as a part of the dialogue manager to produce adaptive responses. (Janarthanam and Lemon, 2008) presented a user simulation model that simulates a variety of users with different domain knowledge profiles. Although this model incorporated clarification acts at the conceptual level, these users ignore the issues concerning the user's understanding of the REs used by the system. In this work, in contrast to the above, we present a User Simulation model which explicitly encodes the user's lexical knowledge of the domain, understands descriptive expressions, and issues clarification requests at the lexical level.

3 User Simulation

Our User Simulation module simulates dialogue behaviour of different users, and interacts with the dialogue system by exchanging both dialogue acts and REs. It produces users with different knowledge profiles. The user population produced by the simulation comprises a spectrum from complete novices to experts in the domain. Simulated users behave differently from one another because of differences in their knowledge profiles. Simulated users are also able to learn new REs during interaction with the SDS. These new expressions are held in the user simulation's short term memory for later use in the conversation. Simulated users interact with the environment using an interactive mechanism that allows them to observe and manipulate the states of various domain objects. The interaction between the user and the other components is given in figure 1 (notations explained in later sections).

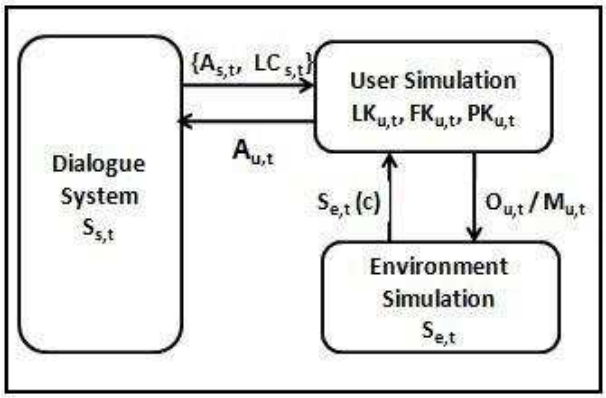

Figure 1: Experimental setup

3.1 Domain knowledge model

Domain experts know most of the technical terms that are used to refer to domain objects whereas novice users can only reliably identify them when descriptive expressions are used. While in the model of (Janarthanam and Lemon, 2008) knowledge profiles were presented only at conceptual levels (e.g. does the user know what a modem is?), we present them in a more granular fashion. In this model, the user's domain knowledge profile is factored into lexical ($LK_{u,t}$), factual ($FK_{u,t}$) and procedural knowledge ($PK_{u,t}$) components.

Lexical knowledge $LK_{u,t}$
vocab([modem, router], dobj1)
vocab([wireless, WiFi], dobj3)
vocab([modem power light], dobj7)
Factual knowledge $FK_{u,t}$
location(dobj1)
location(dobj7)
Procedural knowledge $PK_{u,t}$
procedure(replace_filter)
procedure(refresh_page)

Table 1: Knowledge profile - Intermediate user.

A user's lexical knowledge is encoded in the format:

vocab(referring_expressions, domain_object)

where *referring_expressions* can be a list of expressions that the user knows can be used to talk about each $domain\_object$.

Whether the user knows facts like the location of the domain objects *(location(domain_object))* is encoded in the factual component. Similarly, the procedural component encodes the user's knowledge of how to find or manipulate domain objects (*procedure(domain_action)*). Table 1 shows an example user knowledge profile.

In order to create a knowledge spectrum, a Bayesian knowledge model is used. The current model incorporates patterns of only the lexical knowledge among the users. For instance, people who know the word "router" most likely also know "DSL light" and "modem" and so on. These dependencies between REs are encoded as conditional probabilities in the Bayesian model. Figure 2 shows the dependencies between knowledge of REs.

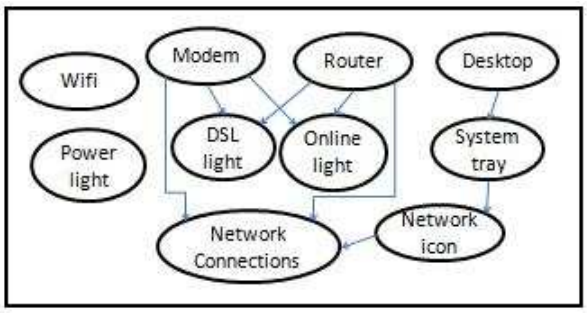

Figure 2: Bayes Net for User Lexical Knowledge

Using this Bayesian model, we instantiate different knowledge profiles for different users. The current conditional probabilities were set by hand based on intuition. In future work, these values will be populated based on simple knowledge surveys performed on real users (Janarthanam and Lemon, 2009). This method creates a spectrum of users from ones who have no knowledge of technical terms to ones who know all the technical jargon, though every profile will have a different frequency of occurrence. This difference in frequency reflects that expert users are less common than novice users.

The user's domain knowledge can be dynamically updated. The new REs, both technical and descriptive, presented by the system through clarification moves are stored in the user's short term memory. Exactly how long (in terms of dialogue turns) to retain the newly acquired knowledge is given by a retention index RI_u. At the end of RI_u turns, the lexical item is removed from user's short term memory.

3.2 User Dialogue Action set

Apart from environment-directed acts, simulated users issue a number of dialogue acts. The list of dialogue actions that the user can perform in this model is given in Table 2. It consists of default moves like *provide_info* and *acknowledge* as well as some clarification moves. *Request_description* is issued when the SDS uses technical terms that the simulated user does not know, e.g. "What is a router?". *Request_verification* is issued when the SDS uses descriptive lexical items for domain objects that the user knows more technical terms for, e.g. System: "Is the black box plugged in?" User: "Do you mean the router?". *Request_disambiguation* is issued when the user faces an underspecified and ambiguous descriptive expression, e.g."User: I have two black boxes here - one with lights and one without. Which one is it?". These clarification strategies have been modeled based on (Schlangen, 2004). The user simulation also issues *request_location* and *request_procedure* dialogue acts, when it does not know the location of domain objects or how to manipulate them, respectively.

3.3 Environment simulation

The environment simulation includes both physical objects, such as the computer, modem, ADSL filter, etc and virtual objects, such as the browser, control panel, etc in the user's environment. Physical and virtual connections between these objects

report_problem
provide_info(dobj, info)
acknowledge
request_verification(x, y)
request_description(x)
request_disambiguation(x, [y1,y2])
request_location(dobj)
request_procedure(daction)
thank_system

Table 2: User Dialogue Acts.

are also simulated. At the start of every dialogue, the environment is initiated to a faulty condition. Following a system instruction or question, the user issues two kinds of environment acts. It issues an observation act $O_{u,t}$ to observe the status of a domain object and a manipulation act $M_{u,t}$ to change the state of the environment ($S_{e,t}$). The simulation also includes task irrelevant objects in order to confuse the users with underspecified descriptive expressions. For instance, we simulate two domain objects that are black in colour - an external hard disk and a router. So, the users may get confused when the system uses the expression, "black box".

3.4 User Action Selection

User Action selection has several steps. The user's dialogue behaviour is described in the action selection algorithm (Table 3). Firstly, the user must identify all the RE choices ($REC_{s,t}$) that are used to refer to different domain objects ($dobj$) and domain actions ($daction$) in the system instruction (step 1). Secondly, the user's knowledge of the prerequisite factual (FK_{prereq}) and procedural (PK_{prereq}) knowledge components connected to the observation or manipulation action is checked. If the user does not satisfy the knowledge requirements, the user simulation issues an appropriate clarification request (steps 2 & 3). After the knowledge requirements are satisfied, the user issues environment directed actions and responds to system instruction $A_{s,t}$ (steps 4 & 5). When the system provides the user specific information, they are added to the user's short term memory (steps 6-8). Although, the action selection process is deterministic at this level, it is dependent on the users' diverse knowledge profiles, which ensures stochastic dialogue behaviour amongst different users created by the module.

greet_the_user
request_status(x)
request_action(x)
give_description(x)
accept_verification(x,y)
give_location(dobj)
give_procedure(daction)
close_dialogue

Table 4: System Dialogue acts.

4 Dialogue System Model

The dialogue system is modeled as a reinforcement learning agent in a Markov Decision Process framework (Levin et al., 1997). At every turn, it interacts with the Simulated User by issuing a System Dialogue Act ($A_{s,t}$) along with a set of REs, called the System RE Choices ($REC_{s,t}$). $REC_{s,t}$ contains the REs that refer to various domain objects in the dialogue act $A_{s,t}$. First, the system decides the dialogue act to issue using a hand-coded dialogue strategy. Troubleshooting instructions are coded in the troubleshooting decision tree[1]. Dialogue repair moves include selecting clarification moves in response to user's request. The list of system dialogue acts is given Table 4.

The system issues various repair moves when the users are unable to carry out the system's instructions due to ignorance, non-understanding or the ambiguous nature of the instructions. The *give_description* act is used to give the user a description of the domain object previously referred to using a technical term. It is also used when the user requests disambiguation. Similarly, *accept_verification* is given when the user wants to verify whether the system is referring to a certain domain object *y* using the expression *x*.

After selecting the dialogue act $A_{s,t}$, a set of REs must be chosen to refer to each of the domain objects/actions used in the dialogue act. For instance, the dialogue act *request_status(router_dsl_light)* requires references to be made to domain objects "router" and "DSL light". For each of these references, the system chooses a RE, creating the System RE Choice $REC_{s,t}$. In this study, we have 7 domain objects and they can either be referred to using technical

[1]The Troubleshooting decision tree was hand-built using guidelines from www.orange.co.uk and is similar to the one used by their Customer Support personnel

Input:	System Dialogue Act $A_{s,t}$, System Referring Expressions Choice $REC_{s,t}$ and User State $S_{u,t}$: $LK_{u,t}$, $FK_{u,t}$, $PK_{u,t}$
Step 1.	$\forall$ x $\in REC_{s,t}$
Step 1a.	if (vocab(x, dobj)$\in LK_{u,t}$) then next x.
Step 1b.	else if (description(x, dobj) & $\exists$ j ((is_jargon(j) & vocab(j, dobj) $\notin LK_{u,t}$))) then next x.
Step 1c.	else if (is_jargon(x) & (vocab(x, dobj) $\notin LK_{u,t}$)) then return request_description(x).
Step 1d.	else if (is_ambiguous(x)) then return request_disambiguation(x).
Step 1e.	else if (description(x, dobj) & $\exists$ j ((is_jargon(j) & vocab(j, dobj) $\in LK_{u,t}$))) then return request_verification(x, j).
Step 2.	if ($\exists$dobj location(dobj) $\in FK_{prereq}$ & location(dobj) $\notin FK_{u,t}$) then return request_location(dobj).
Step 3.	else if ($\exists$daction procedure(daction) $\in PK_{prereq}$ & procedure(daction) $\notin PK_{u,t}$) then return request_procedure(daction).
Step 4.	else if ($A_{s,t}$ = request_status(dobj)) then observe_env(dobj, status), return provide_info(dobj, status)
Step 5.	else if ($A_{s,t}$ = request_action(daction)) then manipulate_env(daction), return acknowledge.
Step 6.	else if ($A_{s,t}$ = give_description(j, d) & description(d, dobj)) then add_to_short_term_memory(vocab(j, dobj)), return acknowledge.
Step 7.	else if ($A_{s,t}$ = give_location(dobj)) then add_to_short_term_memory(location(dobj)), return acknowledge.
Step 8.	else if ($A_{s,t}$ = give_procedure(daction)) then add_to_short_term_memory(procedure(daction)), return acknowledge.

Table 3: Algorithm: Simulated User Action Selection

terms or descriptive expressions. For instance, the DSL light on the router can be descriptively referred to as the "second light on the panel" or using the technical term, "DSL light". Sometimes the system has to choose between a lesser known technical term and a well-known one. Some descriptive expressions may be underspecified and therefore can be ambiguous to the user (for example, "the black box"). Choosing inappropriate expressions can make the conversation longer with lots of clarification and repair episodes. This can lead to long frustrating dialogues, affecting the task success rate. Therefore, the dialogue system must learn to use appropriate REs in its utterances. The RE choices available to the system are given in Table 5.

The system's RE choices are based on a part of the dialogue state that records which of the technical terms the user knows. These variables are initially set to *unknown* (u). During the dialogue, they are updated to *user_knows* (y) or *user_doesnot_know* (n) states. We therefore record the user's lexical knowledge during the course of the dialogue and let the system learn the statistical usage patterns by itself. Part of the dialogue state relevant to system's RE choices is given in Table 6.

1. router / black box / black box with lights
2. power light / first light on the panel
3. DSL light / second light on the panel
4. online light / third light on the panel
5. network icon / flashing computer symbol
6. network connections / earth with plug
7. WiFi / wireless

Table 5: System RE choices.

The state can be extended to include other relevant information like the usage of various REs by the user as well to enable alignment with the user through priming (Pickering and Garrod, 2004) and personal experience (Clark, 1996). However they are not yet implemented in the present work.

5 Reward function

The reward function calculates the reward awarded to the reinforcement learning agent at the end of each dialogue session. Successful task completion is rewarded with 1000 points. Dialogues running beyond 50 turns are deemed

Feature	Values
user_knows_router	y/n/u
user_knows_power_light	y/n/u
user_knows_dsl_light	y/n/u
user_knows_online_light	y/n/u
user_knows_network_icon	y/n/u
user_knows_network_connections	y/n/u
user_knows_wifi	y/n/u

Table 6: (Part of) Dialogue state for Lexical Alignment.

unsuccessful and are awarded 0 points. The number of turns in each dialogue varies according to the system's RE choices and the simulated user's response moves. Each turn costs 10 points. The final reward is calculated as follows:

$TaskCompletionReward(TCR) = 1000$
$TurnCost(TC) = 10$
$TotalTurnCost(TTC) = \#(Turns) * TC$
$FinalReward = TCR - TTC$

The reward function therefore gives high rewards when the system produces shorter dialogues, which is possible by adaptively using appropriate REs for each user.

6 Training

The system was trained to produce an adaptive lexical alignment policy, which can adapt to users with different lexical knowledge profiles. Ideally, the system must interact with a number of different users in order to learn to align with them. However, with a large number of distinct Bayesian user profiles (there are 90 possible user profiles), the time taken for learning to converge is exorbitantly high. Hence the system was trained with selected profiles from the distribution. It was initially trained using two user profiles from the very extremes of the knowledge spectrum produced by the Bayesian model - complete experts and complete novices. In this study, we calibrated all users to know all the factual and procedural knowledge components, because the learning exercise was targeted only at the lexical level. With respect to the lexical knowledge, complete experts knew all the technical terms in the domain. Complete novices, on the other hand, knew only one: $power\_light$. We set the RI_u to 10, so that the users do not forget newly learned lexical items for 10 subsequent turns. Ideally, we expected the system to learn to use technical terms with experts and to use descriptive expressions with novices and a mixture for intermediates. The system was trained using SARSA reinforcement learning algorithm (Sutton and Barto, 1998), with linear function approximation, for 50000 cycles. It produced around 1500 dialogues and produced an alignment policy (RL1) that adapted to users after the first turn which provides evidence about the kind of user the system is dealing with.

The system learns to get high reward by producing shorter dialogues. By learning to choose REs by adapting to the lexical knowledge of the user, it avoids unnecessary clarification and repair episodes. It learns to choose descriptive expressions for novice users and jargon for expert users. It also learns to use technical terms when all users know them (for instance, "power_light"). Due to the user's high retention (10 turns), the system learned to use newly learned items later in the dialogue.

We also trained another alignment policy (RL2) with two other intermediate high frequency user lexical profiles. These profiles (Int1 and Int2) were chosen from either ends of the knowledge spectrum close to the extremes. Int1 is a knowledge profile that is close to the novice end. It only knows two technical terms: "power_light" and "WiFi". On the other hand, Int2 is profile that is close to the expert end and knows all technical terms except: "dsl_light" and "online_light" (which are the least well-known technical terms in the user population). With respect to the other knowledge components - factual and procedural, both users know every component equally. We trained the system for 50000 cycles following the same procedure as above. This produced an alignment policy (RL2) that learned to optimize the moves, similar to RL1, but with respect to the given distinct intermediate users.

Figure 3 shows the overall dialogue reward for the 2 policies during training.

Both policies RL1 and RL2, apart from learning to adapt to the users, also learned not to use ambiguous expressions. Ambiguous expressions lead to confusion and the system has to spend extra turns for clarification. Therefore both policies learnt to avoid using ambiguous expressions.

Figure 4 shows the dialogue length variation for the 2 policies during training.

7 Evaluation and baselines

We evaluated both the learned policies using a testing simulation and compared the results to other baseline hand-coded policies. Unlike the training simulation, the testing simulation used the Bayesian knowledge model to produce all different kinds of user knowledge profiles. It produced around 90 different profiles in varying distribution, resembling a realistic user population. The tests were run over 250 simulated dialogues each.

Several rule-based baseline policies were manually created for the sake of comparison:

1. Random - Choose REs at random.
2. Descriptive only - Only choose descriptive expressions. If there is more than one descriptive expression it picks one randomly.
3. Jargon only - Chooses the technical terms.
4. Adaptive 1 - It starts with a descriptive expression. If the user asks for verification, it switches to technical terms for the rest of the dialogue.
5. Adaptive 2 - It starts with a technical term and switches to descriptive expressions if the user does not understand in the first turn.
6. Adaptive 3 - This rule-based policy adapts continuously based on the previous expression. For instance, if the user did not understand the technical reference to the current object, it uses a descriptive expression for the next object in the dialogue.

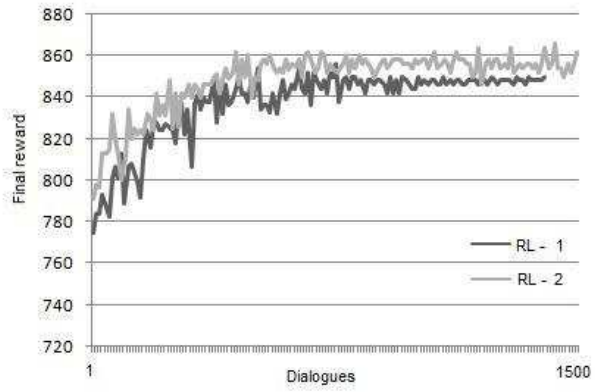

Figure 3: Final reward for RL1 & RL2.

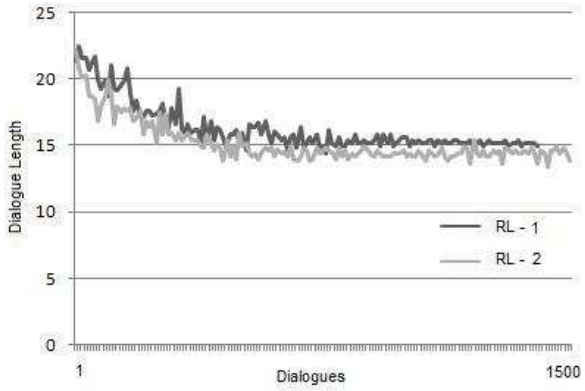

Figure 4: Dialogue length for RL1 & RL2.

Policy	Avg. Reward	Avg. Length
RL2	830.4	16.98
RL1	812.3	18.77
Adaptive 1	809.6	19.04
Adaptive 2	792.1	20.79
Adaptive 3	780.2	21.98
Random	749.8	25.02
Desc only	796.6	20.34
Jargon only	762.0	23.8

Table 7: Rewards and Dialogue Length.

The first three policies (random, descriptive only and jargon only) are equivalent to policies learned using user simulations that are not sensitive to system's RE choices. In such cases, the learned policies will not have a well-defined strategy to choose REs based on user's lexical knowledge. Table 7 shows the comparative results for the different policies. RL (1 & 2) are significantly better than all the hand-coded policies. Also, RL2 is significantly better than RL1 ($p < 0.05$).

Ideally the system with complete knowledge of the user would be able to finish the dialogue in 13 turns. Similarly, if it got it wrong every time it would take 28 turns. From table 7 we see that RL2 performs better than other policies, with an average dialogue length of around 17 turns. The learned policies were able to discover the hidden dependencies between lexical items that were encoded in the Bayesian knowledge model. Although trained only on two knowledge profiles, the learned policies adapt well to unseen users, due to the generalisation properties of the linear function approximation method. Many unseen states arise when interacting with users with new profiles and both the learned policies generalise very well in such situations, whereas the baseline policies do not.

8 Conclusion

In this paper, we have shown that by using a statistical User Simulation that is sensitive to RE choices we are able to learn NLG policies that adaptively decide which REs to use based on audience design. We have shown that the lexical alignment policies learned with this type of simulation are better than a range of hand-coded policies.

Although lexical alignment policies could be hand-coded, the designers would need to invest significant resources every time the list of referring expressions is revised or the conditions of the dialogue change. Using reinforcement learning, near-optimal lexical alignment policies can be learned quickly and automatically. This model can be used in any task where interactions need to be tailored to different users' lexical knowledge of the domain.

8.1 Future work

Lexical alignment in dialogue also happens due to priming (Pickering and Garrod, 2004) and personal experience (Clark, 1996). We will examine trade-offs in various conditions, like 'instruct' versus 'teach' and low versus high retention users. Using Wizard-of-Oz studies and knowledge surveys, we plan to make the model more data-driven and realistic (Janarthanam and Lemon, 2009). We will also evaluate the learned policies with real users.

Acknowledgements

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

References

A. Bell. 1984. Language style as audience design. *Language in Society*, 13(2):145–204.

H. H. Clark. 1996. *Using Language*. Cambridge University Press, Cambridge.

K. Georgila, J. Henderson, and O. Lemon. 2005. Learning User Simulations for Information State Update Dialogue Systems. In *Proceedings of Eurospeech/Interspeech*.

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

S. Janarthanam and O. Lemon. 2008. User simulations for online adaptation and knowledge-alignment in Troubleshooting dialogue systems. In *Proc. SEMdial'08*.

S. Janarthanam and O. Lemon. 2009. A Wizard-of-Oz environment to study Referring Expression Generation in a Situated Spoken Dialogue Task. In *Proc. ENLG'09*.

K. Komatani, S. Ueno, T. Kawahara, and H. G. Okuno. 2003. Flexible Guidance Generation using User Model in Spoken Dialogue Systems. In *Proc. ACL'03*.

O. Lemon. 2008. Adaptive Natural Language Generation in Dialogue using Reinforcement Learning. In *Proc. SEMdial'08*.

E. Levin, R. Pieraccini, and W. Eckert. 1997. Learning Dialogue Strategies within the Markov Decision Process Framework. In *Proceedings of ASRU97*.

R. Molich and J. Nielsen. 1990. Improving a Human-Computer Dialogue. *Communications of the ACM*, 33-3:338–348.

M. J. Pickering and S. Garrod. 2004. Toward a mechanistic psychology of dialogue. *Behavioral and Brain Sciences*, 27:169–225.

V. Rieser and O. Lemon. 2009. Natural Language Generation as Planning Under Uncertainty for Spoken Dialogue Systems. In *Proc. EACL'09*.

J. Schatzmann, K. WeilHammer, M. N. Stuttle, and S. J. Young. 2006. A Survey of Statistical User Simulation Techniques for Reinforcement Learning of Dialogue Management Strategies. *Knowledge Engineering Review*, pages 97–126.

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

D. Schlangen. 2004. Causes and strategies for requesting clarification in dialogue. *Proceedings of the 5th SIGdial Workshop on Discourse and Dialogue (SIGDIAL 04), Boston*.

R. Sutton and A. Barto. 1998. *Reinforcement Learning*. MIT Press.

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