Gaussian Processes for POMDP-Based Dialogue Manager Optimization

Milica Gašić, *Member, IEEE*, and Steve Young, *Fellow, IEEE*

***Abstract*—A partially observable Markov decision process (POMDP) has been proposed as a dialog model that enables automatic optimization of the dialog policy and provides robustness to speech understanding errors. Various approximations allow such a model to be used for building real-world dialog systems. However, they require a large number of dialogs to train the dialog policy and hence they typically rely on the availability of a user simulator. They also require significant designer effort to hand-craft the policy representation. We investigate the use of Gaussian processes (GPs) in policy modeling to overcome these problems. We show that GP policy optimization can be implemented for a real world POMDP dialog manager, and in particular: 1) we examine different formulations of a GP policy to minimize variability in the learning process; 2) we find that the use of GP increases the learning rate by an order of magnitude thereby allowing learning by direct interaction with human users; and 3) we demonstrate that designer effort can be substantially reduced by basing the policy directly on the full belief space thereby avoiding ad hoc feature space modeling. Overall, the GP approach represents an important step forward towards fully automatic dialog policy optimization in real world systems.**

***Index Terms*—Gaussian process, POMDP, statistical dialog systems.**

I. Introduction

Spoken dialog systems enable human-computer interaction where the primary input is speech. As such they have innumerable benefits. However, building such systems to operate robustly is challenging as they are very sensitive to speech recognition errors and require designer effort to define their behavior. The focus of this paper is automatic dialog optimization using a small amount of training data.

A partially observable Markov decision process (POMDP) has been proposed as a dialog model which intrinsically deals with uncertainty from the recognizer, providing more robust system operation [1]–[5]. It assumes that the dialog state s_t is only partially observable and depends on a noisy observation o_t. Since the dialog state is unobservable, at every dialog step a distribution over all states is maintained, which is called the *belief state* b_t. It takes values $\mathbf{b} \in \mathcal{B}$, where $\mathcal{B}$ is a continuous space of dimensionality $|\mathcal{S}|$, namely $[0,1]^{|\mathcal{S}|}$. The dialog policy π then maps the belief state $\mathbf{b}$ into an action a at every dialog turn, $\pi(\mathbf{b}) = a$. In each state, the system receives a reward r_t, which is a measure of how good the system action is for that particular state. The aim is then to find a policy such that the total accumulated reward at the end of the dialog is maximized, which is measured by the Q-function.

The process of exact belief state updating becomes intractable for very large state spaces. However, there exist real-world dialog systems based on the POMDP model which maintain an approximate belief state distribution in real-time throughout the dialog. These include the Hidden Information State (HIS) system [6] and the Bayesian Update of Dialogue State (BUDS) system [5]. Policy optimization, however, involves a large amount of training data and requires the designer to explicitly define the policy representation [7], [8]. This paper explores ways of overcoming these limitations.

II. POMDP Policy Optimization

Exact policy optimization for POMDPs is intractable for everything but very simple cases [9]. Approximate POMDP solutions, as in the point-based value iteration algorithm [10], [11], are only suitable for relatively small state/action problems. However, a POMDP with discrete state and observation spaces can be viewed as an MDP with a continuous state space [9]. This allows standard MDP algorithms to be used for policy optimization and this usually introduces the need for some form of parametrization.

For example, it is proposed in [12] that the Q-function is parametrized as a linear combination of features from the belief state. Optimization is then performed using a modified Sarsa algorithm. In the BUDS system, the policy is parametrized as a softmax function of features of the belief state [13]. Then, gradient methods and the natural-actor critic algorithm are used for policy optimization. While this achieves tractability, in the case of the BUDS system it requires from 10^5 to 10^6 dialogs to train a policy, and this is only possible in interaction with a simulated user [8]. Moreover, the basis feature functions must be chosen by the designer and the solution is only optimal within the chosen basis. In [14], an algorithm is proposed that automatically selects useful features from an initial set of features predefined by a designer. An alternative approach is taken in [15] where Krylov iteration for lossless compression is used to compress the dialog states, however the effectiveness of this approach in practice greatly depends on the definition of the reward function. A more recent thread of research has focused on parametric models of the Q-function where the uncertainty of the approximation can be encoded in the estimate of the parameters. Here, a Gaussian distribution is placed over each parameter and Kalman filters are used for optimization. It is shown

Manuscript received January 17, 2013; revised September 10, 2013; accepted September 11, 2013. Date of publication September 16, 2013; date of current version November 13, 2013. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. James Glass.

The authors are with the Department of Engineering, University of Cambridge, Cambridge, CB2 1PZ, U.K. (e-mail: mg436@eng.cam.ac.u; sjy@eng.cam.ac.uk).

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/TASL.2013.2282190

that this leads to increased speed of training, however it still requires a set of basis functions [16].

In the HIS system [6], the belief space is heuristically mapped into a much smaller summary space, which is then discretized into a grid, allowing the Monte Carlo control algorithm to be used for the policy optimization. This optimization requires 10^5 dialogs, which is still too large for the policy to be optimized in direct interaction with humans and the forced use of a summary space again raises optimality questions.

Our aim is to reduce designer effort in defining a suitable policy representation and to speed up the process of policy optimization to avoid reliance on user simulators. For this purpose, we propose non-parametric policy modeling which achieves operational efficiency by exploiting the similarities between different parts of the belief space. For instance, if the system action is reliably estimated in one belief state, that estimate should contribute to the unexplored parts of the belief space. In addition, it is not only important to estimate the system action for a given belief state, but also to provide a measure of how confident the system is about that estimate. This information is a useful indication of the extent to which different parts of the belief space have been explored during policy optimization. A Gaussian process (GP) is proposed because it is a non-parametric model of Bayesian inference that has these desirable properties.

The next section introduces non-parametric policy modeling. It explains how the Q-function can be modeled as a Gaussian process and how a stochastic policy can be derived from this model that is well-suited for on-line learning. Then, in Section IV, experimental results using the BUDS dialog manager are presented. Several issues concerning policy design are examined with the aim of reducing variability in the learning process. The resulting GP-based policy is shown to learn faster than a standard method and this is confirmed in a human evaluation using the Amazon MTurk crowd-sourcing service [17]. A similar training scheme is conducted in interaction with humans via the Amazon MTurk service and is shown to perform significantly better than a simulator trained policy. The final set of experiments discuss policy training in the full belief space where it is shown that the GP approach will scale to handle the full space obviating the need for hand-crafted summary space mapping. Section V presents conclusions and directions for future work.

III. Non-Parametric Policy Modelling

Non-parametric policy modeling avoids the limitations that occur when the solution is constrained by the chosen basis. This does not necessarily mean that it is parameter-free, but rather that the solution is not restricted by the choice of parameters. Instead, the choice of parameters only influences the speed at which the optimal solution is found.

A Gaussian process is a non-parametric Bayesian model used for function approximation. It has been successfully applied to reinforcement learning for continuous-space MDPs [18]–[21]. An advantage of Bayesian approaches to policy optimization is that they offer a principled way of incorporating prior knowledge about the underlying task and this provides the potential to improve the learning rate. As already noted, it is important that the dependencies of different parts of the belief state space are taken into consideration during learning. Gaussian processes are able to incorporate the knowledge of these dependencies elegantly through the choice of a *kernel function*, the purpose of which is to describe correlations in different parts of the space. In addition, Gaussian processes can provide the uncertainty of the approximation by estimating the variance of the posterior.

In the GP approach, the size of the state space only impacts the evaluation of the kernel. Hence, it is not sensitive to the size of the POMDP summary space and indeed it can be used to model correlations directly in the full belief space. In this paper, the properties of the GP approach are evaluated in summary space in order to allow comparison with conventional gradient methods, but preliminary results using GP in the full belief space are also given.

This section provides an overview of the Gaussian process model of the Q-function based on the description given in [19]. Several policy formulations derived from the GP model are presented which are suitable for dialog management. Issues of computational complexity and sparse approximation solutions are also discussed.

A. Gaussian Process Model for the Q-Function

This section describes how the Q-function can be modeled as a Gaussian process. As outlined in the introduction, a discrete-space POMDP can be viewed as a continuous-space MDP. In dialog management, however, this space is often reduced to a summary space that contains both continuous and discrete variables [6]. Therefore, in the most general case, the approximation framework needs to support modeling of the Q-function in a multidimensional space that consists of both continuous and discrete variables. A Gaussian process allows such modeling. For now, however, an MDP with a full continuous belief state space $\mathcal{B}$ is considered.

The discounted return R_t^π for time step t and a given policy π is the total accumulated reward acquired over time:

$$R_t^\pi = \sum_{i=0}^{\infty} \gamma^i r_{t+i+1}, \tag{1}$$

where r_t is the immediate reward at time step t and $\gamma \in [0,1]$, is the discount factor. If the immediate reward is a random process, the discounted return is also a random process.

The discounted return can be written recursively as:

$$R_t^\pi = r_{t+1} + \gamma R_{t+1}^\pi. \tag{2}$$

The Q-function for policy π is the expectation of the discounted return given belief state $\mathbf{b}$ and action a at time t, over all possible belief state sequences that can be generated:

$$Q^\pi(\mathbf{b}, a) = E_\pi\left(R_t | b(s_t) = \mathbf{b}, a_t = a\right). \tag{3}$$

Due to the stochasticity of transitions, the discounted return is a random variable and can be decomposed into a mean $Q^\pi(\mathbf{b}, a)$ and a residual $\Delta Q^\pi(\mathbf{b}, a)$:

$$R_t^\pi(b(s_t) = \mathbf{b}, a_t = a) = Q^\pi(\mathbf{b}, a) + \Delta Q^\pi(\mathbf{b}, a). \tag{4}$$

Substituting R_t^π and R_{t+1}^π from (4) into (2) yields:

$$r_{t+1}(b(s_t) = \mathbf{b}, a_t = a) = Q^\pi(\mathbf{b}, a) \\ -\gamma Q^\pi(\mathbf{b}', a') + \Delta Q^\pi(\mathbf{b}, a) - \gamma \Delta Q^\pi(\mathbf{b}', a'), \quad (5)$$

where $b(s_{t+1}) = \mathbf{b}'$ is the next belief state and $a' = \pi(\mathbf{b}')$ is the next action, $a_{t+1} = a'$.

Let $\mathbf{B}_t = [(\mathbf{b}^0, a^0), \ldots, (\mathbf{b}^t, a^t)]^\top$ be a sequence of t belief states and action samples[1] generated with policy π. Then, (5) becomes:

$$\begin{aligned} r^1 &= Q^\pi(\mathbf{b}^0, a^0) - \gamma Q^\pi(\mathbf{b}^1, a^1) \\ &\quad + \Delta Q^\pi(\mathbf{b}^0, a^0) - \gamma \Delta Q^\pi(\mathbf{b}^1, a^1) \\ r^2 &= Q^\pi(\mathbf{b}^1, a^1) - \gamma Q^\pi(\mathbf{b}^2, a^2) \\ &\quad + \Delta Q^\pi(\mathbf{b}^1, a^1) - \gamma \Delta Q^\pi(\mathbf{b}^2, a^2) \\ &\vdots \\ r^t &= Q^\pi(\mathbf{b}^{t-1}, a^{t-1}) - \gamma Q^\pi(\mathbf{b}^t, a^t) \\ &\quad + \Delta Q^\pi(\mathbf{b}^{t-1}, a^{t-1}) - \gamma \Delta Q^\pi(\mathbf{b}^t, a^t), \end{aligned} \quad (6)$$

where $r^1, \ldots, r^t$ are the acquired immediate rewards. This can be written in a more compact form as

$$\mathbf{r}_t = \mathbf{H}_t \mathbf{q}_t^\pi + \mathbf{H}_t \mathbf{\Delta q}_t^\pi, \quad (7a)$$

where

$$\begin{aligned} \mathbf{r}_t &= [r^1, \ldots, r^t]^\top & (7b) \\ \mathbf{q}_t^\pi &= [Q^\pi(\mathbf{b}^0, a^0), \ldots, Q^\pi(\mathbf{b}^t, a^t)]^\top, & (7c) \\ \mathbf{\Delta q}_t^\pi &= [\Delta Q^\pi(\mathbf{b}^0, a^0), \ldots, \Delta Q^\pi(\mathbf{b}^t, a^t)]^\top, & (7d) \\ \mathbf{H}_t &= \begin{bmatrix} 1 & -\gamma & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & 1 & -\gamma \end{bmatrix}. & (7e) \end{aligned}$$

If the transition probabilities and policy π do not change over time, $Q^\pi(\mathbf{b}, a)$ is not a random variable, however, during the process of estimation π changes, so $Q^\pi(\mathbf{b}, a)$ can be modeled as a random variable. By modeling $Q^\pi(\mathbf{b}, a)$ as a Gaussian process, $Q^\pi(\mathbf{b}, a) \sim \mathcal{GP}(0, k((\mathbf{b}, a), (\mathbf{b}, a)))$, where the kernel $k(\cdot, \cdot)$ is factored into separate kernels over the belief state and action spaces $k_{\mathcal{B}}(\mathbf{b}, \mathbf{b}')k_{\mathcal{A}}(a, a')$ and $\Delta Q^\pi(\mathbf{b}, a) \sim \mathcal{N}(0, \sigma^2)$ is Gaussian noise, Gaussian process regression [22] can be applied to find the posterior of $Q^\pi(\mathbf{b}, a)$, given a set of belief state-action pairs $\mathbf{B}_t$ and the observed rewards $\mathbf{r}_t$:

$$\begin{aligned} Q^\pi(\mathbf{b}, a) | \mathbf{r}_t, \mathbf{B}_t &\sim \mathcal{N}(\overline{Q}(\mathbf{b}, a), \mathrm{cov}((\mathbf{b}, a), (\mathbf{b}, a))), \\ \overline{Q}(\mathbf{b}, a) &= \mathbf{k}_t(\mathbf{b}, a)^\top \mathbf{H}_t^\top (\mathbf{H}_t \mathbf{K}_t \mathbf{H}_t^\top + \sigma^2 \mathbf{H}_t \mathbf{H}_t^\top)^{-1} \mathbf{r}_t, \\ \mathrm{cov}((\mathbf{b}, a), (\mathbf{b}, a)) &= k((\mathbf{b}, a), (\mathbf{b}, a)) - \mathbf{k}_t(\mathbf{b}, a)^\top \mathbf{H}_t^\top (\mathbf{H}_t \mathbf{K}_t \mathbf{H}_t^\top \\ &\quad + \sigma^2 \mathbf{H}_t \mathbf{H}_t^\top)^{-1} \mathbf{H}_t \mathbf{k}_t(\mathbf{b}, a) \end{aligned} \quad (8)$$

where $\mathbf{k}_t(\mathbf{b}, a) = [k((\mathbf{b}^0, a^0), (\mathbf{b}, a)), \ldots, k((\mathbf{b}^t, a^t), (\mathbf{b}, a))]^\top$ and $\mathbf{K}_t = [\mathbf{k}_t((\mathbf{b}^0, a^0)), \ldots, \mathbf{k}_t((\mathbf{b}^t, a^t))]$.

[1]Random variables are denoted with a subscript, *e.g.*, a_t is a random variable at some time step t. Samples are denoted with a superscript *e.g.*, a^t is the action that was taken at time step t.

This Gaussian process model of the Q-function exploits the relationship between *distributions* of the Q-function at differing time steps. This is in contrast to standard reinforcement learning algorithms which exploit the relationship between *values* of the Q-function at differing time steps. An important implication of this property is that prior knowledge about the Q-function can be incorporated in the prior distribution. Initially, the estimated Q-function distribution is just a zero-mean Gaussian process with the kernel function $k((\mathbf{b}, a), (\mathbf{b}, a))$, since no data has been observed. By time step t, the estimate is the posterior distribution given the set of observed rewards $\mathbf{r}_t$ and associate belief state-action pairs $\mathbf{B}_t$.

It can be shown [23] that the marginal likelihood of the observed rewards is modeled by

$$\mathbf{r}_t | \mathbf{B}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{H}_t(\mathbf{K}_t + \sigma^2 \mathbf{I})\mathbf{H}_t^\top). \quad (9)$$

If the kernel function is parametrized, this relationship can be used to optimize the kernel parameters as well as the noise parameter σ[24].

In the next section the derivation of a policy model from the Gaussian process model of the Q-function is discussed.

B. Gaussian Process-Based Policy Model

The description so far has presented a Gaussian process model for the Q-function associated with a policy π given belief state-action pairs $\mathbf{B}_t$ and rewards $\mathbf{r}_t$(8). What remains is to define the policy.

The simplest way of deriving such a policy to select the action associated with the highest mean of the Q-function:

$$\pi(\mathbf{b}) = \arg\max_a \overline{Q}(\mathbf{b}, a). \quad (10)$$

However, since the objective is to learn the Q-function associated with the optimal policy from interaction, the policy must exhibit some form of stochastic behavior in order to explore alternatives during the process of learning. One way of defining a policy that can be optimized on-line is to use an ϵ-greedy approach. This requires setting an additional parameter ϵ which balances how often an action is taken according to the current best estimate of the Q-function mean (the *exploitation* stage) compared to how often an action is taken randomly (the *exploration* stage). This policy is defined as:

$$\pi(\mathbf{b}) = \begin{cases} \arg\max_a \overline{Q}(\mathbf{b}, a) & \text{with prob } 1 - \epsilon, \\ \text{random } a & \text{with prob } \epsilon. \end{cases} \quad (11)$$

However, such random exploration can be inefficient since not all parts of belief-action space are equally informative. Using active learning to provide more efficient exploration should yield faster learning [25]. Instead of selecting actions randomly, active learning selects actions according to some *utility function* which normally includes some measure of information gain through which various heuristics can be incorporated [26]. Gaussian process reinforcement learning provides a measure of uncertainty at each point in belief-action space. This uncertainty can then be used directly in the active learning utility function [21]. This enables the model to explore the parts of the space it is less certain about. Therefore, during exploration, actions are chosen according to the variance of the

GP estimate for the Q-function and during exploitation, actions are chosen according to the mean:

$$\pi(\mathbf{b}) = \begin{cases} \arg\max_a \overline{Q}(\mathbf{b}, a) & \text{with prob } 1-\epsilon, \\ \arg\max_a \operatorname{cov}((\mathbf{b}, a), (\mathbf{b}, a)) & \text{with prob } \epsilon. \end{cases} \tag{12}$$

Both of the policies defined in (11) and (12) require the proportions of exploration and exploitation to be balanced manually by tuning the parameter ϵ. Since the Gaussian process for the Q-function defines a Gaussian distribution for every belief state-action pair (8), when a new belief point $\mathbf{b}$ is encountered, for each action $a \in \mathcal{A}$, there is a Gaussian distribution $\hat{Q}(\mathbf{b}, a) \sim \mathcal{N}(\overline{Q}(\mathbf{b}, a), \operatorname{cov}((\mathbf{b}, a), (\mathbf{b}, a))))$. Sampling from these Gaussian distributions gives Q-values from which the action with the highest sampled Q-value can be selected:

$$\pi(\mathbf{b}) = \arg\max_a \left\{ \hat{Q}(\mathbf{b}, a) : a \in \mathcal{A} \right\}. \tag{13}$$

If two actions have overlapping distributions then sampling from these distributions will result in the system randomly switching between them. Thus, in contrast to the ϵ-greedy policies defined by (11) and (12), this approach maps the GP approximation of the Q-function into a stochastic policy model. Note however that all three policies are *greedy* in the sense that as either ϵ or the variance tends to zero, they tend to select the locally optimal action. The performance of different policy models is discussed further below in Section IV-B.

```
1: for each episode do
2:   Initialise $\mathbf{b}$
3:   if first episode then
4:     Choose $a$ arbitrary
5:     $\mathbf{B}_0 \leftarrow (\mathbf{b}, a)$
6:     $\mathbf{K}_0 \leftarrow k((\mathbf{b}, a), (\mathbf{b}, a))$
7:     $\mathbf{H}_0 \leftarrow [\, 1 \quad -\gamma \,]$
8:   else
9:     if initial step then
10:      Choose $a \leftarrow \pi(\mathbf{b})$ (11), (12) or (13)
11:    end if
12:  end if
13:  for each step in the episode do
14:    Take $a$, observe $r'$, update $\mathbf{b}'$
15:    if non-terminal step then
16:      Choose new action $a' \leftarrow \pi(\mathbf{b}')$ (11), (12) or (13)
17:      $\mathbf{B}_{t+1} \leftarrow [\, \mathbf{B}_t \quad (\mathbf{b}', a') \,]$
18:      $\mathbf{H}_{t+1} \leftarrow \begin{bmatrix} \mathbf{H}_t & \mathbf{0} \\ \mathbf{u}^\mathsf{T} & -\gamma \end{bmatrix}$, where $\mathbf{u} = [\, \mathbf{0} \quad 1 \,]^\mathsf{T}$
19:      $\mathbf{K}_{t+1} \leftarrow \begin{bmatrix} \mathbf{K}_t & \mathbf{k}_t(\mathbf{b}', a') \\ \mathbf{k}_t(\mathbf{b}', a') & k(\mathbf{b}', a', \mathbf{b}', a') \end{bmatrix}$
20:    else
21:      $\mathbf{B}_{t+1} \leftarrow \mathbf{B}_t$, $\mathbf{K}_{t+1} \leftarrow \mathbf{K}_t$
22:      $\mathbf{H}_{t+1} \leftarrow \begin{bmatrix} \mathbf{H}_t \\ \mathbf{u}^\mathsf{T} \end{bmatrix}$, where $\mathbf{u} = [\, \mathbf{0} \quad 1 \,]^\mathsf{T}$
23:    end if
24:    $\mathbf{r}_{t+1} \leftarrow [\, \mathbf{r}_t \quad r' \,]$
25:    Update $Q$-function posterior $Q^\pi | \mathbf{r}_{t+1}, \mathbf{B}_{t+1}$ (8)
26:    if non-terminal step then
27:      $\mathbf{b} \leftarrow \mathbf{b}'$, $a \leftarrow a'$
28:    end if
29:  end for
30: end for
```

Fig. 1. Episodic GP-Sarsa (without computational constraints).

C. Gaussian Process-Based Policy Optimisation

If the Q-function of a greedy policy is estimated on-line using temporal differences, then it will in the limit tend to the Q-function of the optimal policy. A commonly used embodiment of this idea for MDPs is the Sarsa algorithm which iteratively updates the Q-function on-line using the rule:

$$Q(\mathbf{b}, a) \leftarrow Q(\mathbf{b}, a) + \alpha[r_{t+1}(b_t = \mathbf{b}, a_t = a) \\ + \gamma Q(\mathbf{b}', a') - Q(\mathbf{b}, a)]. \tag{14}$$

If the Q-function correction provided by the right hand side of (14) is replaced by the update of the GP-based Q-function posterior given by (8), the GP-Sarsa algorithm is obtained. This is shown in Fig. 1 amended for episodic reinforcement learning and hence applicable to dialog optimization.

In practice, however, the exact computation of the posterior using (8) is intractable as it requires the inversion of a matrix of dimensionality t. A solution is discussed in the next section.

D. Computational Complexity and Sparse Approximation

Due to the matrix inversion in (8), the computational complexity of calculating the Q-function posterior is $O(t^3)$, where t is the number of data points. In the case of a dialog system, this will be equal to the total number of turns, summed over all dialogs, which poses a serious computational problem.

One solution to this problem is to restrict the set of data points used for the Q-function approximation. This has the obvious drawback that useful information will be discarded and this is exacerbated when the space is continuous since the exact same state is never visited twice. Sparse approximation methods for Gaussian processes aim to take into account all data points while reducing the computational complexity.

A significant research effort has been invested into solving this problem, and a number of methods have been developed [27]. However, these rely on the pre-selection of a fixed set of m support points. This reduces the complexity of calculating the posterior from $O(t^3)$ to $O(tm^2)$ and if the number of support points is significantly smaller than the number of data points, this approach is very effective for reducing the computational cost. This class of methods is not useful for learning on-line, since the support points cannot be determined *a priori*. An alternative algorithm which approximates the Gaussian process without first obtaining a set of support points is the kernel span sparsification method described in [28]. In this case, a representative set of data points is acquired as the belief-action space is traversed during interaction with the user. It is assumed that the environment dynamics do not change over time hence obviating the need to remove support points and thereby avoiding the difficulties discussed in [29].

A kernel function can be thought of as the dot product of a (potentially infinite) set of feature functions $k((\mathbf{b}, a), (\mathbf{b}, a)) = \langle \boldsymbol{\phi}(\mathbf{b}, a), \boldsymbol{\phi}(\mathbf{b}, a) \rangle$ where $\boldsymbol{\phi}(\mathbf{b}, a)$ is the vector of feature functions $[\phi_1(\mathbf{b}, a), \phi_2(\mathbf{b}, a), \ldots]^\mathsf{T}$. Any linear combination of feature vectors $\{\boldsymbol{\phi}(\mathbf{b}^0, a^0), \ldots, \boldsymbol{\phi}(\mathbf{b}^t, a^t)\}$ for a given set of points $\{(\mathbf{b}^0, a^0), \ldots, (\mathbf{b}^t, a^t)\}$ is called the *kernel span*. The aim then is to find the subset of points that approximates this kernel span. These points are called the *representative points* and the set of representative points is called the *dictionary* $\mathcal{D} = \{(\tilde{\mathbf{b}}^0, \tilde{a}^0), \ldots, (\tilde{\mathbf{b}}^m, \tilde{a}^m)\}$.

A sparsification parameter ν places a threshold on the squared distance between the feature function span at the representative points, and the true feature function value at each visited point:

$$\min_{\mathbf{g}_t} \| \sum_{j=0}^{m} g_{tj}\boldsymbol{\phi}(\tilde{\mathbf{b}}^j, \tilde{a}^j) - \boldsymbol{\phi}(\mathbf{b}^t, a^t) \|^2 \leq \nu, \tag{15}$$

where $(\mathbf{b}^t, a^t)$ is the current point, $\mathbf{g}_t = [g_{t1}, \ldots, g_{tj}]$ is a vector of coefficients and m is the size of the current dictionary, $\mathcal{D} = \{(\tilde{\mathbf{b}}^0, \tilde{a}^0), \ldots, (\tilde{\mathbf{b}}^m, \tilde{a}^m)\}$. This is equivalent to [28]:

$$\min_{\mathbf{g}_t} \left(k((\mathbf{b}^t, a^t), (\mathbf{b}^t, a^t)) - \tilde{\mathbf{k}}_{t-1}(\mathbf{b}^t, a^t)^\mathsf{T}\mathbf{g}_t \right) \leq \nu, \tag{16}$$

where

$$\begin{aligned}\tilde{\mathbf{k}}_{t-1}&(\mathbf{b}^t, a^t) \\ &= [k((\mathbf{b}^t, a^t), (\tilde{\mathbf{b}}^0, \tilde{a}^0)), \ldots, k((\mathbf{b}^t, a^t), (\tilde{\mathbf{b}}^m, \tilde{a}^m))]^\mathsf{T}.\end{aligned}$$

It can also be shown that the expression on the left side of (16) is minimized when $\mathbf{g}_t = \tilde{\mathbf{K}}_{t-1}^{-1}\tilde{\mathbf{k}}_{t-1}(\mathbf{b}^t, a^t)$, where $\tilde{\mathbf{K}}_{t-1}$ is the Gram matrix of the current set of representative points. If the threshold ν is exceeded then $(\mathbf{b}^t, a^t)$ is added to the dictionary, otherwise the dictionary stays the same. This constitutes the sparsification criterion and it allows the posterior to be approximated with bounded complexity as follows.

Since the kernel function is the dot product of the feature functions, the Gram matrix is $\mathbf{K}_t = \boldsymbol{\Phi}_t^\mathsf{T}\boldsymbol{\Phi}_t$ where $\boldsymbol{\Phi}_t = [\boldsymbol{\phi}(\mathbf{b}^0, a^0), \ldots, \boldsymbol{\phi}(\mathbf{b}^t, a^t)]$. The feature function values for each point are then approximated as the linear combination of the representative points $\boldsymbol{\phi}(\mathbf{b}^i, a^i) \approx \sum_{j=1}^{m} g_{ij}\boldsymbol{\phi}(\tilde{\mathbf{b}}^j, \tilde{a}^j)$, for all $i \in 0, \ldots, t$. Also, the Gram matrix is approximated as

$$\mathbf{K}_t = \boldsymbol{\Phi}_t^\mathsf{T}\boldsymbol{\Phi}_t \approx \mathbf{G}_t\tilde{\mathbf{K}}_t\mathbf{G}_t^\mathsf{T}, \tag{17}$$

where $\mathbf{G}_t = [\mathbf{g}_1, \ldots, \mathbf{g}_t]$. Similarly, $\mathbf{k}_t(\mathbf{b}, a) \approx \mathbf{G}_t\tilde{\mathbf{k}}_t(\mathbf{b}, a)$.

This allows the posterior (8) to be approximated as:

$$\begin{aligned} Q(\mathbf{b}, a)|\mathbf{B}_t, \mathbf{r}_t &\sim \mathcal{N}\left(\tilde{\overline{Q}}(\mathbf{b}, a), \widetilde{\mathrm{cov}}((\mathbf{b}, a), (\mathbf{b}, a))\right), \\ \tilde{\overline{Q}}(\mathbf{b}, a) &= \tilde{\mathbf{k}}_t(\mathbf{b}, a)^\mathsf{T}(\tilde{\mathbf{H}}_t^\mathsf{T}(\tilde{\mathbf{H}}_t\tilde{\mathbf{K}}_t\tilde{\mathbf{H}}_t^\mathsf{T} + \sigma^2\tilde{\mathbf{H}}_t\tilde{\mathbf{H}}_t^\mathsf{T})^{-1}\mathbf{r}_t), \\ \widetilde{\mathrm{cov}}&((\mathbf{b}, a), (\mathbf{b}, a)) \\ &= k((\mathbf{b}, a), (\mathbf{b}, a)) - \tilde{\mathbf{k}}_t(\mathbf{b}, a)^\mathsf{T}(\tilde{\mathbf{H}}_t^\mathsf{T}(\tilde{\mathbf{H}}_t\tilde{\mathbf{K}}_t\tilde{\mathbf{H}}_t^\mathsf{T} \\ &\quad + \sigma^2\tilde{\mathbf{H}}_t\tilde{\mathbf{H}}_t^\mathsf{T})^{-1}\tilde{\mathbf{H}}_t)\tilde{\mathbf{k}}_t(\mathbf{b}, a), \end{aligned} \tag{18}$$

where $\tilde{\mathbf{H}}_t = \mathbf{H}_t\mathbf{G}_t$. It can be shown that this reduces the complexity to $O(tm^2)$, where m is the size of the dictionary, see [19] for details.

This sparsification method has a drawback that it turns a non-parametric method into a parametric method. More precisely, the kernel is now approximated using only a limited number of points, which is equivalent to defining a functional basis for the kernel function. This can be shown to be equivalent to parametrizing the Q-function [19]. This may limit the accuracy of the solution, however, the fact that the basis is chosen dynamically, without an initial set of basis functions, still allows for more appropriate functions than when using a fixed basis defined by a designer.

```
 1: Initialise $\tilde{\boldsymbol{\mu}} \leftarrow []$, $\tilde{\mathbf{C}} \leftarrow []$, $\tilde{\mathbf{c}} \leftarrow []$, $d \leftarrow 0$, $1/v \leftarrow 0$
 2: for each episode do
 3:   Initialise $\mathbf{b}$
 4:   if first episode then
 5:     Choose $a$ arbitrary, $\mathcal{D} = \{(\mathbf{b}, a)\}$, $\tilde{\mathbf{K}}^{-1} \leftarrow 1/k((\mathbf{b}, a), (\mathbf{b}, a))$
 6:   else
 7:     if initial step then
 8:       Choose $a \leftarrow \pi(\mathbf{b})$
 9:     end if
10:   end if
11:   $\tilde{\mathbf{c}} \leftarrow \mathbf{0}$ {size $|\mathcal{D}|$}, $d \leftarrow 0$, $1/v \leftarrow 0$
12:   $\mathbf{g} \leftarrow \tilde{\mathbf{K}}^{-1}\tilde{\mathbf{k}}(\mathbf{b}, a)$, $\delta \leftarrow k((\mathbf{b}, a), (\mathbf{b}, a)) - \tilde{\mathbf{k}}(\mathbf{b}, a)^\mathsf{T}\mathbf{g}$
13:   if $\delta > \nu$ then
14:     $\mathcal{D} \leftarrow \{(\mathbf{b}, a)\} \cup \mathcal{D}$, $\tilde{\mathbf{K}}^{-1} \leftarrow \frac{1}{\delta}\begin{bmatrix} \delta\tilde{\mathbf{K}}^{-1} + \mathbf{g}\mathbf{g}^\mathsf{T} & -\mathbf{g} \\ -\mathbf{g}^\mathsf{T} & 1 \end{bmatrix}$
15:     $\mathbf{g} \leftarrow [0, \ldots, 0, 1]^\mathsf{T}$ {size $|\mathcal{D}|$}, $\tilde{\boldsymbol{\mu}} \leftarrow \begin{bmatrix} \tilde{\boldsymbol{\mu}} \\ 0 \end{bmatrix}$, $\tilde{\mathbf{C}} \leftarrow \begin{bmatrix} \tilde{\mathbf{C}} & \mathbf{0} \\ \mathbf{0}^\mathsf{T} & 0 \end{bmatrix}$, $\tilde{\mathbf{c}} \leftarrow \begin{bmatrix} \tilde{\mathbf{c}} \\ 0 \end{bmatrix}$
16:   end if
17:   for each step in the episode do
18:     Take $a$, observe $r'$, update $\mathbf{b}'$
19:     if non-terminal step then
20:       Choose new action $a' \leftarrow \pi(\mathbf{b}')$
21:       $\mathbf{g}' \leftarrow \tilde{\mathbf{K}}^{-1}\tilde{\mathbf{k}}(\mathbf{b}', a')$, $\delta \leftarrow k((\mathbf{b}', a'), (\mathbf{b}', a')) - \tilde{\mathbf{k}}(\mathbf{b}', a')^\mathsf{T}\mathbf{g}'$
22:       $\Delta\tilde{\mathbf{k}} \leftarrow \tilde{\mathbf{k}}(\mathbf{b}, a) - \gamma\tilde{\mathbf{k}}(\mathbf{b}', a')$
23:     else
24:       $\mathbf{g}' \leftarrow [\mathbf{0}]$ {size $|\mathcal{D}|$}, $\delta \leftarrow 0$, $\Delta\tilde{\mathbf{k}} \leftarrow \tilde{\mathbf{k}}(\mathbf{b}, a)$
25:     end if
26:     $d \leftarrow \frac{\gamma\sigma^2}{v}d + r' - \Delta\tilde{\mathbf{k}}^\mathsf{T}\tilde{\boldsymbol{\mu}}$
27:     if $\delta > \nu$ and non-terminal step then
28:       $\mathcal{D} \leftarrow \{(\mathbf{b}', a')\} \cup \mathcal{D}$, $\tilde{\mathbf{K}}^{-1} \leftarrow \frac{1}{\delta}\begin{bmatrix} \delta\tilde{\mathbf{K}}^{-1} + \mathbf{g}\mathbf{g}^\mathsf{T} & -\mathbf{g} \\ -\mathbf{g}^\mathsf{T} & 1 \end{bmatrix}$
29:       $\mathbf{g}' \leftarrow [0, \ldots, 0, 1]^\mathsf{T}$, $\tilde{\mathbf{h}} \leftarrow [\mathbf{g}^\mathbf{T}, -\gamma]^\mathsf{T}$
30:       $\Delta k_{tt} \leftarrow \mathbf{g}^\mathsf{T}(\tilde{\mathbf{k}}(\mathbf{b}, a) - 2\gamma\tilde{\mathbf{k}}(\mathbf{b}', a')) + \gamma^2 k((\mathbf{b}', a'), (\mathbf{b}', a'))$
31:       $\tilde{\mathbf{c}}' \leftarrow \frac{\gamma\sigma^2}{v}\begin{bmatrix} \tilde{\mathbf{c}} \\ 0 \end{bmatrix} + \tilde{\mathbf{h}} - \begin{bmatrix} \tilde{\mathbf{C}}\Delta\tilde{\mathbf{k}} \\ 0 \end{bmatrix}$
32:       $v \leftarrow (1+\gamma^2)\sigma^2 + \Delta k_{tt} - \Delta\tilde{\mathbf{k}}^\mathsf{T}\tilde{\mathbf{C}}\Delta\tilde{\mathbf{k}} + \frac{2\gamma\sigma^2}{v}\tilde{\mathbf{c}}\Delta\tilde{\mathbf{k}} - \frac{\gamma^2\sigma^4}{v}$
33:       $\tilde{\boldsymbol{\mu}} \leftarrow \begin{bmatrix} \tilde{\boldsymbol{\mu}} \\ 0 \end{bmatrix}$, $\tilde{\mathbf{C}} \leftarrow \begin{bmatrix} \tilde{\mathbf{C}} & \mathbf{0} \\ \mathbf{0}^\mathsf{T} & 0 \end{bmatrix}$
34:     else
35:       $\tilde{\mathbf{h}} \leftarrow \mathbf{g} - \gamma\mathbf{g}'$, $\tilde{\mathbf{c}}' \leftarrow \frac{\gamma\sigma^2}{v}\tilde{\mathbf{c}} + \tilde{\mathbf{h}} - \tilde{\mathbf{C}}\Delta\tilde{\mathbf{k}}$
36:       if non-terminal step then
37:         $v \leftarrow (1+\gamma^2)\sigma^2 + \Delta\tilde{\mathbf{k}}^\mathsf{T}(\tilde{\mathbf{c}}' + \frac{\gamma\sigma^2}{v}\tilde{\mathbf{c}}) - \frac{\gamma^2\sigma^4}{v}$
38:       else
39:         $v \leftarrow \sigma^2 + \Delta\tilde{\mathbf{k}}^\mathsf{T}(\tilde{\mathbf{c}}' + \frac{\gamma\sigma^2}{v}\tilde{\mathbf{c}}) - \frac{\gamma^2\sigma^4}{v}$
40:       end if
41:     end if
42:     $\tilde{\boldsymbol{\mu}} \leftarrow \tilde{\boldsymbol{\mu}} + \frac{\tilde{\mathbf{c}}}{v}d$, $\tilde{\mathbf{C}} \leftarrow \tilde{\mathbf{C}} + \frac{1}{v}\tilde{\mathbf{c}}'\tilde{\mathbf{c}}'^\mathsf{T}$, $\tilde{\mathbf{c}} \leftarrow \tilde{\mathbf{c}}'$, $\mathbf{g} \leftarrow \mathbf{g}'$
43:     if non-terminal step then
44:       $\mathbf{b} \leftarrow \mathbf{b}'$, $a \leftarrow a'$
45:     end if
46:   end for
47: end for
```

Fig. 2. Episodic GP-Sarsa.

An advantage of this sparsification approach is that it enables non-positive definite kernel functions to be used in the approximation, for example see [24]. This is due to the fact that the sparsification method essentially changes the kernel function in a manner which ensures that the approximated Gram matrix is positive definite and this is sufficient to guarantee that the model remains well-defined [30].

This sparsification method allows observations to be processed sequentially, in direct interaction with the user, and it can be incorporated directly into the GP-Sarsa algorithm out-lined in Fig. 1. The full GP-Sarsa algorithm with kernel sparsification is given in Fig. 2.

IV. Experiments

The previous sections have introduced the theory of Gaussian process-based POMDP policy optimization. Here we examine their application in practice and investigate three important research issues. Firstly, Section IV-B addresses the detailed design trade-offs involved in practical implementation of a GP-based dialog policy. Secondly, Section IV-C investigates the speed of the GP policy optimization process and whether in practice it can be performed in direct interaction with real users. Finally, Section IV-D addresses the extent to which design time effort can be reduced by applying a GP-policy directly to the full belief space.

A. Experimental Set-Up

A spoken dialog system providing restaurant information for Cambridge UK was used as a test-bed for all of the reported experiments. This system uses the BUDS dialog manager in which beliefs are modeled by a Bayesian network. Training and testing was provided by an agenda-based simulated user and by real users recruited using the Amazon MTurk service.

1) Bayesian Update of Dialogue State Dialogue Manager: The Bayesian Update of Dialogue State (BUDS) dialog manager is a POMDP-based dialog manager [13] whose belief state consists of the marginal posterior probability distribution over hidden nodes in a Bayesian network. Each concept in the dialog, e.g. *area*, *food-type*, *address* is represented by a pair of hidden nodes recording the history and the goal. The history nodes define possible states e.g. *system-informed*, *user-informed* and the goal nodes define possible values for a particular concept, e.g. *Chinese*, *Indian*.

In order to train an optimal policy the belief space $\mathcal{B}$ is mapped into a summary space $\mathcal{C}$. Likewise, the action space is mapped into a smaller scale summary action space $\mathcal{A}$. This summary space $\mathcal{C} \times \mathcal{A}$ is then mapped into a feature space $\mathcal{F}$. This space is then used to produce a parametric representation of the policy. A weighted set of these basis functions can then be cast into a probability via a soft-max function

$$\pi(a|\mathbf{c};\theta) = \frac{e^{\theta \cdot f_a(\mathbf{c})}}{\sum_a e^{\theta \cdot f_a(\mathbf{c})}}. \tag{19}$$

where $f_a(\mathbf{c})$ is the vector of features derived from summary state $\mathbf{c}$ for action a. The policy is then optimized using the natural actor critic (NAC) algorithm which is a gradient-based method [31]. In operation, system responses are generated by sampling (19) and then heuristically mapping the summary action back into a full system action. Once the system has chosen the summary action, the mapping to a full action is usually straightforward. For instance, if the action is `ConfirmArea` the system simply confirms the most likely value of the *area* slot.

Mappings $\mathcal{B} \rightarrow \mathcal{C}$ and $\mathcal{C} \times \mathcal{A} \rightarrow \mathcal{F}$, however, require a significant amount of hand-crafting. Also, due to the parametric policy representation, the solution is only optimal within the given basis. Finally, gradient-based optimization methods are inherently slow which prevents direct on-line optimization with real users. Here we show that the GP-Sarsa algorithm can overcome these limitations.

2) The Cambridge Restaurant Domain: The Cambridge restaurant domain consists of a selection of approximately 150 restaurants, referred to as *entities*, which have been automatically extracted from various web-based sources. Each restaurant has 8 attributes (slots) and this results in a belief space consisting of 25 concepts where each concept takes from 3 to 150 values and each value has a probability in $[0, 1]$.[2] The summary space is formed from discrete, as well as continuous, features. Continuous features represent the entropy of the distribution for each hidden node in the Bayesian network. In addition, there are a few discrete features that correspond to history nodes, e.g., if the highest probability method that the user used to inform the system was *by name of the venue* or if the highest probability discourse act used was *repeat*. Also, there is a count of the number of top probability goal values that are greater than 0.8. Finally, there is a list which defines the order in which the goal slots can be accepted. In total, there are 129 features. The summary action space consists of 16 summary actions.

3) The Agenda-Based Simulated User: In training and testing, an agenda-based user simulator was used [32], [33]. The user state is factored into an *agenda* and a *goal*. The goal ensures that the user simulator exhibits consistent, goal-directed behavior. The role of the agenda is to elicit the dialog acts that are needed for the user simulator to fulfil the goal. Both the goal and the agenda are dynamically updated throughout the dialog. These updates are controlled by deterministic or stochastic *decision points*, allowing a wide spread of realistic dialogs to be generated.

In addition, an error model was used to add confusions to the simulated user input such that it resembles those found in real data [34]. The output of the error model is an N-best list of possible user responses. The length of this list was set to 10 and the confusion rate was set to 15%, which means that 15% of time the true hypothesis is not in the N-best list. Intermediate experiments showed that these settings match the confusions typically found in real data.

The reward function was set to give a reward of 20 for successful dialogs, zero otherwise, less the number of dialog turns. A successful dialog is one in which the user goal is fulfilled. If the user goal changes, success depends on satisfying the final the user goal. The discount factor γ is set to 1 and the dialog length is limited to 30 turns.

4) The Evaluation Schedule: When evaluating a policy optimization technique one is typically interested in the performance of the resulting policy. However, it is also important to know how many dialog sessions are needed to reach that performance. Policy optimization is a random process itself. A lucky choice of actions at the beginning may lead to very good performance, whereas an unlucky choice of actions can slow down the process of learning. Therefore, it is important to examine the variability in performance during training.

For this reason, in all the user simulator-based experiments reported here, unless stated otherwise, 10 training sessions were seeded with different random seeds and after every 1000 dialogs the performance of the partially trained policies were evaluated.

[2]It should be emphasised that the system is not sensitive to the number of values in a slot or the number of entities in the database since only values and entities which are actually mentioned in the dialog are modeled at run time.

Each evaluation was performed with 1000 dialogs ensuring that the simulated user had a different random seed to the one used in training. The results were then averaged and presented with one standard error.

5) The MTurk Amazon Service: In order to evaluate different policies with real people we recruited subjects using the Amazon MTurk crowd-sourcing service in a set-up similar to [35]. The BUDS dialog manager was incorporated in a live telephone-based spoken dialog system in which human users were assigned specific tasks which involved finding restaurants that have particular features. To elicit more complex dialogs, users were sometimes asked to find more than one restaurant, or a restaurant that does not exist. In the latter case, they were asked to change one of their constraints, for example find a Chinese restaurant instead of a Vietnamese one. After each dialog, users filled in a feedback form indicating whether they thought the dialog was successful. This rating was then used to calculate the average success and reward.

B. Design of Gaussian Process-Based Policy Optimisation

To use the GP-Sarsa algorithm, the kernel function, the noise parameter σ and the sparsification parameter ν need to be defined. While the choice of the kernel function and the choice of σ do not greatly affect the resulting policy, they do influence the speed of learning. In addition, the parameter ν controls the approximation of the kernel function, essentially turning a non-parametric optimization into a parametric one, and if set inappropriately it can lead to the resulting policy being suboptimal. Below we examine the different design choices.

1) Kernel Function: The kernel function defines prior correlations in different parts of the space and if correctly defined can speed up the process of learning. It ideally should be defined to suit the particular domain. However, there are a number of standard kernel functions that can be used.

As explained in Section IV-A.2, the summary space $\mathcal{C}$ consists of both continuous and discrete variables. We therefore define the kernel function on $\mathcal{C}$ as the sum[3] of kernels on continuous and discrete spaces:

$$k_{\mathcal{C}}(\mathbf{c},\mathbf{c}') = k_{\mathrm{cont}(\mathcal{C})}(\mathbf{c},\mathbf{c}') + k_{\mathrm{disc}(\mathcal{C})}(\mathbf{c},\mathbf{c}') \tag{20}$$

We investigated two standard kernels on the continuous part of the summary space.

The polynomial kernel function is defined as:

$$k(\mathbf{c},\mathbf{c}';\sigma_0,p) = (\langle \mathbf{c},\mathbf{c}'\rangle + \sigma_0^2)^p, \tag{21}$$

where $\langle\cdot,\cdot\rangle$ is the dot-product, with hyper-parameters $\sigma_0 >= 0$ and $p > 0$. In the case where $p = 1$ this is the linear kernel. Higher order polynomial kernels can lead to better results for small input spaces [23]. However, they are less suitable for large input spaces such as the one used here since the prior variance $k(\mathbf{c},\mathbf{c})$ grows rapidly with $|\mathbf{c}| > 1$[22].

The Gaussian kernel function[4] is defined as:

$$k(\mathbf{c},\mathbf{c}';p,\sigma_k) = p^2 \exp\left(-\frac{\|\mathbf{c}-\mathbf{c}'\|^2}{2\sigma_k^2}\right), \tag{22}$$

[3]A product is also possible but has not been examined here.

[4]Also called the squared exponential kernel function in the literature.

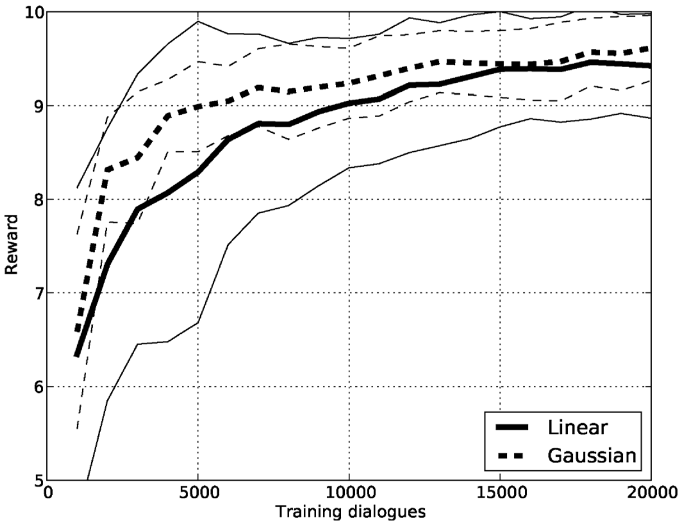

Fig. 3. Influence of the kernel choice on the performance. The performance is measured as the mean average reward as a function of the number of training dialogs. The thinner lines denote one standard error.

where σ_k determines how close the points have to be for the values of the function to be correlated, and p defines the prior variance at each data point since $k(\mathbf{c},\mathbf{c}) = p^2$. The main advantage of the Gaussian kernel over the polynomial kernel is that it is a dot product of an infinite vector of feature functions [22], which gives it the potential to model covariances better. The Gaussian kernel is a stationary kernel meaning that the covariances only depend on the distance between two summary states. This is in contrast to the polynomial kernel which is non-stationary since the covariance depends on the part of the space where the two summary states are.

The GP-Sarsa algorithm requires that the kernel function is defined on the summary actions space too. Since this is a small discrete space, the δ-kernel was used:

$$k(a,a') = \delta_a(a'). \tag{23}$$

The same function is also used on the discrete part of the summary space.

Fig. 3 compares a linear kernel ($\sigma_0 = 1$ and $p = 1$) with a Gaussian kernel ($\sigma_k = 5$ and $p = 4$). The results suggest that the Gaussian kernel is more effective as it results in less variable training. This kernel is therefore used in all subsequent experiments unless stated otherwise.

2) Residual Noise: The second element of policy design is the choice of parameter σ, the noise of the residual ΔQ (see Section III-A). As already noted, this controls how much change in the estimate of the Q-function is expected to take place during the process of learning. It is directly related to the randomness of the return (1) and therefore depends on the reward function. The rule of thumb is that σ should be the square root of the half length of the interval over possible values of the reward function. The intuition behind this is that the return in principle can take any value between the highest and the lowest possible reward. Since the process of learning is non-stationary, the return can vary significantly during the optimization process. The reward function used here gives 20 for a successful dialog less the

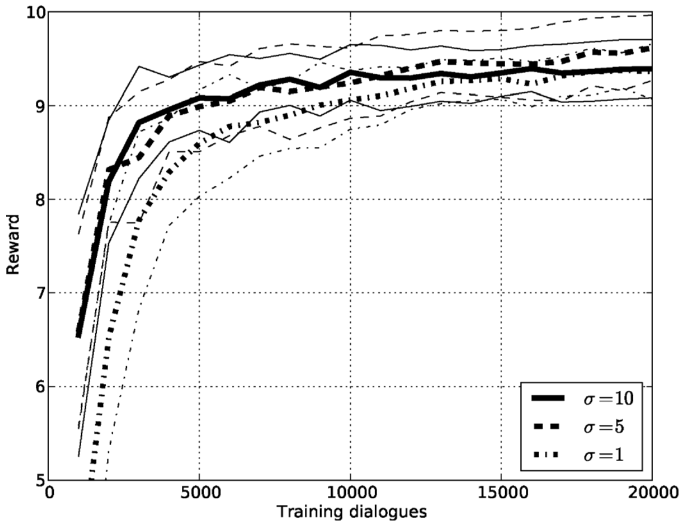

Fig. 4. Influence of the choice of the noise residual parameter σ on the performance. The thinner lines denote one standard error.

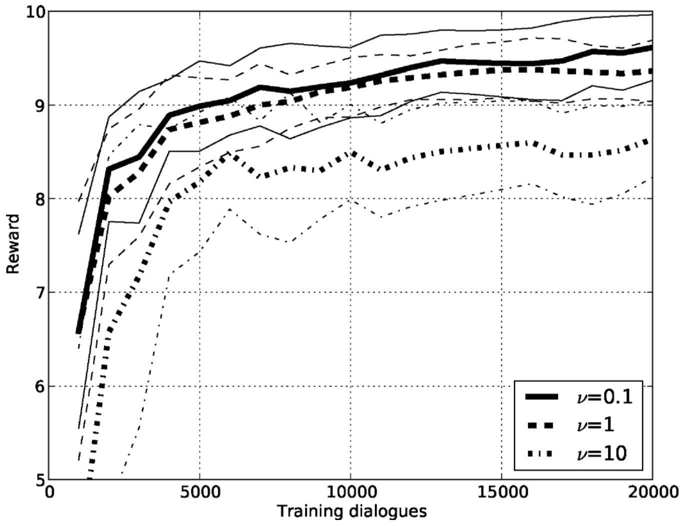

Fig. 5. Influence of the sparsification threshold ν on the performance. The thinner lines denote one standard error.

number of turns, where the user can take up to 30 turns. Therefore, the interval is $[-30, 20]$ so σ is fixed at 5. To experimentally support this claim, values of $\sigma = 1$, $\sigma = 5$ and $\sigma = 10$ were tested and the results are shown in Fig. 4 where it can be seen that $\sigma = 5$ does indeed give the best performance.

3) Sparsification Threshold: The sparsification threshold ν represents a trade-off between the computational complexity and the precision to which the kernel function is computed. Therefore, this should be some small percentage of the minimal value for $k(\mathbf{c}, \mathbf{c}')$. Fig. 5 compares performance for $\nu = 0.1$, $\nu = 1$ and $\nu = 10$. As can be seen, the smaller ν is the better performance is. It is interesting to note however that the difference in performance between $\nu = 0.1$ and $\nu = 1$ is almost negligible in comparison to the difference between $\nu = 1$ and $\nu = 10$. Fig. 6 gives the number of dictionary points for different thresholds. Comparing the two, it can be seen that for a very small improvement in performance the number of dictionary points dramatically increases.

4) Policy Model: As already noted in Section III-B, the policy model can be defined ϵ-greedily (11), using active

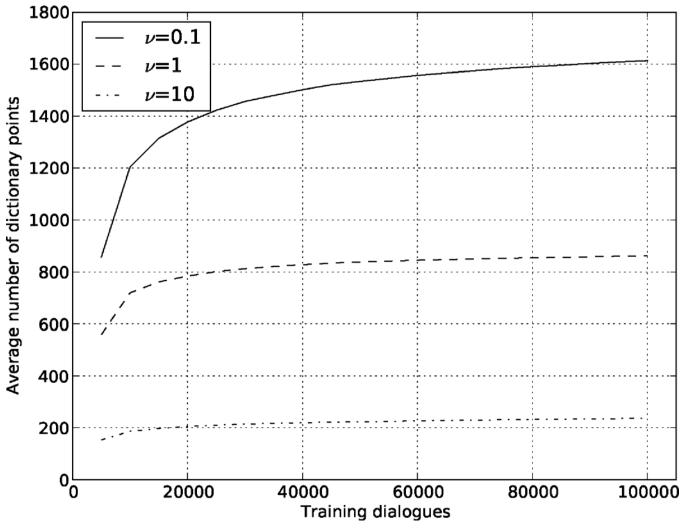

Fig. 6. Average number of dictionary points for different settings of ν as a function of the number of training dialogs.

learning (12), or stochastically (13). It has already been shown that active learning has the potential to lead to faster learning [24] as well as other learning strategies that exploit the variance [36], [37]. Here we focus on the stochastic policy since it avoids the need to explicitly control the exploration rate during training, and it provides a uniform mechanism for both optimization and normal run-time operation.

Gaussian process estimation has an inherent problem in that the estimate of the variance depends only on the number of points from the input space rather than their values in the output space. This is exacerbated in the set-up used here since the form of Gaussian process approximation used can lead to overly confident estimates [27] while at the same time, as input spaces become larger, the same points need to be visited more and more times in order to achieve optimal performance. Hence, when defining a stochastic policy for a large input space, such as the one used here, it is helpful to scale the variances during training to ensure an adequate degree of exploration: $Q(\mathbf{b}, a) \sim \mathcal{N}(\overline{Q}(\mathbf{b}, a), \eta^2 \mathrm{cov}((\mathbf{b}, a), (\mathbf{b}, a))))$ where η is the scaling factor.

Figs. 7 and 8 show performance during training and testing, respectively, for a stochastic policy with variances scaled by $\eta = 1$, $\eta = 2$ and $\eta = 3$.[5] For comparison, the performance of an ϵ-greedy policy is also shown as a baseline.

As can be seen, varying the scale factor η has a significant influence on performance. Setting η to a low value results in the system learning quickly, but also converging to a local optimum. While the best stochastic policy model shows only a small improvement in comparison to ϵ-greedy learning during testing (Fig. 8), the performance during training is significantly higher (Fig. 7) because ϵ-greedy learning requires an action to be chosen randomly some percentage of time, whereas the stochastic policy always follows the current estimate of the Q-function. This is particularly important when training in interaction with real users since poor performance during the early stages of training may impact on the user's willingness to continue to interact with the system.

[5]Higher values of η did not produce any further improvements in performance.

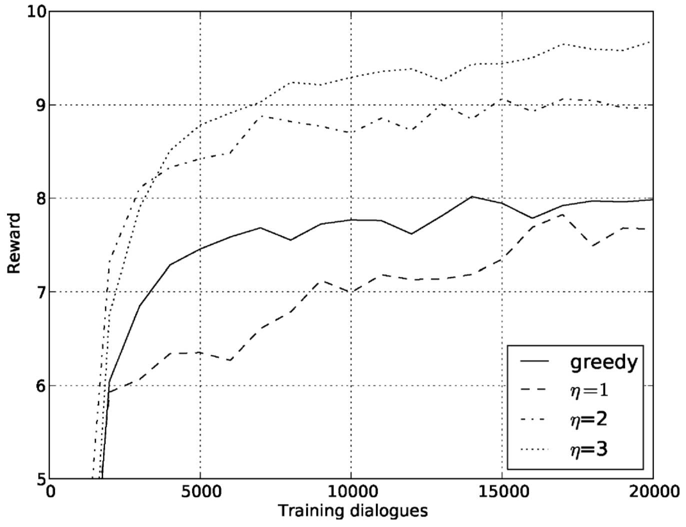

Fig. 7. Influence of variance scaling and the choice of the policy model on the performance during training.

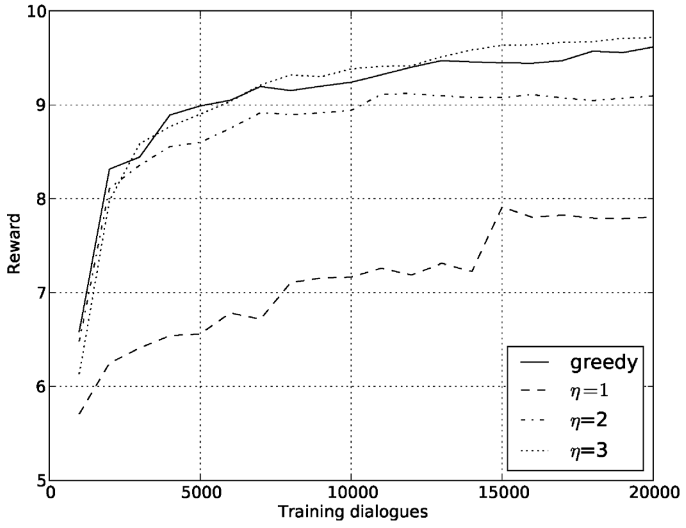

Fig. 8. Influence of variance scaling and the choice of the policy model on the performance during testing.

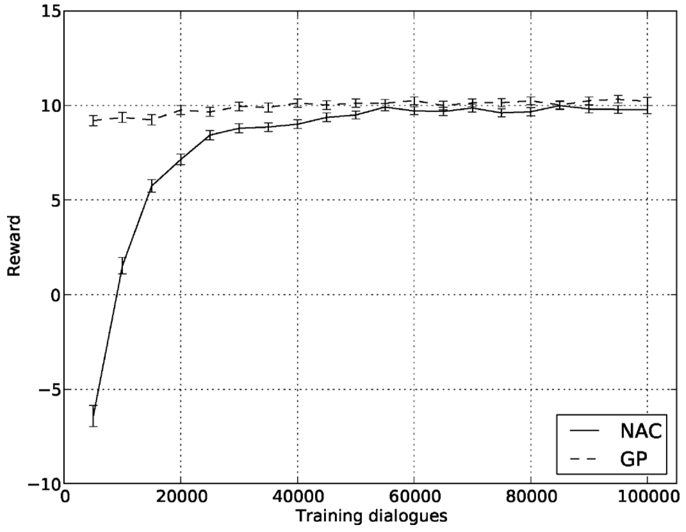

Fig. 9. Comparison of the performance of GP-Sarsa vs the natural actor critic algorithm, where the performance is measured as the average reward as a function of the number of training dialogs.

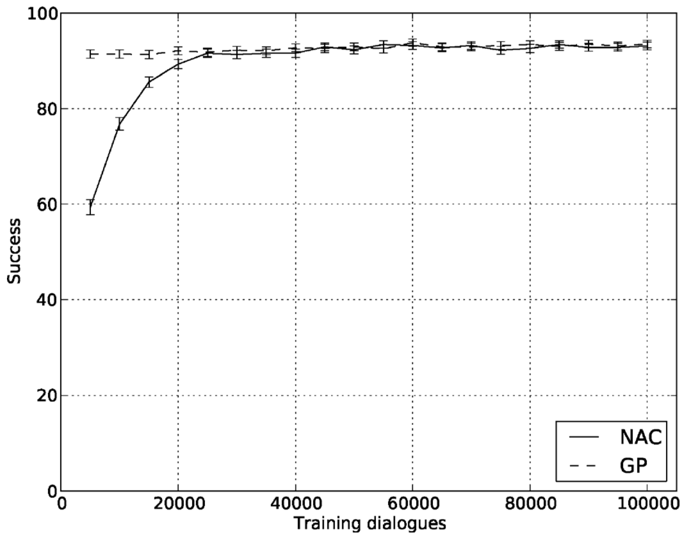

Fig. 10. Comparison of the performance of GP-Sarsa vs the natural actor critic algorithm, where the performance is measured as the average success as a function of the number of training dialogs.

C. Fast Policy Optimisation

One of the claimed advantages of the GP-Sarsa algorithm is its ability to learn from a small amount of data. It has been shown that GP-Sarsa outperforms standard non-parametric policy approaches [24]. In this section, we compare the GP-Sarsa learning rate with the NAC gradient-based algorithm.

1) Comparison With Standard Methods: NAC policy optimization for the BUDS dialog manager relies on the parametric representation of the policy π using a set of feature functions $\mathcal{F}$, as explained in Section IV-A.1. The GP-Sarsa algorithm directly optimizes the policy in the summary state $\mathcal{C}$.

To measure the rate at which each algorithm learns, the partially optimized policies were evaluated on 1000 random dialogs after being trained on 5000 dialogs. This process was repeated until the policies had been trained with 100,000 dialogs in total. The confusion rate was set to 15% throughout. The simulated user had a different random seed in testing mode to the one set in training mode. The average reward, the success rate and the average number of turns are shown in Figs. 9–11 respectively. The error bars represent one standard error.

The results show that the GP-based policy reaches a performance comparable to the fully trained NAC policy in only 10,000 dialogs. This is consistent with related research which has shown that Gaussian processes can be deployed to guide gradient descent and improve its efficiency [38]. Also, the fully trained GP policy outperforms the NAC policy in terms of dialog length showing that parametric policy modeling limits the optimality of the solution.

The fully trained GP and NAC policies were also compared by performing 1000 simulated dialogs over a range of confusion rates. The average reward is given in Fig. 12.

The results show that the GP policy outperforms the NAC policy across confusion rates in the neighborhood of 15% confusion rate where the policies are trained. This shows that while

TABLE I
HUMAN EVALUATION OF SIMULATOR TRAINED POLICIES.

	#of dialogues	Reward	Success	Turns	#of dict points
NAC-PartlyTrained	402	8.9 ± 0.4	87.3 ± 1.7	8.5 ± 0.2	–
NAC-FullyTrained	415	11.9 ± 0.3	91.8 ± 1.3	6.5 ± 0.2	–
GP-PartlyTrained	400	12.5 ± 0.3	93.5 ± 1.2	6.2 ± 0.2	1569
GP-FullyTrained	397	11.6 ± 0.4	91.2 ± 1.4	6.6 ± 0.2	1699

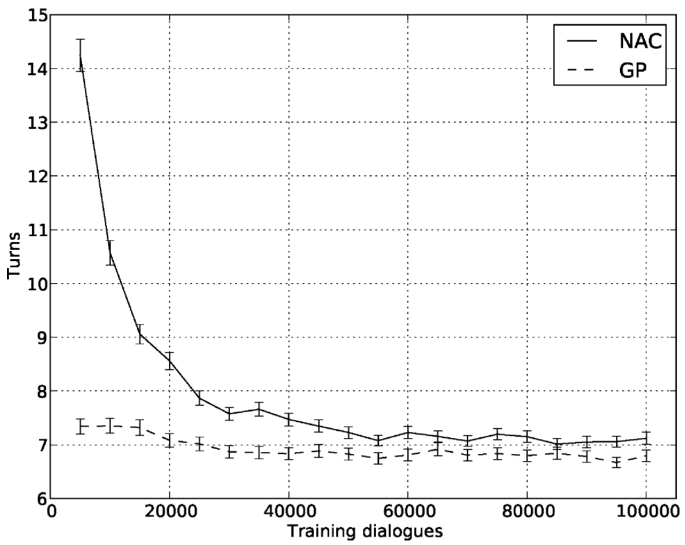

Fig. 11. Comparison of the performance of GP-Sarsa vs the natural actor critic algorithm, where the performance is measured as the average number of turns as a function of the number of training dialogs.

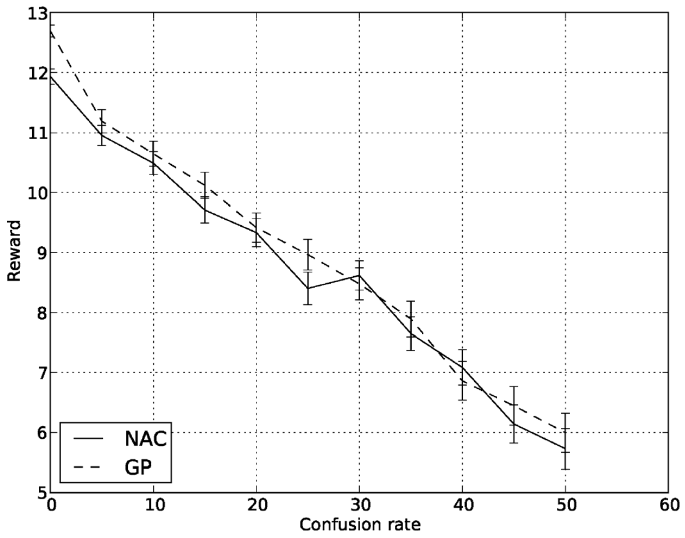

Fig. 12. Comparison of the performance of GP-Sarsa vs the natural actor critic algorithm, where the performance is measured as the average reward as a function of the confusion rate in the user input.

the GP has a potential for more robust performance, it is important that the training conditions match testing conditions.

Evaluating policy performance on the same simulated user as used for training may be misleading. Hence, the policies were also tested with human users recruited using the Amazon MTurk service. Four policies were compared: partially trained GP and NAC policies, using only 10,000 dialogs, and the fully trained GP and NAC policies.

The results are given in Table I, where for each policy the number of dialogs is given together with the average reward, the success rate, the average number of turns as well as their respective one-standard errors, and the number of dictionary points used by the GP sparsification algorithm. The average word error rate was 19%. These results show that the partially trained GP policy outperforms the partially trained NAC policy. Indeed, the partially trained GP policy outperforms all of the other policies including the fully trained GP policy. The latter result may from exploitation of traits in the simulated user that are not normally present in real human interaction.

2) Real World Application: GP-Sarsa has been previously used for training a dialog manager in direct interaction with real users [37], however, the summary state space used in that experiment was only four dimensional. In addition, the learning process exhibited inconsistencies due to the users not providing accurate ratings. Here we report on an on-line learning experiment with the Bayesian Update of Dialogue State manager using a refined reward function.

The GP-Sarsa algorithm was implemented in a live telephone-based spoken dialog system based on the BUDS framework operating with the summary space described in Section IV-A.2. The GP-Sarsa configuration consisted of the Gaussian kernel ($\sigma_k = 5$ and $p = 4$) with a stochastic policy model where $\eta = 3$, the noise of the residual $\sigma = 5$ and the sparsification threshold $\nu = 0.1$.

Human users were assigned specific tasks in the Cambridge restaurant domain using the Amazon MTurk service. At the end of each call, users were asked to press 1 if they were satisfied and 0 otherwise. Previous experience suggests that this subjective user feedback is not sufficiently robust to rely on solely for the reward function. Therefore, at the end of each call the objective success was also calculated by comparing the predefined task with the system actions. The user rating was then used to compute the reward function only if it agreed with the objective measure of success. It has been shown that this can result in better policy performance [37].

The performance achieved during on-line learning on the initial 1274 dialogs was compared to the performance of the policy trained on 10,000 dialogs using the simulated user (GPPartlyTrained policy from the previous section). The results are given in Fig. 13 where the moving average success is given for a window of 400 dialogs along with the 95% confidence interval. Of the 1274 dialogs, 1081 had the same subjective and objective success ratings and were used for training. It can be seen from Fig. 13 that the policy trained on-line reaches a comparable asymptotic reward as the simulator-trained policy but in substantially fewer training dialogs. Moreover, the learning curve

TABLE II
HUMAN EVALUATION OF THE POLICY TRAINED ONLINE.

	#of dialogues	Reward	Success	Turns	#of dict points
GP-OnlineTrained	410	13.4 ± 0.3	96.8 ± 0.9	6.0 ± 0.1	1132

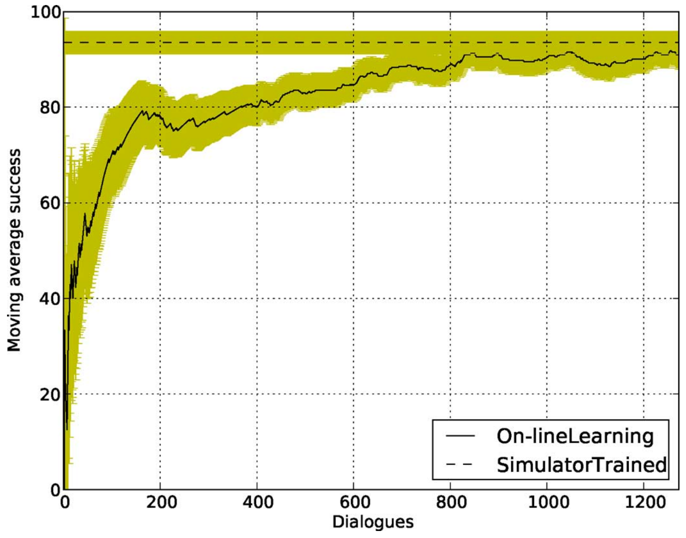

Fig. 13. Comparison of the performance of the policy trained with the simulated user and tested with real people with the policy that is optimized online with interaction with real people. The performance is measured as the moving average success as a function of the number of dialogs.

is smooth and it does not exhibit inconsistencies as was the case in earlier experiments [37] due to the refined definition of the reward function.

Finally, the policy trained on 1000 dialogs with real users was tested under the same conditions as for the simulator trained policies (i.e., the scaling factor $\eta = 1$). Comparing Table II with Table I, it can be seen that the policy trained on real users significantly outperforms the simulator trained policies.

D. *Policy Optimisation on the Full Belief Space*

The second major benefit of using Gaussian process policy optimization is its potential to substantially reduce designer effort in defining the policy representation by avoiding the need to construct *ad hoc* feature mapping functions. In this section, the ability to optimize a policy on the full belief space is examined in practice.

1) The Kernel Function on the Full Belief Space: In the case of the full belief space, the kernel function must be defined over distributions. For the experiments described here, the kernel functions were defined following the approaches described in [39], [40] but modified to suit the dialog system application. The kernel function was constructed from the sum of individual kernels over the distributions $\mathbf{b}_k$ for each hidden node k in the Bayesian network belief state $\mathbf{b}$.

We examined three kernel functions. The first is the expected likelihood kernel, which is also a simple linear inner product:

$$k_{\mathcal{B}}(\mathbf{b}, \mathbf{b}') = \sum_k \langle \mathbf{b}_k, \mathbf{b}'_k \rangle, \tag{24}$$

the expected value of distribution $\mathbf{b}_k$ under distribution $\mathbf{b}'_k$.

The second is the Bhattacharyya kernel

$$k_{\mathcal{B}}(\mathbf{b}, \mathbf{b}') = \sum_k \sum_{i=0}^{i<d_k} \sqrt{\mathbf{b}_k(i)}\sqrt{\mathbf{b}'_k(i)}, \tag{25}$$

where d_k is the dimensionality of the k concept in the Bayesian network. It is related to Hellinger's distance, which can be viewed as a symmetric approximation of the Kullback-Leibler divergence [39].

The third kernel that we compare is given by

$$k_{\mathcal{B}}(\mathbf{b}, \mathbf{b}') = \frac{-1}{\log(2)} \sum_k \sum_{i=0}^{i<d_k} \mathbf{b}_k(i) \log\left(\frac{\mathbf{b}_k(i)}{\mathbf{b}_k(i) + \mathbf{b}'_k(i)}\right) + \mathbf{b}'_k(i) \log\left(\frac{\mathbf{b}'_k(i)}{\mathbf{b}_k(i) + \mathbf{b}'_k(i)}\right). \tag{26}$$

This is related to Jansen-Shannon divergence, a symmetric and smoothed variant of the Kullback-Leibler divergence [40].

The kernel functions of the history nodes are defined directly over their respective distributions. While it is possible to calculate the kernel function for the goal nodes in the same way, in this case, the choice of system action, such as *confirm* or *inform*, does not depend on the distribution of actual values, rather it depends on the shape of the distribution. Therefore, the kernel functions for goal nodes are calculated over vectors, where each vector represents the corresponding distribution sorted into order of probability. The only exceptions are the *method* goal node and the *discourse act* node. The former defines whether the user is searching for a venue *by name* or *by constraints* and the latter defines which discourse act the user used, e.g.. *acknowledgement*, *thank you*. Their kernels are calculated in the same way as for the history nodes.

The sparsification threshold ν for this space is set at 0.001. Some intermediate experimentation showed that the kernel requires a finer span, so the threshold here is low. Since the same reward function is used for both the summary space and the full belief space, the variance of the Q-function residual σ was also fixed at 5. The sampling scale η was set to 3.

2) Training and Evaluation on Simulated User: The GP-Sarsa stochastic policy (13) that operates on the summary space was compared to the two policies operating on the full belief space that use different kernel functions in the same training schedule as described in Section IV-A.4.

The results are shown in Fig. 14. As can be seen, the GP policies trained on the full belief space do converge to an optimum, with the Bhattacharyya kernel performing best. Furthermore, increasing the dimensionality of the input space from the summary space to the full belief space did not slow down the process of learning. Overall performance is not however competitive with the summary space stochastic policy. This is disappointing but perhaps not surprising since defining a feature based summary space mapping can be viewed as designing a special kind of kernel that is hand-crafted to suit this task well. This is also

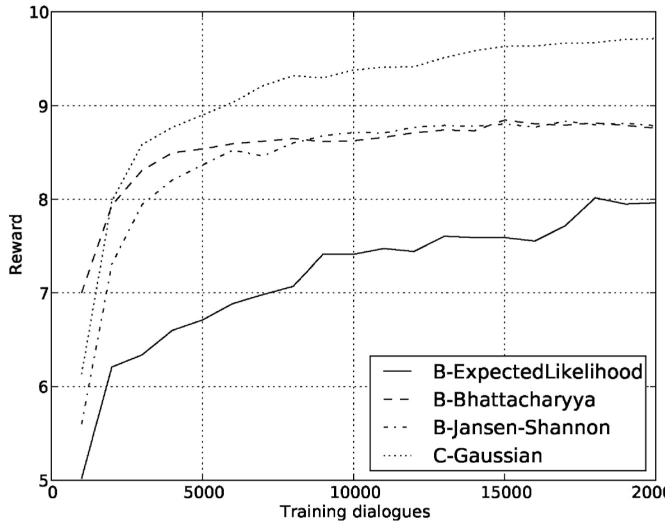

Fig. 14. Comparison of the performance of 3 different policies operating on the full belief space ($\mathcal{B}$) with a policy operating on the summary space ($\mathcal{C}$).

the first attempt at designing a general-purpose kernel to fit an arbitrarily large belief space, and further work is needed to refine this process.

V. Conclusion

This paper has described how Gaussian processes can be successfully applied to POMDP-based dialog management. Various aspects of Gaussian process reinforcement learning have been discussed, including sparse approximations, the use of a GP-based stochastic policy model and the GP-Sarsa algorithm.

Using a system based on the Bayesian Update of Dialogue State (BUDS) dialog manager, various issues have been investigated experimentally both in simulation and with real users. The principal results are firstly that GP-based policy optimization is faster than conventional gradient based methods and furthermore, performance in the early stages of learning becomes acceptable very quickly making it feasible to train systems from scratch directly in interaction with real users. Secondly, policies trained with real users significantly outperform policies trained on user simulators. Thirdly, although there is clearly more work to be done, it is feasible to construct a GP-based policy directly on the full POMDP belief space obviating the need for hand-crafting a summary space mapping. These three results represent an important step towards building flexible dialog systems that can range over large and varied domains dynamically adapting to the local context.

Future research in this area will need to address a number of issues. Firstly, methods are needed for optimizing the kernel function parameters. It has been shown on a toy dialog problem that these parameters can be learnt off-line directly from data by maximizing the marginal likelihood (9)[23], [24]. This type of approach needs to be extended to work effectively in a full-scale real-world dialog manager.

Secondly, the work presented here still requires the use of a summary action space and heuristic mapping functions to generate actual system responses. For the experiments here, a simple δ-kernel was used as the kernel function over the summary action space. However, Gaussian process reinforcement learning should allow more elaborate kernel functions to be defined which can be applied directly to the full action space. Since the full action space is not finite, sampling methods would then need to be developed to select an appropriate system response as in [19].

Finally, more work is needed to develop effective training strategies which allow optimization of GP-Sarsa policies operating on the full belief state space in direct interaction with real users. The preliminary experiments reported in this paper suggest this is possible, but work is clearly needed to achieve performance levels which are competitive with current summary space models. A related issue concerns the design of reward functions, and robust methods of computing them on-line. In all of the work described in this paper, a simple fixed reward for success is given at the end of the dialog in conjunction with a per turn penalty and the primary measure of success is determined by asking the user to give feedback. This learning regime would not be possible in real applications. Ideally the reward needs to be computed without the conscious engagement of users and it needs to be refined to encourage the type of desirable dialog behaviors that are embodied in the design of current hand-crafted spoken dialog systems.

Acknowledgment

The authors would like to thank B. Thomson, who built the Bayesian Update of Dialogue State system used here, J. Schatzmann and S. Keizer, who built the user simulator and P. Tsiakoulis for his help with the user trials. The authors would also like to thank M. Henderson, K. Yu, M. Szummer, C. Breslin and D. Kim for useful comments and discussions.

References

[1] S. Young, "Talking to machines (Statistically speaking)," in *Proc. ICSLP*, 2002.
[2] N. Roy, J. Pineau, and S. Thrun, "Spoken dialogue management using probabilistic reasoning," in *Proc. ACL*, 2000.
[3] B. Zhang, Q. Cai, J. Mao, E. Chang, and B. Guo, "Spoken dialogue management as planning and acting under uncertainty," in *Proc. Eurospeech*, 2001.
[4] J. Williams and S. Young, "Partially observable Markov decision processes for spoken dialog systems," *Comput. Speech Lang.*, vol. 21, no. 2, pp. 393–422, 2007.
[5] B. Thomson, "Statistical methods for spoken dialogue management," Ph.D., Univ. of Cambridge, Cambridge, U.K., 2009.
[6] 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.
[7] M. Gašić, S. Keizer, F. Mairesse, J. Schatzmann, B. Thomson, K. Yu, and S. Young, "Training and evaluation of the HIS-POMDP dialogue system in noise," in *Proc. SIGDIAL*, 2008.
[8] F. Jurčíček, B. Thomson, and S. Young, "Natural actor and belief critic: Reinforcement algorithm for learning parameters of dialogue systems modelled as POMDPs," *ACM Trans. Speech Lang. Process.*, pp. 6:1–6:26, 2011.
[9] L. Kaelbling, M. Littman, and A. Cassandra, "Planning and acting in partially observable stochastic domains," *Artif. Intell.*, vol. 101, pp. 99–134, 1998.
[10] J. Pineau, G. Gordon, and S. Thrun, "Point-based value iteration: An anytime algorithm for POMDPs," in *Proc. IJCAI*, 2003, pp. 1025–1032.
[11] J. Williams and S. Young, "Scaling POMDPs for spoken dialog management," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 15, no. 7, pp. 2116–2129, Sep. 2007.
[12] J. Henderson, O. Lemon, and K. Georgila, "Hybrid reinforcement/supervised learning for dialogue policies from fixed data sets," *Comput. Linguist.*, vol. 34, no. 4, pp. 487–511, 2008.

[13] B. Thomson and S. Young, "Bayesian update of dialogue state: A POMDP framework for spoken dialogue systems," *Comput. Speech Lang.*, vol. 24, no. 4, pp. 562–588, 2010.
[14] L. Li, J. Williams, and S. Balakrishnan, "Reinforcement learning for dialog management using least-squares policy iteration and fast feature selection," in *Proc. Interspeech*, 2009.
[15] P. Crook and O. Lemon, "Lossless value directed compression of complex user goal states for statistical spoken dialogue systems," in *Proc. Interspeech*, 2011.
[16] L. Daubigney, M. Geist, and O. Pietquin, "Off-policy learning in large-scale POMDP-based dialogue systems," in *Proc. ICASSP*, 2012, pp. 4989–4992.
[17] Amazon Mechanical Turk Amazon, 2011 [Online]. Available: https://www.mturk.com/mturk/welcome
[18] Y. Engel, S. Mannor, and R. Meir, "Bayes meets Bellman: The Gaussian process approach to temporal difference learning," in *Proc. ICML*, 2003.
[19] Y. Engel, S. Mannor, and R. Meir, "Reinforcement learning with Gaussian processes," in *Proc. ICML*, 2005.
[20] C. E. Rasmussen and M. Kuss, "Gaussian processes in reinforcement learning," in *Advances in Neural Information Processing Systems*. Cambridge, MA, USA: MIT Press, 2004, vol. 16, pp. 751–759.
[21] M. Deisenroth, C. Rasmussen, and J. Peters, "Gaussian process dynamic programming," *Neurocomputing*, vol. 72, no. 7-9, pp. 1508–1524, 2009.
[22] C. Rasmussen and C. Williams*, Gaussian Processes for Machine Learning*. Cambridge, MA, USA: MIT Press, 2005.
[23] M. Gašić, "Statistical dialogue modelling," Ph.D. dissertation, Univ. of Cambridge, Cambridge, U.K., 2011.
[24] M. Gašić, F. Jurčíček, S. Keizer, F. Mairesse, J. Schatzmann, B. Thomson, K. Yu, and S. Young, "Gaussian processes for fast policy optimisation of POMDP-based dialogue managers," in *Proc. SIGDIAL*, 2010.
[25] D. Cohn, L. Atlas, and R. Ladner, "Improving generalization with active learning," *Mach. Learn.*, vol. 15, pp. 201–221, 1994.
[26] D. J. C. MacKay, "Information-based objective functions for active data selection," *Neural Comput.*, vol. 4, no. 4, pp. 590–604, 1992.
[27] J. Quinonero-Candela and C. Rasmussen, "A unifying view of sparse approximate Gaussian process regression," *J. Mach. Learn. Res.*, vol. 6, pp. 1939–1959, 2005.
[28] Y. Engel, "Algorithms and representations for reinforcement learning," Ph.D. dissertation, Hebrew Univ., Jerusalem, Israel, 2005.
[29] D. Nguyen-Tuong and J. Peters, "Incremental sparsication for real-time online model learning," in *Proc. AISTATS '10*, 2010.
[30] B. Scholkopf and A. Smola*, Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: MIT Press, 2001.
[31] J. Peters and S. Schaal, "Natural actor-critic," *Neurocomputing*, vol. 71, pp. 1180–1190, 2008.
[32] J. Schatzmann, "Statistical user and error modelling for spoken dialogue systems," Ph.D. dissertation, Univ. of Cambridge, Cambridge, U.K., 2008.
[33] S. Keizer, M. Gašić, F. Jurčíček, F. Mairesse, B. Thomson, K. Yu, and S. Young, "Parameter estimation for agenda-based user simulation," in *Proc. SIGDIAL*, 2010.
[34] B. Thomson, M. Gasic, M. Henderson, P. Tsiakoulis, and S. Young, "N-best error simulation for training spoken dialogue systems," in *Proc. SLT*, 2012.
[35] 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*, 2011.
[36] 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*, 2011.
[37] 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. ASRU*, 2011.
[38] H. Jakab and L. Csató, "Reinforcement learning with guided policy search using Gaussian processes," in *Proc. Int. Joint Conf. Neural Networks (IJCNN)*, 2012, pp. 1–8.
[39] T. Jebara, R. Kondor, and A. Howard, "Probability product kernels," *J. Mach. Learn. Res.*, vol. 5, pp. 819–844, Dec. 2004.
[40] M. Hein and O. Bousquet, "Hilbertian metrics and positive definite kernels on probability measures," in *Proc. AISTATS '05*, 2004.

Milica Gašić is a Research Associate in the Dialogue Systems Group at the University of Cambridge. She graduated in Computer Science and Mathematics from the University of Belgrade in 2006. Since then she has been at the University of Cambridge first as an MPhil student reading Computer Speech, Text and Internet Technology. Subsequently, she did a PhD in Statistical Dialogue Modeling in 2011. She has published around 20 peer-reviewed conference and journal papers in the area of dialog management. She served on the organizing committee for Young Researcher's Roundtable on Spoken Dialogue Systems in 2009.

Steve Young received a BA in Electrical Sciences from Cambridge University in 1973 and a PhD in Speech Processing in 1978. He was elected to the Chair of Information Engineering at Cambridge University in 1994. He was a co-founder and Technical Director of Entropic Ltd from 1995 until 1999 when the company was taken over by Microsoft. After a short period as an Architect at Microsoft, he returned full-time to Cambridge University in January 2001 where he was Head of Information Engineering until 2009 and is now Senior Pro-Vice-Chancellor.

His research interests include speech recognition, expressive synthesis and spoken dialog systems. He has written and edited books on software engineering and speech processing, and he has published as author and co-author, more than 200 papers in these areas. He is a Fellow of the UK Royal Academy of Engineering, the IET, IEEE and RSA. He served as the senior editor of Computer Speech and Language from 1993 to 2004, and he was Chair of the IEEE Speech and Language Technical Committee from 2008 to 2010. He was the recipient of an IEEE Signal Processing Society Technical Achievement Award in 2004. He was elected a Fellow of the International Speech Communication Association (ISCA) in 2009 and he was the recipient of the ISCA Medal in 2010 for Scientific Achievement.