A Comprehensive Reinforcement Learning Framework for Dialogue Management Optimization

Lucie Daubigney, Matthieu Geist, Senthilkumar Chandramohan, and Olivier Pietquin, *Senior Member, IEEE*

***Abstract*—Reinforcement learning is now an acknowledged approach for optimizing the interaction strategy of spoken dialogue systems. If the first considered algorithms were quite basic (like SARSA), recent works concentrated on more sophisticated methods. More attention has been paid to off-policy learning, dealing with the exploration-exploitation dilemma, sample efficiency or handling non-stationarity. New algorithms have been proposed to address these issues and have been applied to dialogue management. However, each algorithm often solves a single issue at a time, while dialogue systems exhibit all the problems at once. In this paper, we propose to apply the Kalman Temporal Differences (KTD) framework to the problem of dialogue strategy optimization so as to address all these issues in a comprehensive manner with a single framework. Our claims are illustrated by experiments led on two real-world goal-oriented dialogue management frameworks, DIPPER and HIS.**

***Index Terms*—Dialogue management, reinforcement learning, spoken dialogue system.**

I. Introduction

SPOKEN Dialogue Systems (SDSs) allow the interaction between humans and computers by means of speech. Although several dialogue types exist (social, chats, *etc.*), this paper focuses on goal-oriented dialogues where the user is helped by the machine to achieve a goal. Most often, the goal is to retrieve some information (ticket booking, assistance, appointment scheduling, *etc.*). Although the use of dialogue systems becomes more and more common today, the user experience remains quite frustrating at times. In order to improve the quality of this experience, the dialogue should be efficient (providing the right information at an appropriate time) and natural (acting in the appropriate manner with a human).

Manuscript received July 23, 2012; revised October 12, 2012; accepted November 16, 2012. Date of publication November 21, 2012; date of current version January 03, 2013.This work was supported in part by the 7th European framework program (CLASSiC project) and the INTERREG IVa funding scheme (ALLEGRO project) and in part by the Région Lorraine. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Jason Williams.

L. Daubigney is with the IMS-MaLIS Research Group, Supélec, Metz 57070, France, and also with the MAIA Project-team, INRIA (e-mail: lucie.daubigney@supelec.fr).

M. Geist is with the IMS-MaLIS Research Group, Supélec, Metz 57070, France (e-mail: matthieu.geist@supelec.fr).

S. Chandramohan is with the IMS-MaLIS Research Group, Supélec, Metz 57070, France, and also with the LIA, Université d'Avignon, Avignon 84911, France (e-mail: senthilkumar.chandramohan@supelec.fr).

O. Pietquin is with the IMS-MaLIS Research Group, Supélec, Metz 57070, France, and also with the UMI 2958 (GeorgiaTech - CNRS) (e-mail: olivier.pietquin@supelec.fr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSTSP.2012.2229257

A central component of the SDS is the Dialogue Manager (DM). Provided the current *dialogue context* (that can include the history of the dialogue, some external information like database hits, *etc.*), the dialogue manager must decide what to say next to the user. The ultimate goal is to provide the required information through natural interactions. This problem is made difficult by the fact that the available information is the output of speech processing components (such as speech recognition and understanding modules), which are error-prone. Moreover, the variability induced by the interaction with several users increases the difficulty of building such a component.

Therefore, the DM has to *take decisions* based on uncertain *dialogue contexts* in order to eventually *achieve a goal*. This is actually the definition of a sequential decision making problem under uncertainty. Dialogue management has first been described as such in the mid '90s [1]. To do so, the DM problem is cast into a Markov Decision Process (MDP) [2]. This is a mathematical framework allowing the computation or the learning of an optimal mapping between situations and actions. This mapping is called a *policy*. This framework comes from the optimal control theory and we consider thus the decision making process as a *control* problem. A probabilistic description of the task, which requires the knowledge of some parameters is made. If some of those parameters are unknown, the reinforcement learning (RL) paradigm [3] provides some approximate solutions to the problem of policy optimization. The DM is seen as an agent which has to interact with its environment (the human user here) in order to maximize some expected cumulative discounted reward. This reward is ideally provided only when the user is satisfied.

The first RL algorithm to have been considered for DM training is based on Monte Carlo sampling [4], [5]. This approach requires applying sub-optimal policies a large number of times to measure their performance so as to improve iteratively the quality of the interaction. This process is called *on-policy* learning because the currently learnt policy has to be tested to get improved. However, this is not applicable to real users, as it may be quite annoying if the tested policies are far from optimal. Moreover, this algorithm requires a huge amount of interactions to provide good solutions (*no sample efficiency*). For these reasons, user simulators have been proposed to train DMs by interaction with artificial users [6]–[8]. However, a modeling bias is introduced. The learnt behavior may consequently not fit for any real user but only for the simulated average user [9], [10].

A better alternative would be to directly use data sampled according to a sub-optimal but acceptable policy. This policy can typically be provided by the data which are used to train the user simulator and are obtained by letting the user interact with a handcrafted DM or with a wizard-of-the-Oz system [11]. The corpus could be used to learn the underlying MDP (or some approximation) to compute an exact solution to the control problem [1], [12]. Yet, the dialogue system must be very simple so that all the situations (contexts) can be enumerated. This way, transition probabilities between contexts could be estimated from data and be used to solve the optimisation problem by dynamic programming [13]. The conditions for using such a method are rarely met in real applications because of obvious scalability problems. Another way to use the corpus would be to infer directly an optimal policy by observing the sub-optimal policy followed in the data without explicitly building a complete model of the MDP. This is known as *off-policy learning*: the agent learning an optimal policy does not have to apply its current estimation to test it. The learning can be done offline, using previously collected data. Yet, collecting and annotating data can be very costly so algorithms learning good controllers with as few examples as possible would be preferred. This is known as *sample efficiency*. Such an approach avoids needing a user simulator. The user modeling bias is consequently lowered (some inherent bias from available data still remains). Some recent works based on (batch) approximate dynamic programming [14], [15] follow this line of work. However, approximations are made during the process. Optimality is not strictly guaranteed. Indeed, the user population present in the collected data may not cover all of the possible behaviors. Another batch algorithm has been used in [16] where a hybrid reinforcement learning algorithm based on SARSA(λ) was developed. However, this algorithm is known to slowly converge and in the article, the space spanned by the features used for the approximation is not rich enough.

Therefore, the property of an algorithm to learn online while interacting with the user may still be interesting. Assume that a good sub-optimal initial policy has been found by either some sample efficient off-policy RL algorithm or by hand. Continuing the learning during the interaction with the user while using the currently estimated policy could improve the interaction. Thus, sample efficient algorithms are still required to learn quickly without impacting the user experience. Learning while controlling the system also implies a dilemma between exploration and exploitation. At each interaction the DM must choose between a decision assumed to be optimal so far (exploitation) or a sub-optimal action which aims at collecting more information about the optimal policy without guarantee (exploration). The exploration phase is necessary but should be really cautious in order to avoid annoying the user. Many schemes exist to handle this dilemma but most of them require a confidence information about current estimates to be provided, see for example [17]. The proposed algorithm should be thus based on a *probabilistic model*. Very little work has been done in this direction. In [18] and [19], the authors use Gaussian Processes (GP) while in [20] a Natural Actor Critic (NAC) algorithm is used. It has to be noticed that these algorithms only provide online and on-policy learning. This can be a drawback since they are unable to deal with non-stationarity. Indeed, this kind of learning implies direct changes in the policy thus a non-stationarity. The user may also get used to the DM over the interactions and therefore may change her behavior over time which introduces non-stationarities. The DM should be able to react to this change. In other words, adaptive RL algorithms able to handle non-stationarities would better suit for an online training of the DM.

The learning of a policy is very often done through the learning of a so-called *action-value function*. This function roughly measures the average cumulative reward that can be expected from a context by following a given policy. This function must be computed for every context and even more additionally for each possible action. In the case of complex problems such as DM, this function cannot be represented exactly because the number of possible contexts is too large. The context often lies in a multidimensional continuous space. An approximate architecture must consequently be chosen. Linear parameterizations are standard for RL algorithms (*e.g.*, Radial Basis Function –RBF– networks) while non-linear parameterizations are less usual. However such a representation may be desirable. For example, multi-layered artificial neural networks are known to be universal approximators. They allow handling large input spaces (*i.e.*, dialogue states) therefore addressing the curse of dimensionality. Thus, an RL algorithm for DM should be able *to handle non-linear architectures*.

To sum up, a good RL algorithm for DM training should exhibit the following properties:

- *on-* and *off-policy* learning,
- addressing the dilemma between exploration and exploitation (so estimation of uncertainty),
- supporting *linear* or *non-linear* parameterization,
- *sample efficiency* and *adaptivity*.

This paper argues that the Kalman Temporal Differences (KTD) framework [21] to be presented in Section III, satisfies all of these properties. This framework is thus suitable for all phases of Dialogue Management policy learning.

Before this, we introduce some necessary background on Markov Decision Processes (MDP) and on the modeling of DM as a sequential decision problem. Our claims are supported by a series of experiments on two well established dialogue management frameworks, DIPPER [22] and HIS [23], see Sections IV and V. Section VI concludes the paper.

II. Background

This section briefly reviews the paradigm on which is based reinforcement learning: the Markov Decision Processes (MDP). Then, the casting of the DM problem as an MDP is presented.

A. Markov Decision Processes

Formally, an MDP is a tuple $\{S, A, P, R, \gamma\}$ where S is the (possibly continuous) state space, A is the discrete action space, P is a set of Markovian transition probabilities, $R : S \times A \times S \rightarrow \mathbb{R}$ is the reward function, and $\gamma \in [0, 1]$ is the discount factor (discounting long term rewards). At each time step t, the environment is in a state s_t and the agent chooses an action a_t according to some mapping from states to actions called a policy, $\pi : S \rightarrow A$. The state then changes to s_{t+1} according to the Markovian transition probability, $s_{t+1} \sim p(.|s_t, a_t)$, and the agent is rewarded by $r_t = R(s_t, a_t, s_{t+1})$. The goal of the agent is to find a policy which maximises the expected discounted cumulative reward.

This quantity is called the value function and is defined for a given policy as $V^\pi(s) = \mathbb{E}[\sum_{t\geq 0}\gamma^t r_t | s_0 = s, \pi] \in \mathbb{R}^S$. In other words, the goal is to find $\pi^* \in \arg\max_\pi V^\pi$. For this sake, the action-value function (or Q-function) is defined. This function adds a degree of freedom on the first action to be chosen: $Q^\pi(s,a) = \mathbb{E}[\sum_{t\geq 0}\gamma^t r_t | s_0 = s, a_0 = a, \pi] \in \mathbb{R}^{S\times A}$.

The (action-) value function of any optimal policy π^* is noted (Q^*) V^*. If Q^* is known, an optimal policy can be directly computed by being greedy according to this Q-function:

$$\pi^*(s) \in \arg\max_a Q^*(s,a).$$

Using the Markovian property, the action-value function of a given policy π satisfies the (linear) Bellman evaluation equation:

$$Q^\pi(s_t,a_t) = \mathbb{E}_{s_{t+1}|s_t,a_t}[r_t + \gamma Q^\pi(s_{t+1}, \pi(s_{t+1}))].$$

In order to find the optimal policy, this equation can be used within a policy iteration scheme. An initial policy π_0 is chosen. At iteration i, the policy π_i is evaluated, that is the related Q-function is computed thanks to the Bellman evaluation equation. Then, the policy π_{i+1} is defined as being greedy respectively to Q^{π_i}: $\pi_{i+1}(s) \in \arg\max_a Q^{\pi_i}(s,a)$. If the model is unknown, the action-value function can be estimated from interactions using the TD-Q (Temporal Difference algorithm applied to the Q-function [3]) algorithm for example (α_t being a learning rate):

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha_t(r_t + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)).$$

Moreover, to learn in an online fashion, an asynchronous and optimistic approach can be adopted. A greedy step can be taken at each iteration. In order not to get stuck, an exploration scheme should be added. For example, an ϵ-greedy policy can be chosen: at each time step, the greedy action is taken with probability $1-\epsilon$ and a random action otherwise. Overall, this gives the well known SARSA algorithm [3].

Now, the direct estimation of the optimal action-value function $Q^* = Q^{\pi^*}$ may be preferred. This is possible using the (non-linear) Bellman optimality equation:

$$Q^*(s_t,a_t) = \mathbb{E}_{s_{t+1}|s_t,a_t}[r_t + \gamma \max_a Q^*(s_{t+1},a)].$$

The value iteration approach uses the fixed-point property of this equation to compute iteratively its unique solution. This is of interest: if a sampling approach is chosen, assuming the model (transition and reward function) is not known (for example, the Q-learning algorithm), the learning of the optimal Q-function from trajectories sampled according to an arbitrary policy is possible (as long as the policy is explorative enough). We recall the Q-learning update:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha_t(r_t + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)).$$

We already mentioned that if the model is unknown (which is the case for SDS), the estimation of the action-value function relies on sampling (that is, interaction with the environment). However, the state space may be too large to allow an exact representation of this function. An approximation architecture has consequently to be adopted. For example, a linear parameterization can be used: $\hat{Q}_\theta(s,a) = \theta^T\Phi(s,a)$. The feature vector $\Phi(s,a)$ is a set of p basis functions to be chosen beforehand and $\theta \in \mathbb{R}^p$ is the set of parameters to be learnt. A non-linear parameterization such as a multi-layered artificial neural network can also be chosen. In this case, the Q-function $\hat{Q}_\theta$ is parameterized by the weights of the synaptic connections. The learning is much more complex in this case, especially for reinforcement learning. Classic algorithms such as TD cannot be shown to be convergent with such architectures [24].

Given some architecture $\hat{Q}_\theta$, the aim is to estimate from samples an approximate solution to one of the Bellman equations.

B. Casting the DM Problem as an MDP

The casting of the dialogue management problem into the Markovian decision process framework is presented now. A first thing to define is the dialogue state. A natural choice would be to consider the output of the speech recognizer/semantic analyzer. However, this would cause a problem of partial observability. This could be handled by the POMDP (Partially Observed MDP) framework, but this is much more complex [25]. Instead of using this approach, we consider here the Information State Paradigm [26]. The SDSs we consider are goal-oriented. The objective can be summarized as filling a certain number of slots (*e.g.*, for a ticket booking problem, the date, the destination and possibly the accommodation). This paradigm provides a component which indicates for each slot some confidence level (zero if never asked, one if the user was well understood, possibly with confirmation asked from the DM), which summarizes the history of the dialogue. For the HIS system, this approach is enhanced with a summary space [27] which selects the most relevant and plausible hypotheses (due to the high and variable number of slots to be filled). In any case, we refer to it as the dialogue state (or state for short).

To conduct the dialogue, the DM may ask to fill a slot or to confirm it, to greet the user, to provide information, or any combination. These acts form the action set. The transition probabilities are not analytically known, but they can be sampled simply by letting the user and the DM interact. The reward is given to penalize long dialogues. So each turn in the dialogue is punished and a more informative reward is provided when the DM gives the required information to the user. Designing the reward is not an easy task and other solutions may be considered [28]–[30].

III. The Kalman Temporal Differences Framework

The Kalman filter [31] is a prominent signal processing algorithm. This filter aims at inferring some hidden variables provided related observations. It is important to notice that Kalman filtering differs from Bayesian filtering. Indeed, the Bayesian filter aims at computing the complete posterior distribution conditioned on past observations of the hidden variables. A Kalman filter only focuses on the first and second moments of this distribution, as a consequence of constrained linear updates. They are equal only when Gaussian distributions and linear models are concerned.

In this framework, the hidden variables are modeled as a random vector. This vector is allowed to evolve over time through an *evolution equation*. The hidden variables are linked to some observations through an *observation equation*. The goal is to infer, in an online manner, the hidden variables at each time step given the past observations and their equations. The Kalman filter provides the best linear estimator which minimises the mean squared error conditioned on past observations. The estimator is linear in the sense that the update of the hidden variables is linear according to the predicted observation error. The Kalman Temporal Differences (KTD) framework [21] adapts this popular approach to the action-value function estimation problem.

More precisely, a parametric representation $\hat{Q}_\theta$ of the action-value function is adopted. The hidden variables to be inferred are the components of the parameter vector θ. The first thing to do is to assess a related evolution equation. As there is no reason to assume a specific evolution model for this parameter vector, an Occam's razor principle is applied and a random walk is considered: $\theta_t = \theta_{t-1} + v_t$, with v_t a white noise of covariance matrix P_{v_t}. This allows taking into account non-stationarity. This is of interest if the behavior of the user evolves (without harming the case where this behavior is stationary). The observations are the rewards (and associated transitions). They are linked to the hidden parameters through one of the sampled Bellman equations. If the evaluation equation is considered, the observation equation is $r_t = \hat{Q}_{\theta_t}(s_t, a_t) - \gamma \hat{Q}_{\theta_t}(s_{t+1}, a_{t+1}) + n_t$. The term n_t is a white noise of covariance matrix P_{n_t}. This noise takes into account the fact that the *sampled* Bellman equation is used and not the true Bellman equation. Similarly, the Bellman optimality equation can be considered: $r_t = \hat{Q}_{\theta_t}(s_t, a_t) - \gamma \max_a \hat{Q}_{\theta_t}(s_{t+1}, a) + n_t$. Put together, this gives the following model of evolution and observation equations:

$$\begin{cases} \theta_t &= \theta_{t-1} + v_t \\ r_t &= g_t(\theta_t) + n_t \end{cases}$$

with

$$g_t(\theta_t) = \begin{cases} \hat{Q}_{\theta_t}(s_t, a_t) - \gamma \hat{Q}_{\theta_t}(s_{t+1}, a_{t+1}) \text{ (evaluation)} \\ \hat{Q}_{\theta_t}(s_t, a_t) - \gamma \max_a \hat{Q}_{\theta_t}(s_{t+1}, a) \text{ (optimality).} \end{cases}$$

If the evaluation equation is used in the algorithm, we call it KTD-SARSA whereas if the optimality equation is used, we call it KTD-Q. Provided the model, KTD is the best linear estimator minimizing the mean squared error between the true parameter vector θ_t and its estimation $\hat{\theta}_t$, conditioned on past observations (past rewards $r_{1:t} = r_1 \dots r_t$). The error is $J_t(\theta) = \mathbb{E}[\|\theta_t - \theta\|^2 | r_{1:t}]$ and $\hat{\theta}_t = \arg\min_{\theta,\text{ linear update}} J_t(\theta)$.

Further details are provided in [21]. Only the main points are given here. The KTD algorithm is made of four steps: an *initialization* step, a *prediction* step, a step to compute some *statistics of interest* and finally a *correction* step.

The initialization step consists in choosing the priors θ_0 and $P_{0|0}$ ($P_{t|t} = cov(\theta - \hat{\theta}_{t|t} | r_{1:t})$). The notation $\bullet_{i|j}$ represents the estimation of the variable $\bullet$ at time step i knowing the previous samples from time step 0 to time step j ($j \leq i$). Then, for each of the transitions, the second, third, and fourth steps are repeated. The second step is the prediction step:

$$\begin{cases} \hat{\theta}_{t|t-1} &= \hat{\theta}_{t-1|t-1}, \\ P_{t|t-1} &= P_{t-1|t-1} + P_{v_t}. \end{cases}$$

During the third step, some statistics of interest are computed:

$$\begin{cases} \hat{r}_{t|t-1} &= \mathbb{E}[g_t(\theta_t) | r_{1:t-1}], \\ P_{\theta r_t} &= \mathbb{E}\left[(\theta_t - \hat{\theta}_{t|t-1})(g_t(\theta_t) - \hat{r}_t) | r_{1:t-1}\right], \\ P_{r_t} &= \mathbb{E}\left[(g_t(\theta_t) - \hat{r}_{t|t-1})^2 | r_{1:t-1}\right] + P_{n_t}. \end{cases}$$

The scalar $\hat{r}_t$ is the prediction of the reward given the currently estimated parameter vector (and thus the current estimate of the Q-function). Finally, the correction step is made using the statistics just updated:

$$\begin{cases} K_t &= P_{\theta r_t} P_{r_t}^{-1}, \\ \hat{\theta}_{t|t} &= \hat{\theta}_{t|t-1} + K_t(r_t - \hat{r}_{t|t-1}), \\ P_{t|t} &= P_{t|t-1} - K_t P_{r_t} K_t^T. \end{cases} \tag{1}$$

The term K_t is the Kalman gain, and $r_t - \hat{r}_t$ is actually a temporal difference error. The second equation of (1) is a Widrow-Hoff equation. Among all Widrow-Hoff-like updates of the parameter vector (for example, the classical TD algorithm is of this form), KTD is the one minimizing the conditional mean squared error, that is $J_t(\theta) = \mathbb{E}[\|\theta_t - \theta\|^2 | r_{1:t}]$. KTD is a second order algorithm and is thus *sample efficient* (contrary to SARSA and Q-learning which are gradient-descent-like thus first-order methods). By means of the random walk evolution model, the parameters are allowed to evolve. In other words, non-stationarity is taken into account. Therefore, KTD can be safely used in an optimistic policy iteration scheme and is able to track the possible changes in the user behavior.

As seen during the description of the KTD framework, some parameters must be chosen: the variance of the observation noise P_{n_t}, the initial values of $\hat{\theta}_{0|0}$ and $P_{0|0}$, and the variance of the process noise P_{v_t}. To have a hint about how to choose them, we recall the expression of the mean squared error (theorem 2 of [21]) minimized by the KTD algorithm (without evolution noise):

$$J_t(\theta) = \sum_{j=0}^{t} \frac{1}{P_{n_j}} (r_j - g_i(\theta))^2 + (\theta - \theta_0)^T P_0^{-1} (\theta - \theta_0) \tag{2}$$

The $\hat{\theta}_{0|0}$ vector can be initialized to a default value or, if known, a value near the optimal one. The $P_{0|0}$ matrix translates the certainty the user has in the prior $\hat{\theta}_{0|0}$. It appears in (2) as a regularization term. A common initialization for this matrix is $\lambda\mathrm{Id}$. Only one parameter (λ) has thus to be chosen instead of p^2 parameters (p being the number of parameters to learn). The higher λ the less certain $\hat{\theta}_{0|0}$ is. The observation noise P_{n_t} represents the confidence about the ability of the chosen parameterization to represent the true Q-function. Additionally, the evolution noise is chosen so that $P_{v_t} = \eta P_{\theta_{t-1|t-1}}$ with $0 < \eta \ll 1$. The effect of this heuristic is to put more emphasis on more recent data (it acts as a forgetting factor).

The statistics of interest required for the Kalman gain such as the predicted reward $\hat{r}_{t|t-1}$ can be analytically computed if the observation equation is linear (Bellman evaluation equation and linear parameterization). Otherwise, they can be computed efficiently using a derivative-free approximation scheme, the unscented transform (UT) [32]. The derivative-free aspect is particularly interesting for two reasons. First, non-linear architectures such as neural networks can easily be taken into account

	State Space	Action Space	Reward function	γ
DIPPER	• 3 continuous components $S = [0, 1]^3$. • Each slot value is the average between the filling and confirmation confidences provided by the speech recognition module.	• Contains 13 different actions. • *E.g.* "ask-slot", "explicit-confirm", "implicit-confim".	• Each turn penalised by -1. • At the end of a dialogue: $+25$ & -75 for resp. correct & incorrect fillings, -300 for empty slots.	0.95
HIS	• 4 components $S = [0, 1]^2 \times [\![0, 21]\!] \times [\![0, 5]\!]$. • $s =< s^1, s^2, s^3, s^4 >$: s^1 and s^2 represent the confidence associated with the two most probable dialogue histories, s^3 is the most probable user action, s^4 is the estimation of the user goal.	• Contains 12 summary actions. • *E.g.* "confirm", "repeat", *etc.*	• Each turn penalised by -1. • At the end of a dialogue: $+20$ if the task is fulfiled, 0 if not.	0.95

Fig. 1. Details about how the dialogue management problem is cast into an MPD for the two systems, DIPPER and HIS.

without requiring the computation of gradient through back-propagation. Second, the Bellman optimality equation, which is not derivable everywhere because of the $\max$ operator, can be used. Therefore, the KTD framework allows *off-policy learning*.

Recall that the parameter vector θ is modeled as a random variable. The Q-function $Q_\theta(s_t, a_t)$ is consequently a random variable itself. It is thus possible to compute the mean of the Q-function $\mu_{Q,\hat{\theta}_t}(s, a) = \mathbb{E}_{\hat{\theta}_t}[\hat{Q}_{\theta_t(s,a)}]$ (possibly using the UT). This quantity can be used to take decisions (*e.g.*, by selecting the greedy action). The associated variance of the Q-function can also be computed (still possibly using the UT): $\sigma^2_{Q,\hat{\theta}_t}(s, a) = \mathrm{var}_{\hat{\theta}_t}[\hat{Q}_{\theta_t}(s, a)]$. As announced before, this can be particularly useful for the exploration-exploitation dilemma [17]. For example, in the experiments we consider the bonus-greedy scheme (see Section V). Others schemes have been proposed in [33]. This one was the most efficient. When using the bonus-greedy exploration, the chosen action is the one which maximizes

$$\mu_{Q,\hat{\theta}_t}(s, a) + \beta \frac{\sigma^2_{Q,\hat{\theta}_t}(s, a)}{\beta_0 + \sigma^2_{Q,\hat{\theta}_t}(s, a)},$$

where β and β_0 are some meta-parameters. When the uncertainty about the action-value function estimate for the state-action couple (s, a) is high, the variance $\sigma^2_{Q,\hat{\theta}_t}(s, a)$ is high and a bonus (of value up to β) is given for the choice of this action, favoring exploration. As the state space is explored, the uncertainty decreases and the variance tends to be low, favoring the choice of the greedy action.

To sum up, KTD is a sample efficient framework by its very principles. Off-policy learning and non-linear parameterization can be performed by means of the UT derivative-free approximation scheme. By using the random walk evolution equation, it is also adaptive. The parameters being modeled as random vectors, an uncertainty information about the currently estimated action-value function is provided (for any state-action couple). This is useful for the exploration-exploitation dilemma. These are all the properties we advocated to be important for DM training in Section I. The next section illustrates this empirically.

IV. Experimental Setup

We present here the two dialogue management frameworks considered in this paper and the Q-function representation used for the experiments.

A. Considered SDSs

Two dialogue management frameworks are considered for the experiments, namely DIPPER and HIS. Both of them are tools which allow a SDS designer to write or learn dialogue management rules.

1) The Choice of Two Systems: To avoid or at least minimize the impact of simulation on our conclusions, the algorithms are trained in two quite different contexts. Firstly, the two user simulators are different: the first one is an agenda-based simulator (with HIS) while the other one is based on n-grams (with DIPPER). Second, the way the state spaces are built is also different (see Tab. Fig. 1). Although this will not replace evaluation on real users, it exhibits the capability of KTD to adapt when experimental conditions change drastically. Switching to real users can thus be considered as a new change in the learning conditions to which we expect KTD to be robust.

2) DIPPER: The first one is the DIPPER system (Dialogue Prototyping Equipment & Resources) built by the University of Edinburgh [22]. This system implements a slot filling problem for a tourist information task. The problem is to fill-in three slots: (i) the location of a restaurant, (ii) the type of cuisine and (iii) its price range. In this article, the experiments on the DIPPER system are conducted using an n-gram user simulator [34]. One has to notice that the system has also been tested with real users in [35]. The MDP which results from the DIPPER system is described Fig. 1.

3) HIS: The second DM framework is the HIS system (Hidden Information State) developed by the University of Cambridge [23]. This system also implements a slot filling problem in a similar domain (tourist information), this time with 12 slots. More information about the venues is thus available and the user can ask for hotels and bars, choose the type of music, *etc*. In this article, experiments on the HIS system are conducted with an agenda-based user simulator [36]. The HIS system has also been successfully tested with real users in [37], [38]. In the experiments, the speech understanding error rate is set to 10% by using an error simulator. Some details about the MDP which results from the HIS system is described Fig. 1.(a whole description is provided in [39]).

B. Baseline

In order to illustrate the different aspects of the KTD framework, a comparison is made with some state-of-the-art algorithms. The two standard algorithms, *SARSA* and Q *-learning* [3] are used since they must be the most common algorithms in

RL. The SARSA algorithm illustrates the on-policy and online approach. Whereas the Q-learning, which is an off-policy algorithm, may be used in both online and offline approaches. For these two algorithms, one must choose a learning rate α.

The third baseline algorithm, *fitted*-Q[40], is based on approximate dynamic programming. The fitted-Q algorithm is an off-policy and batch algorithm and known to be sample efficient.

The fourth algorithm used as a baseline is *Least Square Policy Iteration* (LSPI) [41], [42]. This algorithm is also known to be sample-efficient. The representation of the Q-function for the function approximation needs here to be linear.

The last baseline is the *Gaussian Process Temporal Differences* (GPTD) algorithm [43], which is online and on-policy. In this Bayesian approach, the Q-function is modeled as a Gaussian process (that is, a set of jointly Gaussian random variables). The prior distribution of the process is entirely known by its mean and variance. The representation of the Q-function is non-parametric and linear. The representation is built iteratively using an online sparsification method [44]. GPTD has already been used for DM in [18] and [19].

C. Q-*Function Representation*

1) The DIPPER System: For all of the algorithms tested with the DIPPER system, the Q-function is represented using a linear parameterization. This parameterization is built by means of one Radial Basis Function (RBF) network per action (13 actions are available). Each RBF network has three equi-spaced Gaussian functions φ per dimension, each one with a standard deviation of $\sigma = 1/3$ (recall that state variables range from 0 to 1).

The corresponding feature vector Φ, defined for each $s \in S$ and each $a \in A$, is: $\Phi(s,a) = [\delta_{a,a_1}\phi^T(s), \ldots, \delta_{a,a_{13}}\phi^T(s)]$ with $\phi(s) = [1, \varphi_1(s), \varphi_2(s), \ldots, \varphi_{27}(s)]$ and δ the Kronecker delta. The φ vector is defined as follows: for $i \in 1 \ldots 27$: $\varphi_i(s) = \exp\left(-\|s - s_i\|^2/2\sigma^2\right)$, s_i being the i^{th} RBF center. Therefore, there are 364 (*i.e.*, $(3^3+1) \times 13$) parameters.

2) The HIS System: Two different parameterizations of the Q-function are used with the HIS system: a linear one based on an RBF network and a non-linear one based on an artificial neural network (ANN). All of the algorithms tested with the HIS system use the linear representation. The KTD framework is tested with an ANN. It has to be recalled here that KTD-Q is a second-order algorithm (so sample efficient) able to deal with a double non-linearity. A first non-linearity is introduced by the $\max$ operator in the optimality Bellman's equation. A second non-linearity is introduced with the ANN.

The first parameterization is based on an RBF network per action (12 actions are available). Three equi-spaced Gaussian functions φ are used in each continuous dimension to tile the 2D space spanned by s^1 and s^2. Each of them has a standard deviation $\sigma = \sqrt{0.2}$. The basis function Φ defined for each $s \in S$ and for each $a \in A$ is built as follow: $\Phi^T(s,a) = [\delta_{a,a_1}\phi^T(s), \ldots, \delta_{a,a_{12}}\phi^T(s)]$ with

$$\phi^T(s) = [1, \varphi_1^1(s^1,s^2), \ldots, \varphi_3^3(s^1,s^2), \delta_{s^3,0}, \ldots, \delta_{s^3,21}, \delta_{s^4,0}, \ldots, \delta_{s^4,5}]$$

and $\varphi_i^j(s^1,s^2) = \exp\left(-(\|s^1 - s_i\|^2 + \|s^2 - s_j\|^2)/2\sigma^2\right)$, (s_i, s_j) being the RBF centers equi-spaced in $[0\,,1] \times [0\,,1]$. The dimension of the feature vector which is equal to the number of parameters to learn is thus : $(1 + 9 + 22 + 6) \cdot 12 = 456$.

The second parameterization is a non-linear one based on an ANN. A neural network is considered since it can represent a complex function with a compact structure and a small number of parameters. We use an ANN with one hidden layer. The ANN is built as follows. The input vector $\mathrm{I}(s,a)$ has 42 components: the first two contain the values of s^1 and s^2 respectively, the next 22 are used to represent the s^3 value, the next 6 are used to represent the s^4 value, and the last 12 are used to represent the action a. The vector $\mathrm{H}(s,a)$ representing the values of the hidden neurons has 8 components (this number beeing chosen by the designer). The output vector $\mathrm{O}(s,a)$ has only one component since this neuron regresses the Q-function ($Q : S \times A \to \mathbb{R}$). The parameters of the representation are the synaptic weights between the different layers. The number of parameters to link the input layer to the hidden layer is $42 \cdot 8$ (each neuron of the input layer is linked to the ones of the hidden layer). The number of parameters to link the hidden layer to the output layer is $(8+1) \cdot 1$. A neuron which value is set to 1 is always added when using an ANN. The total number of parameters of this representation is thus $42 \cdot 8 + 9 = 345$. One advantage of using an ANN is its ability to deal with larger state spaces. For example, if we want to add a third confidence score associated with the third most probable dialogue history in the state space (see Section IV-A), the input vector of the ANN will only be increased by one. The new number of parameters would be thus $(42+1) \cdot 8 + (8+1) \cdot 1 = 353$ (only about 2.5% of parameters added). On the other hand, if we would add a continuous dimension in the state space while using an RBF network, three Gaussians would have to be added. Thus, the new number of parameters would be $((1+3^3)+22+6) \cdot 12 = 672$ (about 32% of parameters added). The larger the number of parameters, the more difficult the learning is. The HIS system has been tested with an increased state space in [19].

V. Results

This section using experimental results illustrates the advantages of employing KTD for dialogue optimization. Here, we study each of the problems outlined in the introduction (Section I).

A first important feature for an algorithm applied to dialogue management is its ability to learn in an uncontrolled way from a fixed set of data. This property is useful when logs of interactions have already been collected. In this case, the use of an off-policy approach is more interesting. Indeed, the optimal policy can direclty be learnt by processing data generated with a sub-optimal one. We compare the KTD-Q algorithm with LSPI and Q-learning while using the DIPPER and HIS systems. The comparison with fitted-Q is added in the DIPPER case. An on-policy approach could also be used with a fixed set of data but the approach is less interesting since the policy can only be evaluated (no better policy can be computed). This aspect is not presented in this article but is in [16].

The other interesting aspect a DM should have is the ability to learn a policy while being controlled. The answers of the user influence the learning of the policy: an improved policy is presented to the user each time an interaction takes place.

The learning can be on-policy and controlled: after each interaction, the policy presented to the user is improved until a stabilization on a good policy is reached. The KTD-SARSA algorithm is compared to SARSA for both DIPPER and HIS systems. A comparison with GPTD is added with the HIS system. Since the learning is controlled, it must be safe. Indeed the learnt policy is directly shown to the user. Algorithms which provide uncertainty information to lead safe exploration are thus of great interest. This aspect of the KTD-SARSA algorithm is shown with both the DIPPER and HIS systems.

Finally, the controlled setting can be used in an off-policy approach. The exploration should still be safe to show a good policy to the user. In an off-policy approach, the learnt policy (here the optimal one) is different from the one used for control. The KTD-Q algorithm is compared to Q-learning with the DIPPER and HIS systems. The choice of the on- or off-policy learning is in the designer's hands. An advantage could be that, if the learning stops for any reason, the off-policy approach would provide an estimate of the optimal policy whereas an on-policy approach would go on playing the same policy.

Since the sample efficiency aspect is an intrinsic property of the KTD framework, it is considered in each presented result.

A. Experimental Conditions

The values of the parameters under different experimental conditions are presented here. All the curves represent the mean discounted cumulative reward with respect to the size of the training datasets. There are two types of curves. The first type is where the learnt policy is tested. Here, the next action is chosen greedily with respect to the learnt Q-function. The second type of curve presents the mean discounted cumulative reward got during the learning of the control policy. A different exploration scheme can be adopted during this learning. The associated standard deviations are added to the results. Because of implementation constraints, results are shown with respect to a number of samples for the DIPPER system while they are presented with respect to the number of dialogues for the HIS system. A sample is a dialogue turn between the user and the system. In the case of the HIS system, for sake of comparison, the number of samples can be deduced from the number of dialogues. Indeed, the reward depends on the length of the dialogue. For example, if the mean discounted cumulative reward is 14 for a set of 50 training dialogues, since the maximum reward that can be obtained is 20 and that each turn a penalty of -1 is given, the average length of the dialogue is $20 - 14 = 6$ turns. The number of samples is thus approximately $6 \cdot 50 = 300$. For the sake of readability of the figures, the conversion into number of samples has not been done.

1) The DIPPER System: The learning rate α for the Q-learning and SARSA algorithms is chosen to be 0.2 and divided by two after every 10^4 interactions (similarly to what is done in [45]). For all the algorithms, the initial parameter vector θ_0 is set to 0.001. For KTD, $\eta = 0$, $P_{n_t} = 1$ and $P_{0|0} = 1 \cdot \mathrm{Id}$. When an ϵ-greedy scheme is used for exploration, ϵ is set to 0.1. When a bonus-greedy exploration is used, $\beta = 5$ and $\beta_0 = 1$. As far as the curves are considered, each algorithm is trained 8 times and each obtained policy is tested 50 times.

2) The HIS System: The learning rate α is set to 0.1 for Q-learning and SARSA. For all the algorithms, the initial parameter vector θ_0 is set to 0. For KTD, $\eta = 0.001$, $P_{n_t} = 0.1$ and $P_{0|0} = 1 \cdot \mathrm{Id}$. When an ϵ-greedy scheme is used for exploration, ϵ is set to 0.1. When a bonus-greedy exploration is used, $\beta = 1000$ and $\beta_0 = 100$. As far as the curves are considered, the Q-learning and SARSA are trained 250 times and LSPI, KTD, and GPTD are trained 50 times. The number of trials is different because more time is needed to compute solutions with LSPI, KTD and GPTD since there are matrix inversions. Each obtained policy is tested 1000 times.

For both systems, the parameters have been found by trials and errors. They may not be the best ones but they gave consistent results.

B. Batch Learning in an Off-Policy Approach

In this section, we present the ability of KTD-Q to learn in a batch and off-policy fashion. The optimal policy is learnt while observing data generated with a sub-optimal policy. We will call this policy the "*behavioral policy*". This approach is similar to the ones used in [14]–[16].

For both systems, the behavioral policy is recalled on the graphs. The policy is not provided as a baseline but only to show that there is no need for a good initial policy. The behavioral policy must nevertheless sufficiently explore the state-space. Indeed, the off-policy algorithms cannot infer a good policy in situations which have not been encountered in the training set.

A first experiment is done with the **DIPPER** system and a n-gram user model. A similar experiment has been done in [46]. The behavioral policy is a handcrafted policy aiming at filling each slot by explicitly asking before closing the dialogue combined with random actions. With a probability ϵ, the system acts randomly, and with a probability $1-\epsilon$, the system acts according to the specified policy. Random actions are chosen among "reasonable actions": a set of hand-crafted rules prevents the DM from making inconsistent decisions. For example, a rule prevents the DM from confirming a slot while it has never been asked for before. Random choices ensure the state-action space to be sufficiently explored.

Results are shown in Fig. 2 (abscissa in logarithmic scale). No standard deviation is plotted for the behavioral policy for sake of legibility since the variance was really high. For the other curves, when the average cumulative reward is around 0, that means that the agent did not learn to stop the dialogue. Therefore the associated standard deviation is low. Fitted-Q and LSPI curves start only at $5 \cdot 10^3$ samples: these algorithms require inverting some matrices which are too badly conditioned below a few thousands of samples (knowing that we have a few hundreds of parameters). A common method to avoid poor conditioned matrix is to introduce an L_2 regularization. The method has been tested without any success on this particular problem: the learnt policies return poor results. That is why results are not reported here. Q-learning learns very slowly. The best learnt policy is not even as good as the hand-coded one after $5 \cdot 10^4$ interactions. This is compliant with the literature [45].

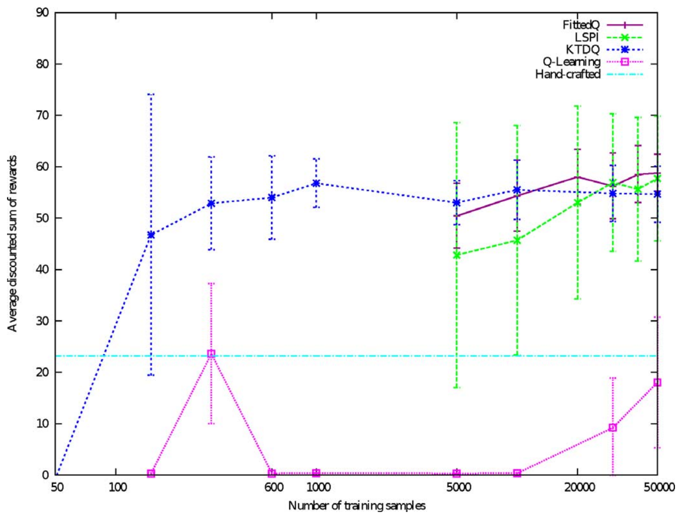

Fig. 2. [DIPPER] Results of KTD-Q (and other algorithms), data being sampled according to some handcrafted policy (no control).

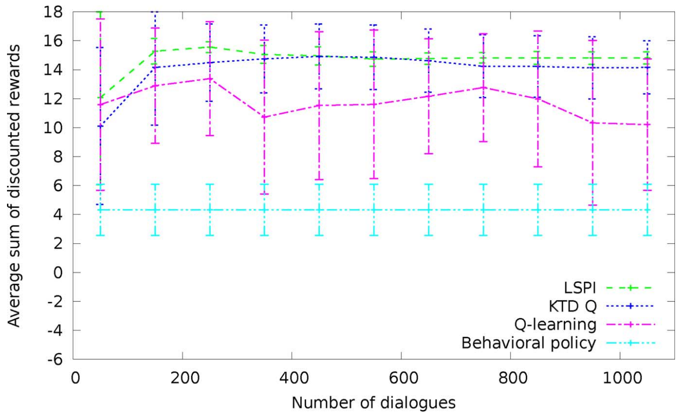

Fig. 3. [HIS] Results of KTD-Q compared to LSPI and Q-learning when a set of fixed data is provided (no control).

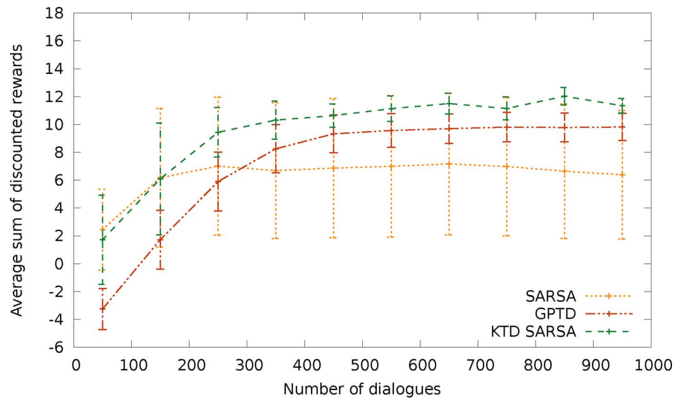

Fig. 4. [HIS] Results of KTD-SARSA compared to GPTD and SARSA during the learning of the policy (controlled case).

With the DIPPER system, the KTD-Q algorithm learns good policies after only a few hundreds of transitions and in a stable manner (variance is decreasing). Another interesting aspect is its ability to provide good policies when there are not enough samples for fitted-Q or LSPI to learn a good policy. The corresponding amount of dialogues can be easily collected in real applications to learn optimal policies with KTD-Q. Also, the variance is much lower with KTD after 5000 turns than for LSPI meaning that results are more reproducible even if the mean is a little bit lower with the KTD algorithm. The off-policy and batch approach has also been tested with the **HIS** system. The data of the behavioral policy have been collected while learning policies with a SARSA algorithm with an ϵ-greedy exploration ($\epsilon = 0.3$ to ensure a good exploration of the state space). The results are presented in Fig. 3. For each of the training phases, the algorithms have used the same data. Both LSPI and KTD-Q give satisfying results. With the HIS, the regularized version of LSPI gives satisfying results even when the size of the training data set is not large. All the more, one has to pay attention while comparing the Figs. 2 and 3. The first point of the LSPI curve of Fig. 3 corresponds of a training set of 50 dialogues which average length are around 15 turns. The number of training samples is thus $50 \cdot 15 = 750$ (for a number of parameters to learn around 350). In Fig. 2, the first point of the LSPI curve (pay attention, log scale) is plotted for 5000 samples.

With few samples, the learnt optimal policy is good. The results given by the Q-learning algorithm are not so bad but not as good as those returned by KTD-Q and LSPI. It has to be noticed that the learning seems to be easier with the HIS system since the state space is carefully tuned: the summary state space is quite informative.

To conclude this subsection, the KTD-Q algorithm gives consistent results with both systems. The variance seems to be higher in the KTD-Q case with the HIS system. The fact that the data are processed several times with fitted-Q and LSPI can be mentioned to explain the differences between the algorithms. Indeed, each data sample is only processed one time with KTD since it is an online algorithm.

C. *Online Learning in an On-Policy Approach: The Need to Deal With Safe Exploration*

In this section, we compare KTD-based algorithms when used in an on-policy and online setting. We present first a comparison of online and on-policy algorithms with a classical ϵ-greedy exploration used for the control. KTD-SARSA is compared with SARSA and GPTD while using the **HIS** system. First, the parameterization used for the Q-function is linear (see Section IV-C). Results are shown in Fig. 4. For these curves, each point is the result of an average made with a sliding window of 100 points width. The results obtained with KTD-SARSA are better of about 1.5 turn on average than those got with GPTD. The results with SARSA are worst. Two explanations arise to understand the fact that KTD-SARSA gives better results than GPTD. The first reason can be the differences in the Q-function representation. The second reason can be that the KTD-SARSA is able to deal with non-stationarity by means of the evolution noise. Indeed, since we are in a controlled case, the current policy varies which introduce non-stationarity. KTD-SARSA tracks the solution while GPTD tries to converge to a fixed one.

Since the KTD-SARSA algorithm is able to handle non-linearities, the on-policy and online approach is tested with the ANN described Section IV-C. Results are shown in Fig. 5. The different runs used to compute the mean are added on the graph. They show the detailed behavior of the ANN. Contrary to the linear case, the learning is either very good or very poor. The

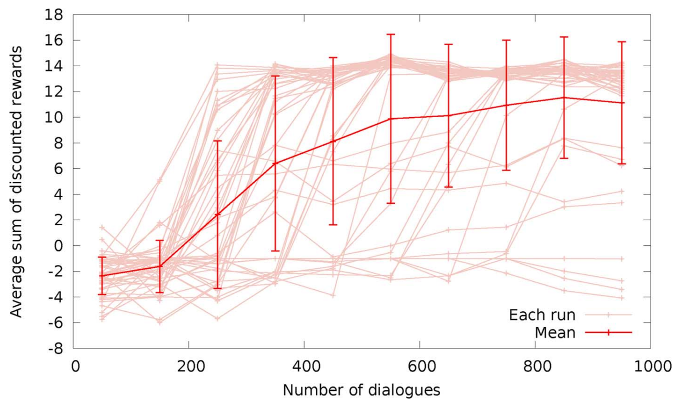

Fig. 5. [HIS] Results of KTD-SARSA with a non-linear representation of the Q-function (controlled case).

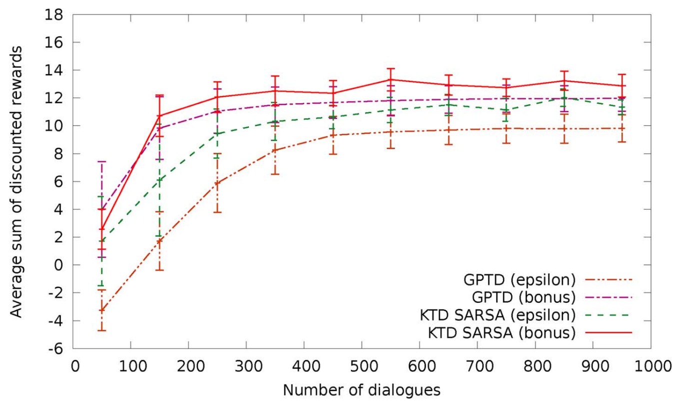

Fig. 7. [HIS] Results of KTD-SARSA compared to GPTD when different exploration schemes are used during the learning.

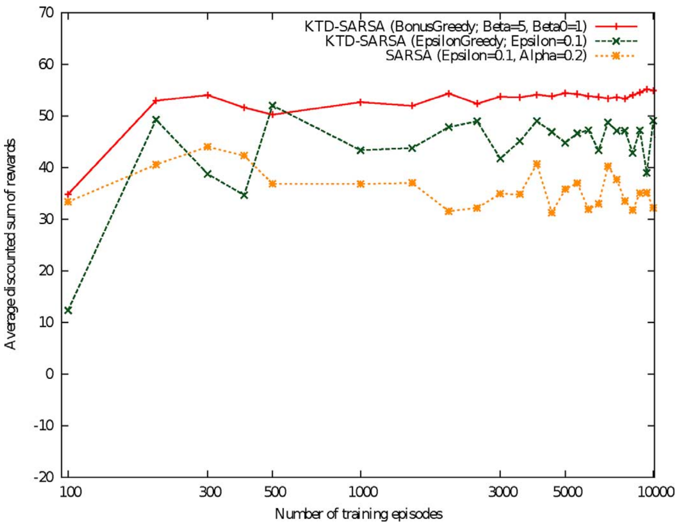

Fig. 6. [DIPPER] Results of KTD-SARSA compared to SARSA when two exploration schemes are tested.

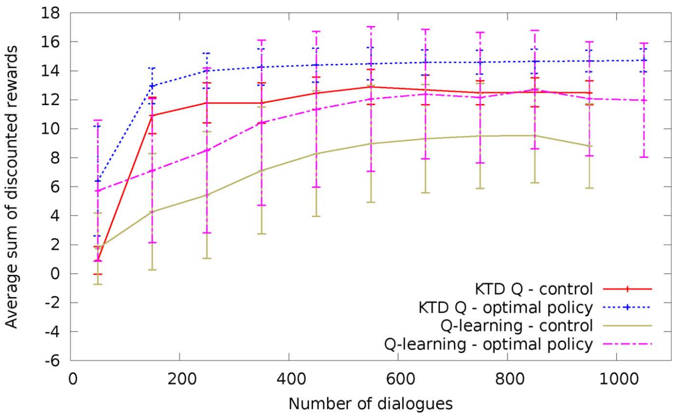

Fig. 8. [HIS] Results of KTD-Q compared to Q-learning, data being sampled according to some behavioral policy (controlled case).

curve representing the average results presents thus poor results with small sets of data. Anyway, after a reasonable number of episodes, the non-linear parameterization provides as good results as those got with the RBF parameterization.

Here, all the changes in the learnt policy are visible to the user and might be annoying. This is why a safe exploration scheme should be used. An uncertainty on the quality of the estimation is provided by KTD so it is used to lead a safe exploration. The bonus-greedy scheme is compared to a standard ϵ-greedy one (see Section III).

For the **DIPPER** system, the results are shown in Fig. 6. The SARSA algorithm using an ϵ-greedy exploration is recalled on the graph. The KTD-SARSA algorithm is more sample efficient than SARSA meaning that the user is less disturbed by the exploration. Among the exploration schemes reported, only one can be said "safe". Indeed, the ϵ-greedy policy always explores. Therefore, the bonus-greedy policy performs better and is more stable after the learning has converged.

Different exploration schemes are also compared for the **HIS** system. Since the GPTD is also able to deal with safe exploration (variance information is provided), the results obtained with this algorithm are also plotted on Fig. 7. When a safe exploration is led, the learning is of better quality. Indeed, in both cases (KTD-SARSA and GPTD) the average of the mean discounted cumulative reward got during the learning with safe exploration is higher. The dialogue is shortened by 2 turns on average and KTD still outperforms GPTD.

To conclude this subsection, with both systems when safe exploration and the KTD framework are used, the performance is improved.

D. *Online Learning in an Off-Policy Approach*

In this section, dialogue optimization is performed using KTD based algorithms in a online and off-policy setting. Here for control, the bonus-greedy method based on the estimated Q-function is employed. This guarantees safe user experience. The system is tested with both linear and non-linear parameterizations.

Fig. 8 shows the performance of the system while using a linear parameterization. As a baseline, the performance of the Q-learning algorithm is given. The GPTD results do not appear on the graph since the off-policy approach is not handled by this algorithm. Each point on the curves which represents the control policy is the result of an average made with a sliding window of 100 points width. This explains the fact that one point is missing on these curves. At the end of the learning, the two KTD-Q curves should converge towards the same value. The convergence is not reached (training set too small), probably because the exploration is not yet zero.

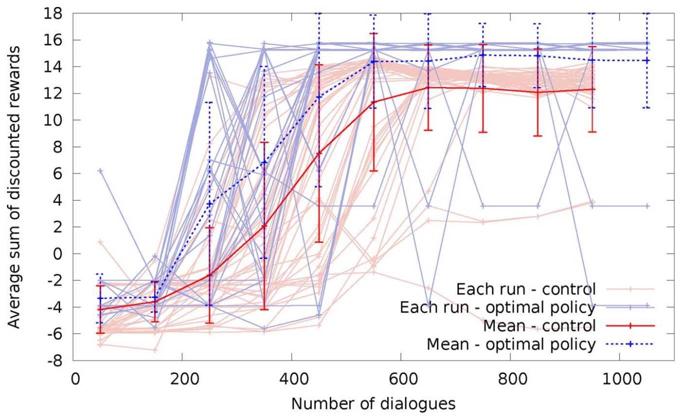

Fig. 9. [HIS] Results of KTD-Q with a non-linear representation of the Q-function (controlled case).

Since KTD is able to handle non-linearities, the online learning in an off-policy and controlled approach is tested with the ANN described Section IV-C. Results are given in Fig. 9. The curves are plotted the same way as in Fig. 8. As discussed earlier, for the Fig. 5, the learning is either very good or very poor. The number of runs which give poor results is not large. If we compare the two controlled approaches (Figs. 5 and 9), the KTD-Q curve presents better results than KTD-SARSA. KTD-Q using a linear (Fig. 8) and KTD-Q using a non-linear parameterization (Fig. 9) with training sets of data big enough give roughly the same results.

E. Discussion About the Non-Linear Parameterization

The results of the KTD framework tested with a non-linear parameterization have been presented in Figs. 5 and 9. Each run has been plotted on the figures in order to show precisely the behavior of the ANN-based KTD. For a single run, the policy is either very poor or very good. The moment where the learning switches and becomes good is not precisely defined. This is typical of non-linear learning. It must be underlined that KTD-Q has to deal with both the non-linearity introduced by the optimality of Bellman's operator and the non-linearity of the neural network.

With a reasonable number of dialogues, the results are as good as those obtained with a linear parameterization. The performance is not significantly improved but this result is nevertheless interesting. Indeed, good results are not guaranteed (if not impossible) when a nonlinear parameterization is used in combination with other algorithms. Even if several examples are provided with satisfactory results, the most famous being the TD-Gammon presented in [47], the convergence of TD algorithms is not guaranteed as shown in [24]. Moreover, the KTD framework deals with second order algorithms. To our knowledge, KTD is the only second order algorithm able to deal with nonlinear value function approximation, apart [48].

Another advantage of using ANNs is that they can represent richer function spaces with less parameters and could be easier to design than RBF networks. This aspect has been presented in [19].

VI. Conclusion

The goal of this paper is to provide with a unified framework to address most of the aspects of the dialogue management optimization problem. In this context, we proposed to use the Kalman Temporal Differences framework. This framework allows learning efficiently in an off-policy manner: fixed batches of data can thus be used to learn a strategy without having to disturb users while learning. It also allows learning in an online and controlled manner. Thus, starting from a good policy learnt off-line, improvement can be learnt online with real users. To minimize the impact on the user experience, smart exploration schemes, based on uncertainty management, are made possible. Finally, scalability properties are promising. Indeed, non-linear parameterizations are handled by the KTD framework, which makes compact representations of the Q-function possible. This aspect requires more work to determine automatically trials leading to poor results. The results seem promising since the performance obtained with the unified KTD framework compares favorably to the one obtained with each of the different state-of-the-art algorithms able to deal with one single property at once. Experiments were made in quite different settings. A good clue is thus given about the ability of the algorithms to adapt to various conditions, among which interactions with real users.

Acknowledgment

The authors would like to thank S. Young, M. Gašić and the Cambridge Dialogue Systems Group for their help in using the CamInfo system.

References

[1] E. Levin, R. Pieraccini, and W. Eckert, "Learning dialogue strategies within the Markov decision process framework," in *Proc. Autom. Speech Recognit. Understand. Workshop (ASRU'97)*, 1997.
[2] R. Bellman, "A Markovian Decision Process," *J. Math. Mech.*, vol. 6, pp. 679–684, 1957.
[3] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An introduction*. Cambridge, MA: MIT Press, 1998.
[4] E. Levin and R. Pieraccini, "Using Markov decision process for learning dialogue strategies," in *Proc. Int. Conf. Accoust., Speech, Signal Process. (ICASSP'98)*, 1998, pp. 201–204.
[5] E. Levin, R. Pieraccini, and W. Eckert, "A stochastic model of human-machine interaction for learning dialog strategies," *IEEE Trans. Speech Audio Process.*, vol. 8, no. 1, pp. 11–23, Jan. 2000.
[6] W. Eckert, E. Levin, and R. Pieraccini, "User modeling for spoken dialogue system evaluation," in *Proc. Autom. Speech Recognit. Understand. Workshop (ASRU'97)*, 1997.
[7] O. Pietquin and T. Dutoit, "A probabilistic framework for dialog simulation and optimal strategy learning," *IEEE Trans. Speech Audio Process.*, vol. 14, no. 2, pp. 589–599, Mar. 2006.
[8] J. Schatzmann, K. Weilhammer, M. Stuttle, and S. Young, "A survey of statistical user simulation techniques for RL of dialogue management strategies," *Knowl. Eng. Rev.*, vol. 21, no. 2, pp. 97–126, 2006.
[9] J. Schatzmann, M. N. Stuttle, K. Weilhammer, and S. Young, "Effects of the user model on simulation-based learning of dialogue strategies," in *Proc. Autom. Speech Recognit. Understand. Workshop (ASRU'05)*, 2005.
[10] O. Pietquin and H. Hastie, "A survey on metrics for the evaluation of user simulations," *Knowledge Eng. Rev.*, 2011.
[11] V. Rieser and O. Lemon, *Reinforcement learning for adaptive dialogue systems: A data-driven methodology for dialogue management and natural language generation*. New York: Springer, 2011, Theory and Applications of Natural Language Processing.
[12] S. Singh, M. Kearns, D. Litman, and M. Walker, "Reinforcement learning for spoken dialogue systems," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS'99)*, 1999.

[13] R. Bellman*, Dynamic Programming*, 6th ed. New York: Dover, 1957.

[14] L. Li, S. Balakrishnan, and J. Williams, "Reinforcement learning for dialog management using least-squares policy iteration and fast feature selection," in *Proc. Interspeech'09*, 2009.

[15] O. Pietquin, M. Geist, S. Chandramohan, and H. Frezza-Buet, "Sample-efficient batch reinforcement learning for dialogue management optimization," *ACM Trans. Speech Audio Process.*, vol. 7, no. 3, pp. 1–21, 2011.

[16] J. Henderson, O. Lemon, and K. Georgila, "Hybrid reinforcement/supervised learning of dialogue policies from fixed data sets," *Comput. Linguist.*, vol. 34, no. 4, pp. 487–511, 2008.

[17] M. Geist and O. Pietquin, "Managing uncertainty within the KTD framework," in *Proc. AL&E Workshop*, 2011, Journal of Machine Learning Research C& WP.

[18] M. Gašić, F. Jurčíček, S. Keizer, F. Mairesse, B. Thomson, K. Yu, and S. Young, "Gaussian processes for fast policy optimisation of POMDP-based dialogue managers," in *Proc. SIGdial'10*, 2010.

[19] L. Daubigney, M. Geist, and O. Pietquin, "Off-policy learning in large-scale POMDP-based dialogue systems," in *Proc. Int. Conf. Acoust., Speech Signal Process. (ICASSP'12)*, 2012, pp. 4989–4992.

[20] F. Jurčíček, B. Thomson, S. Keizer, M. Gašić, F. Mairesse, K. Yu, and S. Young, "Natural belief-critic: A reinforcement algorithm for parameter estimation in statistical spoken dialogue systems," in *Proc. Interspeech'10*, 2010.

[21] M. Geist and O. Pietquin, "Kalman temporal differences," *J. Artif. Intell. Res. (JAIR)*, vol. 39, pp. 483–532, 2010.

[22] J. Bos, E. Klein, O. Lemon, and T. Oka, "DIPPER: Description and formalisation of an information-state update dialogue system architecture," in *Proc. SIGdial'03*, 2003.

[23] S. Young, J. Schatzmann, B. Thomson, H. Ye, and K. Weilhammer, "The HIS dialogue manager," in *Proc. IEEE/ACL Workshop Spoken Lang. Technol. (SLT'06)*, 2006.

[24] J. Tsitsiklis and B. Van Roy, "An analysis of temporal-difference learning with function approximation," *IEEE Trans. Autom. Control*, vol. 42, no. 5, pp. 674–690, May 1997.

[25] S. Png and J. Pineau, "Bayesian reinforcement learning for POMDP-based dialogue systems," in *Proc. Int. Conf. Acoust., Speech, Signal Process. (ICASSP'11)*, 2011, pp. 2156–2159.

[26] S. Larsson and D. R. Traum, "Information state and dialogue management in the trindi dialogue move engine toolkit," *Natural Lang. Eng.*, vol. 6, pp. 323–340, 2000.

[27] J. Williams and S. Young, "Scaling up POMDPs for dialogue management: The summary POMDP method," in *Proc. Autom. Speech Recognit. Understanding Workshop (ASRU'05)*, 2005.

[28] M. Walker, D. Litman, C. Kamm, and A. Abella, "PARADISE: A framework for evaluating spoken dialogue agents," in *Proc. Meeting Assoc. Comput. Linguist. (ACL'97)*, 1997.

[29] A. Boularias, H. Chinaei, and B. Chaib-draa, "Learning the reward model of dialogue POMDPs from data," in *Proc. NIPS Workshop of Mach. Learn. for Assistive Tech.*, 2010.

[30] L. E. Asri, R. Laroche, and O. Pietquin, "Reward function learning for dialogue management," in *Proc. Starting Artif. Intell. Res. Symp. (STAIRS'12)*, 2012, pp. 95–106.

[31] R. Kalman, "A new approach to linear filtering and prediction problems," *Trans. ASME–J. Basic Eng.*, vol. 82, no. Series D, pp. 35–45, 1960.

[32] E. Wan and R. Van Der Merwe, "The unscented Kalman filter for nonlinear estimation," in *Adaptive Syst. for Signal Process., Commun., Control Symp. (AS-SPCC'00)*, 2000, pp. 153–158.

[33] L. Daubigney, M. Gašić, S. Chandramohan, M. Geist, O. Pietquin, and S. Young, "Uncertainty management for on-line optimisation of a POMDP-based large-scale spoken dialogue system," in *Proc. Interspeech'11*, 2011.

[34] K. Georgila, J. Henderson, and O. Lemon, "Learning user simulations for information state update dialogue systems," in *Proc. Eur. Conf. Speech Commun. Technol. (Interspeech – Eurospeech'05)*, 2005.

[35] V. Rieser, S. Keizer, X. Liu, and O. Lemon, "Adaptive information presentation for spoken dialogue systems: Evaluation with human subjects," in *Proc. Eur. Workshop Natural Lang. Generat. (ENLG'11)*, 2011.

[36] J. Schatzmann, B. Thomson, K. Weilhammer, H. Ye, and S. Young, "Agenda-based user simulation for bootstrapping a POMDP dialogue system," in *Proc. NAACL Conf. Human Lang. Technol. (HLT/NAACL'07)*, 2007.

[37] F. Jurčíček, S. Keizer, M. Gašić, F. Mairesse, B. Thomson, K. Yu, and S. Young, "Real user evaluation of spoken dialogue systems using Amazon Mechanical Turk," in *Proc. Interspeech'11*, 2011.

[38] M. Gašić, F. Jurčíček, B. Thomson, K. Yu, and S. Young, "On-line policy optimisation of spoken dialogue systems via live interaction with human subjects," in *Proc. Autom. Speech Recognit. Understand. Workshop (ASRU'11)*, 2011, pp. 312–317.

[39] S. Young, M. Gašić, S. Keizer, F. Mairesse, J. Schatzmann, B. Thomson, and K. Yu, "The hidden information state model: A practical framework for POMDP-based spoken dialogue management," *Comput. Speech Lang.*, vol. 24, no. 2, pp. 150–174, 2010.

[40] G. Gordon, "Stable function approximation in dynamic programming," in *Proc. Int. Conf. Mach. Learn. (ICML'95)*, 1995.

[41] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *J. Mach. Learn. Res. (JMLR)*, vol. 4, pp. 1107–1149, 2003.

[42] L. Li, M. L. Littman, and C. R. Mansley, "Online exploration in least-squares policy iteration," in *Proc. Int. Conf. Autonom. Agents Multiagent Syst. (AAMAS'09)*, 2009, vol. 2, pp. 733–739.

[43] Y. Engel, S. Mannor, and R. Meir, "Reinforcement learning with Gaussian processes," in *Proc. Int. Conf. Mach. Learn. (ICML'05)*, 2005.

[44] Y. Engel, S. Mannor, and R. Meir, "Sparse online greedy support vector regression," in *Proc. Eur. Conf. Mach. Learn. (ECML'02*, 2002, vol. 2430, pp. 84–96.

[45] O. Lemon, K. Georgila, J. Henderson, and M. Stuttle, "An ISU dialogue system exhibiting reinforcement learning of dialogue policies: Generic slot-filling in the TALK in-car system," in *Proc. Conf. Eur. Chapter Assoc. for Comput. Linguist. (EACL'06)*, 2006, pp. 119–122.

[46] O. Pietquin, M. Geist, and S. Chandramohan, "Sample efficient on-line learning of optimal dialogue policies with Kalman temporal differences," in *Proc. Int. Joint Conf. Artif. Intell. (IJCAI'11)*, 2011.

[47] G. Tesauro, "Temporal difference learning and TD-Gammon," *Commun. Assoc. for Comput. Mach. (ACM)*, vol. 38, no. 3, pp. 58–68, 1995.

[48] M. Geist and O. Pietquin, "Statistically linearized least-squares temporal differences," in *Proc. IEEE Int. Conf. Ultra Modern Control Syst. (ICUMT'10)*, 2010, pp. 450–457.

Lucie Daubigney obtained her degree of Electrical Engineering from the Phelma (Physics, Applied physics, Electronics and Materials Science) engineering school of the Grenoble Institute of Technology in 2010. Then, she joined in 2010 the Machine Learning and Interactive Systems research group of Supélec and the MAIA project-team of INRIA to prepare a PhD degree. She is interested in machine learning, dialogue management and e-learning.

Matthieu Geist obtained an Electrical Engineering degree and an Msc degree in Mathematics from Supélec (France), both in September 2006, as well as a PhD degree in Mathematics from the "Université Paul Verlaine de Metz" (France) in November 2009. From January 2007 to January 2010, he was a member of the Measure and Control lab (MC cluster) of ArcelorMittal Research and a member of the CORIDA project-team of INRIA. In February 2010, he joined the IMS-MaLIS research group of Supélec as an assistant professor. His research interests include statistical machine learning (especially reinforcement learning), as well as applications to spoken dialogue systems. He authored or co-authored more than 50 publications in these fields.

Senthilkumar Chandramohan received his Bachelor of Engineering (Electronics and Communication) degree from the Bharathiar University (Coimbatore, India) in 2004. Following which he worked as a software engineer for Torry Harris Business Solutions (Bangalore, India) from June 2004 until June 2006. Between 2006 and 2008, he pursued European Masters in Informatics (EuMI): a dual degree program resulting in a Master of Science (Informatics) degree from the University of Edinburgh (Scotland) and Master of Science (Media

Informatics) degree from RWTH Aachen (Germany). He was a recipient of European Commission's Erasmus Mundus scholarship (2006/08). Since early 2009, he has been affiliated with the IMS-MaLIS research group at the Metz campus of Supélec. He started his doctoral research (in conjunction with LIA, University d'Avignon, France) in October 2009. He obtained his PhD (Computer Science) in September 2012. His research focuses on statistical approaches for spoken dialogue optimization and user modeling.

Olivier Pietquin (M'01–SM'11) obtained an Electrical Engineering degree from the Faculty of Engineering, Mons (FPMs, Belgium) in June 1999 and a PhD degree in April 2004. In 2011, he received the Habilitationà Diriger des Recherches (French Tenure) from the University Paul Sabatier (Toulouse, France). He joined the FPMs Signal Processing department (TCTS Lab.) in September 1999. In 2001, he has been a visiting researcher at the Speech and Hearing lab of the University of Sheffield (UK). Between 2004 and 2005, he was a Marie-Curie Fellow at the Philips Research lab in Aachen (Germany). Now he is a Professor at the Metz campus of the Ecole Supérieure d'Electricité (Supélec, France). He has also been a full member of the UMI 2958 (joint lab with GeorgiaTech and CNRS) since 2010 where he coordinates the computer science departments and heads the Machine Learning and Interactive Systems group. From 2007 to 2011, he was also a member of the IADI INSERM research team (in biomedical signal processing). Since 2010, Olivier Pietquin has sat on the IEEE Speech and Language Technical Committee. His research interests include spoken dialog systems evaluation, simulation and automatic optimization, machine learning, speech and signal processing. He authored or co-authored more than 100 publications in these domains.