Evaluation of Online Dialogue Policy Learning Techniques

Alexandros Papangelis[1,2], Vangelis Karkaletsis[1], Fillia Makedon[2]

[1]Institute of Informatics and Telecommunications, N.C.S.R. "Demokritos", Athens, Greece
[2]Heracleia Human-Centered Computing Lab, Dept. of CSE, University of Texas at Arlington, Arlington, TX, USA
alexandros.papangelis@mavs.uta.edu, vangelis@iit.demokritos.gr, makedon@uta.edu

Abstract

The number of applied Dialogue Systems is ever increasing in several service providing and other applications as a way to efficiently and inexpensively serve large numbers of customers. A DS that employs some form of adaptation to the environment and its users is called an Adaptive Dialogue System (ADS). A significant part of the research community has lately focused on ADS and many existing or novel techniques are being applied to this problem. One of the most promising techniques is Reinforcement Learning (RL) and especially online RL. This paper focuses on online RL techniques used to achieve adaptation in Dialogue Management and provides an evaluation of various such methods in an effort to aid the designers of ADS in deciding which method to use. To the best of our knowledge there is no other work to compare online RL techniques on the dialogue management problem.

Keywords: Adaptive Dialogue Systems, Evaluation, Reinforcement Learning

1. Introduction

A Dialogue System (DS) is a system that is able to make human-like conversation with its users in order to retrieve information and meet the users' goals. Such a system can be used for a variety of purposes, from helping users book flights and hotels to keeping them company, helping them learn a new subject or helping and motivating them in their rehabilitation process. An Adaptive Dialogue System (ADS) is a DS that is moreover able to adjust to its environment, to individual users and their current needs. ADS is a rapidly growing field and is receiving more and more attention from the research community. One critical aspect of it is how to achieve adaptation in the several modules of a DS. There has been much work in this direction, in adaptive (trainable) Natural Language Generation (NLG) (Rieser and Lemon, 2009), adaptive Referring Expression Generation (REG) (Janarthanam and Lemon, 2009) and adaptive Dialogue Management. The current trend for achieving adaptation is training with Reinforcement Learning (RL) techniques, since it is easy to cast the dialogue problem to RL formulation and take advantage of the algorithms that have been developed in this field.

ADS that have been proposed in the literature and apply RL techniques for adaptation include the work of Cuayáhuitl et al. (2010) who propose a system targeted for travel planning. The authors take a hierarchical RL approach to learn optimal dialogue policies for hybrid actions (i.e. complex actions, composed of basic actions). Young et al. (2010) propose an ADS for providing touristic information in a fictitious city. Their system can scale to real world problems and is able to handle uncertainty due to noise or misunderstandings. Konstantopoulos (2010) proposes a museum guide ADS, able to provide exhibit descriptions to the museum's visitors. The system can adapt to the users' personality by computing their mood or long term emotional state. It is also able to adapt its output according to how experienced the visitor is. Last, it maintains a system emotional state that is affected by the user's utterance and affects its output accordingly.

All the systems described so far, while having many benefits such as error handling or scalability, learn optimal dialogue policies offline. This means that when they actually interact with users they follow a static policy, which may be optimal but is not flexible to environmental changes (such as a user switching goals or a major environmental change that affects all users). To tackle this problem, Pietquin et al. (2011) propose an ADS which provides information about restaurants in a fictitious town and is able to learn optimal dialogue policies online, using a particle filtering approach. Gašić et al. (2011) also propose an ADS that builds upon the ADS proposed by Young et al. (2010) and is able to learn optimal dialogue policies online, using a Gaussian Process approach. Jurčíček et al. (2010) propose a novel online algorithm for DS parameter estimation, called Natural Actor - Belief Critic (NABC), which based on observed rewards, estimates the natural gradient of the expected cumulative reward and then performs gradient ascend following that gradient.

This paper focuses on the evaluation of RL techniques applied to online Dialogue Management, an under-explored research direction. Several existing online RL algorithms are compared, on the Olympus/RavenClaw platform (Bohus and Rudnicky, 2009), against each other on a simple generic slot filling scenario (up to 2^{10} states). Evaluation was done through extensive user simulations. We also attempt to explain the results and provide insight on why some algorithms perform better than others. Our contribution is that, for the first time to the best of our knowledge, we directly compare several online RL algorithms on a real ADS and on a high dimensional generic problem.

The rest of the paper is structured as follows: section 2 provides a brief overview of the basics of RL, section 3 presents our dialogue policy learning architecture and describes our problem formulation and experimental setup, section 4 presents our results and section 5 concludes this paper.

2. Background

To understand what reinforcement learning is about we first need to introduce Markov Decision Processes (MDP). An MDP is defined as a triplet $M = \{X, A, P\}$, where X is a non empty set of states, A is a non empty set of actions and P is a transition probability kernel that assigns probability measures over $X \times \mathbb{R}$ for each state-action pair $(x, a) \in X \times A$. Each transition from a state to another is associated with an immediate reward, as dictated by the reward function:

$$R(x, a) = \mathbb{E}[r(x, a)] \tag{1}$$

where $r(x, a)$ is the actual immediate reward, received after each transition. The cumulative discounted rewards, collected during transitions from state to state until a final state is reached, are called *return* and are defined as:

$$\rho = \sum_{t=0}^{\infty} \gamma^t R(x_t, a_t) \tag{2}$$

The parameter $\gamma \in [0, 1]$ is a discount factor that regulates the importance of future rewards. When $\gamma \to 1$ then future rewards become more important and when $\gamma \to 0$ they become less important and more weight is given in the current (immediate) reward. In most problems that are modelled as MDPs we are interested in maximizing the cumulative discounted rewards. We therefore need to select an action for each state that will ultimately maximize the return. An MDP policy $\pi : X \to A$ is a mapping that dictates which action to take from each state the system can be in. An optimal policy is a policy that maximizes the cumulative discounted expected rewards and finding this policy is the goal of RL (Szepesvári, 2010). In order to evaluate a given policy, we define a function, called the *value function* V:

$$V^{\pi}(x) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R_{t+1} | X_0 = x] \tag{3}$$

This function yields the expected cumulative rewards the system receives when it follows policy π and its initial state is x. The *return* of a given policy π is defined as:

$$\rho^{\pi} = \sum_{t=0}^{\infty} \gamma^t R_t(x_t, \pi(x_t)) \tag{4}$$

An optimal policy $\pi^{\star}$ satisfies $\rho^{\pi^{\star}}(x) = V^{\pi^{\star}}(x), \forall x \in X$. Similar to function V, we define another function, called the *action-value function* Q:

$$Q^{\pi}(x, \alpha) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R_{t+1} | X_0 = x, A_0 = \alpha] \tag{5}$$

This function yields the expected cumulative rewards the system receives when it follows policy π, its initial state is x and its initial action is a (Sutton and Barto, 1998).

RL is preferable in ADS because the generic slot filling dialogue problem can be easily cast in RL formulation. It models required information as slots that need to be filled by the user, for example a slot could be the date of travel and another could be a restaurant type. Once the system has enough slots filled (i.e. enough information) it attempts to answer the user's query. If we model slots as states and system actions as requests for one or more slot values we get the MDP representation of the slot filling problem.

RL algorithms can be divided in two categories: planning (or model-based) and learning (or model-free) algorithms. Their main difference is that learning algorithms use experience (samples or training examples) from interactions with the environment while planning algorithms also use experience simulated by a trained model of the environment (Sutton and Barto, 1998). According to their properties and the way they are trained, RL algorithms can be further categorized as online or offline learning algorithms. Note here that we only compared online learning and planning algorithms. In (Atkeson and Santamaria, 1997) the authors compare model based and model free algorithms but not the ones we evaluate here. Depending on whether they follow the policy being learned or another they can be characterized as on or off policy algorithms. RL algorithms can also be characterized as policy iteration or value iteration algorithms depending on how they update and evaluate their policy. For our evaluation to be complete we have selected algorithms from each category, as shown in Table 1. To evaluate model based approaches we used the Dyna architecture (Sutton and Barto, 1998) combined with all learning algorithms we evaluated.

Algorithm	Model	Policy	Iteration
SARSA(λ)	No	On	Value
Q-Learning	No	Off	Value
Q(λ)	No	Off	Value
Actor Critic-QV	No	On	Policy
DynaSARSA(λ)	Yes	On	Value
DynaQ	Yes	Off	Value
DynaQ(λ)	Yes	Off	Value
DynaActorCritic-QV	Yes	On	Policy

Table 1: Algorithm characteristics.

SARSA(λ)

This is one of the most widely used algorithms in ADS, mostly for dialogue policy learning. This algorithm employs temporal difference techniques to estimate $Q(s, a)$, which is represented as a matrix and thus learn the optimal policy. It also uses a matrix, called eligibility traces, to store past experience (states visited and actions taken) and aid the learning process. SARSA(λ) is a model-free algorithm and we call its model-based version, according to the Dyna framework, DynaSARSA(λ).

Q Learning

As the name implies, Q-Learning learns an estimate of $Q(s, a)$ as well. It was proposed by Watkins (1989) and the most significant difference it has with SARSA(λ) is that it uses the difference of $Q(s', a^*) - Q(s, a)$ in order to make the next update, where s' is the next state and $a^{\star}$ is the optimal action to take from that state according to a greedy policy. Also, that update is applied to a single entry of $Q(s, a)$, rather than the whole matrix, as in the case of SARSA(λ). Last, Q-Learning does not apply eligibility traces to aid learning. DynaQ-Learning is the Dyna (i.e. model-based) version of this algorithm.

Q(λ)

This algorithm is an extension of Q-Learning. It uses the same temporal difference to make updates but updates the whole matrix $Q(s, a)$ instead of a single entry and also uses eligibility traces to take past experience into account (Watkins, 1989; Peng and Williams, 1996). The model-based version of this algorithm is DynaQ(λ).

Actor Critic QV

This algorithm is an actor-critic algorithm, meaning it uses an actor to follow a policy and a critic to evaluate it. Note that these policies need not be the same. ActorCritic-QV makes actor or critic updates according to the temporal difference of $Q(s, a) - V(s)$ (Wiering and Van Hasselt, 2009). Its model-based version, according to Dyna, is the DynaActorCritic-QV algorithm.

The advantage of online methods is that at each dialogue turn we only need to take into account states that can be reached from the current state. We thus have to compute the maximum value of only the current state and not of every one. Another advantage is that online algorithms are applicable to dynamic environments as learning never stops and the system is able to adapt to environmental changes. Off-policy online learning has the advantage of considering actions that the behaviour policy does not select, which cannot be done in on-policy learning if the target policy is not stochastic. Moreover, even if the target policy is nondeterministic the algorithm will most likely produce bad estimates for the less often visited actions (Szepesvári, 2010).

3. Experimental Setup

Our system is based on CMU's Olympus/RavenClaw platform (Bohus and Rudnicky, 2009), which is an open source platform for developing spoken dialogue systems. To achieve adaptation in dialogue management we have developed a learning component for each algorithm, located at the backend server, where all data processing is performed. The dialogue manager calls the appropriate update function after each interaction with the real or simulated user. Figure 1 depicts the Dialogue Manager (DM) and the Backend Server of our system.

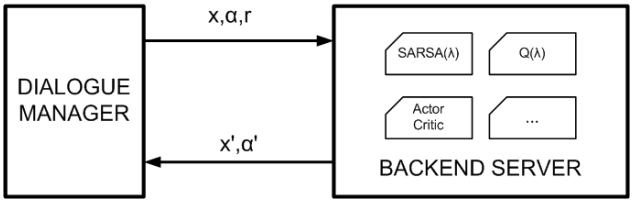

Figure 1: Dialogue Manager and Learning Algorithms.

The DM forwards the current dialogue state x, last action a and reward $R(x, a)$ to the appropriate learning component in the back-end server. The component then computes the new system action a' and the new dialogue state x' according to an online RL algorithm and forwards them to the DM. The DM will then forward the selected action a' to the appropriate component of Olympus (for example NLG) which will execute it.

To evaluate the selected methods we needed a model of ADS. A commonly used model is the so called *slot filling* model. This model assumes that the user's query can be formed as a paragraph in every day language. For example, in a museum guide ADS, a user looking for an exhibit with specific properties would form his / her request as: "I would like to see a Greek *amphora* that was found in *Athens*, dating around *50 BC*". The system may therefore have a template to represent generic user queries, such as: "I would like to see a Greek [_____] found in [_____], dating around [_____]". The system then would have to ask questions in order to fill the empty slots in the paragraph, for example to fill the first slot it could ask: "What type of exhibit are you interested in learning about?". Such questions that request slot values are abstractly represented as actions in the ADS (e.g. 'ask for slot value of ExhibitType'). This approach is called slot filling. An example museum guide ADS could use the following setup:

- **Slots:** Exhibit Type, Artist, Location Found, Time Period
  - **Exhibit Type Values:** <empty>, Amphora, Temple, Statue, Sword, Battlefield, Epic, Technology
  - **Artist Values:** <empty>, Iktinos, Kallikrates, Homer, Anonymous
  - **Location Found Values:** <empty>, Athens, Sparta, Crete, Delos
  - **Time Period Values:** <empty>, 1000BC, 900BC, ..., 0, 100AD
- **Actions:** Welcome, Request Exhibit Type, Request Artist, Request Location Found, Request Time Period, Show Results, Greet Goodbye

RL can be easily applied to this problem, if we model it as a MDP. The dialogue states can be represented as a vector containing the slots to be filled and the system actions as requests for slots (filled when the user provides the answer). The reward function also needs to be defined (along with transition probabilities etc) in a way that reflects what we want to optimize, for example that we most likely want to minimize dialogue length, maximize user satisfaction and so on. Formally the slot filling problem is defined as:

- $S = \{s_1, ..., s_N\}$, the slots to be filled
- $M_i = \{1, ..., |T_i|\}$, the values for each slot
- $A \in \{1, ..., |S|\}$, the available system actions
- $d =< s_1, ..., s_N >\in M$, the dialogue state
- $M = M_0 \times M_1 \times ... \times M_N$, the set of values for d
- $q \subset S$, the user's query

where T_i are the different values slot s_i can be filled with, S are the N slots to be filled and each slot s_i can take values from M_i. Dialogue state is defined as a vector $d \in M$, where each dimension corresponds to a slot and its value

corresponds to the slot's value. System actions A are defined as requests for slots to be filled, where action a_i requests the value of slot s_i. At each dialogue state d_i we define a set of available actions $\tilde{a}_i \subset A$. The user query q is defined as the slots that need to be filled so that the system will be able to accurately provide an answer. We assume action a_N always means *Give Answer* and, consequently, slot s_N's value shows if an answer has been provided to the user or not. The reward function is defined as:

$$\text{R(d,a)} = \begin{cases} \text{-1}, & \text{if } a \neq a_N \\ \text{-100}, & \text{if } a = a_N \text{ AND } \exists q_i | q_i = \emptyset \\ 0, & \text{if } a = a_N \text{ AND } \neg\exists q_i | q_i = \emptyset \end{cases} \quad (6)$$

Thus, the optimal reward for each problem is: $-|q|$ assuming $|q| < |S|$. This reward function penalizes long dialogues (-1 for each action taken) or inaccurate responses (-100 if the system provides results without all slots filled) and rewards accurate responses (when all slots have been filled). For example, if a system has 4 slots, 4 actions, binary values for each slot and a user query $q = \{1, 2, 3\}$, then a transition from state $d_1 =< 0, 0, 0, 0 >$ to state $d_2 =< 1, 0, 0, 0 >$ would incur a reward of -1. A transition to state $d_3 =< 0, 0, 0, 1 >$ would incur a reward of -100, since the last action is *Give Answer* and obviously the system has no slot filled yet. On the other hand, a transition from state $d_4 =< 1, 1, 1, 0 >$ to state $d_5 =< 1, 1, 1, 1 >$ would incur a reward of 0, since all slots of the query q have been filled and the system correctly attempts to provide an answer.

In order to compare algorithms to the best of each one's abilities we optimized their parameters in an exhaustive manner. We assessed how each algorithm performed on a given problem, varying its parameters by a small value at each run. Table 2 shows the parameter values we used, where α is the (actor) learning rate, β is the critic learning rate, γ is the discount factor, ϵ is the probability by which we select a random action in ϵ-greedy policies and I is the number of iterations the model is trained for, after each interaction with the user in the Dyna architecture. Note here that ϵ, α and β in general were decaying as the episodes progressed to reduce exploration and learning rates respectively. We made sure however that $\epsilon, \alpha, \beta > 0$ to make sure learning did not stop. Note here that ϵ was set to 0.01 for every algorithm.

Algorithm	α	β	γ	λ	I
SARSA(λ)	0.95	-	0.55	0.4	-
Q-Learning	0.8	-	0.4	-	-
Q(λ)	0.95	-	0.95	0.05	-
Actor Critic-QV	0.9	0.25	0.75	-	-
DynaSARSA(λ)	0.95	-	0.25	0.25	15
DynaQ	0.8	-	0.4	-	15
DynaQ(λ)	0.8	-	0.4	0.05	15
DynaActorCritic-QV	0.9	0.05	0.75	-	15

Table 2: Optimized parameter values.

4. Evaluation

Dialogue Systems' evaluation is an open problem and the current trend in ADS evaluation, when RL is used for training, is comparison of the total reward each system received. The reward function typically depends on common parameters, such as length of dialogue, achievement of goals etc. In our case $R(d, a)$ favours short dialogues and penalises attempts to answer without the required information.

Evaluation was done with user simulations, in a noise free environment. We opted for user simulations since they have many advantages, such as the fact that we can easily train large systems and see how they perform and the fact that they are cost effective and not time consuming. Of course the ultimate goal of a dialogue system is to interact with humans so a real user trial is imperative (although real user trials do have disadvantages). Online learning systems however can easily adapt to real users, even if initially trained using simulations, since learning never stops.

The user simulation model we used for our evaluation was a noise free model where the simulated user is assumed to always respond correctly to system requests. We opted for this type of simulation to maintain simplicity. The interested reader may easily extend our model to account for uncertainty due to noise, misunderstandings, changes in the environment or user goals etc.

We conducted three experiments to see how fast each algorithm learns, how well it scales and how it responds to environmental changes. Note here that we constrained the number of iterations per episode to $2^{|S|+1}$. This ensures enough opportunities for exploration and does not allow bad directions to be followed for too long. To evaluate learning speed we marked the episode on which each algorithm converged, averaged over 20 runs. An algorithm is considered to have converged if it yields the optimal reward for at least 20 consecutive episodes. The episode of convergence though is the first episode of that series, where optimal reward was achieved. We opted for as high a dimensional problem as implementation and evaluation restrictions would allow to be as close as possible to real world problems. Table 3 below, shows the results on a problem with 1024 dialogue states, 10 slots, 2 values per slot (filled/empty) and 10 actions.

Algorithm	Converge After Episode
SARSA(λ)	645
Q-Learning	856
Q(λ)	717
Actor Critic-QV	1135
DynaSARSA(λ)	564
DynaQ	851
DynaQ(λ)	**435**
DynaActorCritic-QV	961

Table 3: Algorithms' learning speed.

We can see here that DynaQ(λ) greatly outperforms the other algorithms with DynaSARSA(λ) being second best but with a considerable difference. While Q(λ), like SARSA(λ), disseminates new information to previously visited states, it converges slower mainly due to its instability that disappears as episodes progress. Moreover, Q-Learning and Actor Critic-QV perform much worse during the initial episodes. Actor Critic-QV exhibits instabilities as well even though it yields some optimal rewards quickly enough. Due to this it requires more than 1000 episodes to

converge and DynaActorCritic-QV requires close to 1000 but both are generally stable afterwards. Q-Learning performs worse than Q(λ) because it updates a single entry of its Q matrix after each interaction. It is important to note here, however, that model based algorithms need much more absolute time to converge, since at each iteration of each episode they also make I additional iterations while processing simulated data derived from their models. Moreover, the models themselves require additional memory space, compared to the model-free algorithms.

Algorithm	7x7	8x8	9x9	10x10
SARSA(λ)	-16.53	-19.94	-29.50	-40.08
Q-Learning	-29.16	-41.31	-57.50	-56.40
Q(λ)	**-15.48**	**-19.75**	-37.60	-41.34
AC-QV	-25.94	-34.48	-45.75	-52.79
DSARSA(λ)	-19.91	-31.29	-42.50	-57.75
DynaQ	-29.08	-41.07	-57.81	-56.69
DynaQ(λ)	-18.72	-23.95	**-24.76**	**-29.36**
DynaAC-QV	-56.81	-54.60	-53.22	-53.05
Optimal	-6	-7	-8	-9
Episodes	300	400	500	600

Table 4: Average reward for various problem sizes.

To evaluate the scalability of each algorithm we conducted experiments varying the problem dimension, where by dimension we mean the number of slots and actions N. We run each algorithm 20 times and measured the average rewards received after each experiment, for 7 slots and 7 actions (7x7) up to 10 slots and 10 actions (10x10). Table 4 shows the results of this experiment, where the *Optimal* row shows the optimal reward that an optimal policy would achieve in each problem dimension. The learning algorithms try to converge to such a policy, but while learning they follow suboptimal policies, leading to lower rewards. The average reward of an algorithm indicates how fast the algorithm converged to an optimal policy, as the more optimal rewards it receives in the allowed episodes, the closer the average will get to the *Optimal* value. Lower average reward values show either slow convergence or instability.

Algorithm	Converg. Ep.	Additional Ep.
SARSA(λ)	659	59
Q-Learning	971	371
Q(λ)	**632**	**32**
Actor Critic-QV	729	129
DynaSARSA(λ)	905	305
DynaQ	>1000	>400
DynaQ(λ)	675	75
DynaActorCritic-QV	797	197

Table 5: Additional episodes required to converge again after environmental changes at episode 600.

We can see here that in smaller problems Q(λ) performs best with SARSA(λ) and DynaQ(λ) following close, and on higher problem sizes DynaQ(λ) performs best with SARSA(λ) following in performance. In the 7x7 and 8x8 problems the difference in performance between SARSA(λ), Q(λ) and DynaQ(λ) is not statistically significant, while in higher problem sizes those differences are statistically significant. The rest of the algorithms perform worse, having statistically significant differences between them, with Dyna algorithms (except for DynaQ(λ)) usually being among the worst. DynaActorCritic-QV seems to perform similarly on all problem sizes mainly because it needs more episodes than those allowed to converge. This also explains the fact that its performance slightly increases on higher problem sizes, where more episodes are allowed.

To evaluate how each algorithm reacts to environmental changes, something typical in ADS, we changed the user's query and consequently the reward function, after the algorithms had converged, to see how fast they could converge to the new setting. We allowed 600 episodes for each algorithm to converge, then changed the query and allowed another 400 episodes to converge to the new problem, thus allowing 1000 episodes in total. The problem had 512 dialogue states, 9 slots and 9 actions. Table 5 shows the results where Q(λ) appears to perform better than any other algorithm, contrary to Q-Learning and DynaQ-Learning who do not react well to change. SARSA(λ) and DynaQ(λ) follow in terms of performance with a considerable difference from the rest algorithms. Model based algorithms, except for DynaQ(λ) perform worse than their respective model free versions, probably due to the fact that their models were trained on pre-change interactions with the environment and continue to be trained this way until the majority of table entries are updated by post-change interactions. Note that the higher the number of iterations the model is trained for (I), the worse model based algorithms reacted to change.

5. Conclusion and Future Work

Atkeson and Santamaria (1997) conclude that model based algorithms can handle changes in the pendulum problem better than model free ones. In the slot filling dialogue problem, however, this might not be the case as not all model-based algorithms performed well on the three experiments, except for DynaQ(λ) that performed very well on all of them. Q(λ) and SARSA(λ) performed slightly worse but, including ActorCritic-QV they are much simpler and cheaper to implement and much faster to run, with Actor Critic-QV being the fastest of our evaluation. Note also that Q(λ) performs best in the presence of changes, but cannot scale as well as DynaQ(λ). DynaQ(λ) performs best when the query does not change and well enough when it does, but with an additional cost in time and memory space, due to the model representation and the additional iterations the algorithm must perform.

As a conclusion, learning algorithms perform generally better than model-based ones when Dyna architecture is adopted, again excluding DynaQ(λ). In ADS, changes in the environment are frequent and one should expect this when designing a learning algorithm for ADS dialogue management and should avoid algorithms that do not adapt well to environmental changes. ActorCritic-QV is very fast and reacts well to change so it should be chosen if computational speed is important. Another aspect of environmental changes is changes in the user's goal, in the middle of the interaction. This means the algorithm would need to converge to the optimal solution in the same episode the change occurs. None of the algorithms we selected is able

to do this hinting that more focused techniques are required rather than simple RL algorithms.

Apart from their performance, a major drawback of the evaluated algorithms is the size of the table needed to store state-action values in high dimensional or continuous problems. Function approximation can tackle this problem but one must take care when selecting features and their distribution (Wu and Meleis, 2008; Allen and Fritzsche, 2011). In the future we plan to explore several function approximation techniques and more state of the art algorithms such as Kallman Temporal Differences (Geist and Pietquin, 2010), Gaussian Process - SARSA (Gašić et al., 2010) and Natural Actor - Belief Critic (Jurčíček et al., 2011), on a model that will be able to account for uncertainty and misunderstandings. We also plan to explore other model based architectures such as Rmax and conduct experiments with real users.

6. References

Allen, M., Fritzsche, P., 2011, *Reinforcement learning with adaptive Kanerva encoding for Xpilot game AI*, In IEEE Annual Congress on Evolutionary Computation, pp. 1521–1528.

Atkeson, C., Santamaria, J., 1997, *A comparison of direct and model-based reinforcement learning*, Robotics and Automation, volume 4, pp. 3557-3564.

Bohus, D., Rudnicky, A.I., 2009, *The ravenclaw dialog management framework: Architecture and systems*, Computer Speech & Language, 23(3), pp.332-361.

Cuayáhuitl, H., Renals, S., Lemon, O., Shimodaira, H., 2010, *Evaluation of a hierarchical reinforcement learning spoken dialogue system*, Computer Speech & Language, Academic Press Ltd., vol 24:2, pp. 395–429.

Gašić, M., Jurčíček, F., Keizer, S., Mairesse, F., Thomson, B., Yu, K., Young, S., 2010, *Gaussian processes for fast policy optimisation of pomdp-based dialogue managers*, In Proceedings of the SIGDIAL 2010 Conference, pp. 201-204.

Geist, M., Pietquin, O., 2010, *Kalman temporal differences*, Journal of Artificial Intelligence Research, vol 39, pp. 483-532.

Janarthanam, S., Lemon, O., 2009, *A Two-Tier User Simulation Model for Reinforcement Learning of Adaptive Referring Expression Generation Policies*, SIGDIAL Conference, pp. 120–123.

Jurčíček, F., Thomson, B., Young, S., 2011, *Natural actor and belief critic: Reinforcement algorithm for learning parameters of dialogue systems modelled as pomdps*, ACM Transactions on Speech and Language Processing (TSLP), vol 7(3):6, pp. 1–26.

Konstantopoulos S., 2010, *An Embodied Dialogue System with Personality and Emotions*, Proceedings of the 2010 Workshop on Companionable Dialogue Systems, ACL 2010, p.p 31-36.

Peng, J., Williams, R., 1996, *Incremental multi-step Q-Learning*, Machine Learning, vol. 22:1, pp. 283–290.

Rieser, V., Lemon, O., 2009, *Natural Language Generation as Planning Under Uncertainty for Spoken Dialogue Systems*, Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009), pp. 683–691.

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

Szepesvári, C., 2010, *Algorithms for Reinforcement Learning*, Morgan & Claypool Publishers, Synthesis Lectures on Artificial Intelligence and Machine Learning, vol 4:1, pp. 1–103.

Watkins, C.J.C.H., 1989, *Learning from delayed rewards*, PhD Thesis, University of Cambridge, England.

Wiering, M.A., Van Hasselt, H., 2009, *The QV family compared to other reinforcement learning algorithms*, IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning, pp. 101–108.

Wu, C., Meleis, W.M., 1998, *Adaptive kanerva-based function approximation for multi-agent systems*, In Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems, vol. 3, pp. 1361-1364.

Young, S., Gašić, M., Keizer, S., Mairesse, F., Schatzmann, J., Thomson, B., Yu, K., 2010, *The Hidden Information State model: A practical framework for POMDP-based spoken dialogue management*, Computer Speech & Language, vol. 24:2, pp. 150–174.