

*Does this list contain what you were searching
for? Learning adaptive dialogue strategies for
Interactive Question Answering*

V. RIESER

*Department of Computational Linguistics, Saarland University
Saarbrücken, D-66041*

O. LEMON

*School of Informatics, University of Edinburgh
Edinburgh, EH8 9LW, GB*

(*Received 17 January 2009*)

Abstract

Policy learning is an active topic in dialogue systems research, but it has not been explored in relation to Interactive Question Answering (IQA). We take a first step in learning adaptive interaction policies for QA: we address the question of how to acquire enough reliable query constraints, how many database results to present to the user and when to present them, given the competing trade-offs between the length of the answer list, the length of the interaction, the type of database, and the noise in the communication channel.

The operating conditions are reflected in an objective function which we use to derive a hand-coded threshold-based policy and rewards to train a Reinforcement Learning policy. The same objective function is used for evaluation. We show that we can learn strategies for this complex trade-off problem which perform significantly better than a variety of hand-coded policies, for a wide range of noise conditions, user types, types of DB, and turn-penalties. Our policy learning framework thus covers a wide spectrum of operating conditions. The learned policies produce an average *relative* increase in reward of 86.78% over the hand-coded policies. In 93% of the cases the learned policies perform highly significantly better than the hand-coded ones ($p < .001$). Furthermore we show that the type of database has a significant effect on learning and we give qualitative descriptions of the learned IQA policies.

1 Introduction

The problems of determining desirable system behaviour for Interactive Question Answering (IQA) and of optimizing dialogue strategies are strongly connected. Dialogue policy learning has focussed on the issue of determining the best system “dialogue move” in any situation, leading to efficient completion of a domain task. These domain tasks are usually “slot-filling” tasks such as booking a flight (Walker et al.2001) or finding tourist information (Singh et al.2002). However, when we

consider the more open-domain tasks associated with IQA, it becomes clear that simply gathering sufficient query constraints or “slots” (e.g. destination city, food type, artist name etc.) and presenting suitable item(s) to the user is only a special case of the more general IQA problem.

In this work we address the question *how many* database search results to present to the user and *when* to present them. In other words, when to stop gathering query constraints and present a list of possible answer candidates.¹ This question becomes especially relevant as search engines now become available on mobile devices such as Google Voice Local Search and Mobile Search² with voice and text input. One of the challenges of using search engines for mobile devices is that the screen size is limited and the channel quality (typing on a small handset or speech recognition) is quite noisy. In the following we present IQA policies to address this problem. We learn optimal solutions for confirming noisy input and when to present a list of items on the screen. When displaying a list we also verbally notify the user. In current work we learn a more fine-grained strategy when to use multimodal output or verbal summaries only. The learned strategies are optimal with respect to the overall noise condition, database retrieval, and the importance of query efficiency (i.e. time constraints).

To combine statistical approaches to dialogue systems with IQA we need to address various shortcomings of the current approaches as outlined below.

1.1 Relation of IQA and multimodal dialogue systems

A long-standing strand of research in NLP is in natural language access to databases (Androutsopoulos and Ritchie2000). It has mainly focused on mapping natural language input to database queries. More recently, work on question answering (QA) is moving toward interactive question answering that gives the user a greater role in the QA process. This paper explores multimodal dialogue systems as IQA systems for database access. Our main focus is on interactive query refinement and answer presentations, i.e. creating strategies to give useful feedback to the user while s/he is querying the database. QA systems mostly operate on free text whereas we use relational databases. In current work we explore combined techniques which also allow us to retrieve answers from free text (Zhao2007).

Below, we create strategies for different application environments where dialogue systems are typically employed, such as in-car or in-home applications. For example, in eyes-busy scenarios such as in moving cars speech-in list-out strategies have been shown to reduce the cognitive load of the driver (Divi et al.2004). We assume that this will depend on the driving situation and the screen size, so that we need a policy for presenting only a very few items to the driver. In the home or office, where we assume that users have large displays available to them, we may learn policies which

¹ Note that we are still in the domain of factoid QA, i.e. a named entity serves as an answer candidate, but we don't restrict ourselves to a single best answer. We can let the user choose from a list of possible candidates.

² <http://labs.google.com/goog411/> and www.google.com/mobile/search/

present more answers to the user for further clarification. Given a noisy multimodal context, dialogue complexity increases, the feasibility of handcrafted strategy design becomes questionable, and automatic strategy learning becomes promising. Work in interactive QA has not previously considered the “optimality” of policies to assist the user in the query phase.

1.2 Prior work: Information-seeking dialogue systems

Information seeking dialogue systems (such as COMMUNICATOR (Walker et al.2001) or NJFUN (Singh et al.2002)) attempt to determine the user’s goal and eventually present a unique result to the user. For these systems one of the major under-explored problems is the nature of the database (DB) and corresponding decisions regarding presentation of results to the user, i.e. how many items are retrieved at each stage in the dialogue? and how do these items correspond to the (hypothesized) user’s goal? Should the current results be presented to the user, or should the system continue asking for more search constraints? In this paper we consider multimodal question answering dialogues as shown in example 1.³

- (1) User: ‘‘Please search for titles by Madonna.’’
 Wizard: ‘‘Please wait a moment [...]’’
 I found seventeen hundred and eleven items.
 The items are displayed on the screen’’
 User: ‘‘Please select ’Secret’.’’

This example is taken from the SAMMIE Wizard-of-Oz corpus (Rieser et al.2005) (translated from German). In this study, different wizards play the role of a multimodal interface. Note that the wizard decides to display a list of 1,711 database items while the user is driving. This example illustrates that even for humans it is difficult to find an “optimal” solution to the problem we are trying to solve.

There has been substantial work on using dialogue systems for accessing databases. One line of research aims to assist the user to *browse* the data by generating intelligent summaries at each step of the dialogue, e.g. (Demberg and Moore2006; Chung2004; Varges et al.2006), the other line helps the user to *search* for a concrete item by gathering more constraints from the user until a manageable number of results to enumerate are retrieved, e.g. (Pietquin2006; Levin et al.2000). We are focusing on the latter question.

In prior work (see also section 6) the problem of when to present a list of results has been addressed by setting thresholds in rule-based systems (Chung2004; Varges et al.2006). This problem has also been addressed using Reinforcement Learning (Pietquin2006; Levin et al.2000). In this work database matches have been modelled in simple ways, with a quantised state variable monitoring e.g. “High” “Medium” or “Low” numbers of database search results. The vast majority of dialogue policy

³ Our example domain is music search and therefore all the given dialogue examples are from this domain. Note that our approach is not limited to one specific domain and learned slot-filling strategies do carry over to other domains (Lemon et al.2006a).

learning research does not consider the number of database results at all, resulting in policies which are not sensitive to the current number of search results. This is obviously deficient: if there is only one DB search result (or “hit”), we should tell it to the user immediately, regardless of how many information search slots have been filled in the dialogue. Conversely, if there are very many DB hits, we should probably persevere in getting more information (i.e. search constraints) from the user, depending on the penalty for longer dialogues and the reliability of the ASR channel.

Note that there has been substantial work using decision theoretic approaches to (sub-)dialogue policy control, e.g. (Paek and Horvitz2000; Dohsaka et al.2003; Skantze2007). In future work it will be interesting to apply these approaches to the current problem and investigate how their performance compares to global optimisation methods such as RL.

1.3 Structure

The structure of the paper is as follows: we describe the experimental setup in section 2, the hand-coded baselines in section 3, and the methodology in section 4. Section 5 presents our learning results, for a range of noise and turn penalty conditions and database models. We then compare our results with previous work in section 6, and conclude, with directions for future work, in section 7.

2 Experimental Framework

2.1 Dialogue as a Markov Decision Process

The Markov-Decision-Process (MDP) model serves as a formal representation of human-machine dialogue (Levin and Pieraccini1997) and provides the basis for formulating strategy learning problems (Sutton and Barto1998). Every MDP is formally described by a state space, an action set, a set of transition probabilities, and a reward function. Reinforcement Learning (RL) methods are used to determine optimal dialogue policies: mappings of dialogue states to dialogue actions. For training these systems researchers have modelled various factors in the environment. These factors include noise introduced by ASR (e.g. (Pietquin and Dutoit2006)) or user simulations (e.g. (Georgila et al.2006)).

2.2 Database retrieval simulations

To the authors’ knowledge this paper is the first which uses simulated database retrieval to investigate the effects that the nature of the database, together with noise, has on policy learning.

We developed 2 types of database retrieval simulations (“monotonic” and “random”) for training IQA policies, both reflecting different ways a user query can be interpreted. These retrieval types are also known as “narrowing” and “expanding” (Strzalkowski et al.2008). In both cases, we assume a total database size of 100

items. We assume that presenting more than 100 results to a user is never going to be desirable, so in this paper we learn IQA strategies for the phase of the QA process where there are 100 or fewer possible answers. When the number of results is more than 100, we assume that the correct strategy is to ask for more constraints from the user. Thus for DB sizes of >100 we would ask for user constraints until 100 or fewer results are obtained, in which case the learned strategies will be applied. We also assume that the user’s goal item is contained somewhere in the database. Here we don’t consider when to ask for constraint relaxation (i.e. when zero results are retrieved).

2.2.1 Monotonic DB retrieval simulation

The first, “monotonic” simulation, models search tasks where each additional search constraint strictly reduces the number of search results obtained. Thus, if the user fills search slots (e.g. “I want a song by Arcade Fire” or “Is there a Sushi place near the castle?”) the number of results returned is strictly less than in the prior state. This models boolean “AND” search or narrowing retrieval. Conversely, if a slot becomes unfilled (e.g. by the user rejecting a system confirmation move), the number of search results will increase. Here we sample from a normally distributed database. Every search constraint (i.e. filled slot) lowers the mean (μ_{db} , equation 2) and narrows down the standard deviation (δ_{db} , equation 3), i.e. for fewer search constraints the distribution will be flatter (number of hits clusters more widely around the mean and the curve is closer to a random distribution); the more the search becomes constrained the sharper the distribution becomes, and the number of possible results clusters closer to the mean.

$$(2) \quad \mu_{db} = \frac{databaseSize}{2^{no. filledslots}}$$

$$(3) \quad \delta_{db} = \frac{databaseSize}{e^{no. filledslots}}$$

2.2.2 Random DB retrieval simulation

For the second database model (“random”) a user-provided search constraint (slot) can either be interpreted as an “AND” or an “OR” constraint., i.e. expanding retrieval is also possible. That is, the number of DB hits can either increase or decrease, and this is approximated in a random model, sampling between 1 and 100 hits. For this model newly provided information may in fact open up new possibilities in the data. For example the user might say “how about restaurants in the Old Town?” thus shifting focus away from the current set of results and opening up a new (possibly larger) set of search results. Here we simply generate a new number of DB search results after every information-providing or retracting user move.

2.3 MDP state and action spaces

We represent a 4-slot dialogue search problem as an MDP. An overview of the employed state-action space used for learning is given in Figure 1.

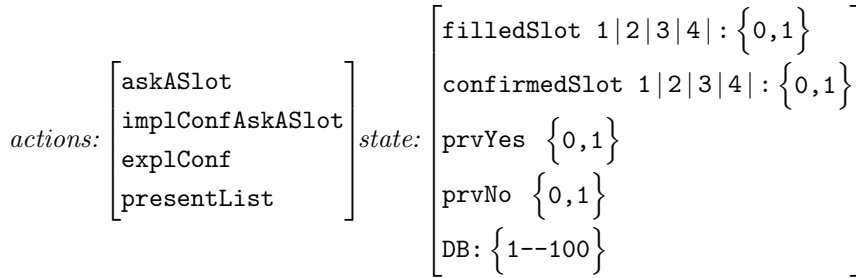


Fig. 1. State-action space for learning

There are 10 binary state variables (for $1 \leq N \leq 4$, `filledSlotN` for whether each slot number N is filled, `confirmedSlotN` for whether each slot number N is confirmed, `prvYes` for whether the last user move was “yes” and `prvNo` for whether the last user move was “no”), and 1 variable `DB` for the current number of DB hits, which takes integer values between 1 and 100, resulting in $2^{10} \times 100 = 102,400$ distinct dialogue states.

The following system actions are available for exploration in every state:

- `ask a slot` (`askASlot`), e.g. “Which artist would you like?”
- `explicit confirm` (`explConf`), e.g. “Did you say Radiohead?”
- `implicit confirm and ask a slot` (`implConfAskASlot`) e.g. “Okay, music by Radiohead. Which album?”
- `present information` (`presentList`) e.g. “There are N items you might be looking for...”

The action “`greet`” is an open-initiative question for as many search constraints as the user wishes to give: “How may help you?”, and the asked slot name (for `askASlot` and `implConfAskASlot`) is controlled by a process model for the domain which describes a default ordering on slots for the QA task (e.g. first greet the user, then ask for music, then artist, then album, then song title, or ask `city_name`, `food_type`, `price_range`, then location for restaurants). The user is not constrained to follow this ordering, and can also over-provide information using mixed-initiative behaviour such as over-answering. Related work replaces the default ordering of slots by a cluster-based method which determines the next most informative slot to ask at each point in the dialogue (Becker et al.2007).

2.4 User simulations

Model-free (or simulation-based) RL needs to explore every action in every state during training, but therefore allows the exploration of strategies that are not present in existing corpora of human-machine dialogues. Exploratory trial-and-error learning with real users is an expensive, time-consuming, and sadistic procedure, so

in current research user simulations are applied for learning (Pietquin2006; Levin et al.2000; Scheffler and Young2002).

Simulated users are also commonly used when evaluating learned policies: i.e. for testing as well as training. Evaluation with real users is of course preferable, and is our ultimate goal, but previous studies have shown that policy learning results obtained for simulated users do in fact carry over to results for real users (Lemon et al.2006a). The standard methodology is to obtain promising results in simulation first, before testing on real users. This paper therefore presents simulation results that are the natural precursor to a full test with real users.

The user simulations employed here are on the intention level, where the possible user acts are stochastic estimates conditioned on the previous system action $P(a_{user}|a_{system})$. Possible user actions are **yes-answer**, **provide-other** (e.g. “I want ABBA” when asked “What type of music do you want?”), **yes-provide-asked** (e.g. “yes, I want ABBA” when asked “Ok, pop music. What band do you want?”), **no-answer**, **provide-two-slots** (e.g. “I want a Radiohead song from the album OK Computer”), or remain silent. The stochastic estimates are set to simulate a relatively collaborative user⁴. For example, if the system’s previous move was to ask for a slot (**askASlot**), the user has a 20% chance of providing a different slot value, a 70% chance of providing the requested slot value, a 6% chance of providing two slot values, and a 4% chance of remaining silent. For system confirmation moves the likelihood of the user simulations rejecting the confirmed information is the same as the probability of a filled slot being incorrect (Pf), i.e. the user simulation also incorporates the noise model (see Section 2.5).

The stochastic estimates are a valid approximation for our simulation-based experiments. To learn a strategy for a realistic application we have employed a user model learned from data (Rieser and Lemon2006a).

2.5 Objective function

The objective function specifies desired behaviour for a dialogue system and is typically used to evaluate an implemented strategy with real users (Paek2006; Walker2005). In this work we also employ the objective function when designing dialogue policies: the objective function specifies the reward used for training a RL policy and is used to manually tune thresholds in a heuristic/rule-based policy.

The following objective function incorporates noise modelling, an estimate of the importance of dialogue length (dialogue efficiency), and the (visual) cognitive capacity of the user. Our objective is to simulate a wide range of possible application environments rather than modelling one realistic application. Therefore we assume objective functions which correspond to possible application scenarios to be known. In a real application the objective function should be estimated from data, e.g. using the PARADISE framework (Walker et al.2000). We take our objective function to be a function of task completion, dialogue length, and the number of presented items.

⁴ Less collaborative users are explored in (Lemon and Liu2007)

The task completion value is a function of noise in the communication channel where the estimates for the individual slots are independent. We also assume that dialogue length and number of presented items have a negative linear relationship to dialogue quality. In current work we estimate the objective function from Wizard-of-Oz data (Rieser et al.2005).

The parameters are set according to the following scenarios: low noise versus high noise environment, impatient versus patient user, low visual capacity (e.g. small screen, in-car) versus high visual capacity (large screen, in-home). In total there are 8 application scenarios. For each dialogue we have:

$$(4) \quad \begin{aligned} FinalReward = & completionValue \\ & - dialogueLengthPenalty \\ & - DBhitsPenalty \end{aligned}$$

Where dialogueLengthPenalty penalises every system turn (via a TurnPenalty TP per turn) and DBhitsPenalty penalises every item which is presented to the user (via an itemPenalty IP per presented item).

The completionValue of a dialogue is defined as the percentage probability that the user goal is in the result set that they are presented with. For example, if we know with 100% certainty that the user wants Sushi in the Old Town (i.e. 2 slots, both confirmed at 100% probability), then we have a 100% chance of supplying the user with an item that meets their goal. On the other hand, if we are in the same situation but we are only 80% sure that they want Sushi then the probability of their goal being in the list we present them with is only $.8 * 1 * 100 = 80\%$. Thus the completion value of a dialogue is directly related to the probability of search slots being correctly filled, which is in turn related to the noise conditions under which the dialogue is being conducted. Thus, where Pc is the probability of a confirmed slot being correct, and Pf is the probability of a filled slot being correct, where C and F are the number of confirmed slots and filled (but not confirmed) slots respectively, we have:

$$(5) \quad completionValue = 100 * (Pc)^C * (Pf)^F$$

For example, in a High Noise environment, we might set $Pc = 1.0$ and $Pf = 0.5$, reflecting the fact that in a noisy environment unconfirmed slots are fairly likely to be incorrect (50% chance). In a real application domain these probabilities can be estimated from Wizard-of-Oz data and ASR confidence scores (Walker et al.2000). The maximum possible reward for any dialogue is 100 (where TP=0, IP=0 and all slots are confirmed in cases where $Pc = 1$). For an example of the computation of *FinalReward*, consider a 4-slot problem where, turnPenalty (TP) =1, itemPenalty (IP) =10, and where 2 items have been presented to the user at the end of the dialogue after 6 system turns. In the case where 4 slots are filled, but only 3 were confirmed, we would then have (in the same High Noise model as above):

$$\begin{aligned} completionValue &= 100 * 1^3 * 0.5^1 = 50 \\ dialogueLengthPenalty &= 6 * 1 = 6 \end{aligned}$$

$$DBhitsPenalty = 2 * 10 = 20$$

and so ultimately, $FinalReward = 50 - 6 - 20 = 24$.

Unfilled slots are not penalised since the task is completed as soon as the user selects one of the presented items. That is, a successful dialogue still has a single best answer (out of a set of possible candidates), but the user goal can be underspecified in various ways.

We now use this objective function to specify intended behaviour using two techniques: first we manually tune threshold-based policies to maximise the expected objective function, and second we use the objective function to specify the reward functions for RL.

3 The hand-coded baseline IQA policies

We construct a range of hand-coded policies for comparison with learned policies. These baselines are “state of the art” policies in the sense that they allow mixed-initiative interaction and use thresholds (for dialogue length and number of DB hits)

following a similar threshold based approach to that described in (Varges et al.2006). The thresholds are manually adjusted to the objective function for the respective environments.⁵ The different hand-coded baseline policies all use the following basic strategy pattern, modified by contextually appropriate global thresholds for dialogue length (measured in system turns) and number of DB results:

1. **Greet** the user,
2. either **askASlot** (if no slots need to be confirmed) or **implConfAskASlot** (if there are remaining slots to ask and slots to confirm),
3. then repeat 2 until there are no slots left to fill, or one of the thresholds is reached,
4. then **explConf** (explicitly confirm) the remaining filled slots (if any)
5. **PresentInfo**: present the answer set to the user.

The hand-coded baselines will thus always **presentInfo** if all slots are confirmed or the number of items returned from the database is less than a threshold (for example < 7) or the dialogue length exceeds a threshold (for example > 6). The different thresholds for the hand-coded policies thus determine when to stop asking for new constraints. In the hand-coded policies, every constraint needs to be confirmed (either explicitly or implicitly), unless a threshold has been reached.

For the baseline systems, we designed four hand-coded policies where we manually tuned the following combination of thresholds to maximise the objective function:⁶

⁵ Note that other techniques for threshold setting do exist: (Chung2004) proposes to automatically tune thresholds in simulation.

⁶ Note that we did not adapt the hand-coded policy to the noise level, but rather employed a standard confirmation technique.

H_{ss} : short dialogues (≤ 6), short lists (≤ 7): e.g. impatient user, low visual capacity
 H_{ls} : long dialogues (≤ 10), short lists (≤ 7): e.g. patient user, low visual capacity
 H_{sl} : short dialogues (≤ 6), long lists (≤ 14): e.g. impatient user, higher visual capacity
 H_{ll} : long dialogues (≤ 10), long lists (≤ 14): e.g. patient user, higher visual capacity

This partially addresses one of the criticisms made by (Paek2006), that learned policies are usually not compared to hand-coded policies that have been designed for the same objective function.

4 Reinforcement Learning Method

For policy learning we use the system REALL described in (Lemon et al.2006b), which uses the SARSA Reinforcement Learning algorithm (with linear function approximation) to learn over the entire policy space for obtaining 4 information slots.

In the following we report on 16 experiments where we systematically varied the database definitions and objective functions, to explore policy learning for combinations of noise level (LowNoise and HiNoise), database model (monotonicDB and randomDB), hit penalty (LowHit and HiHit), and turn penalty (LowTurn and HiTurn). The objective function for the different environments is now reflected in how we formulate the reward functions for policy learning (i.e. we are “programming by reward”). We used the following reward structures for both monotonic and random DB models:

- HiNoise,HiHit,HiTurn: 50% chance of filled slots being correct, turn penalty=-10, hit penalty=-10; e.g. in-car, low visual capacity, impatient user
- HiNoise,LowHit,HiTurn: 50% chance of filled slots being correct, turn penalty=-10, hit penalty=-1; e.g. in-car, higher visual capacity, impatient user
- HiNoise,HiHit,LowTurn: 50% chance of filled slots being correct, turn penalty=-1, hit penalty=-10; e.g. in-car, low visual capacity, patient user
- HiNoise,LowHit,LowTurn: 50% chance of filled slots being correct, turn penalty=-1, hit penalty=-1; e.g. in-car, higher visual capacity, patient user
- LowNoise,HiHit,HiTurn: 80% chance of filled slots being correct, turn penalty=-10, hit penalty=-10; e.g. in-home, low visual capacity, impatient user
- LowNoise,HiHit,LowTurn: 80% chance of filled slots being correct, turn penalty=-1, hit penalty=-10; e.g. in-home, low visual capacity, patient user
- LowNoise,LowHit,HiTurn: 80% chance of filled slots being correct, turn penalty=-10, hit penalty=-1; e.g. in-home, higher visual capacity, impatient user
- LowNoise,LowHit,LowTurn: 80% chance of filled slots being correct, turn penalty=-1, hit penalty=-1; e.g. in-home, higher visual capacity, patient user.

4.1 Training the IQA policies

In all experiments we ran with the following parameters: learningRate = 0.2, discountRate = 0.95, lambdaOfTemporalDifference = 0.9, AhalvingTime = 16,000, expansion halvingTime = none, exploration halvingTime = 16,000. Please see (Sutton and Barto1998) for details of these RL parameters. Each policy was trained over 96,000 cycles of the system, which resulted in about 12,000 simulated dialogues per training run. Figure 2 shows a training run where the policy learns to reduce the number of database hits presented to the user (grey \star , +1 per presented item) and dialogue length (light grey $+$, -1 per turn) while obtaining a high completionValue (dark grey \circ). The average dialogue reward, computed over windows of 50 dialogues, is shown by the jagged line. After about 6000 dialogues the learner has settled on a policy of confirming all information slots while presenting fewer than 10 items in fewer than 15 turns.

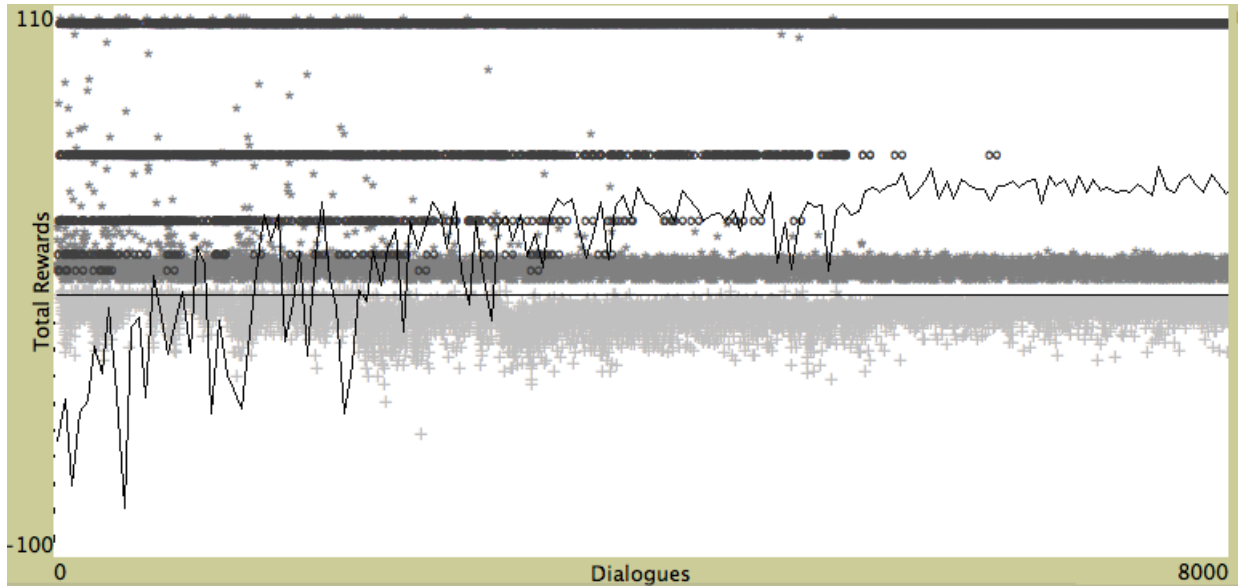


Fig. 2. Learning curve for training HiNoise,HiHit,LowTurn policy, Monotonic DB

In contrast to the hand-coded strategies the graphs in Figure 2 illustrate that learning does not settle on fixed thresholds, but the learner learns the best local trade-off for every dialogue context. After learning converged, the graph still shows some variance for presented items (grey dots) and dialogue length (black dots) illustrating the local trade-offs. In this operating situation efficiency is less important (reflected in a low turn penalty), but confirming and showing short lists are crucial. Therefore the learner is willing to sometimes engage in longer dialogues ($+$) in order to limit the number of items presented (\star) while confirming every filled slot (shown by dark grey \circ at 100% completion rate). For a threshold based policy

the grey and black dots would be distributed to a strict line, i.e. dialogue length and presented items never exceed the specified thresholds. In contrast to previous work we do not quantise the state space features. (Levin et al.2000; Pietquin2006) quantise the number of retrieved database items in order to handle large state spaces for RL. For a quantised DB feature the strategy can only adapt to “low”, “medium”, or “high” numbers of DB results, i.e. implicit thresholds (top and lower boundaries for quantisation) are set. Our approach is not to quantise the features in the state space, but to handle large state spaces by linear function approximation (Sutton and Barto1998). In this way very fine-grained local trade-offs between dialogue length and the number of presented items can be learned.

4.2 Evaluation

We test each (learned and hand-coded) policy in each condition by running 550 test dialogues in simulation. We compare the policies in respect of their average final reward per dialogue over the test runs. We then perform a paired t-test (with Bonferroni correction) on the final rewards, to determine statistical significance.

5 Results

The results produced by the learned policies (denoted RL) and the different hand-coded baselines ($H_{ss} \dots H_{ll}$) for four of the operating conditions (Noise, DB, Hit-Penalty etc.) can be seen in Table 1⁷. In general the learned policies significantly outperform the hand-coded policies with respect to the objective function (the final cumulative reward).

Consider, for example, the pair of graphs in the top row of Table 1. These show the differences between the monotonic and random DB cases, for the case Hi-Noise,LowHit,HiTurn. In the monotonic case, we can see that the learned policy (the graph furthest left) has the fewest turns, but presents the longest lists, and outperforms all the hand-coded configurations. We can see that all policies perform worse in the more challenging random DB case (right column of Table 1), as expected, but the learned policy for the random case has learned that in some cases not all filled slots need to be confirmed (presumably because sometimes users provide new information when asked for confirmation, which can increase the number of DB hits in the random model), and to keep dialogues short. The learned policy here significantly outperforms all the hand-coded configurations (at $p < .001$ or $p < .01$).

Now consider the first column of Table 1. These two graphs show the differences between the LowHit,HiTurn and HiHit,LowTurn cases for monotonic DBs in HighNoise. Moving from the LowHit,HiTurn (top left) to the HiHit,LowTurn (bottom left) case, we see that the learner earns more reward by increasing dialogue length and decreasing the number of presented items, as expected. In both cases

⁷ The others can be viewed online at www.coli.uni-saarland.de/~vrieser/when-to-present-list.html

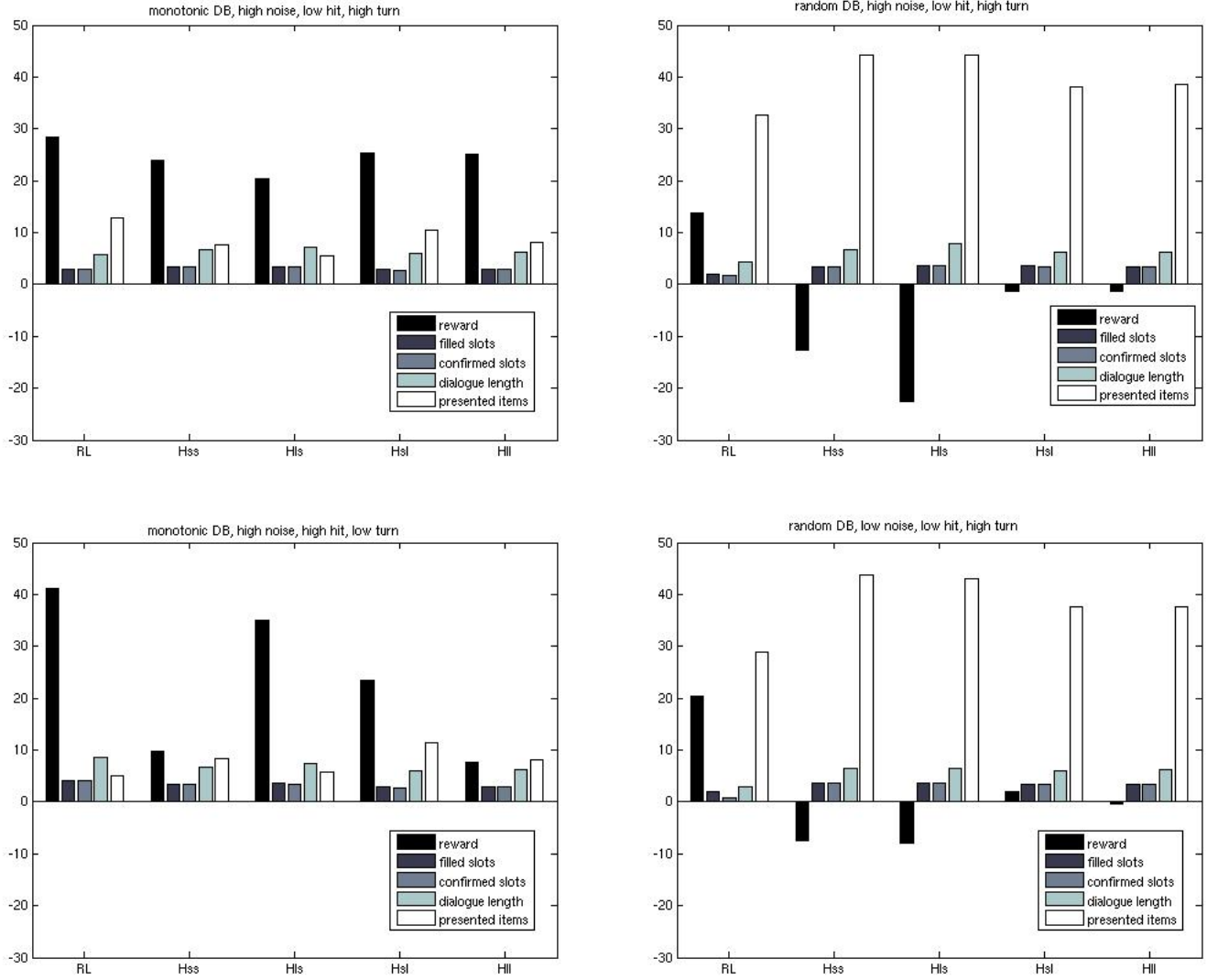


Table 1. Comparison of mean performance measures for learned and hand-coded policies, for monotonic (left) and random (right) DB models

the learned policies significantly outperform all the different hand-coded (threshold-based) configurations ($p < .001$).

Finally, consider the top right (HiNoise,LowHit,HiTurn) and bottom right (LowNoise,LowHit,HiTurn) graphs, which are both for the random DB model. Moving down, we see that in the LowNoise case, the learner has settled on a policy of

confirming fewer slots than in the high noise case (because filled slots are more reliable in low noise). Interactions are thus shorter, and more reward is gained. In both cases the learned policies significantly outperform all the hand-coded configurations ($p < .001$).

Across all the results, the learned policies produce an average *relative* increase in reward of 86.78% over the various hand-coded baseline policies. In 93% of the cases the learned policies perform highly significantly better than hand-coded ($p < .001$). For the remaining 6 out of 84 runs the learned policy still performed better, but not significantly so.

5.1 Discussion

The above results show that, for a complex decision problem, policy learning is able to find more effective and fine-grained IQA strategies than the global strategies implemented by threshold setting in hand-coding. In order to maximise the objective function RL learns local trade-offs, e.g. if dialogue efficiency is less important than reducing the cognitive load, a RL-based policy is willing to engage in longer dialogues in order to reduce the number of presented items.

Considering learning for the different DB models, for monotonic DB simulations more slots were filled and confirmed since there is a direct relation between a smaller number of retrieved items and a greater number of filled slots, i.e. learning adapted to regularities present in the environment without having to explicitly encode them in the reward function. When learning with the random DB simulation we see that the reward definition had a stronger impact on what was learned (as no other regularities were present). For example for LowHit,HiTurn the learned policy for the random DB resulted in extremely short dialogues (3.64 system turns on average) whereas the number of presented items was quite high (30.80 on average). In contrast, for HiHit,LowTurn on the random DB resulted in longer dialogues (6.11 on average) and a lower number of presented items (18.33 on average).⁸ This leads us to the observation that the database which the strategy was learned for has a significant impact. A regression analysis relating *noise levels*, *reward definition*, and *database simulation type* to performance measures, showed that for the measures **confirming slots**, **dialogue length**, and **presented items**, the *database simulation type* is the most predictive factor. For the **filling slots** measure *database simulation type* was the second most important factor.⁹ While (Pietquin2006; Shapiro and Langley2002) show that changes in the reward definition lead to different strategies, this result shows how transition probabilities implicit in the environment can also have a large effect.

Furthermore, for rule-based systems it is quite difficult to manually determine the

⁸ Note that in general the number of presented items for the random DB simulation is significantly higher than for monotonic DB simulation (24.55 vs. 9.73). Detailed results can be viewed online at www.coli.uni-saarland.de/~vrieser/when-to-present-list.html.

⁹ Detailed results can be viewed online.

right combinations of global thresholds *a priori* when the problem becomes complex. Such heuristics are also difficult to transfer to new situations. Even though (for a real application) the specific parameters of the objective function can be estimated from preliminary data (Walker et al.2000), the final application environment might differ (e.g. the noise level changed). Whereas manual thresholds will need to be carefully re-calibrated, RL-based policies are shown to be robustly portable to different conditions (Lemon and Liu2007).

5.2 Qualitative descriptions of learned policies

We now provide a qualitative description of the learned policies, based on examples. We report the number of database hits at each stage of the interaction (“db:”). In example 6 the policy trained for monotonic DB and HiNoise,LowTurn,HiHit has learned to attempt to fill all 4 slots and confirm them before presenting items. It has also learned not to again explicitly confirm if the user rejects (U4), if the dialogue becomes too long. This strategy prefers to present a list of results not including the rejected slot as an search constraint (S5). For the HiNoise environment presenting a list of options rather than asking for explicit confirmation helps to reduce the risk of error spirals (Oviatt et al.1996).

```
(6) S1: ‘‘Hello, how can I help you?’’
      greet
U1:   ‘‘I want a Radiohead song’’
      provide_info(artist_name)
S2:   ‘‘Ok, a Radiohead song. From what album?’’
      implConf(artist_name)AskASlot(album_name) (db: 77)
U2:   ‘‘From OK Computer’’
      provide_info(album)
S3   ‘‘OK from OK Computer. What song title?’’
      implConf(album_name)AskASlot(song_title) (db: 12)
U3:   ‘‘Paranoid Android’’
      provide_info(song_title)
S4:   ‘‘Did you say Karma Police?’’
      explicitConfirm(song_title) (db: 1)
U4:   ‘‘No’’
      no_answer
S5:   ‘‘OK, there are 12 items you might be looking for,
      which are now shown on the screen.’’
      presentList(artist_name,album_name)(db:12)
```

Example 7 shows an example of the policy learned for random DB, LowNoise, LowHit, and HiTurn. The learned policy here immediately presents the list after only one slot is filled, since it has already found a relatively low number of hits, and for a random DB filling more slots does not necessarily result in a lower number of DB hits. Due to the LowNoise condition it has also learned not to confirm the

slot in this situation. This strategy has learned to “take its chance” in this type of situation, once a relatively small number of results has been retrieved.

```
(7) S1 ‘‘Hello, how can I help you?’’
      greet
U1:   ‘‘I want a song from OK Computer.’’
      provide_info(album_name)
S2:   ‘‘OK, there are 12 items you might be looking for,
      which are now shown on the screen.’’
      presentList(artist_name,album_name)(db:12)
```

6 Related work

Prior work using Reinforcement Learning has learned policies for only one specific monotonic database model (Levin et al.2000; Pietquin2006) where two implicit thresholds are used; one for representing the number of database hits in the state space by quantising them (low-medium-high), and one in the reward function.

The state space used for learning is quite limited. (Levin et al.2000)’s system had 411 possible states, (Pietquin2006)’s $3^7 = 2187$ states, whereas this work employs 102,400 distinct dialogue states. These authors do not compare their learned policies to any baseline. (Pietquin2006) shows that the reward definition has an effect on learning, but doesn’t report on statistical significance. With respect to the prior work in this area our approach thus has the following advantages:

- linear function approximation allows us to use integer-valued features for DB hits, and we do not need to quantise. This means we can learn precise (contextual) thresholds and trade-offs for this feature.
- our strategy uses a much richer state space (over 46 times the size of (Pietquin2006)’s and almost 250 times the size of (Levin et al.2000)’s state space) which allows us to solve more complex problems.
- we explore the use of different types of database (monotonic and random).
- we use a reward function modelling ASR noise.
- the learning results are compared to several “state of the art” hand-coded baseline strategies tuned to the same objective function
- our results are shown to be statistically significant.

We believe that the overall stochastic framework that we have presented here is also extensible to a wider range of policy learning problems for Interactive QA in different domains and operating situations. For example, system action sets can be extended in many ways, as can the reward functions and state spaces, to reflect richer IQA problems. The largest problem we have tested the learning environment for has 2^{20} possible states and we do not yet know the practical limit of this technique.

7 Conclusion

We showed how to use a statistical policy learning framework (Reinforcement Learning with the SARSA algorithm and linear function approximation) to address a

complex problem for real IQA applications: how to acquire enough (reliable) query constraints, how many database search results to present to the user, and when to present them, given the competing trade-offs between the length of the answer list, the length of the interaction, the type of database, and the noise in the communication channel. We show that our policy learning framework covers a wide spectrum (in total 16) of possible operating conditions.

The different operating conditions are reflected in an objective function (incorporating likelihood of misrecognition, efficiency, and cognitive capacity) from which we derive hand-coded threshold-based policies and rewards to train our RL policy. The same objective function is used for evaluation. We showed that we can learn strategies for this trade-off complex problem which perform significantly better than a variety of hand-coded policies, for a wide range of noise conditions, user types, types of DB, and turn-penalties. The learned policies produce an average *relative* increase in reward of 86.78% over the hand-coded policies. In 93% of the cases the learned policies perform highly significantly better than the hand-coded ones ($p < .001$). Furthermore we showed that the type of database has a significant effect on learning and we gave qualitative descriptions of the learned IQA policies.

7.1 Future work

In this work we showed that RL adapts optimal policies to a simulated learning environment, but when learning a policy for a real dialogue system the exact parameters are not known in advance. (Lemon and Liu2007) showed that for some training conditions learned policies are transferable to different conditions. However, it is still preferable to have a close estimate of the targeted application environment. In ongoing work we use data from a Wizard-of-Oz data collection to 'bootstrap' a complete learning environment tailored to the application (Rieser et al.2005). We then use feature selection techniques to define the state space (Rieser and Lemon2006b), learn a cluster-based user simulation (Rieser and Lemon2006a), and estimate a (non-linear) objective function. Furthermore we have tested our learned strategies with real users, comparing strategies employed by our human wizards against RL-based policies (forthcoming).

All these avenues will lead to the development of robust, optimal dialogue policies for data-driven IQA applications of real-world complexity. One long term goal of this research is to further integrate information seeking dialogue systems with question answering operating on free text.

Acknowledgements

This research has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n 216594 (CLASSIC project: www.classic-project.org). This work also received funding by the International Research Training Group Language Technology and Cognitive Systems, Saarland University, the EPSRC project number EP/E019501/1 and by Scottish Enterprise under the Edinburgh-

Stanford Link. We thank the referees, as well as Xingkun Liu, Dan Shapiro, and Carl Tollander for their help and comments.

References

- Ion Androutsopoulos and Graeme Ritchie. 2000. Database interfaces. In and Harold Somers Robert Dale, Hermann Moisl, editor, *Handbook of Natural Language Processing*, pages 209–240. Marcel Dekker Inc.
- Tilman Becker, Nate Blaylock, Ciprian Gerstenberger, Andreas Korthauer, Nadine Perera, Michael Pitz, Peter Poller, Jan Schehl, Frank Steffens, Rosmary Stegmann, and Jochen Steigner. 2007. In-car showcase based on TALK libraries. Technical report, Deliverable 5.3, TALK Project.
- Grace Chung. 2004. Developing a flexible spoken dialog system using simulation. In *Proceedings of Association for Computational Linguistics (ACL)*.
- Vera Demberg and Johanna Moore. 2006. Information presentation in spoken dialogue systems. In *Proceedings of European Association for Computational Linguistics (EACL)*.
- Vijay Divi, Clifton Forlines, Jan Van Gemert, Bhiksha Raj, Bent Schmidt-Nielsen, Kent Wittenburg, Joseph Woelfel, Peter Wolf, and Fang fang Zhang. 2004. A speech-in list-out approach to spoken user interfaces. In *Proceedings of Human Language Technologies Conference (HLT)*.
- Kohji Dohsaka, Norhito Yasuda, and Kiyooki Aikawa. 2003. Efficient spoken dialogue control depending on the speech recognition rate and system’s database. In *Proceedings of Eurospeech*.
- Kallirroi Georgila, James Henderson, and Oliver Lemon. 2006. User simulation for spoken dialogue systems: Learning and evaluation. In *Proceedings of Interspeech/ICSLP 2006*.
- Oliver Lemon and Xingkun Liu. 2007. Dialogue policy learning for combinations of noise and user simulations: transfer results. In *Proceedings of 8th SIGdial Workshop*, pages 55–58.
- Oliver Lemon, Kallirroi Georgila, and James Henderson. 2006a. Evaluating effectiveness and portability of Reinforcement Learned dialogue strategies with real users: the TALK TownInfo evaluation. In *Proceedings of Spoken Language Technology (SLT)*.
- Oliver Lemon, Xingkun Liu, Daniel Shapiro, and Carl Tollander. 2006b. Hierarchical Reinforcement Learning of dialogue policies in a development environment for dialogue systems: REALL-DUDE. In *Proceedings of 10th SEMdial Workshop on the Semantics and Pragmatics of Dialogue (BRANDIAL)*.
- Esther Levin and Roberto Pieraccini. 1997. A stochastic model of computer-human interaction for learning dialogue strategies. In *Proceedings of Eurospeech*.
- Esther Levin, Roberto Pieraccini, and Wieland Eckert. 2000. A stochastic model of human-machine interaction for learning dialog strategies. *IEEE Transactions on Speech and Audio Processing*, 8(1):11–23.
- Sharon Oviatt, Gina-Anne Levow, Margaret MacEarchern, and Karen Kuhn. 1996. Modeling hyperarticulate speech during human-computer error resolution. In *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*.
- Tim Paek and Eric Horvitz. 2000. Conversation as action under uncertainty. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI)*.
- Tim Paek. 2006. Reinforcement Learning for spoken dialogue systems: Comparing strengths and weaknesses for practical deployment. In *Dialogue on Dialogues*. Interspeech/ICSLP Satellite Workshop.
- Olivier Pietquin and Thierry Dutoit. 2006. A probabilistic framework for dialog simulation and optimal strategy learning. *IEEE Transactions on Audio, Speech and Language Processing*, 14(2):589–599.

- Olivier Pietquin. 2006. Machine learning for spoken dialogue management: an experiment with speech-based database querying. In J. Euzenat & J. Domingue, editor, *Artificial Intelligence : Methodology, Systems & Applications*, volume 4183 of *Lecture Notes in Artificial Intelligence*, pages 172–180. Springer Verlag.
- Verena Rieser and Oliver Lemon. 2006a. Cluster-based user simulations for learning dialogue strategies. In *Proceedings of Interspeech/ICSLP 2006*.
- Verena Rieser and Oliver Lemon. 2006b. Using machine learning to explore human multimodal clarification strategies. In *Proceedings of Association for Computational Linguistics (ACL)*.
- Verena Rieser, Ivana Kruijff-Korbayová, and Oliver Lemon. 2005. A corpus collection and annotation framework for learning multimodal clarification strategies. In *Proceedings of 6th SIGdial Workshop*.
- Konrad Scheffler and Steve J. Young. 2002. Automatic learning of dialogue strategy using dialogue simulation and Reinforcement Learning. In *Proceedings of Human Language Technology (HTL)*.
- Daniel Shapiro and Pat Langley. 2002. Separating skills from preference: using learning to program by reward. In *Proceedings of 19th International Conference on Machine Learning*.
- Satinder Singh, Diane Litman, Michael Kearns, and Marilyn Walker. 2002. Optimizing dialogue management with Reinforcement Learning: Experiments with the NJFun system. *Journal of Artificial Intelligence Research (JAIR)*, Volume 16:105–133.
- Gabriel Skantze. 2007. Making grounding decisions: Data-driven estimation of dialogue costs and confidence thresholds. In *Proceedings of 8th SIGdial*.
- Tomek Strzalkowski, Sharon Small, Hilda Hardy, Paul Kantor, Wu Min, Sean Ryan, Nobuyuki Shimizu, Liu Ting, Nina Wacholder, and Boris Yamrom. 2008. Question answering as dialogue with data. In Tomek Strzalkowski and Sandrda Harabagiu, editors, *Advances in Open Domain Question Answering*, volume 32 of *Text, Speech and Language Technology*. Springer.
- Richard Sutton and Andrew Barto. 1998. *Reinforcement Learning*. MIT Press.
- Sebastian Varges, Fuliang Weng, and Heather Pon-Barry. 2006. Interactive question answering and constraint relaxation in spoken dialogue systems. In *Proceedings of 7th SIGdial Workshop*.
- Marilyn A. Walker, Candace A. Kamm, and Diane J. Litman. 2000. Towards developing general models of usability with PARADISE. *Natural Language Engineering*, 6(3-4):363–377.
- Marilyn A. Walker, Rebecca J. Passonneau, and Julie E. Boland. 2001. Quantitative and qualitative evaluation of DARPA Communicator spoken dialogue systems. In *Proceedings of Association for Computational Linguistics (ACL)*.
- Marilyn Walker. 2005. Can we talk? methods for evaluation and training of spoken dialogue system. *Language Resources and Evaluation*, 39(1):65–75.
- Xingtao Zhao. 2007. Integrating a QA System with Dialogue Management for the Music Domain. Master’s thesis, School of Informatics, University of Edinburgh.