Sample-Efficient Batch Reinforcement Learning for Dialogue Management Optimization

Olivier PIETQUIN[†,‡]
Matthieu GEIST[†]
Senthilkumar CHANDRAMOHAN[†]
Hervé FREZZA-BUET[†,‡]
[†]*Supélec* / [‡]*UMI 2958 (GeorgiaTech - CNRS)*

Spoken Dialogue Systems (SDS) are systems which have the ability to interact with human beings using natural language as the medium of interaction. A dialogue policy plays a crucial role in determining the functioning of the dialogue management module. Hand-crafting the dialogue policy is not always an option considering the complexity of the dialogue task and the stochastic behavior of users. In recent years approaches based on Reinforcement Learning (RL) for policy optimization in dialogue management have been proved to be an e cient approach for dialogue policy optimization. Yet most of the conventional RL algorithms are data intensive and demand techniques such as user simulation. Doing so, additional modeling errors are likely to occur. This paper explores the possibility of using a set of approximate dynamic programming algorithms for policy optimization in SDS. Moreover, these algorithms are combined to a method for learning a sparse representation of the value function. Experimental results show that these algorithms when applied to dialogue management optimization are particularly *sample efficient* since they learn from few hundreds of dialogue examples. These algorithms learn in an *off-policy* manner meaning that they can learn optimal policies with dialogue examples generated with a quite simple strategy. Thus they can learn good dialogue policies directly *from data*, avoiding user modeling errors.

Categories and Subject Descriptors: Information Systems [**Information Interfaces and Presentation**]: User Interfaces | *Natural Language*

General Terms: Theory, Experimentation

Additional Key Words and Phrases: Spoken Dialogue Systems, Reinforcement Learning

1. INTRODUCTION

Due to the progresses made in the fields of speech and natural language processing during the last decades, speech-based interfaces are now widespread. Spoken Dialogue Systems (SDS) are such interfaces allowing people to interact with a computer or any other machine (like a robot) using speech, which is the most natural communication means for human beings. They are usually *task oriented* meaning that they aim at helping a user to accomplish a specific task such as accessing

Authors' address: O. Pietquin, M. Geist, S. Chandramohan, H. Frezza-Buet, Supelec, Metz Campus, IMS Research group & UMI 2958 (GeorgiaTech - CNRS) 57070 Metz, France

tourism information [Lemon et al. 2006], flight booking [Williams and Young 2007], accessing a database [Pietquin 2006b], controlling an in-car entertainment system [Rieser and Lemon 2008], *etc.* Yet, building a SDS is much more complex than putting together automatic speech recognition and speech synthesis systems. Of course, other modules such as natural language understanding and generation units are mandatory but also, a Dialogue Management (DM) subsystem. This subsystem rules the sequencing of the interaction by defining what is said by the SDS at each dialogue turn, which information is asked for or transmitted to the user. The design of a DM system is of a crucial importance since that naturalness of the interaction depends on it. It often happens that users dislike interacting with a SDS because the interaction seems unnatural due to many confirmation subdialogues occurring too systematically for instance.

The design of a dialogue management strategy is a tailoring task often let to human experts. Handcrafting such a strategy is a time consuming job, which requires highly qualified manpower which makes it very expensive. The designers have to list all the possible dialogue situations so as to associate the appropriate system behavior. Moreover, most of the time the design made for a particular task cannot be reused for another task. This slows down the diffusion of speech-based interfaces into the general public. Although high level scripting languages exist [W3C 2008] for the rapid design of new interfaces, their use is still limited when the dialog task becomes complex.

For these reasons, there has been an important focus on machine learning methods for automatic learning of dialogue management strategies since the late 90's [Levin and Pieraccini 1998; Singh et al. 1999; Levin et al. 2000; Scheffler and Young 2001; Pietquin 2004; Williams and Young 2007; Lemon and Pietquin 2007]. Most of these works are based on a similar two-step model :

(1) Casting the dialogue management problem into a Markov Decision Process (MDP) [Levin and Pieraccini 1998; Singh et al. 1999; Pietquin 2004] or a Partially Observable MDP (POMDP) [Williams and Young 2007] (see Section 3);
(2) Training a reinforcement learning (RL) algorithm [Sutton and Barto 1998] on artificially generated dialogues obtained by simulating the user [Eckert et al. 1997; Scheffler and Young 2001; Pietquin 2006a] and the errors introduced by the speech and language processing modules [Pietquin and Renals 2002; Schatzmann et al. 2007].

RL is a machine learning paradigm aiming at learning an optimal control policy by defining rewards associated to each control decision taken in a given situation. RL algorithms tries to find the control policy that maximizes a cumulative function of rewards over the long run for every situation (this is measured by a so-called *value function* which will be defined formally in Section 3). The reason for using dialogue simulation is that standard RL algorithms are data intensive and typically require several hundreds of thousands of dialogues to converge even for very simple dialogue tasks. Indeed, in [Levin and Pieraccini 1998], the authors mention that 710,000 simulated dialogues were required to learn an optimal policy in a quite simple task. Yet, collecting and annotating data in the spoken dialogue domain is quite hard since a first prototype is required or Wizard-of-Oz techniques have to be employed [Rieser and Lemon 2008]. Dialogue simulation implies introducing ad-

ditional modeling assumptions in the strategy learning process which undoubtedly results in suboptimal results, even if the effect of simulation is not formally known [Schatzmann et al. 2005].

The slow convergence of standard RL algorithms (such as Q-leaning [Watkins 1989] or SARSA [Sutton and Barto 1998]) is due to a very poor sample efficiency, that is the inability of extracting the maximum of information from each observed dialogue turn in our case. This results in a increased number of dialogues required for learning. Another problem is also their inability to handle large state spaces. It is often the case that dialogue management requires handling continuous state spaces since continuous confidence levels of speech recognition and understanding are introduced in the state representation. Most of nowadays systems use a quantization of the state space or a rough linear approximation. Since the late 90's, when RL was first used for dialogue management optimization, the field of RL has known a tremendous amount of progress with a particular focus on sample-efficiency, off-policy learning and scaling up algorithms to real-world problems. Although generalization solutions for solving MDP in large state spaces were proposed very early, such as Approximate Dynamic Programming (ADP) algorithms [Bellman and Dreyfus 1959; Samuel 1959], sample efficient and off-policy algorithms with acceptable computational complexity were only proposed during the last 15 years [Gordon 1995; Bradtke and Barto 1996; Lagoudakis and Parr 2003; Ernst et al. 2005]. Let us mention that learning in an off-policy manner roughly means learning the optimal policy while observing another policy behaving, potentially without being allowed to modify this behavioral policy. It is typically required when learning from a fixed set of data since there is no way to modify the data. An alternative to dialogue simulation would therefore be to make use of such algorithms.

To the knowledge of the authors, very limited work has been done in the field of *batch* approximate dynamic programming applied to SDS management which does not benefit of the large machine learning literature devoted to ADP. Of all the recent works in the field of machine learning for SDS, one can list two closely related approaches, (i) generalization of state-action value function in large state space dialogue management problems using hybrid learning [Henderson et al. 2008] and (ii) using Least-Squares Policy Iteration (LSPI) [Lagoudakis and Parr 2003] for dialogue management [Li et al. 2009]. The former approach (i) suffers from the fact that the used algorithm (SARSA(λ)) was designed for online and on-policy learning but it is used in a batch setting in the reported research with some combination of supervised learning to render it off-policy. SARSA(λ) is known to converge very slowly. Moreover, the choice of the features used for the linear approximation of the value function is particularly simple since features are the state variables themselves. The approximated function can therefore not be more complex than an hyper-plane in the state variables space. In the latter approach (ii), a batch algorithm (LSPI) is combined with a pruning method to select the most meaningful features. This approach shares the same drawback as the former one concerning the choice of the features. In addition, this choice requires prior knowledge on the task and about the features the SDS can provide. Other sample-efficient but *online* and *on-policy* learning algorithms using Gaussian Processes [Gasic et al. 2010] or

Natural Actor Critic [Jurcicek et al. 2010] have also been proposed. These works report the use of *on policy* algorithms which require to permanently changing the policy and test it to learn. These changes to the policy made during learning prevent using fixed sets of data. Moreover they are visible to the user which may cause problems in real applications at the early stage of learning where the changes in the policy can lead to very bad behaviors of the dialogue manager. Thus, user simulation is still required although learning is clearly more efficient in terms of samples used. Both these works are based on a POMDP modeling of an SDS but the POMDP is considered as a continuous MDP, so standard RL algorithms can be used.

In this paper, several contributions are proposed. First, the algorithms are extensively described in a progressive manner. Different *off-policy* Approximate Dynamic Programming (*batch*) algorithms, namely Fitted-Q and LSPI are applied to the problem of learning an optimal policy for a specific dialog management task. Although LSPI has already been applied to SDS optimization [Li et al. 2009], it is not the case of Fitted-Q and comparison is worth doing. Moreover, a generic kernel-based function approximation is used here which was not the case in previously published works. Therefore, no prior knowledge about the task is required which makes the method task-independant. These algorithms, described in Section 4, are applied together with a generic function approximation scheme in a first experiment [Chandramohan et al. 2010a]. Another contribution is that an automatic feature selection technique is added to these algorithms to build a more compact function representation so as to minimize the number of parameters to learn while keeping an efficient strategy [Chandramohan et al. 2010b]. The combination of Fitted-Q and of this method is new in the machine learning literature and the use of both LSPI and Fitted-Q in combination with this method for SDS is also novel. These algorithms have been tested on a slot filling problem with 3 slots and are compared to a state-of-the-art algorithm, namely Q-learning with function approximation. The dialogue system has 3 continuous states (no quantization). Results, presented in Section 5, show that an optimal strategy can be learnt from a *fixed* and *small* set of data. Moreover, the data were collected with a very simple initial strategy showing that no expert knowledge is required to collect data. From all this, it is concluded that user simulation has to be rethought so as to produce more interesting behaviors than just generating statistically consistent data since sample-efficient algorithms can catch all the information in smaller sets of data..

2. DIALOGUE MANAGEMENT AS A SEQUENTIAL DECISION MAKING PROCESS

A spoken dialogue can be seen as a turn-taking process where two participants exchange pieces of information one after the other. At each turn, each participant has to decide what s/he will say, which information to ask or to provide to the other. This decision is generally based on the history of the dialogue and it influences the following turns. When one of the two participants is a machine, it has a limited number of possible decisions it can take at each turn and it has to select among these. More formally, each participant has to select one or several *dialogue acts* to perform so as to modify the situation (this is why it is named an *act*). There can be different dialogue act types such as *questions*, *assertions*, *explicit confirmations*,

implicit confirmations, *dialogue closure*, *etc.*

The dialogue management problem can therefore be stated as a sequential decision making problem where the dialogue manager has to select a sequence of dialogue acts (called *system acts*) so as to accomplish the task it as been designed for in the best possible manner. Each dialogue act can influence the future dialogue acts produced by the user (called *user acts*) and therefore the results of each decision can be delayed in time.

3. MARKOV DECISION PROCESS AND DIALOG MANAGEMENT

Reinforcement learning (RL) [Sutton and Barto 1998] is a general paradigm for representing sequential decision making problems in which an agent learns to control a dynamic system (also called *environment*) through actual interactions with it. In this paradigm, the system to be controlled is considered as made up of *states* and can be controlled by the application of *actions*. At each time step i, the system to control is in a state or configuration s_i. According to its own control *policy* and to the current state, the controlling agent chooses an action a_i to be applied to the system. This causes a (possibly stochastic) transition to a new state s_{i+1}. Moreover, according to this transition an "*oracle*" (often part of the environment) provides a (numeric) reward to the agent, this reward being a local hint about the quality of the control (about the decision). The goal of the agent is to learn a control policy which maximizes not the immediate reward but some cumulative function of immediate rewards over the long run. Delayed rewards are therefore taken into account into the optimization process.

This way, the oracle does not specify (through the reward) how the system should be controlled (this is the underlying idea of the supervised learning paragon), but rather what is the objective of this control ("what to do" and not "how to do"). For example, in the dialogue management problem, the oracle should provide a reward to the agent if the user is satisfied after having interacted with the system, a punishment otherwise. However, it should ideally not provide rewards for intermediate decisions, it is up to the agent to find how to satisfy the user given the (sparse) obtained rewards.

A common formal framework for RL is the Markov Decision Processes, which are presented formally in the following section. Section 3.2 explains how the dialog management problem can be cast into such a framework, which is the basis of all algorithms to be presented in section 4 and experimented in section 5.

3.1 Markov Decision Processes

Markov Decision Processes (MDP) [Puterman 1994] are probably one of the most famous frameworks for modeling sequential decision making problems in machine learning. Formally, an MDP is a tuple $\{S, A, P, R, \gamma\}$ where S is the (finite) state space defining the set of states that can be occupied by the system, A the (finite) action space defining the set of actions the learning agent can perform, $P : S \times A \rightarrow \mathcal{P}(S)$ a set of Markovian transition probabilities defining the one-step dynamics of the system, $R : S \times A \times S \rightarrow \mathbb{R}$ the reward function defining the immediate reward distribution and γ a discount factor that will serve to weight long-term rewards as explained hereafter.

At each time step, the system is in a state $s_i \in S$, the agent chooses an action

$a_i \in A$, the system transits to s_{i+1} according to $p(.|s_i, a_i)$ and the agent receives the reward $r_i = R(s_i, a_i, s_{i+1})$. The Markovian property is very important, it means that the transition of the system depends on the current state-action couple, but not on the path followed to reach it. It can appear to be a strong assumption. However, one can always theoretically make a system Markovian by taking into account all the history of interactions into the state representation. This is actually done for the dialogue management problem (in a smart way, this history being summarized in a compact representation).

The MDP being defined, one has still to define how the system is controlled (the *policy*) and how to quantify the quality of this control (the *value function*). A policy is simply a mapping from states to actions: $\pi : S \rightarrow A$. Its quality is quantified by the value function $V^{\pi} : S \rightarrow \mathbb{R}$, which associates to each state the expected cumulative discounted reward for starting in this state and then following the policy π:

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

The expectation term comes from the fact that trajectories are stochastic (due to the randomness of transitions). The optimal policy π^* is therefore the one for which the value is maximum for each state:

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

Computing the value function V^{π} from its definition (1) is not practical. However, thanks to the Markovian property, it can be shown that the value function satisfies the Bellman evaluation equation:

$$V^{\pi}(s) = E_{s'|s}[R(s, \pi(s), s') + \gamma V^{\pi}(s')] \tag{3}$$

$$= \sum_{s' \in S} p(s'|s, \pi(s))(R(s, \pi(s), s') + \gamma V^{\pi}(s')) \tag{4}$$

Let introduce the associated Bellman evaluation operator T^{π} defined as:

$$T^{\pi} : V \in \mathbb{R}^S \rightarrow T^{\pi} V \in \mathbb{R}^S : T^{\pi} V(s) = E_{s'|s}[R(s, \pi(s), s') + \gamma V(s')] \tag{5}$$

This operator is a contraction [Puterman 1994] from which the value function is the unique fixed-point:

$$V^{\pi} = T^{\pi} V^{\pi} \tag{6}$$

This operator view is very important for algorithms presented in section 4. There is another Bellman equation providing directly the optimal value function V^*, the Bellman optimality equation:

$$V^*(s) = \max_{a \in A} E_{s'|s;a}[R(s, a, s') + \gamma V^*(s')] \tag{7}$$

$$= \max_{a \in A} \sum_{s' \in S} p(s'|s, a)(R(s, a, s') + \gamma V^*(s')) \tag{8}$$

As before, the optimal value function can be defined as the unique fixed-point of a similarly defined Bellman optimality operator T^*, which can be also shown to be a contraction [Puterman 1994].

Notice that knowing the value function V^* is not sufficient to compute an optimal policy. Indeed, the transition probabilities and the reward function have to be known to obtain an optimal policy from V^* (that is a *greedy* policy according to V^*). The state-action value function $Q \ : S \times A \rightarrow \mathbb{R}$ is thus also defined. It associates to each state-action couple the expected cumulative discounted reward for starting in this state, taking this action and then following the policy π:

$$Q \ (s,a) = E[\sum_{i=0}^{\infty} \gamma^i r_i | s_0 = s, a_0 = a, \pi] \tag{9}$$

Compared to the value function, the Q-function provides an additional degree of freedom concerning the choice of the first action to be taken, which will be useful later. Actually, an important concept in RL is the greedy policy, which associates to a state the action which maximizes the expected cumulative discounted reward according to a currently estimated value function. Considering the value function, computing a greedy policy requires knowing the model, that is transition probabilities and the reward function. In the less constrained context, this model is not known neither learned. Thanks to this additional degree of freedom, a greedy policy π_g can be computed from the Q-function without knowing the model: $\pi_g(s) = \text{argmax}_a \, Q \ (s,a)$

The state-action value functions (resp. of a given policy π and the optimal one) also satisfy a Bellman (resp. evaluation and optimality) equation:

$$Q \ (s,a) = E_{s'|s;a}[R(s,a,s') + \gamma Q \ (s', \pi(s'))] \tag{10}$$

$$Q^*(s,a) = E_{s'|s;a}[R(s,a,s') + \gamma \max_{b \in A} Q^*(s',b)] \tag{11}$$

As for the value function, Q and Q^* are the fixed-points of associated Bellman operators T and T^* (with a slight abuse of notation).

Before showing how the optimal policy can be practically computed or estimated, depending on the context, the next section shows how the dialog management problem can be cast into an MDP.

3.2 Dialog Management as an MDP

The casting of the spoken dialogue management problem into the MDP framework (MDP-SDS) comes from the equivalence of this problem to a sequential decision making problem as explained in Section 2. Yet, the dialogue management process has to be defined in terms of states, actions and rewards so as to fit the MDP paradigm and benefit from solving algorithms.

The dialogue state is usually represented efficiently by the Information State paradigm [Larsson and Traum 2000]. Using this representation, the dialogue state contains a compact representation of the history of the dialogue in terms of system and user acts. It summarizes the information exchanged between the user and the system until the considered state is reached. Of course, system actions consist in selecting system dialogue acts. A dialogue management strategy is thus a mapping between dialogue states and dialogue acts.

The optimal strategy is the one that maximizes some cumulative function of rewards collected all along the interaction. A common choice for the immediate reward is the contribution of each action to the user's satisfaction [Singh et al.

1999]. This subjective reward is usually approximated by a linear combination of objective measures (dialogue duration, number of ASR errors, task completion, *etc.*) [Walker et al. 1997].

4. SOLVING MDP

The adopted formal framework for sequential decision making problems has been presented so far, but not how an optimal policy can be computed or estimated. This is the aim of this section.

First, it is assumed that the MDP model (transition probabilities and reward function) is known, and that the state space is small enough for a tabular representation (in other words, the value function can be efficiently represented by a vector). In this case, the optimal policy can be exactly computed by Dynamic Programming approaches, namely *value iteration* and *policy iteration* [Bellman 1957]. These algorithms are not used to solve the dialog management problem, mainly because underlying assumptions are too strong. However, they are the basis of most of reinforcement learning algorithms, so they are briefly presented in section 4.1.

In section 4.2, assumptions are relaxed. Notably, the state space is too large for a tabular representation and the model is no longer known. In Section 4.3, algorithms working under these relaxed assumptions are presented.

4.1 Dynamic Programming

Here, it is assumed that the model is known and that the state space is small enough for an exact tabular representation of the value function. Two approaches for computing the optimal policy are presented, namely policy iteration and value iteration.

4.1.1 *Policy Iteration.* Policy iteration is an iterative scheme which converges to the optimal policy by alternating policy evaluation (that is computing the value function of the current policy) and policy improvement (the new policy is defined as being greedy respectively to the last estimated value function).

More formally, the algorithm is initialized with a policy π_1. At iteration k, the policy π_k is evaluated thanks to the Bellman evaluation equation (5). Actually, this defines a set of $|S|$ linear equations with $|S|$ variables. This system can be easily solved, as $|S|$ is assumed to be small here. The value function V^{π_k} being computed, the policy is improved, that is π_{k+1} is greedy respectively to V^{π_k}:

$$\pi_{k+1} = \pi_{\text{greedy}}(V^{\pi_k}) : s \rightarrow \underset{a \in A}{\text{argmax}}\, E_{s'|s,a}[R(s,a,s') + \gamma V^{\pi_k}(s')] \qquad (12)$$

It can be shown that under this scheme, $V^{\pi_{k+1}} \geq V^{\pi_k}$ [Puterman 1994]. If the equality holds, the optimal policy is reached (the Bellman optimality equation being satisfied in this case), which happens in a finite number of iterations (as the number of different policies is finite).

Notice that computing the greedy policy requires the model. This is not a problem here, as this model is actually known. However, this assumption is released practically, therefore it is more practical to work directly with the Q-function, as announced in section 3.1. The policy iteration algorithm can be straightforwardly extended to the state-action value function, and in this case the greedy policy is

defined as :

$$\pi_{k+1} = \pi_{\text{greedy}}(Q^{k}) : s \rightarrow \underset{a \in A}{\text{argmax}}\, Q^{k}(s,a) \tag{13}$$

Therefore, in the next sections, the focus will be on state-action value functions. Before this, the value iteration algorithm is presented.

4.1.2 *Value Iteration.* The value iteration algorithm aims at directly computing the optimal value function V^*, using the Bellman optimality equation. If this function is known, the optimal policy is simply greedy respectively to it:

$$\pi^* = \pi_{\text{greedy}}(V^*) : s \rightarrow \underset{a \in A}{\text{argmax}}\, E_{s'|s,a}[R(s,a,s') + \gamma V^*(s')] \tag{14}$$

However, the Bellman optimality equation is not linear, and cannot be so easily solved. Nevertheless, recall that V^* is the unique fixed-point of the contraction T^*. Thanks to the Banach theorem, for any initial value function V_1, the iterative scheme $V_{k+1} = T^* V_k$ converges to the optimal value function V^* [Puterman 1994]. Therefore, value iteration iterates as follows:

$$\forall s \in S, \quad V_{k+1}(s) = \max_{a \in A} E_{s'|s,a}[R(s,a,s') + \gamma V_k(s')] \tag{15}$$

Notice that practically this convergence is only asymptotical. Therefore, a stopping criterion has to be set.

4.2 Relaxing Assumptions

Dynamic programming relies on strong assumptions, notably the fact that the model is known and that a tabular representation is possible.

First the model assumption is relaxed. Recall that the notion of greedy action is important in RL, and that the state-action value function allows computing it without knowing the model. Therefore, the value function is no longer considered in this paper. To get rid of the model, one has to rely on sampled transitions $(s_j, a_j, r_j, s_{j+1}, a_{j+1})$ to learn an optimal policy. Bellman operators can no longer be used, therefore sampled Bellman operators $\hat{T}$ and $\hat{T}^*$ are defined as follows, for a given transition (s,a,r,s'):

$$\hat{T}\ Q(s,a) = r + \gamma Q(s', \pi(s')) \tag{16}$$

$$\hat{T}^* Q(s,a) = r + \gamma \max_{b \in A} Q(s', b) \tag{17}$$

in these equations $\hat{T}\ Q(s,a)$ and $\hat{T}^* Q(s,a)$ are random variables and their expectations on all possible trajectories provide the true operators (e.g. $E_{s'|s,a}[\hat{T}\ Q(s,a)] = T\ Q(s,a)$). These sampled operators will be useful for all presented algorithms.

Second, the tabular representation assumption is relaxed. If the state space is too large, some approximate representation has to be adopted. In this paper, a general parametric representation $\hat{Q}$ is considered, with θ being the set of parameters describing the state-action value function. More precisely, kernel-based linear parameterizations are considered here [Scholkopf and Smola 2001], that is

the Q-function is approximated as a weighted sum of kernel basis functions:

$$\hat{Q}\ (s,a) = \sum_{i=1}^{p} \alpha_i K((s,a),(s_i,a_i)) = \theta^T \phi(s,a) \tag{18}$$

$$\text{with } \theta^T = \begin{pmatrix} \alpha_1 & \dots & \alpha_p \end{pmatrix} \tag{19}$$

$$\text{and } \phi(s,a)^T = \begin{pmatrix} K((s,a),(s_1,a_1)) & \dots & K((s,a),(s_p,a_p)) \end{pmatrix} \tag{20}$$

Moreover, in this contribution Gaussian kernels are considered (however, this work extends easily to any kernel function):

$$K((s,a),(s_i,a_i)) = \delta_{a=a_i} \exp(-\frac{\|s-s_i\|^2}{2\sigma^2}) \tag{21}$$

with $\delta_{a=a_i}$ being the Kronecker symbol. This linear parameterization defines an hypothesis space $\mathcal{H} = \{\hat{Q}\ \in \mathbb{R}^{S\times A} | \theta \in \mathbb{R}^p\}$ in which a good approximation of a state-action value function of interest will be searched for. Remember that previous works on generalization for SDS optimization did consider only hyperplanes in the state space. Here, more complex structures can be obtained for the value function according to the selected basis functions.

Last but not least, the batch learning aspect can be relaxed too, yet this is not in the scope of this paper. In Section 4.3, approximate dynamic programming algorithms are presented. Basically, they extend policy iteration and value iteration when the two first hypotheses (knowing the model and exact representation) are relaxed. These algorithms are batch: they approximate the optimal policy from a given set of transitions (not necessarily generated using the optimal policy, this aspect is called *off-policy* as mentioned in the introduction).

4.3 Approximate Dynamic Programming

In this section, approximate dynamic programming algorithms are introduced, namely fitted-Q [Gordon 1995; Chandramohan et al. 2010a], which is approximate value iteration, and least-squares policy iteration (LSPI) [Lagoudakis and Parr 2003; Li et al. 2009], which is approximate policy iteration. For these algorithms, the structure of the value function has to be set beforehand. Sparse variations of LSPI [Xu et al. 2007] and of fitted-Q [Chandramohan et al. 2010b] based on a kernel-based sparsification procedure [Engel et al. 2004] are also presented. Through this section, it is assumed that a set of N transitions $(s_i, a_i, r_i, s_{i+1})_{1\leq i\leq N}$ is available (batch learning), generated from any sufficiently explorative policy.

4.3.1 *Least-Squares Policy Iteration.* LSPI [Lagoudakis and Parr 2003; Li et al. 2009] is an approximate policy iteration algorithm. Its principle is therefore to alternate policy evaluation and improvement. This last step is done as before, by taking the greedy policy respectively to the last estimated Q-function:

$$\pi_{k+1} = \pi_{\text{greedy}}(\hat{Q}_k) \tag{22}$$

The policy is evaluated thanks to the Least-Squares Temporal Differences (LSTD) algorithm [Bradtke and Barto 1996] which applies to this relaxed setting (model unknown and linear parametric representation).

The principle of LSTD is to minimize the distance between the estimated Q-function and the projection of its image through the Bellman operator onto the

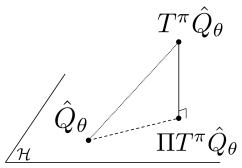

Fig. 1. LSTD principle.

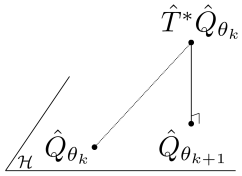

Fig. 2. Fitted-Q iteration.

hypothesis space (as illustrated in figure 1):

$$\theta \ = \underset{\in \mathbb{R}^p}{\operatorname{argmin}} \| \hat{Q} \ - \Pi T \ \hat{Q} \ \|^2 \tag{23}$$

where Π is the projection operator, and generally $T \ \hat{Q} \ \notin \mathcal{H}$. The corresponding empirical cost function is therefore (considering the N available transitions and replacing T by the sampled Bellman operator):

$$\theta \ = \underset{\in \mathbb{R}^p}{\operatorname{argmin}} \sum_{i=1}^{N} (r_i + \gamma \hat{Q} \ _\pi(s_{i+1}, \pi(s_{i+1})) - \hat{Q} \ (s_i, a_i))^2 \tag{24}$$

Notice that θ appears in both sides of this equation, which is normal. Actually, equations (23,24) correspond to two nested optimization problems: projecting $T \ \hat{Q}$ to the hypothesis space, and minimizing the distance between $\hat{Q}$ and $\Pi T \ \hat{Q}$. Thanks to the assumed linear parameterization, this optimization problem admits an analytical solution, the LSTD estimator:

$$\theta \ = \left(\sum_{i=1}^{N} \phi(s_i, a_i)(\phi(s_i, a_i) - \gamma \phi(s_{i+1}, \pi(s_{i+1}))^T \right)^{-1} \sum_{i=1}^{N} \phi(s_i, a_i) r_i \tag{25}$$

Therefore, at iteration k, knowing the policy π_k, the Q-function $\hat{Q} \ _{\pi_k} = \hat{Q} \ _k$ is estimated thanks to LSTD, and π_{k+1} is greedy respectively to it.

4.3.2 *Fitted-Q.* Fitted-Q [Bellman et al. 1973; Gordon 1995; Ernst et al. 2005; Chandramohan et al. 2010a] generalizes value iteration to model-free and large state space problems. Recall that the underlying idea of the value iteration algorithm is to repeatedly apply the Bellman optimality operator to an initial Q-function. Therefore, it seems legitimate to use the same idea for fitted-Q (by considering the sampled operator, the model being unavailable): $\hat{Q} \ _{k+1} = \hat{T}^* \hat{Q} \ _k$. However, this equation is not proper, because $\hat{T}^* \hat{Q} \ _k$ does no generally lies in the hypothesis space. Therefore, fitted-Q projects back the image of the current estimated state-action value function onto $\mathcal{H}$, which defines the new estimate, as illustrated on figure 2:

$$\hat{Q} \ _{k+1} = \Pi \hat{T}^* \hat{Q} \ _k \tag{26}$$

Recall that a linear parametrization is considered here. The projection thus reduces to solving a linear least-square problem:

$$\theta_{k+1} = \text{argmin} \sum_{i=1}^{N} (r_i + \gamma \max_{b \in A} \hat{Q}_{\ k}(s_{i+1}, b) - \hat{Q}\ (s_i, a_i))^2 \tag{27}$$

$$= \left(\sum_{i=1}^{N} \phi(s_i, a_i) \phi(s_i, a_i)^T \right)^{-1} \sum_{i=1}^{N} \phi(s_i, a_i)(r_i + \gamma \max_{b \in A} \hat{Q}_{\ k}(s_{i+1}, b)) \tag{28}$$

The reader may have noted the similarity between LSTD and fitted-Q. Actually, LSTD computes directly the fixed-point of the composed operator $\Pi\hat{T}$ whereas fitted-Q computes iteratively the fixed-point of the composed operator $\Pi\hat{T}^*$. Both approaches heavily rely on the fixed-point aspect of Bellman equations.

4.3.3 *Learning a sparse representation.* Both LSPI and fitted-Q assume that the representation (basis functions) is chosen beforehand. In the considered kernel-based linear representation, it consists in choosing the variance σ^2 of the Gaussian kernels (21), and where to place them (the number p of kernels as well as their centers (s_j, a_j)). What is proposed in this paper is to combine these algorithms with an approach which allows learning this representation from data (namely the Engel's dictionary) [Engel et al. 2004].

Let's write z a state action couple. Recall that the Q-function is approximated as $\hat{Q}\ (z) = \theta^T \phi(z)$, with $\phi(z)$ being a set of (here Gaussian) kernel basis functions:

$$\phi(z) = \left(K(z, \tilde{z}_1)\ \ldots\ K(z, \tilde{z}_p) \right)^T \tag{29}$$

The problem addressed by the Engel's dictionary method is the following: given the training basis $\{z_1, \ldots, z_N\}$ and a kernel K (which comes back to choosing σ in our case), how to choose the number p of basis functions as well as associated kernel centers $\{\tilde{z}_1, \ldots, \tilde{z}_p\} \subset \{z_1, \ldots, z_N\}$.

An important result about kernels is the Mercer theorem: for each kernel K there exists a mapping $\varphi : z \in Z \rightarrow \varphi(z) \in \mathcal{F}$ ($\mathcal{F}$ being called the feature space) such that $\forall z_1, z_2 \in Z$, $K(z_1, z_2) = \langle \varphi(z_1), \varphi(z_2) \rangle$ (in short, K defines a dot product in $\mathcal{F}$). The space $\mathcal{F}$ can be huge (it is an infinite hypersphere for Gaussian kernels), therefore φ cannot always be explicitly built. Given this result and from the bilinearity of the dot product, Q can be rewritten as follows:

$$Q\ (z) = \sum_{i=1}^{p} \theta_i K(z, \tilde{z}_i) = \langle \varphi(z), \sum_{i=1}^{p} \theta_i \varphi(\tilde{z}_i) \rangle \tag{30}$$

Therefore, a kernel-based parametrization corresponds to a linear approximation in the feature space, the weight vector being $\sum_{i=1}^{p} \theta_i \varphi(\tilde{z}_i)$. This is called the *kernel trick*. Consequently, kernel centers $(\tilde{z}_1, \ldots, \tilde{z}_p)$ should be chosen such that $(\varphi(\tilde{z}_1), \ldots, \varphi(\tilde{z}_p))$ are linearly independent in order to avoid redundancy. This can be solved using the Engel's dictionnary approach [Engel et al. 2004].

The training base is sequentially processed, and the dictionary is initiated with the first sample: $\mathcal{D}_1 = \{z_1\}$. At iteration k, a dictionary $\mathcal{D}_{k-1}$ computed from $\{z_1, \ldots, z_{k-1}\}$ is available and the k^{th} sample z_k is considered. If $\varphi(z_k)$ is linearly independent of $\varphi(\mathcal{D}_{k-1})$, then it is added to the dictionary: $\mathcal{D}_k = \mathcal{D}_{k-1} \cup \{z_k\}$.

Otherwise, the dictionary remains unchanged: $\mathcal{D}_k = \mathcal{D}_{k-1}$. Linear dependency can be checked by solving the following optimization problem (p_{k-1} being the size of $\mathcal{D}_{k-1}$):

$$\delta = \underset{w \in \mathbb{R}^{p_{k-1}}}{\operatorname{argmin}} \|\varphi(z_k) - \sum_{i=1}^{p_{k-1}} w_i \varphi(\tilde{z}_i)\|^2 \tag{31}$$

Thanks to the kernel trick, this optimization problem can be solved analytically, without computing explicitly φ. Formally, linear dependency is satisfied if $\delta = 0$. However, an approximate linear dependency is allowed, and $\varphi(z_k)$ will be considered as linearly dependent of $\varphi(\mathcal{D}_{k-1})$ if $\delta < \nu$, where ν is the so-called *sparsification factor*. This allows controlling the trade-off between quality of the representation and its sparsity. See [Engel et al. 2004] for details as well as an efficient implementation of this dictionary approach.

4.3.4 *Sparse LSPI and Sparse fitted-Q.* Combining this dictionary method with fitted-Q has been first proposed in [Chandramohan et al. 2010b]. Actually this is quite simple. In a preprocessing step, the dictionary method is used to compute the set of p basis functions from the N available state-action couples $(s_i, a_i)_{1 \leq i \leq N}$. Then the standard fitted-Q algorithm described previously is applied, this learned structure being considered. The only parameters to choose are the Gaussian standard deviation σ and the sparsification factor ν.

Combining this constructive approach with LSPI is a little bit more tricky, it was first proposed in [Xu et al. 2007]. Recall the LSTD estimate of equation (24). As opposed to fitted-Q, the inputs depend on the iteration: at iteration k, inputs are composed of the set of state-action couples $(s_i, a_i)_{1 \leq N}$ but also on transiting state-action couples $(s_{i+1}, \pi_k(s_{i+1}))$. Therefore, the dictionary has to be computed at each iteration from $\{(s_i, a_i)_{1 \leq i \leq N}, (s_{i+1}, \pi_k(s_{i+1}))_{1 \leq i \leq N}\}$.

Both algorithms are summarized in the following pseudo-codes.

Algorithm 1: Sparse Fitted-Q.

Initialization;
Initialize vector θ_0, choose a kernel K and a sparsification factor ν;
Compute the dictionary;
$\mathcal{D} = \{(\tilde{s}_j, \tilde{a}_j)_{1 \leq j \leq p}\}$ from $\{(s_j, a_j)_{1 \leq j \leq N}\}$;
Define the parametrization;
$Q\ (s,a) = \theta^T \phi(s,a)$ with $\phi(s,a) = (K((s,a),(\tilde{s}_1,\tilde{a}_1)), \ldots, K((s,a),(\tilde{s}_p,\tilde{a}_p)))^T$;
Compute P^{-1};
$P^{-1} = (\sum_{j=1}^{N} \phi_j \phi_j^T)^{-1}$;
for $k = 1, 2, \ldots, k_{max}$ **do**
  Compute θ_k, see Eq. (28);
end
$\hat{\pi}^*_{k_{max}}(s) = \operatorname{argmax}_{a \in A} \hat{Q}_{\ M}(s,a)$;

Algorithm 2: Sparse LSPI.

Initialization;
Initialize policy π_0, choose a kernel K and a sparsification factor ν;
for $k = 1, 2, \ldots, k_{max}$ **do**
  Compute the dictionary;
  $\mathcal{D} = \{(\tilde{s}_j, \tilde{a}_j)_{1 \leq j \leq p_k}\}$ from $\{(s_j, a_j)_{1 \leq j \leq N}, (s'_j, \pi_{k-1}(s'_j))_{1 \leq j \leq N}\}$;
  Define the parametrization;
  $Q\ (s, a) = \theta^T \phi(s, a)$ with
  $\phi(s, a) = (K((s, a), (\tilde{s}_1, \tilde{a}_1)), \ldots, K((s, a), (\tilde{s}_{p_k}, \tilde{a}_{p_k})))^T$;
  Compute θ_{k-1}, see Eq. (25);
  Compute π_k;
  $\pi_k(s) = \text{argmax}_{a \in A}\ \hat{Q}_{\ k-1}(s, a)$;
end

5. EXPERIMENTS

So far in this paper a family of approximate dynamic programming algorithms have been presented. The aim of this contribution is to perform dialogue optimization (to be presented in Section 5.1) using the algorithms discussed so far. All the learning algorithms discussed in Section 4 are evaluated on this dialogue management problem. In this section first the TownInfo dialogue problem is defined and casted as an MDP-SDS.

5.1 TownInfo system (MDP-SDS)

The dialogue management problem studied in this paper is a form-filling, task-oriented restaurant information system. From the literature of spoken dialogue systems it is known that in case of a form-filling system every slot value is associated with filling confidence and confirmation confidence (which is the measure of speech recognition confidence). The state representation (*i.e.,* the information state) of the dialogue problem has three slots representing the confidence of slot values of (1) the location of the restaurant, (2) cuisine of the restaurant and (3) price-range of the restaurant. Each of these 3 slot values presents the average of the filling confidence and confirmation confidence of the slot. Thus the state space of the MDP-SDS for this dialogue problem is continuous (values ranging from 0 to 1) and is 3 dimensional ($s \in \mathbb{R}^3$). The action space of the MDP-SDS includes the following 13 system actions (system-acts); Ask-A-Slot (3 actions, one for each slot), Explicit-Confirm-Slot (3 actions, one for each slot), Implicit-Confirm-And-Ask-A-Slot (6 actions, in combination of 2 slots) and Close-Dialogue action.

The reward function of the MDP-SDS is defined as follows; every correct slot filling is awarded 25, every incorrect slot filling is awarded -75 and every empty slot is awarded -300. The rewards are given at the end of the dialogue episode and the reward function includes no time-penalty to penalize the length of dialogue episode explicitly. However the discount factor γ is set to 0.95 to induce an implicit penalty for the length of the dialogue episode.

5.2 Dialogue corpora generation

To begin with a baseline system which uses a hand-crafted policy is built. It is implemented in a state-of-the-art academic dialogue management system, namely the DIPPER system [Lemon et al. 2006]. The hand-crafted policy tries to fill the three available slots one after the other and terminates the dialogue episode if all the slots are filled. Perhaps this is one of the the simplest way to perform the dialogue task studied here, yet this may not be the best way. Moreover, in the general case it is quite difficult to manually design the optimal dialogue policy. The hand-crafted policy used here for data generation did not include any confirmatory system actions and only tries to fill all the slots. This initial strategy has been kept as simple as possible to show that there is actually no need to build a more sophisticated one for collecting data.

Since the dialogues generated using the hand-crafted policy alone will not provide a good sampling of the problem space, an ϵ-random policy is designed. This ϵ-random policy acts randomly with probability ϵ and acts according to the hand-crafted policy with probability $(1 - \epsilon)$. The value of ϵ is set as 0.9 for corpora generation. By using the random policy in conjunction with the hand-crafted policy it is ensured that the problem space is sampled better and in the same time there are also episodes which have successful task completion reward.

Because this paper aims at showing the sample-efficiency of the proposed methods, a comparison between training data sets of different sizes is required. Yet, it is not possible to collect large amounts of data. So, a simulated user has been used to generate different datasets like in [Li et al. 2009] (the most similar work in the literature). In order to generate simulated dialogues, an N-gram user simulation trained on actual data is plugged into the DIPPER dialogue management software. The ϵ-greedy policy is then used to control the interaction between the DIPPER dialogue manager and simulated user for dialogue simulation.

Using this setup a total of 56,485 dialogue episodes (of which 65% are successful episodes in terms of task completion) resulting in 393,896 dialogue turns (state transitions) are collected. This training data set is divided into 8 smaller training data sets each with approximately 50K dialogue turns.

5.3 Q-function representation

As described in Section 4.2 a parametric representation of the Q-function is considered in this paper. For the experiments, a radial basis function (RBF) [Park and Sandberg 1991] is used to represent the Q-function (as explained in Section 4.2 and Equation 21). The RBF network has 3 Gaussian kernels for every dimension in the state space and considering that the restaurant problem has 13 actions, a total of 351 (*i.e.,* $3^3 \times 13$) features are used for representing the Q-function. Gaussian kernels were centered on a regular grid intersecting at $\mu_i = 0.0, 0.5, 1.0$ and the standard deviation is set to $\sigma_i = \sigma = 0.25$ on the continuous variables. One Q-function is learnt for each action since this is a discrete dimension.

5.4 Policy optimization using Approximate Dynamic Programming (ADP)

Dialogue policy optimization using ADP algorithms such as LSPI and Fitted-Q explained in Section 4.3 is studied in this section. Here the policy optimization is

done using the samples (one sample is a dialogue turn, that is a state transition $\{s, a, r, s'\}$) from the dialogue corpora. These algorithms assume that the representation of the Q-function is known beforehand. Thus for learning the optimal dialogue strategy of the restaurant information problem using the ADP algorithms the parameterized Q-function explained in Section 5.3 is used. One of the primary focus here is to to explore the sample efficiency of these ADP algorithms when applied to dialogue management problems.

LSPI is tested using the samples in the training data sets. Since there are 8 different data sets, several training sessions of LSPI are performed. For each of these training sessions the number of samples used for training is varied from 5K to 50K and the resulting estimated Q-functions are used to compute the estimated optimal policy (greedy with respect to Q-function). The stopping criterion is based on the number of changes made to the policy (being learned by LSPI) and the learning is terminated if the number of changes is less than ξ, during the experiments the value of ξ is set to 1 meaning that a stable representation is reached.

The estimated policies are then tested using the DIPPER dialogue management framework and an N-gram user simulation. It may be useful to recollect that the training is conducted using eight different data sets and using varied number of samples. Thus for example there are eight estimated policies when 5K samples are used to perform LSPI. The average cumulative sum of rewards (and their standard deviation) obtained by the policies trained using varied number of samples is shown in Figure 3.

Following LSPI, the same process is repeated to perform policy optimization using Fitted-Q. The convergence condition for the Fitted-Q is as follows; the learning is terminated if the L_1 norm of weight vectors θ from the succeeding iterations is less than ξ. The threshold is set to 1 *i.e,* convergence if $\|\theta_i^n - \theta_i^{n-1}\|_1 < \xi$, where n is the iteration number. The learned policies are then tested using DIPPER and simulated users and the average discounted reward of the estimated policies is shown in Figure 3.

It can be observed from the Figure 3 that both Fitted-Q and LSPI are sample efficient algorithms for dialogue policy optimization. Both these algorithms have been proven to converge toward the optimal policy. The figure shows that the curves are flat and therefore, the optimal policy is learnt. Anyway, the value of the optimal policy is reached after 50k samples since the curve doesn't increase anymore. This value should serve as a target for comparison of all the algorithms. In addition, the learning curve of the standard Q-learning algorithm with linear value approximation is given for the sake of comparison with the state of the art. So as to prove that the initial policy doesn't need to be very good, the performance of the hand-crafted policy are provided. The aim of this is clearly not to claim that the learnt policies are out-performing the hand-crafted one, but just to show that low performance policies are still providing enough information to learn good policies.

These approaches can learn good dialogue policies from limited number of samples counter to other RL algorithms which are known to be data intensive (such as Q-learning). Even though the performance of Fitted-Q and LSPI are more or less the same, LSPI based policy optimization comes at an additional computa-

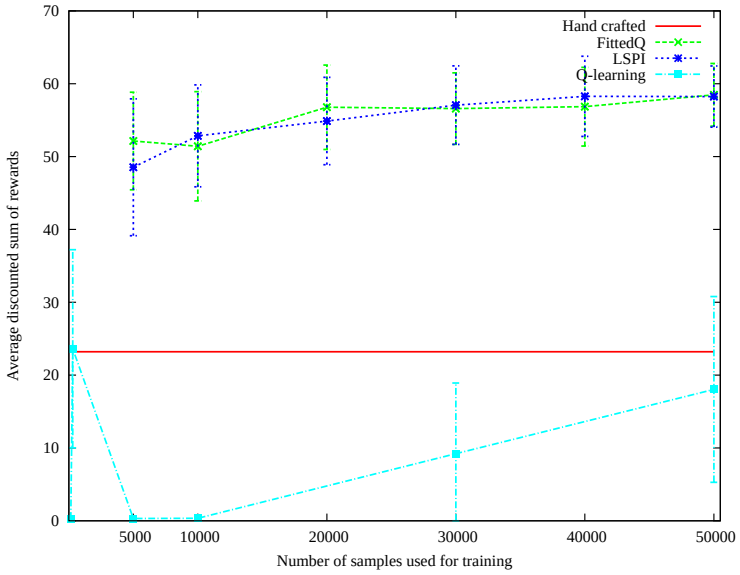

Fig. 3. *Evaluation of policies learned using Fitted-Q and LSPI*

tional cost considering the fact that it needs matrix inversion for every iteration (see Equation (24)), unlike Fitted-Q. If the number of parameters to learn is p, the complexity of LSPI is therefore $O(p^3)$ while Fitted-Q complexity is $O(p^2)$.

Table I shows an example dialogue episode between the DIPPER dialogue manager (using Fitted-Q policy learned using 10k samples) and the simulated user.

From the example dialogue episode it is clear that Fitted-Q indeed learns a good dialogue management strategy to successfully complete the restaurant information task. It may be useful to note that the hand-crafted policy of the baseline system did not include any confirmatory actions. But still Fitted-Q learns to confirm the slot if the filling confidence is less than a threshold. It learns the threshold automatically, given the constrains provided by the reward function which reflects the user's satisfaction (but also the discount factor γ), and the confirmation action is performed only when it is required.

5.5 Dialogue policy optimization using Sparse ADP

As mentioned in the previous section ADP algorithms assume that the representation of the Q-function is known beforehand. Section 5.4 explained how these algorithms can be used for dialogue policy optimization using hand selected features. In this case the Gaussian are positioned uniformly across the problem space. But many real world dialogue problems often tend to have large state space of higher dimensions. Thus hand selecting features (or positioning the Gaussian uniformly across the problem space) to represent the Q-function may not be always

Dialogue State : [0, 0, 0]
System-act : ask-slot3 {location}
User-act : provide-slot1-slot3-value { sh,south}
Dialogue State : [0.180033, 0, 0.319967]
System-act : explicitCon rm-slot3 {location}
User-act : con rm-slot3-value {south}
Dialogue State : [0.0505753, 0, 0.898849]
System-act : implicitCon rm-slot1-ask-slot2 {cuisine,price-range}
User-act : con rm-slot1-value { sh}
Dialogue State : [0.5, 0, 0.87668]
System-act : ask-slot2 {price-range}
User-act : provide-slot2-value {expensive}
Dialogue State : [0.5, 0.5, 0.897227]
System-act : closeDialogue
User goal in dialogue manager's perspective : {[slot1,slot2,slot3],[sh,expensive,south]}
Goal of the simulated user : {[slot1,slot2,slot3],[sh,expensive,south]}
Reward: 75 DialogueLength: 5 DiscountedReward: 58.0336

Table I. Example of a dialogue obtained with Fitted-Q trained on 10k examples

possible. In order to overcome this problem, a dictionary computation scheme explained in Section 4.3.3 is applied to learn the representation of the Q-function of the MDP-SDS.

In this section sparse approximate dynamic programming algorithms such as sparse Fitted-Q and sparse LSPI are used for dialogue policy optimization. Here the learning is carried out using the $\{s, a, r, s'\}$ samples from the dialogue corpora. First these algorithms try to learn the representation of Q-function and then retrieve the estimated optimal Q-function from the underlying data. It was observed during the experiments that including a constant term to the Q-function representation (value set to 1) in addition to features selected by the dictionary method avoided weight divergence. Thus the updated Q-function is used for sparse Fitted-Q and sparse LSPI. The number of features selected by the dictionary scheme varies depending on the value ν (error measure). During the experiments ν is first set to 0.7 and then set to 0.8. Dictionaries computed with ν=0.7 had about 325 to 365 features and had about 286 to 317 features with ν=0.8, depending on the number of samples used for learning.

As mentioned in Section 4.3.4 while using sparse Fitted-Q the dictionary is computed only once at the beginning of the learning using $(s_i, a_i)_{1 \leq N}$ couples from the dialogue corpora. In case of LSPI the dictionary is computed before each iteration using both $(s_i, a_i)_{1 \leq N}$ and $(s_{i+1}, \pi_k(s_{i+1}))$ couples. Similar to the process explained in Section 5.4 multiple learning sessions are performed using the different data sets and the resulting estimated policies are tested using DIPPER and simulated users.

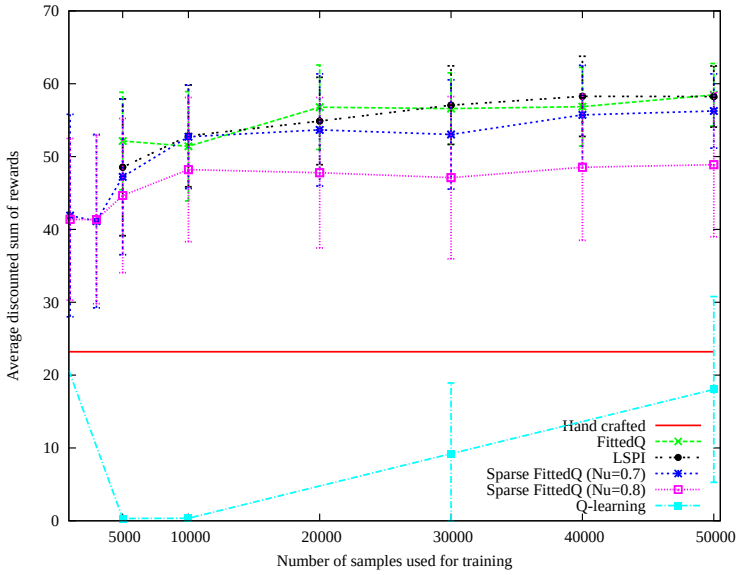

Fig. 4. *Evaluation of policies learned using Fitted-Q and Sparse Fitted-Q*

The convergence criteria for sparse Fitted-Q is similar as that of Fitted-Q whereas the convergence criteria of sparse LSPI has ξ set to 500. In later case since the basis function of the Q-function is updated before each iteration the convergence criteria is relaxed.

Figure 4 shows the comparison between the policies learned using Fitted-Q and sparse Fitted-Q. From the evaluation results it can be observed that Sparse Fitted-Q is also sample efficient and can learn good dialogue policies (as good as Fitted-Q). It can be observed that the performance of Fitted-Q is slightly better when compared to sparse Fitted-Q. However hand selecting a good feature set may not be always an option especially when dealing with larger dialogue problems.

Figure 5 shows the comparison between the policies learned using LSPI and sparse LSPI. It can be observed from the figure that policies learned using Sparse LSPI performed significantly better than the Q-learning policy. But when compared to LSPI the policies learned using Sparse LSPI is moderately inferior. This may be due to the convergence criteria used for stopping the learning. It can also be due to the fact that the learnt parameterization is different for each iteration of the algorithm which is not the case for Fitted-Q. By hand selecting the policy with less policy changes it may be possible to observe better performance. Also in this case the performance of LSPI is good since the hand-selected features set seem to be a good set of features. For many real world problems this may not be possible but sparse LSPI can still be applied.

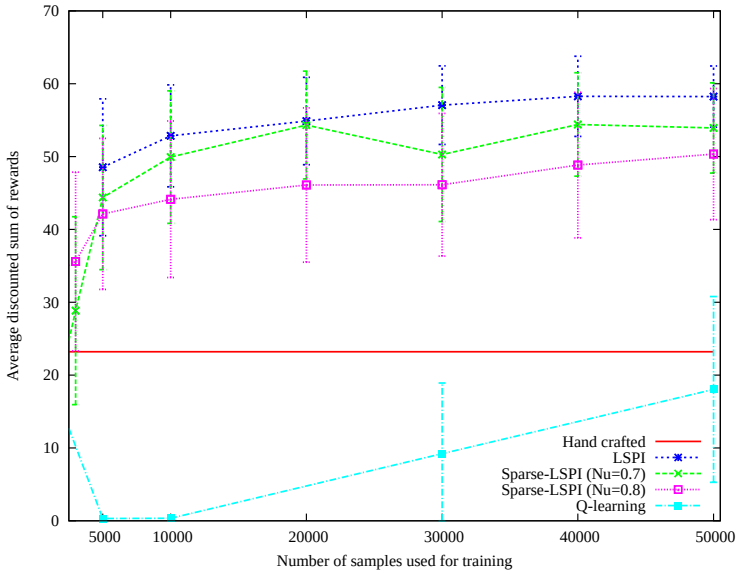

Fig. 5. *Evaluation of policies learned using LSPI and sparse LSPI*

6. DISCUSSION

Results presented in the previous section are very promising. The proposed batch algorithm could learn high performance policies from few hundreds of dialogues collected with a very simple strategy (its performance are quite bad when compared to the learnt ones) and without having to interact directly with users during the learning phase. One should remember that the learning curves are drawn with respect to dialogue turns and not dialogues. In average, a complete learnt dialogue was about 4 to 5 turns long. Since the Fitted-Q and LSPI learn quasi-optimal policies after only 5 to 10k dialogue turns, it means that only 1k to 2k dialogues were required to learn.

Although the task addressed in this paper is relatively simple (3-slots), its size is similar in terms of state-action space to those presented in the state of the art dealing with RL for SDS optimization [Levin and Pieraccini 1998; Levin et al. 2000; Singh et al. 1999; Lemon et al. 2006]. Actually, scaling up in terms of dialogue task is not to be compared to scaling up in terms of RL problems. For example, the dialogue task described in [Gasic et al. 2010; Jurcicek et al. 2010] is a 12-slot dialogue problem (so 4 times more than the system described in this paper). Yet, the RL state space used by the authors to optimize the strategy is only composed of 2 continuous and 2 discrete dimensions which makes this problem of comparable size with the one addressed in this paper in terms of RL complexity.

It is rare that in the literature, algorithms are compared according to their

sample-efficiency, that is the number of required dialogues to reach convergence. Learning curves are generally provided to show that some methods out-perform others but the actual number of dialogues required to learn is not the main focus. In [Levin and Pieraccini 1998; Levin et al. 2000] the authors mention that more than 700k dialogues were needed to learn the optimal policy on a 3 slots problem (for a database querying task). This seminal work encouraged researchers to build user simulation so as to generate enough data for RL algorithms to converge, this is probably why sample-efficiency has not been investigated so much. Yet, some recent works [Gasic et al. 2010; Jurcicek et al. 2010; Geist and Pietquin 2010a] have focused on sample-efficient RL algorithms. Nevertheless they use online and on-policy algorithms which also require user simulation, not only for expanding the datasets but simply to be run. Therefore, all the problems induced by simulation are still present [Schatzmann et al. 2005].

The work presented in this paper shares similarities with the one reported in [Li et al. 2009]. In that paper, the task is not more complex (although the vocabulary is, but the study reported here stands at the intention level) since a maximum of three slots are also requested by the system (most of dialogues only need one slot to be filled). The system is also helped by the introduction of human knowledge to pre-select actions. The results presented here for LSPI are similar to those obtained in [Li et al. 2009] which also validates results obtained for Fitted-Q. Yet, what has been reported here is the use of a generic value function approximation scheme while [Li et al. 2009] use a linear combination of preselected features (such as "is a slot value ambiguous or not") computed from the state representation. This implies prior knowledge about the system. In this paper, a kernel-based function approximation scheme was used which doesn't require any prior knowledge on the SDS nor on the features it can provide. Also, [Li et al. 2009] makes use of a user simulation method to generate different sets of data as was done in this paper. Yet, simulation is not mandatory in the general case. In addition, the user simulation doesn't simplify the task to be learnt (which only depends on the state-action space size and the quality of the exploration done with the handcrafted policy) and results are not losing generality.

The present work also introduced methods allowing the learning of a sparse representation of the value function which reduces the number of parameters to be learnt. These methods have never been applied to SDS before. Moreover, the combination of the sparse representation of the value function and of the Fitted-Q algorithm is also novel (it has only been introduced for LSPI in general by [Xu et al. 2007]). This method shows that it is possible to reduce the number of parameters while keeping acceptable performances.

Finally, the comparison of the reproducibility of the results when using Fitted-Q and LSPI show that Fitted-Q is more stable. Indeed, the variance on the performances seems to be lower with all the versions of Fitted-Q. Notice that the highest variance is obtained with Q-learning.

7. CONCLUSION

Most of the work done in the recent past with regard to use of Reinforcement Learning for policy optimization in spoken dialogue systems focused on coping

with data requirement of standard RL algorithms such as SARSA ([Li et al. 2009] is a notable exception). The technique of using simulated user (trained from real data) along with standard RL algorithms are being widely used today.

This paper explored the possibilities of using several generalization schemes under batch setup for dialogue policy optimization. The primary focus was to explore the possibility of using sample efficient RL algorithms for dialogue management so as to avoid extra modeling assumptions. To start with generalization methods such as Fitted-Q and LSPI were shown to be sample efficient approaches for policy optimization. However these approaches assumed that the representation of the Q-function is known beforehand. In many real world problems often the problem space is very large and is of higher dimensions. Thus an automatic feature selection scheme was combined with Fitted-Q and LSPI. Algorithms such as Sparse Fitted-Q and Sparse LSPI can learn the representation of the Q-function themselves and then retrieve the optimal Q-function from the data. It has been experimentally demonstrated that all these algorithms were really sample efficient and could lead to good policies with sufficiently few samples to be collected and annotated. Indeed after 10,000 samples (that is at least 4 times less in terms of dialogues considering the 3-slot task), a fairly good policy is obtained. This is due to the combination of a generalization framework together with sample efficient algorithms.

All the results presented in the paper are based on simulated users and simulated data. Naturally one task to be addressed in the near future is to learn policies on real data and test the policies learning using real users. This is planned to be done in the immediate future although, as mentioned before, the user simulation doesn't prevent generality. Other interesting directions of future work is to explore the possibility of using compact representation of the Q-function using frameworks such as Neural Networks rather than using linear parameterization (which results in large number of parameters to be learned). A RBF network as we used here can indeed approximate non-linear functions but as a linear combination of features (here Gaussian kernels). Using nonlinear representation is a promising direction of work since it involves fewer parameters to be learned (which may actually be learned using even fewer samples). Works in that direction have been started [Geist and Pietquin 2010b] to make it possible within the LSPI framework.

ACKNOWLEDGMENTS

The work presented here is part of an ongoing research for CLASSiC project (Grant No. 216594, www.classic-project.org) funded by the European Commission's 7th Framework Programme (FP7).

REFERENCES

Bellman, R. 1957. *Dynamic Programming*, sixth ed. Dover Publications.

Bellman, R. and Dreyfus, S. 1959. Functional approximation and dynamic programming. *Mathematical Tables and Other Aids to Computation 13*, 247{251.

Bellman, R., Kalaba, R., and Kotkin, B. 1973. Polynomial approximation - a new computational technique in dynamic programming: allocation processes. *Mathematical Computation 17*, 155{161.

Bradtke, S. J. and Barto, A. G. 1996. Linear Least-Squares algorithms for temporal di erence learning. *Machine Learning 22,* 1-3, 33{57.

CHANDRAMOHAN, S., GEIST, M., AND PIETQUIN, O. 2010a. Optimizing spoken dialogue management with tted value iteration. In *Proceedings of the International Conference on Speech Communication and Technologies (Interspeech 2010)*. ISCA, Makuhari (Japan), 86{89.

CHANDRAMOHAN, S., GEIST, M., AND PIETQUIN, O. 2010b. Sparse approximate dynamic programming for dialog management. In *Proceedings of the 11th SIGDial Conference on Discourse and Dialogue*. ACL, Tokyo (Japan), 107{115.

ECKERT, W., LEVIN, E., AND PIERACCINI, R. 1997. User Modeling for Spoken Dialogue System Evaluation. In *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*. 80{87.

ENGEL, Y., MANNOR, S., AND MEIR, R. 2004. The Kernel Recursive Least Squares Algorithm. *IEEE Transactions on Signal Processing 52*, 2275{2285.

ERNST, D., GEURTS, P., AND WEHENKEL, L. 2005. Tree-Based Batch Mode Reinforcement Learning. *Journal of Machine Learning Research 6*, 503{556.

GASIC, M., JURCICEK, F., KEIZER, S., MAIRESSE, F., THOMSON, B., YU, K., AND YOUNG, S. 2010. Gaussian processes for fast policy optimisation of pomdp-based dialogue managers. In *SIGDIAL'10*. Tokyo, Japan.

GEIST, M. AND PIETQUIN, O. 2010a. Managing uncertainty within the ktd framework. In *Proceedings of the Workshop on Active Learning and Experimental Design (AL&E collocated with AISTAT 2010)*. Journal of Machine Learning Research Conference and Workshop Proceedings. Sardinia (Italy). 12 pages - to appear.

GEIST, M. AND PIETQUIN, O. 2010b. Statistically linearized least-squares temporal di erences. In *Proceedings of the IEEE International Conference on Ultra Modern Control systems (ICUMT 2010)*. IEEE, Moscow (Russia). 8 pages.

GORDON, G. 1995. Stable Function Approximation in Dynamic Programming. In *Proceedings of the International Conference on Machine Learning (ICML)*.

HENDERSON, J., LEMON, O., AND GEORGILA, K. 2008. Hybrid reinforcement/supervised learning of dialogue policies from xed data sets. *Computational Linguistics*.

JURCICEK, F., THOMSON, B., KEIZER, S., GASIC, M., MAIRESSE, F., YU, K., AND YOUNG, S. 2010. Natural Belief-Critic: a reinforcement algorithm for parameter estimation in statistical spoken dialogue systems. In *Interspeech'10*. Makuhari (Japan).

LAGOUDAKIS, M. G. AND PARR, R. 2003. Least-squares policy iteration. *Journal of Machine Learning Research 4*, 1107{1149.

LARSSON, S. AND TRAUM, D. B. 2000. Information state and dialogue management in the TRINDI dialogue move engine toolkit. *Natural Language Engineering*.

LEMON, O., GEORGILA, K., HENDERSON, J., AND STUTTLE, M. 2006. An ISU dialogue system exhibiting reinforcement learning of dialogue policies: generic slot- lling in the TALK in-car system. In *Proceedings of the meeting of the European chapter of the Associaton for Computational Linguistics (EACL'06)*. Morristown, NJ, USA.

LEMON, O. AND PIETQUIN, O. 2007. Machine Learning for Spoken Dialogue Systems. In *Proceedings of the European Conference on Speech Communication and Technologies (Interspeech'07)*. Anvers (Belgium), 2685{2688.

LEVIN, E. AND PIERACCINI, R. 1998. Using markov decision process for learning dialogue strategies. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP'98), Seattle, Washington*.

LEVIN, E., PIERACCINI, R., AND ECKERT, W. 2000. A stochastic model of human-machine interaction for learning dialog strategies. *IEEE Transactions on Speech and Audio Processing 8,* 1, 11{23.

LI, L., BALAKRISHNAN, S., AND WILLIAMS, J. 2009. Reinforcement Learning for Dialog Management using Least-Squares Policy I55(ar)55 and In *Proceedings of the International Conference on Speech Communication and Technologies (InterSpeech'09)*. Brighton (UK).

PARK, J. AND SANDBERG, I.

PIETQUIN, O. 2004. *A Framework for Unsupervised Learning of Dialogue Strategies*. SIMILAR Collection. Presses Universitaires de Louvain. 246 pages.

PIETQUIN, O. 2006a. Consistent Goal-Directed User Model for Realistic Man-Machine Task-Oriented Spoken Dialogue Simulation. In *Proceedings of the 7th IEEE International Conference on Multimedia and Expo*. Toronto (Canada), 425{428.

PIETQUIN, O. 2006b. Machine Learning for Spoken Dialogue Management: an Experiment with Speech-Based Database Querying. In *Artificial Intelligence : Methodology, Systems & Applications*, J. E. . J. Domingue, Ed. Lecture Notes in Arti cial Intelligence, vol. 4183. Springer Verlag, 172{180.

PIETQUIN, O. AND RENALS, S. 2002. ASR System Modeling For Automatic Evaluation And Optimization of Dialogue Systems. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2002)*. Vol. I. Orlando, (USA, FL), 45{48.

PUTERMAN, M. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience.

RIESER, V. AND LEMON, O. 2008. Learning e ective multimodal dialogue strategies from wizard-of-oz data: Bootstrapping and evaluation. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics (ACL/HLT 2008)*.

SAMUEL, A. L. 1959. Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development*, 210{229.

SCHATZMANN, J., STUTTLE, M. N., WEILHAMMER, K., AND YOUNG, S. 2005. E ects of the user model on simulation-based learning of dialogue strategies. In *Proceedings of workshop on Automatic Speech Recognition and Understanding (ASRU'05), San Juan, Puerto Rico*.

SCHATZMANN, J., THOMSON, B., AND YOUNG, S. 2007. Error simulation for training statistical dialogue systems. In *Proceedings of the International Workshop on Automatic Speech Recognition and Understanding (ASRU'07)*. Kyoto (Japan).

SCHEFFLER, K. AND YOUNG, S. 2001. Corpus-based dialogue simulation for automatic strategy learning and evaluation. In *Proceedings of NAACL Workshop on Adaptation in Dialogue Systems*.

SCHOLKOPF, B. AND SMOLA, A. J. 2001. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA.

SINGH, S., KEARNS, M., LITMAN, D., AND WALKER, M. 1999. Reinforcement learning for spoken dialogue systems. In *Proceedings of the Annual meeting of the Neural Iniformation Processing Society (NIPS'99), Denver, USA*. Springer.

SUTTON, R. S. AND BARTO, A. G. 1998. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*, 3rd ed. The MIT Press.

W3C 2008. *VoiceXML 3.0 Specifications*. W3C. http://www.w3.org/TR/voicexml30/.

WALKER, M. A., LITMAN, D. J., KAMM, C. A., AND ABELLA, A. 1997. PARADISE: A framework for evaluating spoken dialogue agents. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL'97)*. Madrid (Spain), 271{280.

WATKINS, C. J. 1989. Learning from Delayed Rewards. Ph.D. thesis, University of Cambridge, England.

WILLIAMS, J. D. AND YOUNG, S. 2007. Partially observable markov decision processes for spoken dialog systems. *Computer Speech Language*.

XU, X., HU, D., AND LU, X. 2007. Kernel-based least squares policy iteration for reinforcement learning. *IEEE Transactions on Neural Networks 18,* 4, 973{992.