# Practical Kernel-Based Reinforcement Learning[†]

André M. S. Barreto
Laboratório Nacional de Computação Científica
Petrópolis, Brazil

Doina Precup and Joelle Pineau
McGill University
Montreal, Canada

## Abstract

*Kernel-based reinforcement learning* (KBRL) stands out among approximate reinforcement learning algorithms for its strong theoretical guarantees. By casting the learning problem as a local kernel approximation, KBRL provides a way of computing a decision policy which is statistically consistent and converges to a unique solution. Unfortunately, the model constructed by KBRL grows with the number of sample transitions, resulting in a computational cost that precludes its application to large-scale or on-line domains. In this paper we introduce an algorithm that turns KBRL into a practical reinforcement learning tool. *Kernel-based stochastic factorization* (KBSF) builds on a simple idea: when a transition probability matrix is represented as the product of two stochastic matrices, one can swap the factors of the multiplication to obtain another transition matrix, potentially much smaller than the original, which retains some fundamental properties of its precursor. KBSF exploits such an insight to compress the information contained in KBRL's model into an approximator of fixed size. This makes it possible to build an approximation that takes into account both the difficulty of the problem and the associated computational cost. KBSF's computational complexity is linear in the number of sample transitions, which is the best one can do without discarding data. Moreover, the algorithm's simple mechanics allow for a fully incremental implementation that makes the amount of memory used independent of the number of sample transitions. The result is a kernel-based reinforcement learning algorithm that can be applied to large-scale problems in both off-line and on-line regimes. We derive upper bounds for the distance between the value functions computed by KBRL and KBSF using the same data. We also prove that it is possible to control the magnitude of the variables appearing in our bounds, which means that, given enough computational resources, we can make KBSF's value function as close as desired to the value function that would be computed by KBRL using the same set of sample transitions. The potential of our algorithm is demonstrated in an extensive empirical study in which KBSF is applied to difficult tasks based on real-world data. Not only does KBSF solve problems that had never been solved before, it also significantly outperforms other state-of-the-art reinforcement learning algorithms on the tasks studied.

[†] Parts of the material presented in this technical report have appeared before in two papers published in the *Neural Information Processing Systems* conference (NIPS, Barreto et al., 2011, 2012). The current manuscript is a substantial extension of the aforementioned works.

# 1 Introduction

Reinforcement learning provides a conceptual framework with the potential to materialize a long-sought goal in artificial intelligence: the construction of situated agents that learn how to behave from direct interaction with the environment (Sutton and Barto, 1998). But such an endeavor does not come without its challenges; among them, extrapolating the field's basic machinery to large-scale domains has been a particularly persistent obstacle.

It has long been recognized that virtually any real-world application of reinforcement learning must involve some form of approximation. Given the mature stage of the supervised-learning theory, and considering the multitude of approximation techniques available today, this realization may not come across as a particularly worrisome issue at first glance. However, it is well known that the sequential nature of the reinforcement learning problem renders the incorporation of function approximators non-trivial (Bertsekas and Tsitsiklis, 1996).

Despite the difficulties, in the last two decades the collective effort of the reinforcement learning community has given rise to many reliable approximate algorithms (Szepesvári, 2010). Among them, Ormoneit and Sen's (2002) kernel-based reinforcement learning (KBRL) stands out for two reasons. First, unlike other approximation schemes, KBRL always converges to a unique solution. Second, KBRL is consistent in the statistical sense, meaning that adding more data always improves the quality of the resulting policy and eventually leads to optimal performance.

Unfortunately, the good theoretical properties of KBRL come at a price: since the model constructed by this algorithm grows with the number of sample transitions, the cost of computing a decision policy quickly becomes prohibitive as more data become available. Such a computational burden severely limits the applicability of KBRL. This may help explain why, in spite of its nice theoretical guarantees, kernel-based learning has not been widely adopted as a practical reinforcement learning tool.

This paper presents an algorithm that can potentially change this situation. *Kernel-based stochastic factorization* (KBSF) builds on a simple idea: when a transition probability matrix is represented as the product of two stochastic matrices, one can swap the factors of the multiplication to obtain another transition matrix, potentially much smaller than the original, which retains some fundamental properties of its precursor (Barreto and Fragoso, 2011). KBSF exploits this insight to compress the information contained in KBRL's model into an approximator of fixed size. In other words, KBSF builds a model, whose size is independent of the number of sample transitions, which serves as an approximation of the model that would be constructed by KBRL. Since the size of the model becomes a parameter of the algorithm, KBSF essentially detaches the structure of KBRL's approximator from its configuration. This extra flexibility makes it possible to build an approximation that takes into account *both* the difficulty of the problem *and* the computational cost of finding a policy using the constructed model.

KBSF's computational complexity is linear in the number of sample transitions, which is the best one can do without throwing data away. Moreover, we show in the paper that the amount of memory used by our algorithm is independent of the number of sample transitions. Put together, these two properties make it possible to apply KBSF to large-scale problems in both off-line and on-line regimes. To illustrate this possibility in practice, we present an extensive empirical study in which KBSF is applied to difficult control tasks based on real-world data, some of which had never been solved before. KBSF outperforms least-squares policy iteration and fitted $Q$-iteration on several off-line problems and SARSA on a difficult on-line task.

We also show that KBSF is a sound algorithm from a theoretical point of view. Specifically, we derive results bounding the distance between the value function computed by our algorithm and the one computed by KBRL using the same data. We also prove that it is possible to control the magnitude of the variables appearing in our bounds, which means that we can make the difference between KBSF's and KBRL's solutions arbitrarily small.

We start the paper presenting some background material in Section . Then, in Section , we introduce the stochastic-factorization trick, the insight underlying the development of our algorithm. KBSF itself is presented in Section . This section is divided in two parts, one theoretical and one practical. In Section  we present theoretical results showing not only that the difference between KBSF's and KBRL's value functions is bounded, but also that such a difference can be controlled. Section  brings experiments with KBSF on four reinforcement-learning problems: single and double pole-balancing, HIV drug schedule domain, and epilepsy suppression task. In Section  we introduce the incremental version of our algorithm, which can be applied to on-line problems. This section follows the same structure of Section , with theoretical results followed by experiments. Specifically, in Section  we extend the results of Section  to the on-line scenario, and in Section  we present experiments on the triple pole-balancing and helicopter tasks. In Section  we discuss the impact of deviating from theoretical assumptions over KBSF's performance, and also present a practical guide on how to configure our algorithm to solve a reinforcement learning problem. In Section  we summarize related works and situate KBSF in the context of kernel-based learning. Finally, in Section  we present the main conclusions regarding the current research and discuss some possibilities of future work.

## 2 Background

We consider the standard framework of reinforcement learning, in which an agent interacts with an environment and tries to maximize the amount of reward collected in the long run (Sutton and Barto, 1998). The interaction between agent and environment happens at discrete time steps: at each instant $t$ the agent occupies a state $s_{(t)} \in S$ and must choose an action $a$ from a finite set $A$. The sets $S$ and $A$ are called the state and action spaces, respectively. The execution of action $a$ in state $s_{(t)}$ moves the agent to a new state $s_{(t+1)}$, where a new action must be selected, and so on. Each transition has a certain probability of occurrence and is associated with a reward $r \in \mathbb{R}$. The goal of the agent is to find a policy $\pi : S \mapsto A$, that is, a mapping from states to actions, that maximizes the expected return. Here we define the return from time $t$ as:

$$R_{(t)} = r_{(t+1)} + \gamma r_{(t+2)} + \gamma^2 r_{(t+3)} + ... = \sum_{i=1}^{\infty} \gamma^{i-1} r_{(t+i)}, \tag{1}$$

where $r_{(t+1)}$ is the reward received at the transition from state $s_{(t)}$ to state $s_{(t+1)}$. The parameter $\gamma \in [0, 1)$ is the discount factor, which determines the relative importance of individual rewards depending on how far in the future they are received.

## 2.1 Markov Decision Processes

As usual, we assume that the interaction between agent and environment can be modeled as a *Markov decision process* (MDP, Puterman, 1994). An MDP is a tuple $M \equiv (S, A, P^a, R^a, \gamma)$, where $P^a$ and $R^a$ describe the dynamics of the task at hand. For each action $a \in A$, $P^a(\cdot|s)$ defines the next-state distribution upon taking action $a$ in state $s$. The reward received at transition $s \xrightarrow{a} s'$ is given by $R^a(s, s')$, with $|R^a(s, s')| \leq R_{\max} < \infty$. Usually, one is interested in the expected reward resulting from the execution of action $a$ in state $s$, that is, $r^a(s) = E_{s' \sim P^a(\cdot|s)}\{R^a(s, s')\}$.

Once the interaction between agent and environment has been modeled as an MDP, a natural way of searching for an optimal policy is to resort to *dynamic programming* (Bellman, 1957). Central to the theory of dynamic-programming is the concept of a *value function*. The value of state $s$ under a policy $\pi$, denoted by $V^\pi(s)$, is the expected return the agent will receive from $s$ when following $\pi$, that is, $V^\pi(s) = E_\pi\{R_{(t)}|s_{(t)} = s\}$ (here the expectation is over all possible sequences of rewards in (1) when the agent follows $\pi$). Similarly, the value of the state-action pair $(s, a)$ under policy $\pi$ is defined as $Q^\pi(s, a) = E_{s' \sim P^a(\cdot|s)}\{R^a(s, s') + \gamma V^\pi(s')\} = r^a(s) + \gamma E_{s' \sim P^a(\cdot|s)}\{V^\pi(s')\}$.

The notion of value function makes it possible to impose a partial ordering over decision policies. In particular, a policy $\pi'$ is considered to be at least as good as another policy $\pi$ if $V^{\pi'}(s) \geq V^\pi(s)$ for all $s \in S$. The goal of dynamic programming is to find an optimal policy $\pi^*$ that performs no worse than any other. It is well known that there always exists at least one such policy for a given MDP (Puterman, 1994). When there is more than one optimal policy, they all share the same value function $V^*$.

When both the state and action spaces are finite, an MDP can be represented in matrix form: each function $P^a$ becomes a matrix $\mathbf{P}^a \in \mathbb{R}^{|S| \times |S|}$, with $p^a_{ij} = P^a(s_j|s_i)$, and each function $r^a$ becomes a vector $\mathbf{r}^a \in \mathbb{R}^{|S|}$, where $r^a_i = r^a(s_i)$. Similarly, $V^\pi$ can be represented as a vector $\mathbf{v}^\pi \in \mathbb{R}^{|S|}$ and $Q^\pi$ can be seen as a matrix $\mathbf{Q}^\pi \in \mathbb{R}^{|S| \times |A|}$. Throughout the paper we will use the conventional and matrix notations interchangeably, depending on the context. When using the latter, vectors will be denoted by small boldface letters and matrices will be denoted by capital boldface letters.

When the MDP is finite, dynamic programming can be used to find an optimal decision-policy $\pi^* \in A^{|S|}$ in time polynomial in the number of states $|S|$ and actions $|A|$ (Ye, 2011). Let $\mathbf{v} \in \mathbb{R}^{|S|}$ and let $\mathbf{Q} \in \mathbb{R}^{|S| \times |A|}$. Define the operator $\Gamma : \mathbb{R}^{|S| \times |A|} \mapsto \mathbb{R}^{|S|}$ such that $\Gamma\mathbf{Q} = \mathbf{v}$ if and only if $v_i = \max_j q_{ij}$ for all $i$. Also, given an MDP $M$, define $\Delta : \mathbb{R}^{|S|} \mapsto \mathbb{R}^{|S| \times |A|}$ such that $\Delta\mathbf{v} = \mathbf{Q}$ if and only if $q_{ia} = r^a_i + \gamma \sum_{j=1}^{|S|} p^a_{ij} v_j$ for all $i$ and all $a$. The *Bellman operator* of the MDP $M$ is given by $T \equiv \Gamma\Delta$. A fundamental result in dynamic programming states that, starting from $\mathbf{v}^{(0)} = \mathbf{0}$, the expression $\mathbf{v}^{(t)} = T\mathbf{v}^{(t-1)} = \Gamma\mathbf{Q}^{(t)}$

gives the optimal $t$-step value function, and as $t \to \infty$ the vector $\mathbf{v}^{(t)}$ approaches $\mathbf{v}^*$. At any point, the optimal $t$-step policy can be obtained by selecting $\pi_i^{(t)} \in \operatorname{argmax}_j q_{ij}^{(t)}$ (Puterman, 1994).

In contrast with dynamic programming, in reinforcement learning it is assumed that the MDP is unknown, and the agent must learn a policy based on transitions sampled from the environment. If the process of learning a decision policy is based on a fixed set of sample transitions, we call it *batch* reinforcement learning. On the other hand, in *on-line* reinforcement learning the computation of a decision policy takes place concomitantly with the collection of data (Sutton and Barto, 1998).

## 2.2 Kernel-Based Reinforcement Learning

Kernel-based reinforcement learning (KBRL) is a batch algorithm that uses a finite model approximation to solve a continuous MDP $M \equiv (\mathbb{S}, A, P^a, R^a, \gamma)$, where $\mathbb{S} \subseteq [0,1]^{d_\mathbb{S}}$ (Ormoneit and Sen, 2002). Let $S^a \equiv \{(s_k^a, r_k^a, \hat{s}_k^a) | k = 1, 2, ..., n_a\}$ be sample transitions associated with action $a \in A$, where $s_k^a, \hat{s}_k^a \in \mathbb{S}$ and $r_k^a \in \mathbb{R}$. Let $\phi : \mathbb{R}^+ \mapsto \mathbb{R}^+$ be a Lipschitz continuous function satisfying $\int_0^1 \phi(x)dx = 1$. Let $\mathrm{k}_\tau(s, s')$ be a kernel function defined as

$$\mathrm{k}_\tau(s, s') = \phi\left(\frac{\|s - s'\|}{\tau}\right), \tag{2}$$

where $\tau \in \mathbb{R}$ and $\|\cdot\|$ is a norm in $\mathbb{R}^{d_\mathbb{S}}$ (for concreteness, the reader may think of $\mathrm{k}_\tau(s, s')$ as the Gaussian kernel, although the definition also encompasses other functions). Finally, define the normalized kernel function associated with action $a$ as

$$\kappa_\tau^a(s, s_i^a) = \frac{\mathrm{k}_\tau(s, s_i^a)}{\sum_{j=1}^{n_a} \mathrm{k}_\tau(s, s_j^a)}. \tag{3}$$

KBRL uses (3) to build a finite MDP whose state space $\hat{S}$ is composed solely of the $n = \sum_a n_a$ states $\hat{s}_i^a$ (if a given state $s \in \mathbb{S}$ occurs more than once in the set of sample transitions, each occurrence will be treated as a distinct state in the finite MDP). The transition functions of KBRL's model, $\hat{P}^a : \hat{S} \times \hat{S} \mapsto [0,1]$, are given by:

$$\hat{P}^a\left(\hat{s}_i^b | s\right) = \begin{cases} \kappa_\tau^a(s, s_i^b), \text{ if } a = b, \\ 0, \text{ otherwise,} \end{cases} \tag{4}$$

where $a, b \in A$. Similarly, the reward functions of the MDP constructed by KBRL, $\hat{R}^a : \hat{S} \times \hat{S} \mapsto \mathbb{R}$, are

$$\hat{R}^a(s, \hat{s}_i^b) = \begin{cases} r_i^a, \text{ if } a = b, \\ 0, \text{ otherwise.} \end{cases} \tag{5}$$

Based on (4) and (5) we can define the transition matrices and expected-reward vectors of KBRL's MDP. The matrices $\hat{\mathbf{P}}^a$ are derived directly from the definition of $\hat{P}^a(\hat{s}_i^b|s)$. The vectors of expected rewards $\hat{\mathbf{r}}^a$ are computed as follows. Let $\mathbf{r} \equiv [(\mathbf{r}^1)^\intercal, (\mathbf{r}^2)^\intercal, ..., (\mathbf{r}^{|A|})^\intercal]^\intercal \in$

$\mathbb{R}^n$, where $\mathbf{r}^a \in \mathbb{R}^{n_a}$ are the vectors composed of the sampled rewards $r_i^a$. Since $R^a(s, \hat{s}_i^b)$ does not depend on the start state $s$, we can write

$$\hat{\mathbf{r}}^a = \hat{\mathbf{P}}^a \mathbf{r}. \tag{6}$$

KBRL's MDP is thus given by $\hat{M} \equiv (\hat{S}, A, \hat{\mathbf{P}}^a, \hat{\mathbf{r}}^a, \gamma)$.

Once $\hat{M}$ has been defined, one can use dynamic programming to compute its optimal value function $\hat{V}^*$. Then, the value of any state-action pair of the continuous MDP can be determined as:

$$\hat{Q}(s,a) = \sum_{i=1}^{n_a} \kappa_\tau^a(s, s_i^a) \left[ r_i^a + \gamma \hat{V}^*(\hat{s}_i^a) \right], \tag{7}$$

where $s \in \mathbb{S}$ and $a \in A$. Ormoneit and Sen (2002) have shown that, if $n_a \to \infty$ for all $a \in A$ and the kernel's width $\tau$ shrink at an "admissible" rate, the probability of choosing a suboptimal action based on $\hat{Q}(s,a)$ converges to zero (see their Theorem 4).

As discussed, using dynamic programming one can compute the optimal value function of $\hat{M}$ in time polynomial in the number of sample transitions $n$ (which is also the number of states in $\hat{M}$). However, since each application of the Bellman operator $\hat{T}$ is $O(n^2|A|)$, the computational cost of such a procedure can easily become prohibitive in practice. Thus, the use of KBRL leads to a dilemma: on the one hand one wants as much data as possible to describe the dynamics of the task, but on the other hand the number of transitions should be small enough to allow for the numerical solution of the resulting model. In the following sections we describe a practical approach to weight the relative importance of these two conflicting objectives.

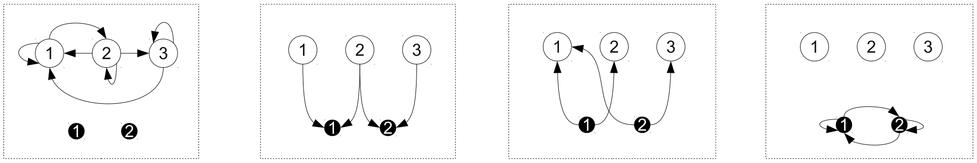
# 3 Stochastic Factorization

A *stochastic matrix* has only nonnegative elements and each of its rows sums to 1. That said, we can introduce the concept that will serve as a cornerstone for the rest of the paper:

**Definition 1.** *Given a stochastic matrix* $\mathbf{P} \in \mathbb{R}^{n \times p}$*, the relation* $\mathbf{P} = \mathbf{DK}$ *is called a* stochastic factorization *of* $\mathbf{P}$ *if* $\mathbf{D} \in \mathbb{R}^{n \times m}$ *and* $\mathbf{K} \in \mathbb{R}^{m \times p}$ *are also stochastic matrices. The integer* $m > 0$ *is the* order *of the factorization.*

This mathematical construct has been explored before. For example, Cohen and Rothblum (1991) briefly discuss it as a special case of nonnegative matrix factorization, while Cutler and Breiman (1994) focus on slightly modified versions of the stochastic factorization for statistical data analysis. However, in this paper we will focus on a useful property of this type of factorization that has only recently been noted (Barreto and Fragoso, 2011).

## 3.1 Stochastic-Factorization Trick



$$\mathbf{P} = \begin{bmatrix} \times & \times & 0 \\ \times & \times & \times \\ \times & 0 & \times \end{bmatrix} \quad \mathbf{D} = \begin{bmatrix} \times & 0 \\ \times & \times \\ 0 & \times \end{bmatrix} \quad \mathbf{K} = \begin{bmatrix} \times & \times & 0 \\ \times & 0 & \times \end{bmatrix} \quad \bar{\mathbf{P}} = \begin{bmatrix} \times & \times \\ \times & \times \end{bmatrix}$$

Figure 1: Reducing the dimension of a transition model from $n = 3$ states to $m = 2$ artificial states. The original states $s_i$ are represented as big white circles; small black circles depict artificial states $\bar{s}_h$. The symbol '×' is used to represent nonzero elements.

Let $\mathbf{P} \in \mathbb{R}^{n \times n}$ be a transition matrix, that is, a square stochastic matrix, and let $\mathbf{P} = \mathbf{DK}$ be an order $m$ stochastic factorization. In this case, one can see the elements of $\mathbf{D}$ and $\mathbf{K}$ as probabilities of transitions between the states $s_i$ and a set of $m$ artificial states $\bar{s}_h$. Specifically, the elements in each row of $\mathbf{D}$ can be interpreted as probabilities of transitions from the original states to the artificial states, while the rows of $\mathbf{K}$ can be seen as probabilities of transitions in the opposite direction. Under this interpretation, each element $p_{ij} = \sum_{h=1}^{m} d_{ih} k_{hj}$ is the sum of the probabilities associated with $m$ two-step transitions: from state $s_i$ to each artificial state $\bar{s}_h$ and from these back to state $s_j$. In other words, $p_{ij}$ is the accumulated probability of all possible paths from $s_i$ to $s_j$ with a stopover in one of the artificial states $\bar{s}_h$. Following similar reasoning, it is not difficult to see that by *swapping* the factors of a stochastic factorization, that is, by switching from $\mathbf{DK}$ to $\mathbf{KD}$, one obtains the transition probabilities between the artificial states $\bar{s}_h$, $\bar{\mathbf{P}} = \mathbf{KD}$. If $m < n$, $\bar{\mathbf{P}} \in \mathbb{R}^{m \times m}$ will be a compact version of $\mathbf{P}$. Figure 1 illustrates this idea for the case in which $n = 3$ and $m = 2$.

The stochasticity of $\bar{\mathbf{P}}$ follows immediately from the same property of $\mathbf{D}$ and $\mathbf{K}$. What is perhaps more surprising is the fact that this matrix shares some fundamental characteristics with the original matrix $\mathbf{P}$. Specifically, it is possible to show that: $(i)$ for each recurrent class in $\mathbf{P}$ there is a corresponding class in $\bar{\mathbf{P}}$ with the same period and, given some simple assumptions about the factorization, $(ii)$ $\mathbf{P}$ is irreducible if and only if $\bar{\mathbf{P}}$ is irreducible and $(iii)$ $\mathbf{P}$ is regular if and only if $\bar{\mathbf{P}}$ is regular (for details, see the article by Barreto and Fragoso, 2011). We will refer to this insight as the "*stochastic-factorization trick*":

*Given a stochastic factorization of a transition matrix,* $\mathbf{P} = \mathbf{DK}$, swapping *the factors of the factorization yields another transition matrix* $\bar{\mathbf{P}} = \mathbf{KD}$, *potentially much smaller than the original, which retains the basic topology and properties of* $\mathbf{P}$.

Given the strong connection between $\mathbf{P} \in \mathbb{R}^{n \times n}$ and $\bar{\mathbf{P}} \in \mathbb{R}^{m \times m}$, the idea of replacing the former by the latter comes almost inevitably. The motivation for this would be, of course, to save computational resources when $m < n$. For example, Barreto and Fragoso

(2011) have shown that it is possible to recover the stationary distribution of $\mathbf{P}$ through a linear transformation of the corresponding distribution of $\bar{\mathbf{P}}$. In this paper we will use the stochastic-factorization trick to reduce the computational cost of KBRL. The strategy will be to summarize the information contained in KBRL's MDP in a model of fixed size.

## 3.2 Reducing a Markov Decision Process

The idea of using stochastic factorization to reduce dynamic programming's computational requirements is straightforward: given factorizations of the transition matrices $\mathbf{P}^a$, we can apply our trick to obtain a reduced MDP that will be solved in place of the original one. In the most general scenario, we would have one independent factorization $\mathbf{P}^a = \mathbf{D}^a\mathbf{K}^a$ for each action $a \in A$. However, in the current work we will focus on the particular case in which there is a single matrix $\mathbf{D}$, which will prove to be convenient both mathematically and computationally.

Obviously, in order to apply the stochastic-factorization trick to an MDP, we have to first *compute* the matrices involved in the factorization. Unfortunately, such a procedure can be computationally demanding, exceeding the number of operations necessary to calculate $\mathbf{v}^*$ (Vavasis, 2009; Barreto et al., 2013). Thus, in practice we may have to replace the exact factorizations $\mathbf{P}^a = \mathbf{D}\mathbf{K}^a$ with approximations $\mathbf{P}^a \approx \mathbf{D}\mathbf{K}^a$. The following proposition bounds the error in the value-function approximation resulting from the application of our trick to approximate stochastic factorizations:

**Proposition 1.** *Let $M \equiv (S, A, \mathbf{P}^a, \mathbf{r}^a, \gamma)$ be a finite MDP with $|S| = n$ and $0 \leq \gamma < 1$. Let $\mathbf{D} \in \mathbb{R}^{n\times m}$ be a stochastic matrix and, for each $a \in A$, let $\mathbf{K}^a \in \mathbb{R}^{m\times n}$ be stochastic and let $\bar{\mathbf{r}}^a$ be a vector in $\mathbb{R}^m$. Define the MDP $\bar{M} \equiv (\bar{S}, A, \bar{\mathbf{P}}^a, \bar{\mathbf{r}}^a, \gamma)$, with $|\bar{S}| = m$ and $\bar{\mathbf{P}}^a = \mathbf{K}^a\mathbf{D}$. Then,*

$$\left\|\mathbf{v}^* - \Gamma\mathbf{D}\bar{\mathbf{Q}}^*\right\|_\infty \leq \xi_v \equiv \frac{1}{1-\gamma}\max_a \|\mathbf{r}^a - \mathbf{D}\bar{\mathbf{r}}^a\|_\infty + \frac{\bar{R}_{\text{dif}}}{(1-\gamma)^2}\left(\frac{\gamma}{2}\max_a \|\mathbf{P}^a - \mathbf{D}\mathbf{K}^a\|_\infty + \sigma(\mathbf{D})\right), \tag{8}$$

*where $\|\cdot\|_\infty$ is the maximum norm, $\bar{R}_{\text{dif}} = \max_{a,i} \bar{r}_i^a - \min_{a,i} \bar{r}_i^a$, and $\sigma(\mathbf{D}) = \max_i (1 - \max_j d_{ij})$.*[1]

*Proof.* Let $\check{M} \equiv (S, A, \check{\mathbf{P}}^a, \check{\mathbf{r}}^a, \gamma)$, with $\check{\mathbf{P}}^a = \mathbf{D}\mathbf{K}^a$ and $\check{\mathbf{r}}^a = \mathbf{D}\bar{\mathbf{r}}^a$. From the triangle inequality, we know that

$$\left\|\mathbf{v}^* - \Gamma\mathbf{D}\bar{\mathbf{Q}}^*\right\|_\infty \leq \|\mathbf{v}^* - \check{\mathbf{v}}^*\|_\infty + \left\|\check{\mathbf{v}}^* - \Gamma\mathbf{D}\bar{\mathbf{Q}}^*\right\|_\infty, \tag{9}$$

where $\check{\mathbf{v}}^*$ is the optimal value function of $\check{M}$. Our strategy will be to bound $\|\mathbf{v}^* - \check{\mathbf{v}}^*\|_\infty$ and $\left\|\check{\mathbf{v}}^* - \Gamma\mathbf{D}\bar{\mathbf{Q}}^*\right\|_\infty$. In order to find an upper bound for $\|\mathbf{v}^* - \check{\mathbf{v}}^*\|_\infty$, we apply Whitt's (1978) Theorem 3.1 and Corollary (b) of his Theorem 6.1, with all mappings between $M$ and $\check{M}$ taken to be identities, to obtain

[1] We recall that $\|\cdot\|_\infty$ induces the following norm over the space of matrices: $\|\mathbf{A}\|_\infty = \max_i \sum_j |a_{ij}|$.

$$\|\mathbf{v}^* - \check{\mathbf{v}}^*\|_\infty \le \frac{1}{1-\gamma}\left(\max_a \|\mathbf{r}^a - \mathbf{D}\bar{\mathbf{r}}^a\|_\infty + \frac{\gamma \bar{R}_{\text{dif}}}{2(1-\gamma)}\max_a \|\mathbf{P}^a - \mathbf{D}\mathbf{K}^a\|_\infty\right), \tag{10}$$

where we used the fact that $\max_{a,i} \check{r}_i^a - \min_{a,i} \check{r}_i^a \le \bar{R}_{\text{dif}}$. It remains to bound $\left\|\check{\mathbf{v}}^* - \Gamma\mathbf{D}\bar{\mathbf{Q}}^*\right\|_\infty$. Since $\check{\mathbf{r}}^a = \mathbf{D}\bar{\mathbf{r}}^a$ and $\mathbf{D}\bar{\mathbf{P}}^a = \mathbf{D}\mathbf{K}^a\mathbf{D} = \check{\mathbf{P}}^a\mathbf{D}$ for all $a \in A$, the stochastic matrix $\mathbf{D}$ satisfies Sorg and Singh's (2009) definition of a *soft homomorphism* between $\check{M}$ and $\bar{M}$ (see equations (25)–(28) in their paper). Applying Theorem 1 by the same authors, we know that

$$\left\|\Gamma(\check{\mathbf{Q}}^* - \mathbf{D}\bar{\mathbf{Q}}^*)\right\|_\infty \le (1-\gamma)^{-1}\sup_{i,t}(1 - \max_j d_{ij})\;\bar{\delta}_i^{(t)}, \tag{11}$$

where $\bar{\delta}_i^{(t)} = \max_{j:d_{ij}>0,k} \bar{q}_{jk}^{(t)} - \min_{j:d_{ij}>0,k} \bar{q}_{jk}^{(t)}$ and $\bar{q}_{jk}^{(t)}$ are elements of $\bar{\mathbf{Q}}^{(t)}$, the optimal $t$-step action-value function of $\bar{M}$. Since $\|\Gamma\check{\mathbf{Q}}^* - \Gamma\mathbf{D}\bar{\mathbf{Q}}^*\|_\infty \le \|\Gamma(\check{\mathbf{Q}}^* - \mathbf{D}\bar{\mathbf{Q}}^*)\|_\infty$ and, for all $t > 0$, $\bar{\delta}_i^{(t)} \le (1-\gamma)^{-1}(\max_{a,k} \bar{r}_k^a - \min_{a,k} \bar{r}_k^a)$, we can write

$$\left\|\check{\mathbf{v}}^* - \Gamma\mathbf{D}\bar{\mathbf{Q}}^*\right\|_\infty \le \frac{\bar{R}_{\text{dif}}}{(1-\gamma)^2}\max_i\,(1 - \max_j d_{ij}) = \frac{\bar{R}_{\text{dif}}}{(1-\gamma)^2}\sigma(\mathbf{D}). \tag{12}$$

Substituting (10) and (12) back into (9), we obtain (8). □

We note that our bound can be made tighter if we replace the right-hand side of (12) with the right-hand side of (11). However, such a replacement would result in a less intelligible bound that cannot be computed in practice. Needless to say, all subsequent developments that depend on Proposition 1 (and on $\xi_v$ in particular) are also valid for the tighter version of the bound. In Appendix we derive another bound for the distance between $\mathbf{v}^*$ and $\Gamma\mathbf{D}\bar{\mathbf{Q}}^*$ which is valid for any norm.

Our bound depends on two factors: the quality of the MDP's factorization, given by $\max_a \|\mathbf{P}^a - \mathbf{D}\mathbf{K}^a\|_\infty$ and $\max_a \|\mathbf{r}^a - \mathbf{D}\bar{\mathbf{r}}^a\|_\infty$, and the "level of stochasticity" of $\mathbf{D}$, measured by $\sigma(\mathbf{D})$. When the MDP factorization is exact, we recover (12), which is a computable version of Sorg and Singh's (2009) bound for soft homomorphisms. On the other hand, when $\mathbf{D}$ is deterministic—that is, when all its nonzero elements are 1—expression (8) reduces to Whitt's (1978) classical result regarding state aggregation in dynamic programming. Finally, if we have exact deterministic factorizations, the right-hand side of (8) reduces to zero. This also makes sense, since in this case the stochastic-factorization trick gives rise to an exact homomorphism (Ravindran, 2004).

Proposition 1 elucidates the basic mechanism through which one can use the stochastic-factorization trick to reduce the number of states in an MDP (and hence the computational cost of finding a policy using dynamic programming). One possible way to exploit this result is to see the computation of $\mathbf{D}$, $\mathbf{K}^a$, and $\bar{\mathbf{r}}^a$ as an optimization problem in which the objective is to minimize some function of $\max_a \|\mathbf{P}^a - \mathbf{D}\mathbf{K}^a\|_\infty$, $\max_a \|\mathbf{r}^a - \mathbf{D}\bar{\mathbf{r}}^a\|_\infty$, and possibly also $\sigma(\mathbf{D})$ (Barreto et al., 2013). However, in this paper we adopt a different approach: as will be shown, we apply our trick in the context of reinforcement learning to *avoid* the construction of $\mathbf{P}^a$ and $\mathbf{r}^a$.

# 4 Kernel-Based Stochastic Factorization

In Section  we presented KBRL, an approximation framework for reinforcement learning whose main drawback is its high computational complexity. In Section  we discussed how the stochastic-factorization trick can in principle be useful to reduce an MDP, as long as one circumvents the computational burden imposed by the calculation of the matrices involved in the process. We now show how to leverage these two components to produce an algorithm called *kernel-based stochastic factorization* (KBSF) that overcomes these computational limitations.

KBSF emerges from the application of the stochastic-factorization trick to KBRL's MDP $\hat{M}$ (Barreto et al., 2011). Similarly to Ormoneit and Sen (2002), we start by defining a "mother kernel" $\bar{\phi}(x) : \mathbb{R}^+ \mapsto \mathbb{R}^+$. In Appendix  we list our assumptions regarding $\bar{\phi}$. Here, it suffices to note that, since our assumptions and Ormoneit and Sen's (2002) are not mutually exclusive, we can have $\phi \equiv \bar{\phi}$ (by using the Gaussian function in both cases, for example). Let $\bar{S} \equiv \{\bar{s}_1, \bar{s}_2, ..., \bar{s}_m\}$ be a set of *representative states.* Analogously to (2) and (3), we define the kernel $\bar{\mathrm{k}}_{\bar{\tau}}(s, s') = \bar{\phi}\left(\|s - s'\|/\bar{\tau}\right)$ and its normalized version $\bar{\kappa}_{\bar{\tau}}(s, \bar{s}_i) = \bar{\mathrm{k}}_{\bar{\tau}}(s, \bar{s}_i)/\sum_{j=1}^{m} \bar{\mathrm{k}}_{\bar{\tau}}(s, \bar{s}_j)$. We will use $\kappa_{\tau}^a$ to build matrices $\mathbf{K}^a$ and $\bar{\kappa}_{\bar{\tau}}$ to build matrix $\mathbf{D}$.

As shown in Figure 2a, KBRL's matrices $\hat{\mathbf{P}}^a$ have a very specific structure, since only transitions ending in states $\hat{s}_i^a \in S^a$ have a nonzero probability of occurrence. Suppose that we want to apply the stochastic-factorization trick to KBRL's MDP. Assuming that the matrices $\mathbf{K}^a$ have the same structure as $\hat{\mathbf{P}}^a$, when computing $\bar{\mathbf{P}}^a = \mathbf{K}^a\mathbf{D}$ we only have to look at the sub-matrices of $\mathbf{K}^a$ and $\mathbf{D}$ corresponding to the $n_a$ nonzero columns of $\mathbf{K}^a$. We call these matrices $\dot{\mathbf{K}}^a \in \mathbb{R}^{m \times n_a}$ and $\dot{\mathbf{D}}^a \in \mathbb{R}^{n_a \times m}$. The strategy of KBSF is to fill out matrices $\dot{\mathbf{K}}^a$ and $\dot{\mathbf{D}}^a$ with elements

$$\dot{k}_{ij}^a = \kappa_{\tau}^a(\bar{s}_i, s_j^a) \quad \text{and} \quad \dot{d}_{ij}^a = \bar{\kappa}_{\bar{\tau}}(\hat{s}_i^a, \bar{s}_j). \tag{13}$$

Note that, based on $\dot{\mathbf{D}}^a$, one can easily recover $\mathbf{D}$ as $\mathbf{D}^\intercal \equiv [(\dot{\mathbf{D}}^1)^\intercal(\dot{\mathbf{D}}^2)^\intercal ... (\dot{\mathbf{D}}^{|A|})^\intercal] \in \mathbb{R}^{n \times m}$. Similarly, if we let $\mathbf{K} \equiv [\dot{\mathbf{K}}^1 \dot{\mathbf{K}}^2 ... \dot{\mathbf{K}}^{|A|}] \in \mathbb{R}^{m \times n}$, then $\mathbf{K}^a \in \mathbb{R}^{m \times n}$ is matrix $\mathbf{K}$ with all elements replaced by zeros except for those corresponding to matrix $\dot{\mathbf{K}}^a$(see Figures 2b and 2c for an illustration). It should be thus obvious that $\bar{\mathbf{P}} = \mathbf{K}^a\mathbf{D} = \dot{\mathbf{K}}^a\dot{\mathbf{D}}^a$.

In order to conclude the construction of KBSF's MDP, we have to define the vectors of expected rewards $\bar{\mathbf{r}}^a$. As shown in expression (5), the reward functions of KBRL's MDP, $\hat{R}^a(s, s')$, only depend on the ending state $s'$. Recalling the interpretation of the rows of $\mathbf{K}^a$ as transition probabilities from the representative states to the original ones, illustrated in Figure 1, it is clear that

$$\bar{\mathbf{r}}^a = \dot{\mathbf{K}}^a\mathbf{r}^a = \mathbf{K}^a\mathbf{r}. \tag{14}$$

$$\hat{\mathbf{P}}^a = \begin{array}{cccccc} & \hat{s}_1^a & \hat{s}_2^a & \hat{s}_3^a & \hat{s}_1^b & \hat{s}_2^b \\ \hat{s}_1^a & \kappa_\tau^a(\hat{s}_1^a, s_1^a) & \kappa_\tau^a(\hat{s}_1^a, s_2^a) & \kappa_\tau^a(\hat{s}_1^a, s_3^a) & 0 & 0 \\ \hat{s}_2^a & \kappa_\tau^a(\hat{s}_2^a, s_1^a) & \kappa_\tau^a(\hat{s}_2^a, s_2^a) & \kappa_\tau^a(\hat{s}_2^a, s_3^a) & 0 & 0 \\ \hat{s}_3^a & \kappa_\tau^a(\hat{s}_3^a, s_1^a) & \kappa_\tau^a(\hat{s}_3^a, s_2^a) & \kappa_\tau^a(\hat{s}_3^a, s_3^a) & 0 & 0 \\ \hat{s}_1^b & \kappa_\tau^a(\hat{s}_1^b, s_1^a) & \kappa_\tau^a(\hat{s}_1^b, s_2^a) & \kappa_\tau^a(\hat{s}_1^b, s_3^a) & 0 & 0 \\ \hat{s}_2^b & \kappa_\tau^a(\hat{s}_2^b, s_1^a) & \kappa_\tau^a(\hat{s}_2^b, s_2^a) & \kappa_\tau^a(\hat{s}_2^b, s_3^a) & 0 & 0 \end{array},$$

$$\hat{\mathbf{P}}^b = \begin{array}{cccccc} & \hat{s}_1^a & \hat{s}_2^a & \hat{s}_3^a & \hat{s}_1^b & \hat{s}_2^b \\ \hat{s}_1^a & 0 & 0 & 0 & \kappa_\tau^a(\hat{s}_1^a, s_1^b) & \kappa_\tau^a(\hat{s}_1^a, s_2^b) \\ \hat{s}_2^a & 0 & 0 & 0 & \kappa_\tau^a(\hat{s}_2^a, s_1^b) & \kappa_\tau^a(\hat{s}_2^a, s_2^b) \\ \hat{s}_3^a & 0 & 0 & 0 & \kappa_\tau^a(\hat{s}_3^a, s_1^b) & \kappa_\tau^a(\hat{s}_3^a, s_2^b) \\ \hat{s}_1^b & 0 & 0 & 0 & \kappa_\tau^a(\hat{s}_1^b, s_1^b) & \kappa_\tau^a(\hat{s}_1^b, s_2^b) \\ \hat{s}_2^b & 0 & 0 & 0 & \kappa_\tau^a(\hat{s}_2^b, s_1^b) & \kappa_\tau^a(\hat{s}_2^b, s_2^b) \end{array}$$

(a) KBRL's matrices

$$\mathbf{D} = \begin{array}{ccc} & \bar{s}_1 & \bar{s}_2 \\ \hat{s}_1^a & \bar{\kappa}_{\bar{\tau}}(\hat{s}_1^a, \bar{s}_1) & \bar{\kappa}_{\bar{\tau}}(\hat{s}_1^a, \bar{s}_2) \\ \hat{s}_2^a & \bar{\kappa}_{\bar{\tau}}(\hat{s}_2^a, \bar{s}_1) & \bar{\kappa}_{\bar{\tau}}(\hat{s}_2^a, \bar{s}_2) \\ \hat{s}_3^a & \bar{\kappa}_{\bar{\tau}}(\hat{s}_3^a, \bar{s}_1) & \bar{\kappa}_{\bar{\tau}}(\hat{s}_3^a, \bar{s}_2) \\ \hat{s}_1^b & \bar{\kappa}_{\bar{\tau}}(\hat{s}_1^b, \bar{s}_1) & \bar{\kappa}_{\bar{\tau}}(\hat{s}_1^b, \bar{s}_2) \\ \hat{s}_2^b & \bar{\kappa}_{\bar{\tau}}(\hat{s}_2^b, \bar{s}_1) & \bar{\kappa}_{\bar{\tau}}(\hat{s}_2^b, \bar{s}_2) \end{array},$$

$$\mathbf{K}^a = \begin{array}{cccccc} & \hat{s}_1^a & \hat{s}_2^a & \hat{s}_3^a & \hat{s}_1^b & \hat{s}_2^b \\ \bar{s}_1 & \kappa_\tau^a(\bar{s}_1, s_1^a) & \kappa_\tau^a(\bar{s}_1, s_2^a) & \kappa_\tau^a(\bar{s}_1, s_3^a) & 0 & 0 \\ \bar{s}_2 & \kappa_\tau^a(\bar{s}_2, s_1^a) & \kappa_\tau^a(\bar{s}_2, s_2^a) & \kappa_\tau^a(\bar{s}_2, s_3^a) & 0 & 0 \end{array},$$

$$\mathbf{K}^b = \begin{array}{cccccc} & \hat{s}_1^a & \hat{s}_2^a & \hat{s}_3^a & \hat{s}_1^b & \hat{s}_2^b \\ \bar{s}_1 & 0 & 0 & 0 & \kappa_\tau^a(\bar{s}_1, s_1^b) & \kappa_\tau^a(\bar{s}_1, s_2^b) \\ \bar{s}_2 & 0 & 0 & 0 & \kappa_\tau^a(\bar{s}_2, s_1^b) & \kappa_\tau^a(\bar{s}_2, s_2^b) \end{array}.$$

(b) KBSF's sparse matrices

$$\dot{\mathbf{D}}^a = \begin{array}{ccc} & \bar{s}_1 & \bar{s}_2 \\ \hat{s}_1^a & \bar{\kappa}_{\bar{\tau}}(\hat{s}_1^a, \bar{s}_1) & \bar{\kappa}_{\bar{\tau}}(\hat{s}_1^a, \bar{s}_2) \\ \hat{s}_2^a & \bar{\kappa}_{\bar{\tau}}(\hat{s}_2^a, \bar{s}_1) & \bar{\kappa}_{\bar{\tau}}(\hat{s}_2^a, \bar{s}_2) \\ \hat{s}_3^a & \bar{\kappa}_{\bar{\tau}}(\hat{s}_3^a, \bar{s}_1) & \bar{\kappa}_{\bar{\tau}}(\hat{s}_3^a, \bar{s}_2) \end{array}, \qquad \dot{\mathbf{K}}^a = \begin{array}{cccc} & \hat{s}_1^a & \hat{s}_2^a & \hat{s}_3^a \\ \bar{s}_1 & \kappa_\tau^a(\bar{s}_1, s_1^a) & \kappa_\tau^a(\bar{s}_1, s_2^a) & \kappa_\tau^a(\bar{s}_1, s_3^a) \\ \bar{s}_2 & \kappa_\tau^a(\bar{s}_2, s_1^a) & \kappa_\tau^a(\bar{s}_2, s_2^a) & \kappa_\tau^a(\bar{s}_2, s_3^a) \end{array},$$

$$\dot{\mathbf{D}}^b = \begin{array}{ccc} & \bar{s}_1 & \bar{s}_2 \\ \hat{s}_1^b & \bar{\kappa}_{\bar{\tau}}(\hat{s}_1^b, \bar{s}_1) & \bar{\kappa}_{\bar{\tau}}(\hat{s}_1^b, \bar{s}_2) \\ \hat{s}_2^b & \bar{\kappa}_{\bar{\tau}}(\hat{s}_2^b, \bar{s}_1) & \bar{\kappa}_{\bar{\tau}}(\hat{s}_2^b, \bar{s}_2) \end{array}, \qquad \dot{\mathbf{K}}^b = \begin{array}{ccc} & \hat{s}_1^b & \hat{s}_2^b \\ \bar{s}_1 & \kappa_\tau^a(\bar{s}_1, s_1^b) & \kappa_\tau^a(\bar{s}_1, s_2^b) \\ \bar{s}_2 & \kappa_\tau^a(\bar{s}_2, s_1^b) & \kappa_\tau^a(\bar{s}_2, s_2^b) \end{array}.$$

(c) KBSF's dense matrices

Figure 2: Matrices built by KBRL and KBSF for the case in which the original MDP has two actions, $a$ and $b$, and $n_a = 3$, $n_b = 2$, and $m = 2$.

Therefore, the formal specification of KBSF's MDP is given by $\bar{M} \equiv (\bar{S}, A, \dot{\mathbf{K}}^a\dot{\mathbf{D}}^a, \dot{\mathbf{K}}^a\mathbf{r}^a, \gamma) = (\bar{S}, A, \mathbf{K}^a\mathbf{D}^a, \mathbf{K}^a\mathbf{r}, \gamma) = (\bar{S}, A, \bar{\mathbf{P}}^a, \bar{\mathbf{r}}^a, \gamma)$.

As discussed in Section , KBRL's approximation scheme can be interpreted as the derivation of a finite MDP. In this case, the sample transitions define both the finite state space $\hat{S}$ and the model's transition and reward functions. This means that the state space and dynamics of KBRL's model are inexorably linked: except maybe for degenerate cases, changing one also changes the other. By defining a set of representative states, KBSF decouples the MDP's structure from its particular instantiation. To see why this is so, note that, if we fix the representative states, different sets of sample transitions will give rise to different models. Conversely, the same set of transitions can generate different MDPs, depending on how the representative states are defined.

A step by step description of KBSF is given in Algorithm 1. As one can see, KBSF is very simple to understand and to implement. It works as follows: first, the MDP $\bar{M}$ is built as described above. Then, its action-value function $\bar{\mathbf{Q}}^*$ is determined through any dynamic programming algorithm. Finally, KBSF returns an approximation of $\hat{\mathbf{v}}^*$—the optimal value function of KBRL's MDP—computed as $\tilde{\mathbf{v}} = \Gamma\mathbf{D}\bar{\mathbf{Q}}^*$. Based on $\tilde{\mathbf{v}}$, one can compute an approximation of KBRL's action-value function $\hat{Q}(s,a)$ by simply replacing $\tilde{V}$ for $\hat{V}^*$ in (7), that is,

$$\tilde{Q}(s,a) = \sum_{i=1}^{n_a} \kappa_\tau^a(s, s_i^a)\left[r_i^a + \gamma\tilde{V}(\hat{s}_i^a)\right], \tag{15}$$

where $s \in \mathbb{S}$ and $a \in A$. Note that $\tilde{V}(\hat{s}_i^a)$ corresponds to one specific entry of vector $\tilde{\mathbf{v}}$, whose index is given by $\sum_{b=0}^{a-1} n_b + i$, where we assume that $n_0 = 0$.

**Algorithm 1** Batch KBSF

```
Input:  $S^a = \{(s^a_k, r^a_k, \hat{s}^a_k)|k = 1, 2, ..., n_a\}$ for all $a \in A$      ▷ Sample transitions
        $\bar{S} = \{\bar{s}_1, \bar{s}_2, ..., \bar{s}_m\}$                                  ▷ Set of representative states
Output: $\tilde{\mathbf{v}} \approx \hat{\mathbf{v}}^*$
for each $a \in A$ do
    Compute matrix $\dot{\mathbf{D}}^a$: $\dot{d}^a_{ij} = \bar{\kappa}_{\bar{\tau}}(\hat{s}^a_i, \bar{s}_j)$
    Compute matrix $\dot{\mathbf{K}}^a$: $\dot{k}^a_{ij} = \kappa^a_\tau(\bar{s}_i, s^a_j)$
    Compute vector $\bar{\mathbf{r}}^a$: $\bar{r}^a_i = \sum_j \dot{k}^a_{ij} r^a_j$
    Compute matrix $\bar{\mathbf{P}}^a = \dot{\mathbf{K}}^a\dot{\mathbf{D}}^a$
Solve $\bar{M} \equiv (\bar{S}, A, \bar{\mathbf{P}}^a, \bar{\mathbf{r}}^a, \gamma)$                       ▷ i.e., compute $\bar{\mathbf{Q}}^*$
Return $\tilde{\mathbf{v}} = \Gamma\mathbf{D}\bar{\mathbf{Q}}^*$, where $\mathbf{D}^\intercal = \left[(\dot{\mathbf{D}}^1)^\intercal(\dot{\mathbf{D}}^2)^\intercal...(\dot{\mathbf{D}}^{|A|})^\intercal\right]$
```

As shown in Algorithm 1, the key point of KBSF's mechanics is the fact that the matrices $\check{\mathbf{P}}^a = \mathbf{D}\mathbf{K}^a$ are never actually computed, but instead we directly solve the MDP $\bar{M}$ containing $m$ states only. This results in an efficient algorithm that requires only $O(nm|A|d_{\mathbb{S}} + \hat{n}m^2|A|)$ operations and $O(\hat{n}m)$ bits to build a reduced version of KBRL's MDP, where $\hat{n} = \max_a n_a$. After the reduced model $\bar{M}$ has been constructed, KBSF's computational cost becomes a function of $m$ only. In particular, the cost of solving $\bar{M}$ through dynamic programming becomes polynomial in $m$ instead of $n$: while one application of $\hat{T}$, the Bellman operator of $\hat{M}$, is $O(n\hat{n}|A|)$, the computation of $\bar{T}$ is $O(m^2|A|)$. Therefore, KBSF's time and memory complexities are only linear in $n$.

We note that, in practice, KBSF's computational requirements can be reduced even further if one enforces the kernels $\kappa_\tau^a$ and $\bar{\kappa}_{\bar{\tau}}$ to be sparse. In particular, given a fixed $\bar{s}_i$, instead of computing $\bar{\mathrm{k}}_{\bar{\tau}}(\bar{s}_i, s_j^a)$ for $j = 1, 2, ..., n_a$, one can evaluate the kernel on a pre-specified neighborhood of $\bar{s}_i$ only. Assuming that $\bar{\mathrm{k}}_{\bar{\tau}}(\bar{s}_i, s_j^a)$ is zero for all $s_j^a$ outside this region, one avoids not only computing the kernel but also storing the resulting values (the same reasoning applies to the computation of $\mathrm{k}_\tau(\hat{s}_i^a, \bar{s}_j)$ for a fixed $\hat{s}_i^a$).

## 4.1 A closer look at KBSF's approximation

As outlined in Section , KBRL defines the probability of a transition from state $\hat{s}_i^b$ to state $\hat{s}_k^a$ as being $\kappa_\tau^a(\hat{s}_i^b, s_k^a)$, where $a, b \in A$ (see Figure 2a). Note that the kernel $\kappa_\tau^a$ is computed with the initial state $s_k^a$, and not $\hat{s}_k^a$ itself. The intuition behind this is simple: since we know the transition $s_k^a \xrightarrow{a} \hat{s}_k^a$ has occurred before, the more "similar" $\hat{s}_i^b$ is to $s_k^a$, the more likely the transition $\hat{s}_i^b \xrightarrow{a} \hat{s}_k^a$ becomes (Ormoneit and Sen, 2002).

From (13), it is clear that the computation of matrices $\mathbf{K}^a$ performed by KBSF follows the same reasoning underlying the computation of KBRL's matrices $\hat{\mathbf{P}}^a$; in particular, $\kappa_\tau^a(\bar{s}_j, s_k^a)$ gives the probability of a transition from $\bar{s}_j$ to $\hat{s}_k^a$. However, when we look at matrix $\mathbf{D}$ things are slightly different: here, the probability of a "transition" from $\hat{s}_i^b$ to representative state $\bar{s}_j$ is given by $\bar{\kappa}_{\bar{\tau}}(\hat{s}_i^b, \bar{s}_j)$—a computation that involves $\bar{s}_j$ itself. If we were to strictly adhere to KBRL's logic when computing the transition probabilities to the representative states $\bar{s}_j$, the probability of transitioning from $\hat{s}_i^b$ to $\bar{s}_j$ upon executing action $a$ should be a function of $\hat{s}_i^b$ and a state $s'$ from which we knew a transition $s' \xrightarrow{a} \bar{s}_j$ had occurred. In this case we would end up with one matrix $\mathbf{D}^a$ for each action $a \in A$. Note though that this formulation of the method is not practical, because the computation of the matrices $\mathbf{D}^a$ would require a transition $(\cdot) \xrightarrow{a} \bar{s}_j$ for each $a \in A$ and each $\bar{s}_j \in \bar{S}$. Clearly, such a requirement is hard to fulfill even if we have a generative model available to generate sample transitions.

In this section we provide an interpretation of the approximation computed by KBSF that supports our definition of matrix $\mathbf{D}$. We start by looking at how KBRL constructs the matrices $\hat{\mathbf{P}}^a$. As shown in Figure 2a, for each action $a \in A$ the state $\hat{s}_i^b$ has an associated stochastic vector $\hat{\mathbf{p}}_j^a \in \mathbb{R}^{1 \times n}$ whose nonzero entries correspond to the kernel $\kappa_\tau^a(\hat{s}_i^b, \cdot)$ evaluated at $s_k^a, k = 1, 2, \ldots, n_a$. Since we are dealing with a continuous state space, it is possible to compute an analogous vector for any $s \in \mathbb{S}$ and any $a \in A$. Focusing on the nonzero entries of $\hat{\mathbf{p}}_j^a$, we define the function

$$\begin{aligned} \hat{\mathcal{P}}_{S^a}: \quad & \mathbb{S} \mapsto \mathbb{R}^{1 \times n_a} \\ & \hat{\mathcal{P}}_{S^a}(s) = \hat{\mathbf{p}}^a \iff \hat{p}_i^a = \kappa_\tau^a(s, s_i^a) \text{ for } i = 1, 2, ..., n_a. \end{aligned} \tag{16}$$

Clearly, full knowledge of the function $\hat{\mathcal{P}}_{S^a}$ allows for an exact computation of KBRL's transition matrix $\hat{\mathbf{P}}^a$. Now suppose we do not know $\hat{\mathcal{P}}_{S^a}$ and we want to compute an approximation of this function in the points $\hat{s}_i^a \in S^a$, for all $a \in A$. Suppose further that we are only given a "training set" composed of $m$ pairs $(\bar{s}_j, \hat{\mathcal{P}}_{S^a}(\bar{s}_j))$. One possible way of approaching this problem is to resort to kernel smoothing techniques. In this case, a particularly common choice is the so-called Nadaraya-Watson kernel-weighted estimator (Hastie et al., 2002, Chapter 6):

$$\bar{\mathcal{P}}_{S^a}(s) = \frac{\sum_{j=1}^{m} \bar{\mathrm{k}}_{\bar{\tau}}(s, \bar{s}_j)\hat{\mathcal{P}}_{S^a}(\bar{s}_j)}{\sum_{j=1}^{m} \bar{\mathrm{k}}_{\bar{\tau}}(s, \bar{s}_j)} = \sum_{j=1}^{m} \bar{\kappa}_{\bar{\tau}}(s, \bar{s}_j)\hat{\mathcal{P}}_{S^a}(\bar{s}_j). \tag{17}$$

Contrasting the expression above with (13), we see that this is exactly how KBSF computes its approximation $\mathbf{DK}^a \approx \hat{\mathbf{P}}^a$, with $\bar{\mathcal{P}}_{S^a}$ evaluated at the points $\hat{s}_i^b \in S^b$, $b = 1, 2, ..., |A|$. In this case, $\bar{\kappa}_{\bar{\tau}}(\hat{s}_i^b, \bar{s}_j)$ are the elements of matrix $\mathbf{D}$, and $\hat{\mathcal{P}}_{S^a}(\bar{s}_j)$ is the $j^{\text{th}}$ row of matrix $\dot{\mathbf{K}}^a$. Thus, in some sense, KBSF uses KBRL's own kernel approximation principle to compute a stochastic factorization of $\hat{M}$.

## 4.2 Theoretical results

Since KBSF comes down to the solution of a finite MDP, it always converges to the same approximation $\tilde{\mathbf{v}}$, whose distance to KBRL's optimal value function $\hat{\mathbf{v}}^*$ is bounded by Proposition 1. Once $\tilde{\mathbf{v}}$ is available, the value of any state-action pair can be determined through (15). The following result generalizes Proposition 1 to the entire continuous state space $\mathbb{S}$:

**Proposition 2.** *Let $\hat{Q}$ be the value function computed by KBRL through (7) and let $\tilde{Q}$ be the value function computed by KBSF through (15). Then, for any $s \in \mathbb{S}$ and any $a \in A$, $|\hat{Q}(s,a) - \tilde{Q}(s,a)| \leq \gamma\xi_v$, with $\xi_v$ defined in (8).*

*Proof.*

$$\begin{aligned}
|\hat{Q}(s,a) - \tilde{Q}(s,a)| &= \left| \sum_{i=1}^{n_a} \kappa_\tau^a(s, s_i^a) \left[ r_i^a + \gamma \hat{V}^*(\hat{s}_i^a) \right] - \sum_{i=1}^{n_a} \kappa_\tau^a(s, s_i^a) \left[ r_i^a + \gamma \tilde{V}(\hat{s}_i^a) \right] \right| \\
&\leq \gamma \sum_{i=1}^{n_a} \kappa_\tau^a(s, s_i^a) \left| \hat{V}^*(\hat{s}_i^a) - \tilde{V}(\hat{s}_i^a) \right| \leq \gamma \sum_{i=1}^{n_a} \kappa_\tau^a(s, s_i^a)\xi_v \leq \gamma\xi_v,
\end{aligned}$$

where the second inequality results from the application of Proposition 1 and the third inequality is a consequence of the fact that $\sum_{i=1}^{n_a} \kappa_\tau^a(s, s_i^a)$ defines a convex combination. □

Proposition 2 makes it clear that the quality of the approximation computed by KBSF depends crucially on $\xi_v$. In the remainder of this section we will show that, if the distances between sampled states and the respective nearest representative states are small enough, then we can make $\xi_v$ as small as desired by setting $\bar{\tau}$ to a sufficiently small value. To be more precise, let $rs : \mathbb{S} \times \{1, 2, ..., m\} \mapsto \bar{S}$ be a function that orders the representative states according to their distance to a given state $s$, that is, if $rs(s,i) = \bar{s}_k$, then $\bar{s}_k$ is the $i^{\text{th}}$ nearest representative state to $s$. Define $dist : \mathbb{S} \times \{1, 2, ..., m\} \mapsto \mathbb{R}$ as $dist(s,i) =$

$\|s - rs(s,i)\|$. Assuming that we have $|A|$ fixed sets of sample transitions $S^a$, we will show that, for any $\epsilon > 0$, there is a $\delta > 0$ such that, if $\max_{a,i} dist(\hat{s}_i^a, 1) < \delta$, then we can set $\bar{\tau}$ in order to guarantee that $\xi_v < \epsilon$. To show that, we will need the following two lemmas, proved in Appendix :

**Lemma 1.** *For any $s_i^a \in S^a$ and any $\epsilon > 0$, there is a $\delta > 0$ such that $|\kappa_\tau^a(s, s_i^a) - \kappa_\tau^a(s', s_i^a)| < \epsilon$ if $\|s - s'\| < \ \delta$.*

**Lemma 2.** *Let $s \in \mathbb{S}$, let $m > 1$, and assume there is a $w \in \{1, 2, ..., m-1\}$ such that $dist(s, w) < dist(s, w+1)$. Define*

$$W \equiv \{k \mid \|s \ - \bar{s}_k\| \leq dist(s, w)\} \textit{ and } \bar{W} \equiv \{1, 2, ..., m\} - W.$$

*Then, for any $\alpha > 0$, $\sum_{k \in W} \bar{\kappa}_{\bar{\tau}}(s, \bar{s}_k) < \alpha \sum_{k \in \bar{W}} \bar{\kappa}_{\bar{\tau}}(s, \bar{s}_k)$ for $\bar{\tau}$ sufficiently small.*

Lemma 1 is basically a continuity argument: it shows that, for any fixed $s_i^a$, $|\kappa_\tau^a(s, s_i^a) - \kappa_\tau^a(s', s_i^a)| \to 0$ as $\|s - s'\| \to 0$. Lemma 2 states that, if we order the representative states according to their distance to a fixed state $s$, and then partition them in two subsets, we can control the relative magnitude of the corresponding kernels's sums by adjusting the parameter $\bar{\tau}$ (we redirect the reader to Appendix for details on how to set $\bar{\tau}$). Based on these two lemmas, we present the main result of this section, also proved in Appendix :

**Proposition 3.** *For any $\epsilon > 0$, there is a $\delta > 0$ such that, if $\max_{a,i} dist(\hat{s}_i^a, 1) < \delta$, then we can guarantee that $\xi_v < \ \epsilon$ by making $\bar{\tau}$ sufficiently small.*

Proposition 3 tells us that, regardless of the specific reinforcement learning problem at hand, if the distances between sampled states $\hat{s}_i^a$ and the respective nearest representative states are small enough, then we can make KBSF's approximation of KBRL's value function as accurate as desired by setting $\bar{\tau}$ to a sufficiently small value (one can see how exactly to set $\bar{\tau}$ in the proof of the proposition). How small the maximum distance $\max_{a,i} dist(\hat{s}_i^a, 1)$ should be depends on the particular choice of kernel $\mathrm{k}_\tau$ and on the sets of sample transitions $S^a$. Here, we deliberately refrained from making assumptions on $\mathrm{k}_\tau$ and $S^a$ in order to present the proposition in its most general form.

Note that a fixed number of representative states $m$ imposes a minimum possible value for $\max_{a,i} dist(\hat{s}_i^a, 1)$, and if this value is not small enough decreasing $\bar{\tau}$ may actually hurt the approximation. The optimal value for $\bar{\tau}$ in this case is again context-dependent. As a positive flip side of this statement, we note that, even if $\max_{a,i} dist(\hat{s}_i^a, 1) > \delta$, it might be possible to make $\xi_v < \ \epsilon$ by setting $\bar{\tau}$ appropriately. Therefore, rather than as a practical guide on how to configure KBSF, Proposition 3 should be seen as a theoretical argument showing that KBSF is a sound algorithm, in the sense that in the limit it recovers KBRL's solution.

### 4.3 Empirical results

We now present a series of computational experiments designed to illustrate the behavior of KBSF in a variety of challenging domains. We start with a simple problem, the "puddle world", to show that KBSF is indeed capable of compressing the information contained in KBRL's model. We then move to more difficult tasks, and compare KBSF with other state-of-the-art reinforcement-learning algorithms. We start with two classical control tasks, single and double pole-balancing. Next we study two medically-related problems based on real data: HIV drug schedule and epilepsy-suppression domains.

All problems considered in this paper have a continuous state space and a finite number of actions, and were modeled as discounted tasks. The algorithms's results correspond to the performance of the greedy decision policy derived from the final value function computed. In all cases, the decision policies were evaluated on challenging test states from which the tasks cannot be easily solved. The details of the experiments are given in Appendix .

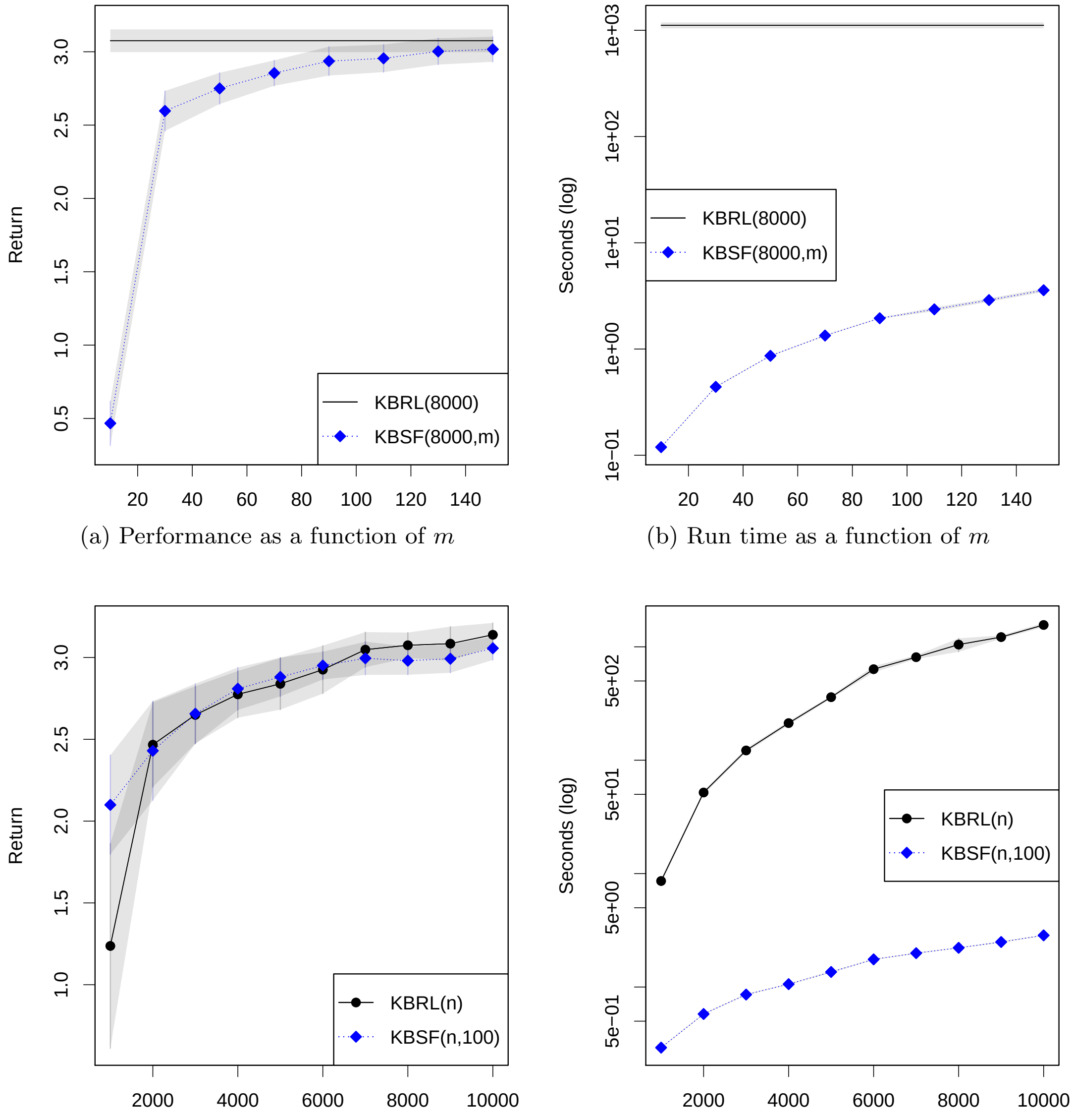
### 4.3.1 Puddle world (proof of concept)

In order to show that KBSF is indeed capable of summarizing the information contained in KBRL's model, we use the puddle world task (Sutton, 1996). The puddle world is a simple two-dimensional problem in which the objective is to reach a goal region avoiding two "puddles" along the way. We implemented the task exactly as described by Sutton (1996), except that we used a discount factor of $\gamma = 0.99$ and evaluated the decision policies on a set of pre-defined test states surrounding the puddles (see Appendix ).

The experiment was carried out as follows: first, we collected a set of $n$ sample transitions $(s^a_k, r^a_k, \hat{s}^a_k)$ using a random exploration policy (that is, a policy that selects actions uniformly at random). In the case of KBRL, this set of sample transitions defined the model used to approximate the value function. In order to define KBSF's model, the states $\hat{s}^a_k$ were grouped by the $k$-means algorithm into $m$ clusters and a representative state $\bar{s}_j$ was placed at the center of each resulting cluster (Kaufman and Rousseeuw, 1990). As for the kernels's widths, we varied both $\tau$ and $\bar{\tau}$ in the set $\{0.01, 0.1, 1\}$ (see Table 1 on page 56). The results reported represent the best performance of the algorithms over 50 runs; that is, for each $n$ and each $m$ we picked the combination of parameters that generated the maximum average return. We use the following convention to refer to specific instances of each method: the first number enclosed in parentheses after an algorithm's name is $n$, the number of sample transitions used in the approximation, and the second one is $m$, the size of the model used to approximate the value function. Note that for KBRL $n$ and $m$ coincide.

In Figure 3a and 3b we observe the effect of fixing the number of transitions $n$ and varying the number of representative states $m$. As expected, KBSF's results improve as $m \to n$. More surprising is the fact that KBSF has essentially the same performance as KBRL using models one order of magnitude smaller. This indicates that KBSF is summarizing well the information contained in the data. Depending on the values of $n$ and $m$, such a compression may represent a significant reduction on the consumption of computational resources. For example, by replacing KBRL(8000) with KBSF(8000, 100), we obtain a decrease of approximately 99.58% on the number of operations performed to find a policy, as shown in Figure 3b (the cost of constructing KBSF's MDP is included in all reported run times).

In Figures 3c and 3d we fix $m$ and vary $n$. Observe in Figure 3c how KBRL and KBSF have similar performances, and both improve as $n$ increases. However, since KBSF is using



(a) Performance as a function of $m$

(b) Run time as a function of $m$

(c) Performance as a function of $n$

(d) Run time as a function of $n$

Figure 3: Results on the puddle-world task averaged over 50 runs. The algorithms were evaluated on a set of test states distributed over a region of the state space surrounding the "puddles" (details in Appendix ). The shadowed regions represent 99% confidence intervals.

a model of fixed size, its computational cost depends only linearly on $n$, whereas KBRL's cost grows with $n^2\hat{n}$, roughly. This explains the huge difference in the algorithms's run times shown in Figure 3d.

### 4.3.2 Single and double pole-balancing (comparison with LSPI)

We now evaluate how KBSF compares to other modern reinforcement learning algorithms on more difficult tasks. We first contrast our method with Lagoudakis and Parr's (2003) least-squares policy iteration algorithm (LSPI). Besides its popularity, LSPI is a natural candidate for such a comparison for three reasons: it also builds an approximator of fixed size out of a batch of sample transitions, it has good theoretical guarantees, and it has been successfully applied to several reinforcement learning tasks.

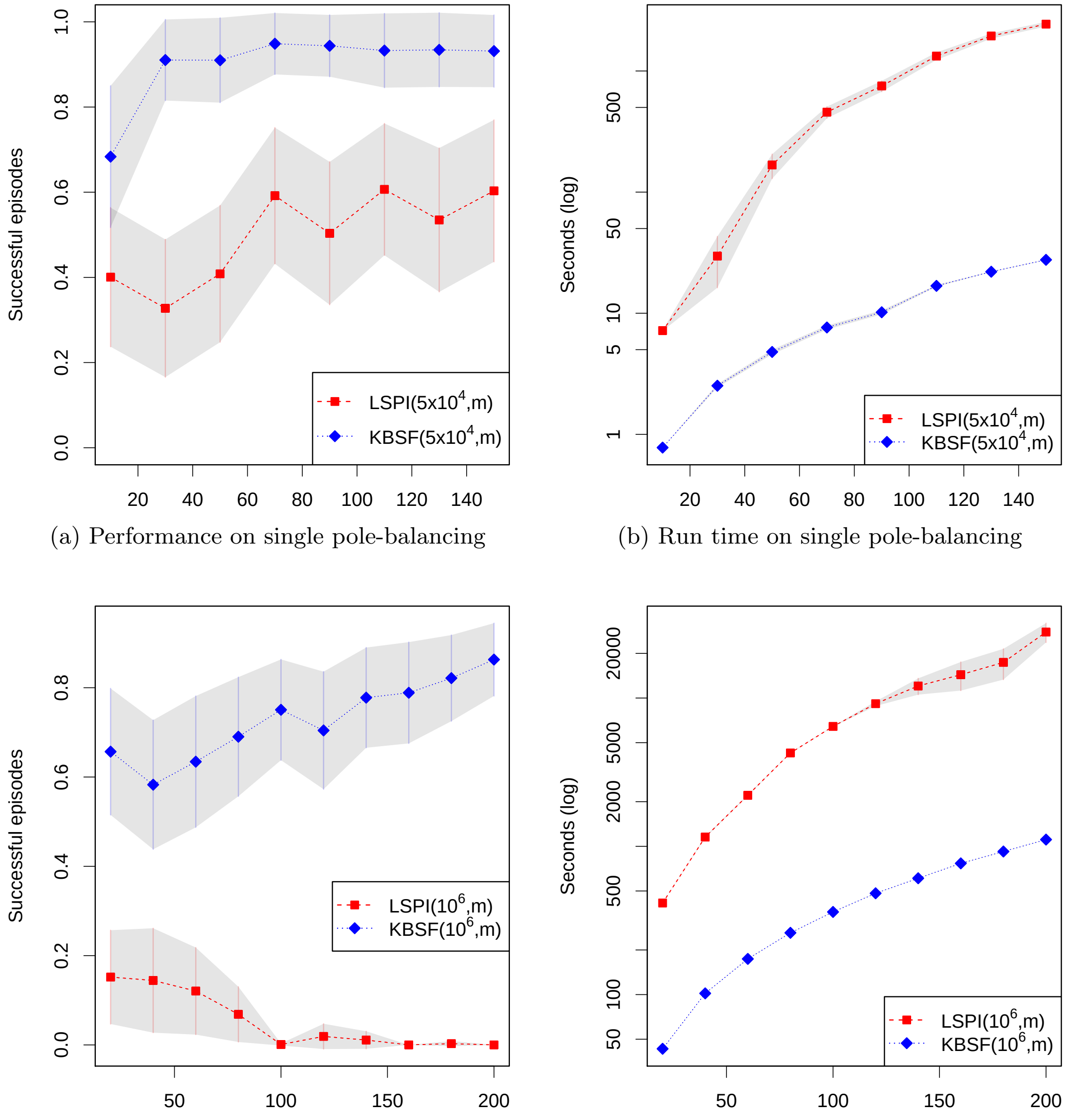
We compare the performance of LSPI and KBSF on the pole balancing task. Pole balancing has a long history as a benchmark problem because it represents a rich class of unstable systems (Michie and Chambers, 1968; Anderson, 1986; Barto et al., 1983). The objective in this problem is to apply forces to a wheeled cart moving along a limited track in order to keep one or more poles hinged to the cart from falling over. There are several variations of the task with different levels of difficulty; among them, balancing two poles side by side is particularly hard (Wieland, 1991). In this paper we compare LSPI and KBSF on both the single- and two-poles versions of the problem. We implemented the tasks using a realistic simulator described by Gomez (2003). We refer the reader to Appendix  for details on the problems's configuration.

The experiments were carried out as described in the previous section, with sample transitions collected by a random policy and then clustered by the $k$-means algorithm. In both versions of the pole-balancing task LSPI used the same data and approximation architectures as KBSF. To make the comparison with LSPI as fair as possible, we fixed the width of KBSF's kernel $\kappa_\tau^a$ at $\tau = 1$ and varied $\bar{\tau}$ in $\{0.01, 0.1, 1\}$ for both algorithms. Also, policy iteration was used to find a decision policy for the MDPs constructed by KBSF, and this algorithm was run for a maximum of 30 iterations, the same limit used for LSPI.

Figure 4 shows the results of LSPI and KBSF on the single and double pole-balancing tasks. We call attention to the fact that the version of the problems used here is significantly harder than the more commonly-used variants in which the decision policies are evaluated on a single state close to the origin. This is probably the reason why LSPI achieves a success rate of no more than 60% on the single pole-balancing task, as shown in Figure 4a. In contrast, KBSF's decision policies are able to balance the pole in 90% of the attempts, on average, using as few as $m = 30$ representative states.

The results of KBSF on the double pole-balancing task are still more impressive. As Wieland (1991) rightly points out, this version of the problem is considerably more difficult than its single pole variant, and previous attempts to apply reinforcement-learning techniques to this domain resulted in disappointing performance (Gomez et al., 2006). As shown in Figure 4c, KBSF($10^6$, 200) is able to achieve a success rate of more than 80%. To put this number in perspective, recall that some of the test states are quite challenging, with the two poles inclined and falling in opposite directions.

The good performance of KBSF comes at a relatively low computational cost. A



(a) Performance on single pole-balancing

(b) Run time on single pole-balancing

(c) Performance on double pole-balancing

(d) Run time on double pole-balancing

Figure 4: Results on the pole-balancing tasks, as a function of the number of representative states $m$, averaged over 50 runs. The values correspond to the fraction of episodes initiated from the test states in which the pole(s) could be balanced for 3000 steps (one minute of simulated time). The test sets were regular grids defined over the hypercube centered at the origin and covering 50% of the state-space axes in each dimension (see Appendix ). Shadowed regions represent 99% confidence intervals.

conservative estimate reveals that, were KBRL($10^6$) run on the same computer used for these experiments, we would have to wait for more than 6 *months* to see the results. KBSF($10^6$, 200) delivers a decision policy in less than 7 minutes. KBSF's computational cost also compares well with that of LSPI, as shown in Figures 4b and 4d. LSPI's policy-evaluation step involves the update and solution of a linear system of equations, which take $O(nm^2)$ and $O(m^3|A|^3)$, respectively. In addition, the policy-update stage requires the definition of $\pi(\hat{s}^a_k)$ for all $n$ states in the set of sample transitions. In contrast, at each iteration KBSF only performs $O(m^3)$ operations to evaluate a decision policy and $O(m^2|A|)$ operations to update it.

### 4.3.3 HIV drug schedule (comparison with fitted $Q$-iteration)

We now compare KBSF with the fitted $Q$-iteration algorithm (Ernst et al., 2005; Antos et al., 2007; Munos and Szepesvári, 2008). Fitted $Q$-iteration is a conceptually simple method that also builds its approximation based solely on sample transitions. Here we adopt this algorithm with an ensemble of trees generated by Geurts et al.'s (2006) extra-trees algorithm. We will refer to the resulting method as FQIT.

We chose FQIT for our comparisons because it has shown excellent performance on both benchmark and real-world reinforcement-learning tasks (Ernst et al., 2005, 2006). In all experiments reported in this paper we used FQIT with ensembles of 30 trees. As detailed in Appendix , besides the number of trees, FQIT has three main parameters. Among them, the minimum number of elements required to split a node in the construction of the trees, denoted here by $\eta_{\min}$, has a particularly strong effect on both the algorithm's performance and computational cost. Thus, in our experiments we fixed FQIT's parameters at reasonable values—selected based on preliminary experiments—and only varied $\eta_{\min}$. The respective instances of the tree-based approach are referred to as FQIT($\eta_{\min}$).

We compare FQIT and KBSF on an important medical problem which we will refer to as the HIV drug schedule domain (Adams et al., 2004; Ernst et al., 2006). Typical HIV treatments use drug cocktails containing two types of medication: reverse transcriptase inhibitors (RTI) and protease inhibitors (PI). Despite the success of drug cocktails in maintaining low viral loads, there are several complications associated with their long-term use. This has attracted the interest of the scientific community to the problem of optimizing drug-scheduling strategies. One strategy that has been receiving a lot of attention recently is structured treatment interruption (STI), in which patients undergo alternate cycles with and without the drugs. Although many successful STI treatments have been reported in the literature, as of now there is no consensus regarding the exact protocol that should be followed (Bajaria et al., 2004).

The scheduling of STI treatments can be seen as a sequential decision problem in which the actions correspond to the types of cocktail that should be administered to a patient (Ernst et al., 2006). To simplify the problem's formulation, it is assumed that RTI and PI drugs are administered at fixed amounts, reducing the actions to the four possible combinations of drugs: none, RTI only, PI only, or both. The goal is to minimize the viral load using as little drugs as possible. Following Ernst et al. (2006), we performed our experiments using a model that describes the interaction of the immune system with HIV. This model was developed by Adams et al. (2004) and has been identified and validated based on real clinical data. The resulting reinforcement learning task has a 6-dimensional continuous state space whose variables describe the overall patient's condition.

We formulated the problem exactly as proposed by Ernst et al. (2006, see Appendix for details). The strategy used to generate the data also followed the protocol proposed by these authors, which we now briefly explain. Starting from a batch of 6000 sample transitions generated by a random policy, each algorithm first computed an initial approximation of the problem's optimal value function. Based on this approximation, a 0.15-greedy policy was used to collect a second batch of 6000 transitions, which was merged with the first.[2] This process was repeated for 10 rounds, resulting in a total of 60000 sample transitions.

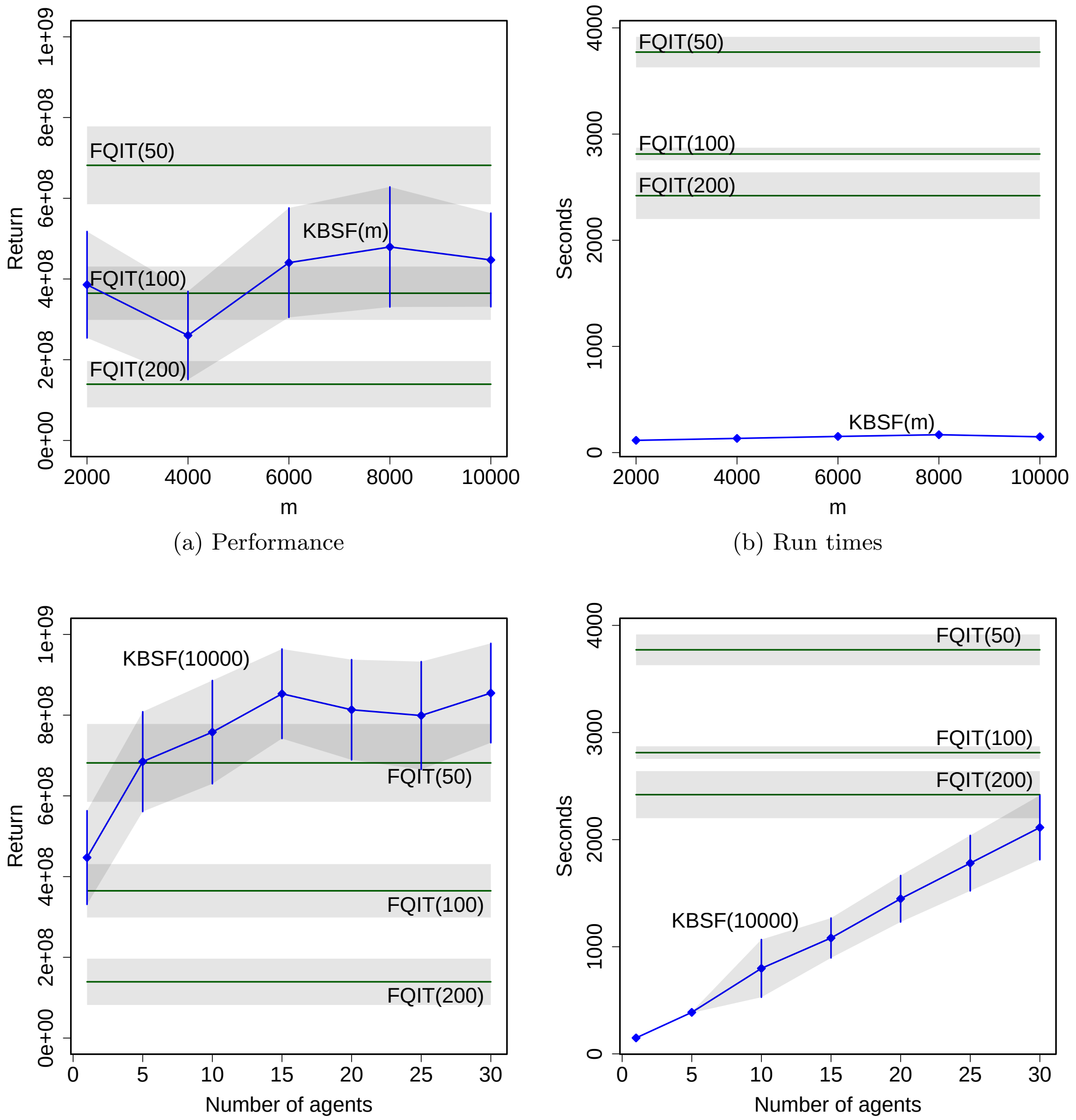
We varied FQIT's parameter $\eta_{\min}$ in the set $\{50, 100, 200\}$. For the experiments with KBSF, we fixed $\tau = \bar{\tau} = 1$ and varied $m$ in $\{2000, 4000, ..., 10000\}$ (in the rounds in which $m \geq n$ we simply used all states $\hat{s}_i^a$ as representative states). As discussed in the beginning of this section, it is possible to reduce KBSF's computational cost with the use of sparse kernels. In our experiments with the HIV drug schedule task, we only computed the $\mu = 2$ largest values of $\mathrm{k}_\tau(\bar{s}_i, \cdot)$ and the $\bar{\mu} = 3$ largest values of $\bar{\mathrm{k}}_{\bar{\tau}}(\hat{s}_i^a, \cdot)$ (see Appendix ). The representative states $\bar{s}_i$ were selected at random from the set of sampled states $\hat{s}_i^a$ (the reason for this will become clear shortly). Since in the current experiments the number of sample transitions $n$ was fixed, we will refer to the particular instances of our algorithm simply as KBSF($m$).

Figure 5 shows the results obtained by FQIT and KBSF on the HIV drug schedule task. As shown in Figure 5a, FQIT's performance improves when $\eta_{\min}$ is decreased, as expected. In contrast, increasing the number of representative states $m$ does not have a strong impact on the quality of KBSF's solutions (in fact, in some cases the average return obtained by the resulting policies decreases slightly when $m$ grows). Overall, the performance of KBSF on the HIV drug schedule task is not nearly as impressive as on the previous problems. For example, even when using $m = 10000$ representative states, which corresponds to one sixth of the sampled states, KBSF is unable to reproduce the performance of FQIT with $\eta_{\min} = 50$.

On the other hand, when we look at Figure 5b, it is clear that the difference on the algorithms's performance is counterbalanced by a substantial difference on the associated computational costs. As an illustration, note that KBSF(10000) is 15 times faster than FQTI(100) and 20 times faster than FQTI(50). This difference on the algorithms's run times is expected, since each iteration of FQIT involves the construction (or update) of an ensemble of trees, each one requiring at least $O(n \log(n/\eta_{\min}))$ operations, and the improvement of the current decision policy, which is $O(n|A|)$ (Geurts et al., 2006). As discussed before, KBSF's efficiency comes from the fact that its computational cost per iteration is independent of the number of sample transitions $n$.

Note that the fact that FQIT uses an ensemble of trees is both a blessing and a curse. If on the one hand this reduces the variance of the approximation, on the other hand it also increases the algorithm's computational cost (Geurts et al., 2006). Given

[2] As explained by Sutton and Barto (1998), an $\epsilon$-greedy policy selects the action with maximum value with probability $1 - \epsilon$, and with probability $\epsilon$ it picks an action uniformly at random.



(a) Performance

(b) Run times

(c) Performance

(d) Run times

Figure 5: Results on the HIV drug schedule task averaged over 50 runs. The STI policies were evaluated for 5000 days starting from a state representing a patient's unhealthy state (see Appendix ). The shadowed regions represent 99% confidence intervals.

the big gap between FQIT's and KBSF's time complexities, one may wonder if the latter can also benefit from averaging over several models. In order to verify this hypothesis, we implemented a very simple model-averaging strategy with KBSF: we trained several agents independently, using Algorithm 1 on the same set of sample transitions, and then put them together on a single "committee". In order to increase the variability within the committee of agents, instead of using $k$-means to determine the representative states $\bar{s}_j$ we simply selected them uniformly at random from the set of sampled states $\hat{s}_i^a$ (note that this has the extra benefit of reducing the method's overall computational cost). The actions selected by the committee of agents were determined by "voting"—that is, we simply picked the action chosen by the majority of agents, with ties broken randomly.

We do not claim that the approach described above is the best model-averaging strategy to be used with KBSF. However, it seems to be sufficient to boost the algorithm's performance considerably, as shown in Figure 5c. Note how KBSF already performs comparably to FQTI(50) when using only 5 agents in the committee. When this number is increased to 15, the expected return of KBSF's agents is considerably larger than that of the best FQIT's agent, with only a small overlap between the 99% confidence intervals associated with the algorithms's results. The good performance of KBSF is still more impressive when we look at Figure 5d, which shows that even when using a committee of 30 agents this algorithm is faster than FQIT(200).

In concluding, we should mention that, overall, our experience with FQIT confirms Ernst et al.'s (2005) report: it is a stable, easy-to-configure method that usually delivers good solutions. In fact, given the algorithm's ease of use, when the problem at hand can be solved off-line using a moderate number of sample transitions, FQIT may be a very good alternative. On the other hand, for on-line problems or off-line problems involving a large number of sample transitions, FQIT's computational cost can be prohibitive in practice. In Section  we will discuss an experiment in which such a computational demand effectively precludes the use of this algorithm.

#### 4.3.4 Epilepsy suppression (comparison with LSPI and fitted $Q$-iteration)

We conclude our empirical evaluation of KBSF by using it to learn a neuro-stimulation policy for the treatment of epilepsy. It has been shown that the electrical stimulation of specific structures in the neural system at fixed frequencies can effectively suppress the occurrence of seizures (Durand and Bikson, 2001). Unfortunately, *in vitro* neuro-stimulation experiments suggest that fixed-frequency pulses are not equally effective across epileptic systems. Moreover, the long term use of this treatment may potentially damage the patients's neural tissues. Therefore, it is desirable to develop neuro-stimulation policies that replace the fixed-stimulation regime with an adaptive scheme.

The search for efficient neuro-stimulation strategies can be seen as a reinforcement learning problem. Here we study it using a generative model developed by Bush et al. (2009) based on real data collected from epileptic rat hippocampus slices. This model was shown to reproduce the seizure pattern of the original dynamical system and was later validated through the deployment of a learned treatment policy on a real brain slice (Bush and Pineau, 2009). The associated decision problem has a five-dimensional continuous state space and highly non-linear dynamics. At each time step the agent must choose whether or not to apply an electrical pulse. The goal is to suppress seizures as much as possible while minimizing the total amount of stimulation needed to do so.

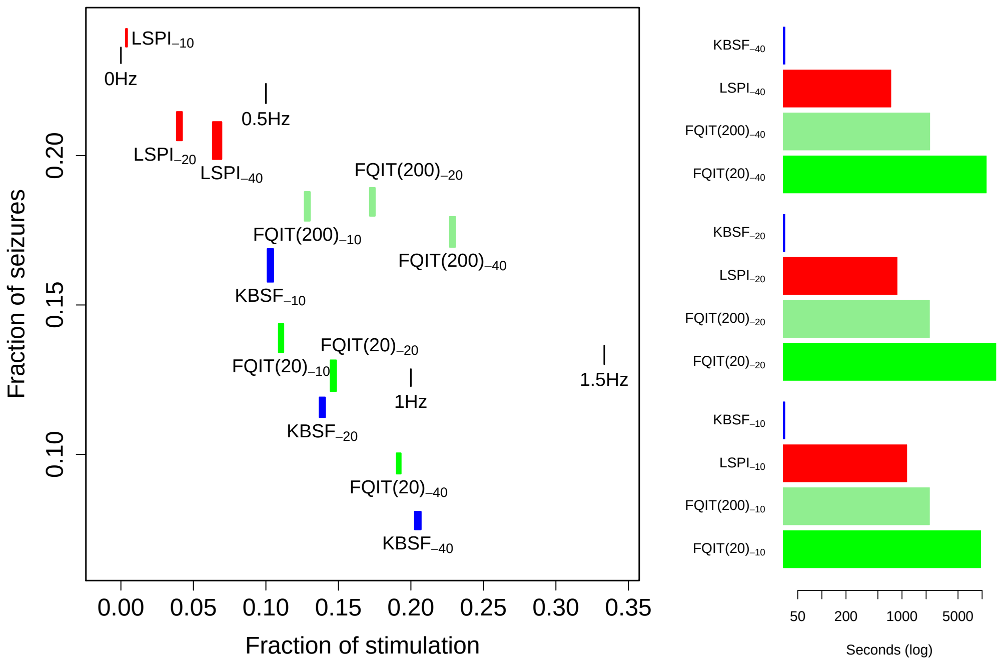
The experiments were performed as described in Section , with a single batch of sample transitions collected by a policy that selects actions uniformly at random. Specifically, the random policy was used to collect 50 trajectories of length 10000, resulting in a total of 500000 sample transitions. We use as a baseline for our comparisons the already mentioned fixed-frequency stimulation policies usually adopted in *in vitro* clinical studies (Bush and Pineau, 2009). In particular, we considered policies that apply electrical pulses at frequencies of 0 Hz, 0.5 Hz, 1 Hz, and 1.5 Hz.

We compare KBSF with LSPI and FQIT. For this task we ran both LSPI and KBSF with sparse kernels, that is, we only computed the kernels at the 6-nearest neighbors of a given state ($\mu = \bar{\mu} = 6$; see Appendix  for details). This modification made it possible to use $m = 50000$ representative states with KBSF. Since for LSPI the reduction on the computational cost was not very significant, we fixed $m = 50$ to keep its run time within reasonable bounds. Again, KBSF and LSPI used the same approximation architectures, with representative states defined by the $k$-means algorithm. We fixed $\tau = 1$ and varied $\bar{\tau}$ in $\{0.01, 0.1, 1\}$. FQIT was configured as described in the previous section, with the parameter $\eta_{\min}$ varying in $\{20, 30, ..., 200\}$. In general, we observed that the performance of the tree-based method improved with smaller values for $\eta_{\min}$, with an expected increase in the computational cost. Thus, in order to give an overall characterization of FQIT's performance, we only report the results obtained with the extreme values of $\eta_{\min}$.

Figure 6 shows the results on the epilepsy-suppression task. In order to obtain different compromises between the problem's two conflicting objectives, we varied the relative magnitude of the penalties associated with the occurrence of seizures and with the application of an electrical pulse (Bush et al., 2009; Bush and Pineau, 2009). Specifically, we fixed the latter at $-1$ and varied the former with values in $\{-10, -20, -40\}$. This appears in the plots as subscripts next to the algorithms's names. As shown in Figure 6a, LSPI's policies seem to prioritize reduction of stimulation at the expense of higher seizure occurrence, which is clearly sub-optimal from a clinical point of view. FQIT(200) also performs poorly, with solutions representing no advance over the fixed-frequency stimulation strategies. In contrast, FQTI(20) and KBSF are both able to generate decision policies that are superior to the 1 Hz policy, which is the most efficient stimulation regime known to date in the clinical literature (Jerger and Schiff, 1995). However, as shown in Figure 6b, KBSF is able to do it at least 100 times faster than the tree-based method.

# 5 Incremental KBSF

As clear in the previous section, one characteristic of KBSF that sets it apart from other methods is its low demand in terms of computational resources. Specifically, both time and memory complexities of our algorithm are linear in the number of sample transitions $n$. In terms of the number of operations performed by the algorithm, this is the best one can do without discarding transitions. However, in terms of memory usage, it is possible to do even better. In this section we show how to build KBSF's approximation incrementally, without ever having access to the entire set of sample transitions at once. Besides reducing the memory complexity of the algorithm, this modification has the additional advantage



(a) Performance. The length of the rectangles's edges represent 99% confidence intervals.

(b) Run times (confidence intervals do not show up in logarithmic scale)

Figure 6: Results on the epilepsy-suppression problem averaged over 50 runs. The decision policies were evaluated on episodes of $10^5$ transitions starting from a fixed set of 10 test states drawn uniformly at random.

of making KBSF suitable for on-line reinforcement learning.

In the batch version of KBSF, described in Section , the matrices $\bar{\mathbf{P}}^a$ and vectors $\bar{\mathbf{r}}^a$ are determined using all the transitions in the corresponding sets $S^a$. This has two undesirable consequences. First, the construction of the MDP $\bar{M}$ requires an amount of memory of $O(\hat{n}m)$. Although this is a significant improvement over KBRL's memory usage, which is lower bounded by $(\min_a n_a)^2|A|$, in more challenging domains even a linear dependence on $\hat{n}$ may be impractical. Second, in the batch version of KBSF the only way to incorporate new data into the model $\bar{M}$ is to recompute the multiplication $\bar{\mathbf{P}}^a = \dot{\mathbf{K}}^a\dot{\mathbf{D}}^a$ for all actions $a$ for which there are new sample transitions available. Even if we ignore the issue with memory usage, this is clearly inefficient in terms of computation. In what follows we present an incremental version of KBSF that circumvents these important limitations (Barreto et al., 2012).

We assume the same scenario considered in Section : there is a set of sample transitions $S^a = \{(s_k^a, r_k^a, \hat{s}_k^a) | k = 1, 2, ..., n_a\}$ associated with each action $a \in A$, where $s_k^a, \hat{s}_k^a \in \mathbb{S}$ and $r_k^a \in \mathbb{R}$, and a set of representative states $\bar{S} = \{\bar{s}_1, \bar{s}_2, ..., \bar{s}_m\}$, with $\bar{s}_i \in \mathbb{S}$. Suppose now that we split the set of sample transitions $S^a$ in two subsets $S_1$ and $S_2$ such that $S_1 \cap S_2 = \emptyset$ and $S_1 \cup S_2 = S^a$ (we drop the "$a$" superscript in the sets $S_1$ and $S_2$ to improve clarity). Without loss of generality, suppose that the sample transitions are indexed so that

$$S_1 \equiv \{(s_k^a, r_k^a, \hat{s}_k^a) | k = 1, 2, ..., n_1\} \text{ and } S_2 \equiv \{(s_k^a, r_k^a, \hat{s}_k^a) | k = n_1{+}1, n_1{+}2, ..., n_1{+}n_2 = n_a\}.$$

Let $\bar{\mathbf{P}}^{S_1}$ and $\bar{\mathbf{r}}^{S_1}$ be matrix $\bar{\mathbf{P}}^a$ and vector $\bar{\mathbf{r}}^a$ computed by KBSF using only the $n_1$ transitions in $S_1$ (if $n_1 = 0$, we define $\bar{\mathbf{P}}^{S_1} = \mathbf{0} \in \mathbb{R}^{m \times m}$ and $\bar{\mathbf{r}}^{S_1} = \mathbf{0} \in \mathbb{R}^m$ for all $a \in A$). We want to compute $\bar{\mathbf{P}}^{S_1 \cup S_2}$ and $\bar{\mathbf{r}}^{S_1 \cup S_2}$ from $\bar{\mathbf{P}}^{S_1}$, $\bar{\mathbf{r}}^{S_1}$, and $S_2$, without using the set of sample transitions $S_1$.

We start with the transition matrices $\bar{\mathbf{P}}^a$. We know that

$$\begin{aligned}\bar{p}_{ij}^{S_1} &= \textstyle\sum_{t=1}^{n_1} \dot{k}_{it}^a \dot{d}_{tj}^a = \sum_{t=1}^{n_1} \dfrac{\mathrm{k}_\tau(\bar{s}_i, s_t^a)}{\sum_{l=1}^{n_1} \mathrm{k}_\tau(\bar{s}_i, s_l^a)} \dfrac{\bar{\mathrm{k}}_{\bar{\tau}}(\hat{s}_t^a, \bar{s}_j)}{\sum_{l=1}^{m} \bar{\mathrm{k}}_{\bar{\tau}}(\hat{s}_t^a, \bar{s}_l)} \\ &= \dfrac{1}{\sum_{l=1}^{n_1} \mathrm{k}_\tau(\bar{s}_i, s_l^a)} \textstyle\sum_{t=1}^{n_1} \dfrac{\mathrm{k}_\tau(\bar{s}_i, s_t^a)\bar{\mathrm{k}}_{\bar{\tau}}(\hat{s}_t^a, \bar{s}_j)}{\sum_{l=1}^{m} \bar{\mathrm{k}}_{\bar{\tau}}(\hat{s}_t^a, \bar{s}_l)}.\end{aligned}$$

To simplify the notation, define

$$w_i^{S_1} = \sum_{l=1}^{n_1} \mathrm{k}_\tau(\bar{s}_i, s_l^a), \; w_i^{S_2} = \sum_{l=n_1+1}^{n_1+n_2} \mathrm{k}_\tau(\bar{s}_i, s_l^a), \text{ and } b_{ij}^t = \frac{\mathrm{k}_\tau(\bar{s}_i, s_t^a)\bar{\mathrm{k}}_{\bar{\tau}}(\hat{s}_t^a, \bar{s}_j)}{\sum_{l=1}^{m} \bar{\mathrm{k}}_{\bar{\tau}}(\hat{s}_t^a, \bar{s}_l)},$$

with $t \in \{1, 2, ..., n_1 + n_2\}$. Then, we can write

$$\bar{p}_{ij}^{S_1 \cup S_2} = \frac{1}{w_i^{S_1} + w_i^{S_2}} \left( \textstyle\sum_{t=1}^{n_1} b_{ij}^t + \sum_{t=n_1+1}^{n_1+n_2} b_{ij}^t \right) = \frac{1}{w_i^{S_1} + w_i^{S_2}} \left( \bar{p}_{ij}^{S_1} w_i^{S_1} + \textstyle\sum_{t=n_1+1}^{n_1+n_2} b_{ij}^t \right).$$

Now, defining $b_{ij}^{S_2} = \sum_{t=n_1+1}^{n_1+n_2} b_{ij}^t$, we have the simple update rule:

$$\boxed{\bar{p}_{ij}^{S_1 \cup S_2} = \frac{1}{w_i^{S_1} + w_i^{S_2}} \left( b_{ij}^{S_2} + \bar{p}_{ij}^{S_1} w_i^{S_1} \right)}\,. \tag{18}$$

We can apply similar reasoning to derive an update rule for the rewards $\bar{r}_i^a$. We know that

$$\bar{r}_i^{S_1} = \frac{1}{\sum_{l=1}^{n_1} \mathrm{k}_\tau(\bar{s}_i, s_l^a)} \sum_{t=1}^{n_1} \mathrm{k}_\tau(\bar{s}_i, s_t^a) r_t^a = \frac{1}{w_i^{S_1}} \sum_{t=1}^{n_1} \mathrm{k}_\tau(\bar{s}_i, s_t^a) r_t^a.$$

Let $e_i^t = \mathrm{k}_\tau(\bar{s}_i, s_t^a) r_t^a$, with $t \in \{1, 2, ..., n_1 + n_2\}$. Then,

$$\bar{r}_i^{S_1 \cup S_2} = \frac{1}{w_i^{S_1} + w_i^{S_2}} \left( \sum_{t=1}^{n_1} e_i^t + \sum_{t=n_1+1}^{n_1+n_2} e_i^t \right) = \frac{1}{w_i^{S_1} + w_i^{S_2}} \left( w_i^{S_1} \bar{r}_i^{S_1} + \sum_{t=n_1+1}^{n_1+n_2} e_i^t \right) .$$

Defining $e_i^{S_2} = \sum_{t=n_1+1}^{n_1+n_2} e_i^t$, we have the following update rule:

$$\boxed{\bar{r}_i^{S_1 \cup S_2} = \frac{1}{w_i^{S_1} + w_i^{S_2}} \left( e_i^{S_2} + \bar{r}_i^{S_1} w_i^{S_1} \right)} . \tag{19}$$

Since $b_{ij}^{S_2}$, $e_i^{S_2}$, and $w_i^{S_2}$ can be computed based on $S_2$ only, we can discard the sample transitions in $S_1$ after computing $\bar{\mathbf{P}}^{S_1}$ and $\bar{\mathbf{r}}^{S_1}$. To do that, we only have to keep the variables $w_i^{S_1}$. These variables can be stored in $|A|$ vectors $\mathbf{w}^a \in \mathbb{R}^m$, resulting in a modest memory overhead. Note that we can apply the ideas above recursively, further splitting the sets $S_1$ and $S_2$ in subsets of smaller size. Thus, we have a fully incremental way of computing KBSF's MDP which requires almost no extra memory.

Algorithm 2 shows a step-by-step description of how to update $\bar{M}$ based on a set of sample transitions. Using this method to update its model, KBSF's space complexity drops from $O(\hat{n}m)$ to $O(m^2)$. Since the amount of memory used by KBSF is now independent of $n$, it can process an arbitrary number of sample transitions (or, more precisely, the limit on the amount of data it can process is dictated by time only, not space).

Instead of assuming that $S_1$ and $S_2$ are a partition of a fixed data set $S^a$, we can consider that $S_2$ was generated based on the policy learned by KBSF using the transitions in $S_1$. Thus, Algorithm 2 provides a flexible framework for integrating learning and planning within KBSF. Specifically, our algorithm can cycle between learning a model of the problem based on sample transitions, using such a model to derive a policy, and resorting to this policy to collect more data. Algorithm 3 shows a possible implementation of this framework. In order to distinguish it from its batch counterpart, we will call the incremental version of our algorithm *i*KBSF. *i*KBSF updates the model $\bar{M}$ and the value function $\bar{\mathbf{Q}}$ at fixed intervals $t_m$ and $t_v$, respectively. When $t_m = t_v = n$, we recover the batch version of KBSF; when $t_m = t_v = 1$, we have a fully on-line method which stores no sample transitions.

Algorithm 3 also allows for the inclusion of new representative states to the model $\bar{M}$. Using Algorithm 2 this is easy to do: given a new representative state $\bar{s}_{m+1}$, it suffices to set $w_{m+1}^a = 0$, $\bar{r}_{m+1}^a = 0$, and $\bar{p}_{m+1,j} = \bar{p}_{j,m+1} = 0$ for $j = 1, 2, ..., m + 1$ and all $a \in A$. Then, in the following applications of update rules (18) and (19), the dynamics of $\bar{M}$ will naturally reflect the existence of state $\bar{s}_{m+1}$. Note that the inclusion of new representative states does not destroy the information already in the model. This allows *i*KBSF to refine its approximation on the fly, as needed. One can think of several

**Algorithm 2** Update KBSF's MDP

```
Input:  $\bar{\mathbf{P}}^a$, $\bar{\mathbf{r}}^a$, $\mathbf{w}^a$ for all $a \in A$                         ▷ Current model
        $S^a = \{(s_k^a, r_k^a, \hat{s}_k^a) | k = 1, 2, ..., n_a\}$ for all $a \in A$    ▷ Sample transitions
Output: Updated $\bar{M}$ and $\mathbf{w}^a$

for $a \in A$ do
    for $t = 1, ..., n_a$ do $z_t \leftarrow \sum_{l=1}^{m} \bar{\mathrm{k}}_{\bar{\tau}}(\hat{s}_t^a, \bar{s}_l)$
    $n_a \leftarrow |S^a|$
    for $i = 1, 2, ..., m$ do
        $w' \leftarrow \sum_{t=1}^{n_a} \mathrm{k}_{\tau}(\bar{s}_i, s_t^a)$
        for $j = 1, 2, ..., m$ do
            $b \leftarrow \sum_{t=1}^{n_a} \mathrm{k}_{\tau}(\bar{s}_i, s_t^a) \bar{\mathrm{k}}_{\bar{\tau}}(\hat{s}_t^a, \bar{s}_j) / z_t$
            $\bar{p}_{ij} \leftarrow \frac{1}{w_i^a + w'} (b + \bar{p}_{ij} w_i^a)$        ▷ Update transition probabilities
        $e \leftarrow \sum_{t=1}^{n_a} \mathrm{k}_{\tau}(\bar{s}_i, s_t^a) r_t^a$
        $\bar{r}_i \leftarrow \frac{1}{w_i^a + w'} (e + \bar{r}_i w_i^a)$                   ▷ Update rewards
        $w_i^a \leftarrow w_i^a + w'$                                  ▷ Update normalization factor
```

**Algorithm 3** Incremental KBSF ($i$KBSF)

```
        $\bar{S} = \{\bar{s}_1, \bar{s}_2, ..., \bar{s}_m\}$                   ▷ Set of representative states
Input:  $t_m$                                             ▷ Interval to update model
        $t_v$                                    ▷ Interval to update value function
Output: Approximate value function $\tilde{Q}(s, a)$

$\bar{\mathbf{P}}^a \leftarrow \mathbf{0} \in \mathbb{R}^{m \times m}$, $\bar{\mathbf{r}}^a \leftarrow \mathbf{0} \in \mathbb{R}^m$, $\mathbf{w}^a \leftarrow \mathbf{0} \in \mathbb{R}^m$, for all $a \in A$
$\bar{\mathbf{Q}} \leftarrow$ arbitrary matrix in $\mathbb{R}^{m \times |A|}$
$s \leftarrow$ initial state
$a \leftarrow$ random action
for $t \leftarrow 1, 2, ...$ do
    Execute $a$ in $s$ and observe $r$ and $\hat{s}$
    $S^a \leftarrow S^a \bigcup \{(s, r, \hat{s})\}$
    if $(t \mod t_m = 0)$ then                                    ▷ Update model
        Add new representative states to $\bar{M}$ using $S^a$          ▷ This step is optional
        Update $\bar{M}$ and $\mathbf{w}^a$ using Algorithm 2 and $S^a$
        $S^a \leftarrow \emptyset$ for all $a \in A$                          ▷ Discard transitions
    if $(t \mod t_v = 0)$ update $\bar{\mathbf{Q}}$                        ▷ Update value function
    $s \leftarrow \hat{s}$
    Select $a$ based on $\tilde{Q}(s, a) = \sum_{i=1}^{m} \bar{\kappa}_{\bar{\tau}}(s, \bar{s}_i) \bar{q}_{ia}$
```

ways of detecting the need for new representative states. A simple strategy, based on Proposition 3, is to impose a maximum distance allowed between a sampled state $\hat{s}_i^a$ and the nearest representative state, $dist(\hat{s}_i^a, 1)$. Thus, anytime the agent encounters a new state $\hat{s}_i^a$ for which $dist(\hat{s}_i^a, 1)$ is above a given threshold, $\hat{s}_i^a$ is added to the model as $\bar{s}_{m+1}$. In Section  we report experiments with *i*KBSF using this approach. Before that, though, we discuss the theoretical properties of the incremental version of our algorithm.

## 5.1 Theoretical results

As discussed, *i*KBSF does not need to store sample transitions to build its approximation. However, the computation of $\tilde{Q}(s,a)$ through (15) requires all the tuples $(s_i^a, r_i^a, \hat{s}_i^a)$ to be available. In some situations, it may be feasible to keep the transitions in order to compute $\tilde{Q}(s,a)$. However, if we want to use *i*KBSF to its full extend, we need a way of computing $\tilde{Q}(s,a)$ without using the sample transitions. This is why upon reaching state $s$ at time step $t$ *i*KBSF selects the action to be performed based on

$$\tilde{Q}_t(s,a) = \sum_{i=1}^{m} \bar{\kappa}_{\bar{\tau}}(s, \bar{s}_i) \bar{Q}_t(\bar{s}_i, a), \tag{20}$$

where $\bar{Q}_t(\bar{s}_i, a)$ is the action-value function available to *i*KBSF at the $t^{\text{th}}$ iteration (see Algorithm 3). Note that we do not assume that *i*KBSF has computed the optimal value function of its current model $\bar{M}_t$—that is, it may be the case that $\bar{Q}_t(\bar{s}_i, a) \neq \bar{Q}_t^*(\bar{s}_i, a)$.

Unfortunately, when we replace (15) with (20) Proposition 2 no longer applies. In this section we address this issue by deriving an upper bound for the difference between $\tilde{Q}_t(s,a)$ and $\hat{Q}_t(s,a)$, the action-value function that would be computed by KBRL using all the transitions processed by *i*KBSF up to time step $t$. In order to derive our bound, we assume that *i*KBSF uses a fixed set $\bar{S}$—meaning that no representative states are added to the model $\bar{M}$—and that it never stops refining its model, doing so at every iteration $t$ (*i.e.*, $t_m = 1$ in Algorithm 3). We start by showing the following lemma, proved in Appendix :

**Lemma 3.** *Let* $M \equiv (S, A, \mathbf{P}^a, \mathbf{r}^a, \gamma)$ *and* $\tilde{M} \equiv (S, A, \tilde{\mathbf{P}}^a, \tilde{\mathbf{r}}^a, \gamma)$ *be two finite MDPs. Then, for any* $s \in S$ *and any* $a \in A$,

$$|Q^*(s,a) - \tilde{Q}^*(s,a)| \leq \frac{1}{1-\gamma} \max_a \|\mathbf{r}^a - \tilde{\mathbf{r}}^a\|_\infty + \frac{\gamma(2-\gamma)}{2(1-\gamma)^2} R_{\text{dif}} \max_a \|\mathbf{P}^a - \tilde{\mathbf{P}}^a\|_\infty,$$

*where* $R_{\text{dif}} = \max_{a,i} r_i^a - \min_{a,i} r_i^a$.

Lemma 3 provides an upper bound for the difference in the action-value functions of any two MDPs having the same state space $S$, action space $A$, and discount factor $\gamma$.[3] Our strategy will be to use this result to bound the error introduced by the application of the stochastic-factorization trick in the context of *i*KBSF.

When $t_m = 1$, at any time step $t$ *i*KBSF has a model $\bar{M}_t$ built based on the $t$ transitions observed thus far. As shown in the beginning of this section, $\bar{M}_t$ exactly matches the model that would be computed by batch KBSF using the same data and the same set

[3] Strehl and Littman's (2008) Lemma 1 is similar to our result. Their bound is more general than ours, as it applies to any $Q^\pi$, but it is also slightly looser.

of representative states. Thus, we can think of matrices $\bar{\mathbf{P}}_t^a$ and vectors $\bar{\mathbf{r}}_t^a$ available at the $t^{\text{th}}$ iteration of $i$KBSF as the result of the stochastic-factorization trick applied with matrices $\mathbf{D}_t$ and $\mathbf{K}_t^a$. Although $i$KBSF does not explicitly compute such matrices, they serve as a solid theoretical ground to build our result on.

**Proposition 4.** *Suppose iKBSF is executed with a fixed set of representative states $\bar{S}$ using $t_m = 1$. Let $\mathbf{D}_t$, $\mathbf{K}_t^a$ and $\bar{\mathbf{r}}_t^a$ be the matrices and the vector (implicitly) computed by this algorithm at iteration $t$. Then, if $s$ is the state encountered by iKBSF at time step $t$,*

$$|\hat{Q}_t(s,a)-\tilde{Q}_t(s,a)| \leq \frac{1}{1-\gamma}\max_a \|\hat{\mathbf{r}}_t^a - \mathbf{D}_t\bar{\mathbf{r}}_t^a\|_\infty + \frac{\bar{R}_{\text{dif},t}}{(1-\gamma)^2}\left(\frac{\gamma(2-\gamma)}{2}\max_a \|\hat{\mathbf{P}}_t^a - \mathbf{D}_t\mathbf{K}_t^a\|_\infty + \sigma(\mathbf{D}_t)\right) + \epsilon_{\bar{Q}_t},$$

*for any $a \in A$, where $\tilde{Q}_t$ is the value function computed by iKBSF at time step $t$ through (20), $\hat{Q}_t$ is the value function computed by KBRL through (7) based on the same data, $\bar{R}_{\text{dif},t} = \max_{a,i} \bar{r}_{i,t}^a - \min_{a,i} \bar{r}_{i,t}^a$, $\sigma(\mathbf{D}_t) = \max_i (1 - \max_j d_{ij,t})$, and $\epsilon_{\bar{Q}_t} = \max_{i,a} |\bar{Q}_t^*(\bar{s}_i, a) - \bar{Q}_t(\bar{s}_i, a)|$.*

*Proof.* Let $\check{M}_t \equiv (\hat{S}_t, A, \check{\mathbf{P}}_t^a, \check{\mathbf{r}}_t^a, \gamma)$, with $\check{\mathbf{P}}_t^a = \mathbf{D}_t\mathbf{K}_t^a$ and $\check{\mathbf{r}}_t^a = \mathbf{D}_t\bar{\mathbf{r}}_t^a$. From the triangle inequality, we know that

$$|\hat{Q}_t(s,a)-\tilde{Q}_t(s,a)| \leq |\hat{Q}_t(s,a)-\check{Q}_t^*(s,a)| + |\check{Q}_t^*(s,a)-\tilde{Q}_t^*(s,a)| + |\tilde{Q}_t^*(s,a)-\tilde{Q}_t(s,a)|, \tag{21}$$

where $\hat{Q}_t$ and $\tilde{Q}_t$ are defined in the proposition's statement, $\check{Q}_t^*$ is the optimal action-value function of $\check{M}_t$, and $\tilde{Q}_t^*(s,a) = \sum_{i=1}^m \bar{\kappa}_{\bar{\tau}}(s,\bar{s}_i)\bar{Q}_t^*(\bar{s}_i,a)$ (the reader will forgive a slight abuse of notation here, since in general $\tilde{Q}_t^*$ is not the optimal value function of any MDP). Our strategy will be to bound each term on the right-hand side of (21). Since $\hat{M}_t$ is the model constructed by KBRL using all the data seen by $i$KBSF up to time step $t$, state $s$ will correspond to one of the states $\hat{s}_i^b$ in this MDP. Thus, from (7), we see that $\hat{Q}_t(s,a) = \hat{Q}_t^*(\hat{s}_i^b, a)$ for some $i$ and some $b$. Therefore, applying Lemma 3 to $\hat{M}_t$ and $\check{M}_t$, we can write

$$|\hat{Q}_t(s,a) - \check{Q}_t^*(s,a)| \leq \frac{1}{1-\gamma}\max_a \|\hat{\mathbf{r}}_t^a - \mathbf{D}_t\bar{\mathbf{r}}_t^a\|_\infty + \frac{\gamma(2-\gamma)}{2(1-\gamma)^2}\bar{R}_{\text{dif},t}\max_a \|\hat{\mathbf{P}}_t^a - \mathbf{D}_t\mathbf{K}_t^a\|_\infty. \tag{22}$$

In order to bound $|\check{Q}_t^*(s,a) - \tilde{Q}_t^*(s,a)|$, we note that, since the information contained in the transition to state $s$ has been incorporated to $i$KBSF's model $\bar{M}$ at time $t$, $\tilde{Q}_t^*(s,a) = \sum_{i=1}^m d_{ti,t}\bar{Q}_t^*(\bar{s}_i,a)$, for any $a \in A$, where $d_{ti,t}$ is the element in the $t^{\text{th}}$ row and $i^{\text{th}}$ column of $\mathbf{D}_t$ (see Figure 2b). In matrix form, we have $\tilde{\mathbf{Q}}_t^* = \mathbf{D}_t\bar{\mathbf{Q}}_t^*$. As $\mathbf{D}_t$ is a soft homomorphism between $\check{M}_t$ and $\bar{M}_t$, we can resort to Sorg and Singh's (2009) Theorem 1, as done in Proposition 1, to write:

$$|\check{Q}_t^*(s,a) - \tilde{Q}_t^*(s,a)| \leq \frac{\bar{R}_{\text{dif},t}}{(1-\gamma)^2}\sigma(\mathbf{D}_t) \tag{23}$$

(see (11) and (12)). Finally,

$$
\begin{aligned}
|\tilde{Q}^*_t(s,a) - \tilde{Q}_t(s,a)| &= \left| \sum_{i=1}^{m} \bar{\kappa}_{\bar{\tau}}(s,\bar{s}_i)\bar{Q}^*_t(\bar{s}_i,a) - \sum_{i=1}^{m} \bar{\kappa}_{\bar{\tau}}(s,\bar{s}_i)\bar{Q}_t(\bar{s}_i,a) \right| \\
&\le \sum_{i=1}^{m} \bar{\kappa}_{\bar{\tau}}(s,\bar{s}_i) \left| \bar{Q}^*_t(\bar{s}_i,a) - \bar{Q}_t(\bar{s}_i,a) \right| \le \epsilon_{\bar{Q}_t}, \qquad (24)
\end{aligned}
$$

where the last step follows from the fact that $\sum_{i=1}^{m} \bar{\kappa}_{\bar{\tau}}(s,\bar{s}_i)$ is a convex combination. Substituting (22), (23), and (24) in (21), we obtain the desired bound. □

Proposition 4 shows that, at any time step $t$, the error in the action-value function computed by $i$KBSF is bounded above by the quality and the level of stochasticity of the stochastic factorization implicitly computed by the algorithm. The term $\epsilon_{\bar{Q}_t}$ accounts for the possibility that $i$KBSF has not computed the optimal value function of its model at step $t$, either because $t_m \neq t_v$ or because the update of $\bar{\mathbf{Q}}$ in Algorithm 3 is not done to completion (for example, one can apply the Bellman operator $\bar{T}$ a fixed number of times, stopping short of convergence). We note that the restriction $t_m = 1$ is not strictly necessary if we are willing to compare $\tilde{Q}_t(s,a)$ with $\hat{Q}_z(s,a)$, where $z = \lfloor (t + t_m)/t \rfloor$ (the next time step scheduled for a model update). However, such a result would be somewhat circular, since the sample transitions used to build $\hat{Q}_z(s,a)$ may depend on $\tilde{Q}_t(s,a)$.

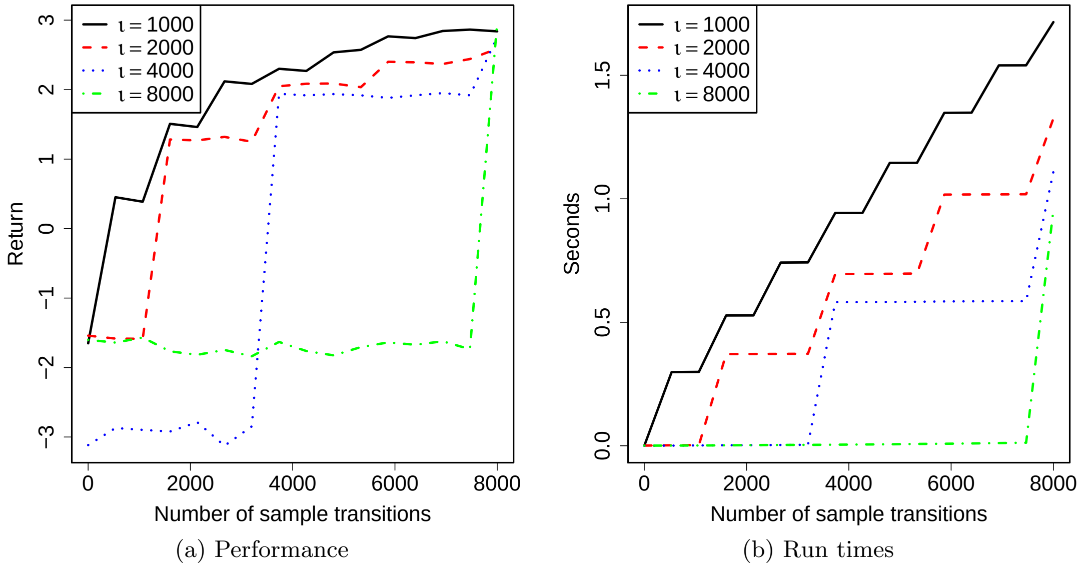
## 5.2 Empirical results

We now look at the empirical performance of the incremental version of KBSF. Following the structure of Section , we start with the puddle world task to show that $i$KBSF is indeed able to match the performance of batch KBSF without storing all sample transitions. Next we exploit the scalability of $i$KBSF to solve two difficult control tasks, triple pole-balancing and helicopter hovering. We also compare $i$KBSF's performance with that of other reinforcement learning algorithms.

### 5.2.1 Puddle world (proof of concept)

We use the puddle world problem as a proof of concept (Sutton, 1996). In this first experiment we show that $i$KBSF is able to recover the model that would be computed by its batch counterpart. In order to do so, we applied Algorithm 3 to the puddle-world task using a random policy to select actions.

Figure 7a shows the result of the experiment when we vary the parameters $t_m$ and $t_v$. Note that the case in which $t_m = t_v = 8000$ corresponds to the batch version of KBSF, whose results on the puddle world are shown in Figure 3. As expected, the performance of KBSF policies improves gradually as the algorithm goes through more sample transitions, and in general the intensity of the improvement is proportional to the amount of data processed. More important, the performance of the decision policies after all sample transitions have been processed is essentially the same for all values of $t_m$ and $t_v$, which



(a) Performance

(b) Run times

Figure 7: Results on the puddle-world task averaged over 50 runs. KBSF used 100 representative states evenly distributed over the state space and $t_m = t_v = \iota$ (see legends). Sample transitions were collected by a random policy. The agents were tested on two sets of states surrounding the "puddles" (see Appendix ).

confirms that $i$KBSF can be used as an instrument to circumvent KBSF's memory demand. Thus, if one has a batch of sample transitions that does not fit in the available memory, it is possible to split the data in chunks of smaller sizes and still get the same value-function approximation that would be computed if the entire data set were processed at once. As shown in Figure 7b, there is only a small computational overhead associated with such a strategy (this results from unnormalizing and normalizing the elements of $\bar{\mathbf{P}}^a$ and $\bar{\mathbf{r}}^a$ several times through update rules (18) and (19)).

#### 5.2.2 Triple pole-balancing (comparison with fitted $Q$-iteration)

As discussed in Section , the pole balancing task has been addressed in several different versions, and among them simultaneously balancing two poles is particularly challenging (Wieland, 1991). Figures 4c and 4d show that the batch version of KBSF was able to satisfactorily solve the double pole-balancing task. In order to show the scalability of the incremental version of our algorithm, in this section we raise the bar, adding a third pole to the problem. We perform our simulations using the parameters usually adopted with the two-pole problem, with the extra pole having the same length and mass as the longer pole (Gomez, 2003, see Appendix ). This results in a difficult control problem with an 8-dimensional state space $\mathbb{S}$.

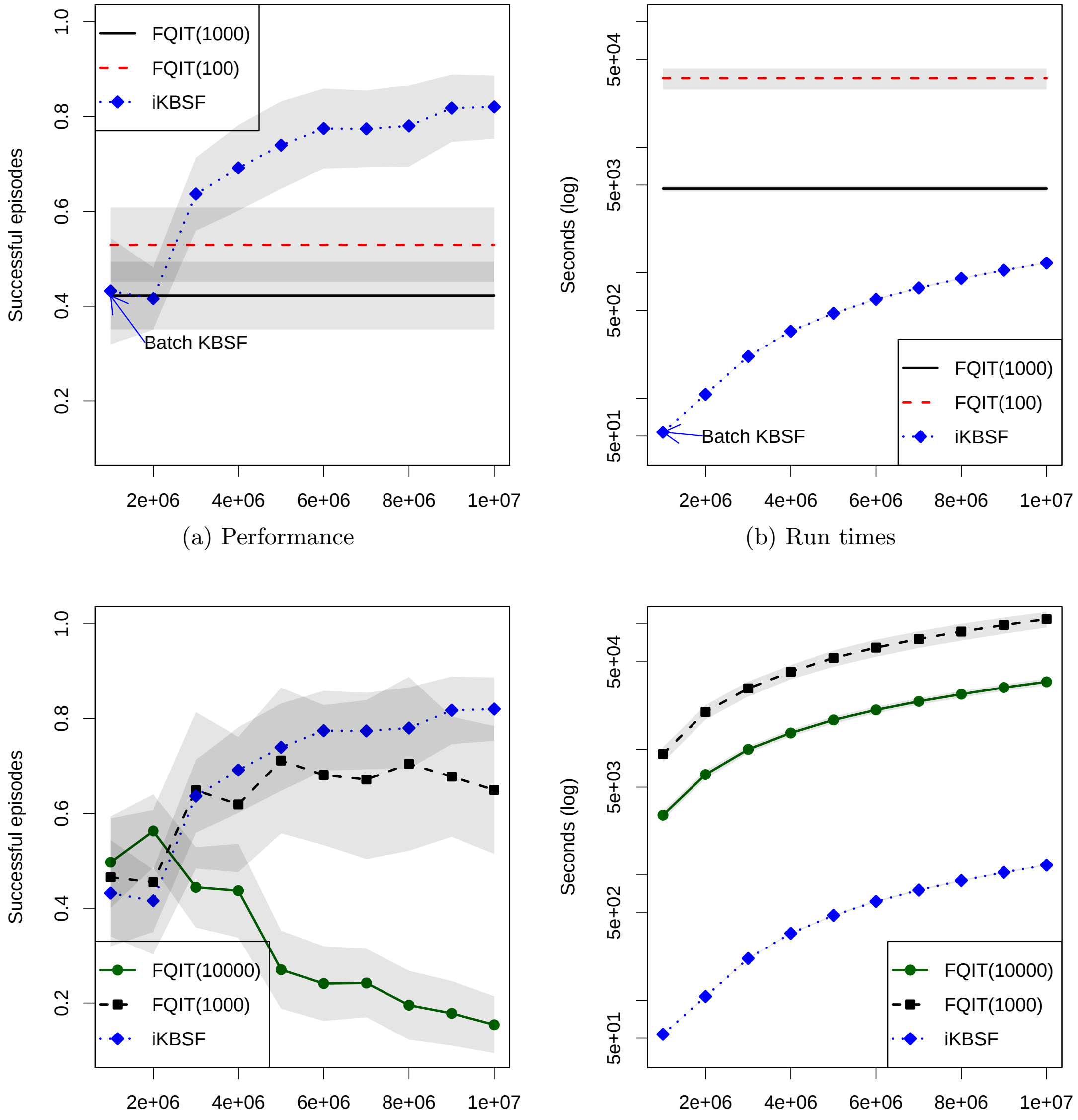
In our experiments with KBSF on the two-pole task, we used 200 representative states and $10^6$ sample transitions collected by a random policy. Here we start our experiment with triple pole-balancing using exactly the same configuration, and then we let $i$KBSF refine its model $\bar{M}$ by incorporating more sample transitions through update rules (18) and (19). We also let *i*KBSF grow its model if necessary. Specifically, a new representative state is added to $\bar{M}$ on-line every time the agent encounters a sample state $\hat{s}_i^a$ for which $\bar{\mathrm{k}}_{\bar{\tau}}(\hat{s}_i^a, \bar{s}_j) < 0.01$ for all $j \in 1, 2, ..., m$. This corresponds to setting the maximum allowed distance from a sampled state to the closest representative state, $\max_{a,i} dist(\hat{s}_i^a, 1)$.

Given the poor performance of LSPI on the double pole-balancing task, shown in Figures 4c and 4d, on the three-pole version of the problem we only compare KBSF with FQIT. We used FQIT with the same configuration adopted in Sections  and , with the parameter $\eta_{\min}$ varying in the set $\{10000, 1000, 100\}$. As for KBSF, the widths of the kernels were fixed at $\tau = 100$ and $\bar{\tau} = 1$ and sparse kernels were used ($\mu = 50$ and $\bar{\mu} = 10$).

In order to show the benefits provided by the incremental version of our algorithm, we assumed that both KBSF and FQIT could store at most $10^6$ sample transitions in memory. In the case of *i*KBSF, this is not a problem, since we can always split the data in subsets of smaller size and process them incrementally. Here, we used Algorithm 3 with a 0.3-greedy policy, $t_m = t_v = 10^6$, and $n = 10^7$. In the case of FQIT, we have two options to circumvent the limited amount of memory available. The first one is to use a single batch of $10^6$ sample transitions. The other option is to use the initial batch of transitions to compute an approximation of the problem's value function, then use an 0.3-greedy policy induced by this approximation to collect a second batch, and so on. Here we show the performance of FQIT using both strategies.

We first compare the performance of *i*KBSF with that of FQIT using a single batch of sample transitions. This is shown in Figure 8a and 8b. For reference, we also show the results of batch KBSF—that is, we show the performance of the policy that would be computed by our algorithm if we did not have a way of computing its approximation incrementally. As shown in Figure 8a, both FQIT and batch KBSF perform poorly in the triple pole-balancing task, with average success rates below 55%. These results suggest that the amount of data used by these algorithms is insufficient to describe the dynamics of the control task. Of course, we could give more sample transitions to FQIT and batch KBSF. Note however that, since they are batch-learning methods, there is an inherent limit on the amount of data that these algorithms can use to construct their approximation. In contrast, the amount of memory required by *i*KBSF is independent of the number of sample transitions $n$. This fact together with the fact that KBSF's computational complexity is only linear in $n$ allow our algorithm to process a large amount of data in reasonable time. This can be clearly observed in Figure 8b, which shows that *i*KBSF can build an approximation using $10^7$ sample transitions in under 20 minutes. As a reference for comparison, FQIT(1000) took an average of 1 hour and 18 minutes to process 10 times less data.

As shown in Figure 8a, *i*KBSF's ability to process a large number of sample transitions allows our algorithm to achieve a success rate of approximately 80%. This is similar to the performance of batch KBSF on the two-pole version of the problem (*cf.* Figure 4). The good performance of *i*KBSF on the triple pole-balancing task is especially impressive when we recall that the decision policies were evaluated on a set of test states representing all possible directions of inclination of the three poles. In order to achieve the same level of



(a) Performance

(b) Run times

(c) Performance

(d) Run times

Figure 8: Results on the triple pole-balancing task, as a function of the number of sample transitions $n$, averaged over 50 runs. The values correspond to the fraction of episodes initiated from the test states in which the 3 poles could be balanced for 3000 steps (one minute of simulated time). The test sets were regular grids of 256 cells defined over the hypercube centered at the origin and covering 50% of the state-space axes in each dimension (see Appendix  for details). Shadowed regions represent 99% confidence intervals.

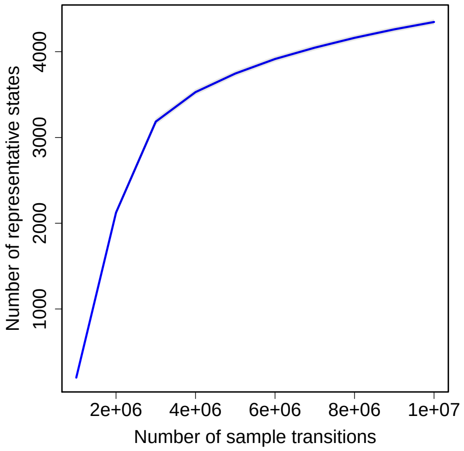


Figure 9: Number of representative states used by *i*KBSF on the triple pole-balancing task. Results were averaged over 50 runs (99% confidence intervals are almost imperceptible in the figure).

performance with KBSF, approximately 2 Gb of memory would be necessary, even using sparse kernels, whereas *i*KBSF used less than 0.03 Gb of memory.

One may argue that the comparison between FQIT and KBSF is not fair, since the latter used ten times the amount of data used by the former. Thus, in Figures 8c and 8d we show the results of FQIT using 10 batches of $10^6$ transitions—exactly the same number of transitions processed by *i*KBSF. Here we cannot compare *i*KBSF with FQIT(100) because the computational cost of the tree-based approach is prohibitively large (it would take over 4 days only to train a single agent, not counting the test phase). When we look at the other instances of the algorithm, we see two opposite trends. Surprisingly, the extra sample transitions actually made the performance of FQIT(10000) *worse*. On the other hand, FQIT(1000) performs significantly better using more data, though still not as well as *i*KBSF (both in terms of performance and computing time).

To conclude, observe in Figure 9 how the number of representative states $m$ grows as a function of the number of sample transitions processed by KBSF. As expected, in the beginning of the learning process $m$ grows fast, reflecting the fact that some relevant regions of the state space have not been visited yet. As more and more data come in, the number of representative states starts to stabilize.

### 5.2.3 Helicopter hovering (comparison with SARSA)

In the previous two sections we showed how *i*KBSF can be used to circumvent the inherent memory limitations of batch learning. We now show how our algorithm performs in a fully on-line regime. For that, we focus on a challenging reinforcement learning task in which the goal is to control an autonomous helicopter.

Helicopters have unique control capabilities, such as low speed flight and in-place hovering, that make them indispensable instruments in many contexts. Such flexibility comes at a price, though: it is widely recognized that a helicopter is significantly harder to control than a fixed-wing aircraft (Ng et al., 2003; Abbeel et al., 2007). Part of this difficulty is due to the complex dynamics of the helicopter, which is not only non-linear, noisy, and asymmetric, but also counterintuitive in some aspects (Ng et al., 2003).

An additional complication of controlling an autonomous helicopter is the fact that a wrong action can easily lead to a crash, which is both dangerous and expensive. Thus, the usual practice is to first develop a model of the helicopter's dynamics and then use the model to design a controller (Ng et al., 2003). Here we use the model constructed by Abbeel et al. (2005) based on data collected on actual flights of an XCell Tempest helicopter (see Appendix ). The resulting reinforcement learning problem has a 12-dimensional state space whose variables represent the aircraft's position, orientation, and the corresponding velocities and angular velocities along each axis.

In the version of the task considered here the goal is to keep the helicopter hovering as close as possible to a fixed position. All episodes start at the target location, and at each time step the agent receives a negative reward proportional to the distance from the current state to the desired position. Because the tail rotor's thrust exerts a sideways force on the helicopter, the aircraft cannot be held stationary in the zero-cost state even in the absence of wind. The episode ends when the helicopter leaves the hover regime, that is, when any of the state's variables exceeds pre-specified thresholds.

The helicopter is controlled via a 4-dimensional continuous vector whose variables represent the longitudinal cyclic pitch, the latitudinal cyclic pitch, the tail rotor collective pitch, and the main rotor collective pitch. By adjusting the value of these variables the pilot can rotate the helicopter around its axes and control the thrust generated by the main rotor. Since KBSF was designed to deal with a finite number of actions, we discretized the set $A$ using 4 values per dimension, resulting in 256 possible actions. The details of the discretization process are given below.

Here we compare $i$KBSF with the SARSA($\lambda$) algorithm using tile coding for value function approximation (Rummery and Niranjan, 1994, Sutton, 1996—see Appendix ). We applied SARSA with $\lambda = 0.05$, a learning rate of 0.001, and 24 tilings containing $4^{12}$ tiles each. Except for $\lambda$, all the parameters were adjusted in a set of preliminary experiments in order to improve the performance of the SARSA agent. We also defined the action-space discretization based on SARSA's performance. In particular, instead of partitioning each dimension in equally-sized intervals, we spread the break points unevenly along each axis in order to maximize the agent's return. The result of this process is described in Appendix . The interaction of the SARSA agent with the helicopter hovering task was dictated by an $\epsilon$-greedy policy. Initially we set $\epsilon = 1$, and at every 50000 transitions the value of $\epsilon$ was decreased in 30%.

The $i$KBSF agent collected sample transitions using the same exploration regime. Based on the first batch of 50000 transitions, $m = 500$ representative states were determined by the $k$-means algorithm. No representative states were added to $i$KBSF's model after that. Both the value function and the model were updated at fixed intervals of $t_v = t_m = 50000$ transitions. We fixed $\tau = \bar{\tau} = 1$ and $\mu = \bar{\mu} = 4$.

Figure 10 shows the results obtained by SARSA and KBSF on the helicopter hovering task. Note in Figure 10a how the average episode length increases abruptly at the points in which the value of $\epsilon$ is decreased. This is true for both SARSA and KBSF. Also, since the number of steps executed per episode increases over time, the interval in between such abrupt changes decreases in length, as expected. Finally, observe how the performance of both agents stabilizes after around 70 000 episodes, probably because at this point there is almost no exploration taking place anymore.

When we compare KBSF and SARSA, it is clear that the former significantly outperforms the latter. Specifically, after the cut-point of 70 000 episodes, the KBSF agent executes approximately 2.25 times the number of steps performed by the SARSA agent before crashing. Looking at Figures 10a and 10b, one may argue at first that there is nothing surprising here: being a model-based algorithm, KBSF is more sample efficient than SARSA, but it is also considerably slower (Atkeson and Santamaria, 1997). Notice though that the difference between the run times of SARSA and KBSF shown in Figure 10b is in part a consequence of the good performance of the latter: since KBSF is able to control the helicopter for a larger number of steps, the corresponding episodes will obviously take longer. A better measure of the algorithms's computational cost can be seen in Figure 10c, which shows the average time taken by each method to perform one transition. Observe how KBSF's computing time peaks at the points in which the model and the value function are updated. In the beginning KBSF's MDP changes considerably, and as a result the value function updates take longer. As more and more data come in, the model starts to stabilize, accelerating the computation of $\bar{\mathbf{Q}}^*$ (we "warm start" policy iteration with the value function computed in the previous round). At this point, KBSF's computational cost per step is only slightly higher than SARSA's, even though the former computes a model of the environment while the latter directly updates the value function approximation.

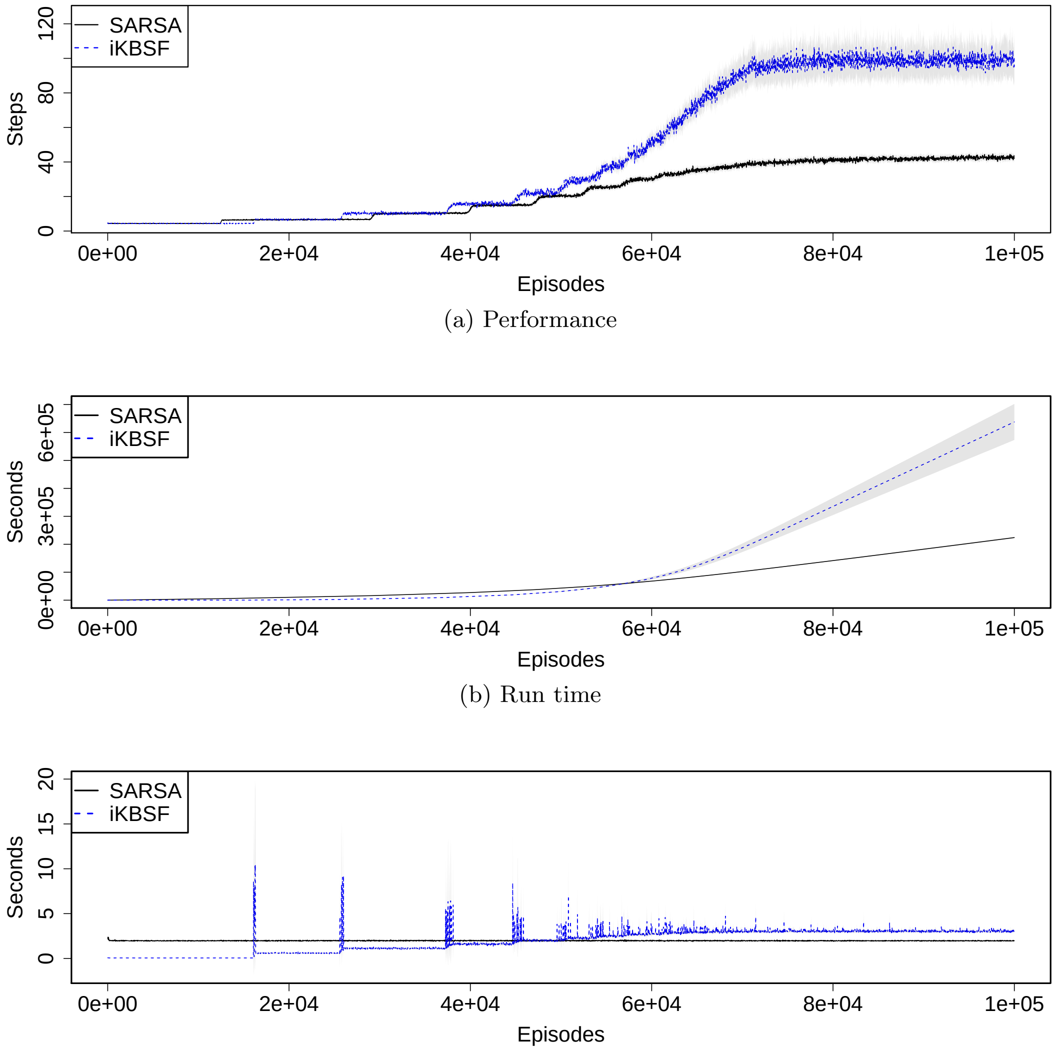
To conclude, we note that our objective in this section was exclusively to show that KBSF can outperform a well-known on-line algorithm with compatible computational cost. Therefore, we focused on the comparison of the algorithms rather than on obtaining the best possible performance on the task. Also, it is important to mention that more difficult versions of the helicopter task have been addressed in the literature, usually using domain knowledge in the configuration of the algorithms or to guide the collection of data (Ng et al., 2003; Abbeel et al., 2007). Since our focus here was on evaluating the on-line performance of KBSF, we addressed the problem in its purest form, without using any prior information to help the algorithms solve the task.

# 6 Discussion

During the execution of our experiments we observed several interesting facts about KBSF which are not immediate from its conceptual definition. In this section we share some of the lessons learned with the reader. We start by discussing the impact of deviating from the theoretical assumptions over the performance of our algorithm. We then present general guidelines on how to configure KBSF to solve reinforcement learning problems.

## 6.1 KBSF's applicability

The theoretical guarantees regarding KBRL's solution assume that the initial states $s_i^a$ in the transitions $(s_i^a, r_i^a, \hat{s}_i^a)$ are uniformly sampled from $\mathbb{S}$ (Ormoneit and Sen, 2002, see Assumption 3). This is somewhat restrictive because it precludes the collection of data



(a) Performance

(b) Run time

(c) Average time per step (time of an episode divided by the number of steps)

Figure 10: Results on the helicopter hovering task averaged over 50 runs. The learned controllers were tested from a fixed state (see text for details). The shadowed regions represent 99% confidence intervals.

through direct interaction with the environment. Ormoneit and Sen conjectured that sampling the states $s_i^a$ from an uniform distribution is not strictly necessary, and indeed later Ormoneit and Glynn (2002) relaxed this assumption for the case in which KBRL is applied to an average-reward MDP. In this case, it is only required that the exploration policy used to collect data chooses all actions with positive probability. As described in Sections  and , in our computational experiments we collected data through an $\epsilon$-greedy policy (in many cases with $\epsilon = 1$). The good performance of KBSF corroborates Ormoneit and Sen's conjecture and suggests that Ormoneit and Glynn's results can be generalized to the discounted reward case, but more theoretical analysis is needed.

Ormoneit and Sen (2002) also make some assumptions regarding the smoothness of the reward function and the transition kernel of the continuous MDP (Assumptions 1 and 2). Unfortunately, such assumptions are usually not verifiable in practice. Empirically, we observed that KBSF indeed performs better in problems with "smooth dynamics"—loosely speaking, problems in which a small perturbation in $s_i^a$ results in a small perturbation in $\hat{s}_i^a$, such as the pole balancing task. In problems with "rougher" dynamics, like the epilepsy-suppression task, it is still possible to get good results with KBSF, but in this case it is necessary to use more representative states and narrower kernels (that is, smaller values for $\bar{\tau}$). As a result, in problems of this type KBSF is less effective in reducing KBRL's computational cost.

## 6.2 KBSF's configuration

The performance of KBSF depends crucially on the definition of the representative states $\bar{s}_j$. Looking at expression (17), we see that ideally these states would be such that the rows of the matrices $\mathbf{K}^a$ would form a convex hull containing the rows of the corresponding $\hat{\mathbf{P}}^a$. However, it is easy to see that when $m < n$ such a set of states may not exist. Even when it does exist, finding this set is not a trivial problem.

Instead of insisting on finding representative states that allow for an exact representation of the matrices $\hat{\mathbf{P}}^a$, it sounds more realistic to content oneself with an approximate solution for this problem. Proposition 3 suggests that a reasonable strategy to define the representative states is to control the magnitude of $\max_{a,i} dist(\hat{s}_i^a, 1)$, the maximum distance from a sampled state $\hat{s}_i^a$ to the nearest representative state. Based on this observation, in our experiments we clustered the states $\hat{s}_i^a$ and used the clusters's centers as our representative states. Despite its simplicity, this strategy usually results in good performance, as shown in Sections  and .

Of course, other approaches are possible. The simplest technique is perhaps to select representative states at random from the set of sampled states $\hat{s}_i^a$. As shown in Section , this strategy seems to work reasonably well when adopted together with model averaging. Another alternative is to resort to quantization approaches other than $k$-means (Kaufman and Rousseeuw, 1990). Among them, a promising method is Beygelzimer et al.'s (2006) cover tree, since it directly tries to minimize $\max_{a,i} dist(\hat{s}_i^a, 1)$ and can be easily updated on-line (the idea of using cover trees for kernel-based reinforcement learning was first proposed by Kveton and Theocharous, 2012). Yet another possibility is to fit a mixture of Gaussians to the sampled states $\hat{s}_i^a$ (Hastie et al., 2002, Chapter 6).

The definition of the representative states can also be seen as an opportunity to incorporate prior knowledge about the domain of interest into the approximation model. For example, if one knows that some regions of the state space are more important than others, this information can be used to allocate more representative states to those regions. Similar reasoning applies to tasks in which the level of accuracy required from the decision policy varies across the state space. Regardless of how exactly the representative states are defined, by using *i*KBSF one can always add new ones on-line if necessary (see Section ).

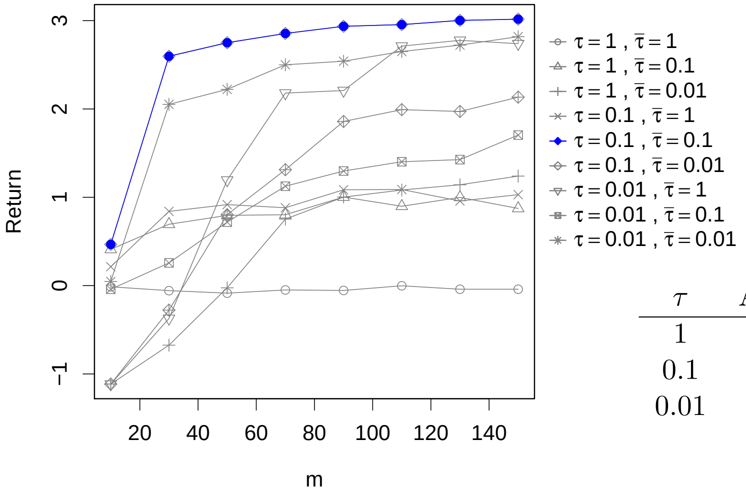
Given a well-defined strategy to select representative states, the use of KBSF requires the definition of three parameters: the number of representative states, $m$, and the widths of the kernels used by the algorithm, $\tau$ and $\bar{\tau}$. Both theory and practice indicate that KBSF's performance generally improves when $m$ is increased. Thus, a "rule of thumb" to define the number of representative states is to simply set $m$ to the largest value allowed by the available computational resources. This reduces KBSF's configuration to the definition of the kernels's widths.

The parameters $\tau$ and $\bar{\tau}$ may have a strong effect on KBSF's performance. To illustrate this point, we show in Figure 11 the results of this algorithm on the puddle world task when $\tau$ and $\bar{\tau}$ are varied in the set $\{0.01, 0.1, 1\}$ (these were the results used to generate Figure 3). Of course, the best combination of values for $\tau$ and $\bar{\tau}$ depends on the specific problem at hand and on the particular choice of kernels. Here we give some general advice as to how to set these parameters, based on both theory in practice. Since $\tau$ is the same parameter used by KBRL, it should decrease with the number of sample transitions $n$ at an "admissible rate" (see Ormoneit and Sen's Lemma 2, 2002). Analogously, Proposition 3 suggests that $\bar{\tau}$ should get smaller as $m \to n$. Empirically, we found out that a simple strategy that usually facilitates the configuration of KBSF is to rescale the data so that all the variables have approximately the same magnitude—which corresponds to using a weighted norm in the computation of the kernels. Using this strategy we were able to obtain good results with KBSF on all problems by performing a coarse search in the space of parameters in which we only varied the order of magnitude of $\tau$ and $\bar{\tau}$ (see Table 1 on page 56).

Alternatively, one can fix $\tau$ and $\bar{\tau}$ and define the neighborhood used to compute $\mathrm{k}_\tau(\bar{s}_j, \cdot)$ and $\bar{\mathrm{k}}_{\bar{\tau}}(\hat{s}_i^a, \cdot)$. As explained in Appendix , in some of our experiments we only computed $\mathrm{k}_\tau(\bar{s}_j, \cdot)$ for the $\mu$ closest sampled states $s_i^a$ from $\bar{s}_j$, and only computed $\bar{\mathrm{k}}_{\bar{\tau}}(\hat{s}_i^a, \cdot)$ for the $\bar{\mu}$ closest representative states from $\hat{s}_i^a$. When using this approach, a possible way of configuring KBSF is to set $\tau$ and $\bar{\tau}$ to sufficiently large values (so as to guarantee a minimum level of overlap between the kernels) and then adjust $\mu$ and $\bar{\mu}$. The advantage is that adjusting $\mu$ and $\bar{\mu}$ may be more intuitive than directly configuring $\tau$ and $\bar{\tau}$ (*cf.* Table 1).

# 7 Previous work

In our experiments we compared KBSF with KBRL, LSPI, fitted $Q$-iteration, and SARSA, both in terms of computational cost and in terms of the quality of the resulting decision policies. In this section we situate our algorithm in the broader context of approximate reinforcement learning. Approximation in reinforcement learning is an important topic that has generated a huge body of literature. For a broad overview of the subject, we



| $\tau$ | Average return |
|---|---|
| 1 | $1.47 \pm 0.42$ |
| 0.1 | $3.01 \pm 0.08$ |
| 0.01 | $3.00 \pm 0.08$ |

(a) Performance of KBSF(8000, ·)

(b) Performance of KBRL(8000)

Figure 11: The impact of the kernels's widths on the performance of KBSF and KBRL. Results on the puddle-world task averaged over 50 runs. The errors around the mean correspond to the 99% confidence intervals. See Figure 3 for details.

refer the reader to the books by Sutton and Barto (1998), Bertsekas and Tsitsiklis (1996), and Szepesvári (2010). Here we will narrow our attention to *kernel-based* approximation techniques.

We start by noting that the label "kernel based" is used with two different meanings in the literature. On one side we have kernel smoothing techniques like KBRL and KBSF, which use local kernels essentially as a device to implement smooth instance-based approximation (Hastie et al., 2002). On the other side we have methods that use reproducing kernels to implicitly represent an inner product in a high-dimensional state space (Schölkopf and Smola, 2002). Although these two frameworks can give rise to approximators with similar structures, they rest on different theoretical foundations. Since reproducing-kernels methods are less directly related to KBSF, we will only describe them briefly. We will then discuss the kernel smoothing approaches in more detail.

The basic idea of reproducing-kernel methods is to apply the "kernel trick" in the context of reinforcement learning (Schölkopf and Smola, 2002). Roughly speaking, the approximation problem is rewritten in terms of inner products only, which are then replaced by a properly-defined kernel. This modification corresponds to mapping the problem to a high-dimensional feature space, resulting in more expressiveness of the function approximator. Perhaps the most natural way of applying the kernel trick in the context of reinforcement learning is to "kernelize" some formulation of the value-function approximation problem (Xu et al., 2005; Engel et al., 2005; Farahmand, 2011). Another alternative is to approximate the *dynamics* of an MDP using a kernel-based regression method (Rasmussen and Kuss, 2004; Taylor and Parr, 2009). Following a slightly different line of work, Bhat et al. (2012) propose to kernelize the linear programming formulation of dynamic programming. However, this method is not directly applicable to reinforcement learning, since it is based on the assumption that one has full knowledge of the MDP. A weaker assumption is to suppose that only the reward function is known and focus on the approximation of the transition function. This is the approach taken by Grunewalder et al. (2012), who propose to embed the conditional distributions defining the transitions of an MDP into a Hilbert space induced by a reproducing kernel.

We now turn our attention to kernel-smoothing techniques, which are more closely related to KBRL and KBSF. Kroemer and Peters (2011) propose to apply kernel density estimation to the problem of policy evaluation. They call their method *non-parametric dynamic programming* (NPDP). If we use KBRL to compute the value function of a fixed policy, we see many similarities with NPDP, but also some important differences. Like KBRL, NPDP is statistically consistent. Unlike KBRL, which assumes a finite action space $A$ and directly approximates the conditional density functions $P^a(s'|s)$, NPDP assumes that $A$ is continuous and models the joint density $P(s, a, s')$. Kroemer and Peters (2011) showed that the value function of NPDP has a Nadaraya-Watson kernel regression form. Not surprisingly, this is also the form of KBRL's solution if we fix the policy being evaluated (*cf.* equation (7)). In both cases, the coefficients of the kernel-based approximation are derived from the value function of the approximate MDP. The key difference is the way the transition matrices are computed in each algorithm. As shown in (4), the transition probabilities of KBRL's model are given by the kernel values themselves. In contrast, the computation of each element of NDPD's transition matrix requires an integration over the continuous state space $\mathbb{S}$. In practice, this is done by numerical integration techniques that may be very computationally demanding (see for example the experiments performed by Grunewalder et al., 2012).

We directly compared NPDP with KBRL because both algorithms build a model whose number of states is dictated by the number of sample transitions $n$, and neither method explicitly attempts to keep $n$ small. Since in this case each application of the Bellman operator is $O(n^2)$, these methods are not suitable for problems in which a large number of transitions are required, nor are they applicable to on-line reinforcement learning.[4] There are however kernel-smoothing methods that try to avoid this computational issue by either keeping $n$ small or by executing a number of operations that grows only linearly with $n$. These algorithms are directly comparable with KBSF.

One of the first attempts to adapt KBRL to the on-line scenario was that of Jong and Stone (2006). Instead of collecting a batch of sample transitions before the learning process starts, the authors propose to grow such a set incrementally, based on an exploration policy derived from KBRL's current model. To avoid running a dynamic-programming algorithm to completion in between two transitions, which may not be computationally feasible, Jong and Stone (2006) resort to Moore and Atkeson's (1993) "prioritized sweeping" method to propagate the changes in the value function every time the model is modified. The idea of exploiting the interpretation of KBRL as the derivation of a finite MDP in order

[4] We note that, incidentally, all the reproducing-kernel methods discussed in this section also have a computational complexity super-linear in $n$.

to use tabular exploration methods is insightful. However, it is not clear whether smart exploration is sufficient to overcome the computational difficulties arising from the fact that the size of the underlying model is inexorably linked to the number of sample transitions. For example, even using sparse kernels in their experiments, Jong and Stone (2006) had to fix an upper limit for the size of KBRL's model. In this case, once the number of sample transitions has reached the upper limit, all subsequent data must be ignored.

Following the same line of work, Jong and Stone (2009) later proposed to guide KBRL's exploration of the state space using Brafman and Tennenholtz's (2003) R-MAX algorithm. In this new paper the authors address the issue with KBRL's scalability more aggressively. First, they show how to combine their approach with Dietterich's (2000) MAX-Q algorithm, allowing the decomposition of KBRL's MDP into a hierarchy of simpler models. While this can potentially reduce the computational burden of finding a policy, such a strategy transfer to the user the responsibility of identifying a useful decomposition of the task. A more practical approach is to combine KBRL with some stable form of value-function approximation. For that, Jong and Stone (2009) suggest the use of Gordon's (1995) averagers. As shown in Appendix , this setting corresponds to a particular case of KBSF in which representative states are selected among the set of sampled states $\hat{s}_i^a$. It should be noted that, even when using temporal abstraction and function approximation, Jong and Stone's (2009) approach requires recomputing KBRL's transition probabilities at each new sample, which can be infeasible in reasonably large problems.

Kveton and Theocharous (2012) propose a more practical algorithm to reduce KBRL's computational cost. Their method closely resembles the batch version of KBSF. As with our algorithm, Kveton and Theocharous's (2012) method defines a set of representative states $\bar{s}_i$ that give rise to a reduced MDP. The main difference in the construction of the models is that, instead of computing a similarity measure between each sampled state $\hat{s}_i^a$ and all representative states $\bar{s}_j$, their algorithm associates each $\hat{s}_i^a$ with a single $\bar{s}_j$—which comes down to computing a hard aggregation of the state space $\hat{S}$. Such an aggregation corresponds to having a matrix $\mathbf{D}$ with a single nonzero element per row. In fact, it is possible to rewrite Kveton and Theocharous's (2012) algorithm using KBSF's formalism. In this case, the elements of $\dot{\mathbf{D}}^a$ and $\dot{\mathbf{K}}^a$ would be defined as:

$$\dot{k}_{ij}^a = \kappa_\tau^a(\bar{s}_i, rs(s_j^a, 1)), \quad \text{and} \quad \dot{d}_{ij}^a = \bar{\kappa}_0(rs(\hat{s}_i^a, 1), \bar{s}_j) \tag{25}$$

where $\bar{\kappa}_0$ is the normalized kernel induced by an infinitely "narrow" kernel $\bar{\mathrm{k}}_0(s, s')$ whose value is greater than zero if and only if $s = s'$ (recall from Section  that $rs(s, 1)$ gives the closest representative state from $s$). It is easy to see that we can make matrix $\mathbf{D}$ computed by KBSF as close as desired to a hard aggregation by setting $\bar{\tau}$ to a sufficiently small value (see Lemma 2). More practically, we can simply plug (25) in place of (13) in Algorithm 1 to exactly recover Kveton and Theocharous's method. Note though that, by replacing $\kappa_\tau^a(\bar{s}_i, s_j^a)$ with $\kappa_\tau^a(\bar{s}_i, rs(s_j^a, 1))$ in the computation of $\dot{\mathbf{K}}^a$, we would be actually deviating from KBRL's framework. To see why this is so, note that if the representative states $\bar{s}_i$ are sampled from the set of states $\hat{s}_i^a$, the rows of matrix $\mathbf{K}^a$ computed by KBSF would coincide with a subset of the rows of the corresponding KBRL's matrix $\hat{\mathbf{P}}^a$ (*cf.* (16)).

However, this property is lost if one uses (25) instead of (13).[5]

# 8 Conclusion

This paper presented KBSF, a reinforcement learning algorithm that results from the application of the stochastic-factorization trick to KBRL. KBSF summarizes the information contained in KBRL's MDP in a model of fixed size. By doing so, our algorithm decouples the structure of the model from its configuration. This makes it possible to build an approximation which accounts for both the difficulty of the problem *and* the computational resources available.

One of the main strengths of KBSF is its simplicity. As shown in the paper, its uncomplicated mechanics can be unfolded into two update rules that allow for a fully incremental version of the algorithm. This makes the amount of memory used by KBSF independent of the number of sample transitions. Therefore, with a few lines of code one has a reinforcement-learning algorithm that can be applied to large-scale problems, in both off-line and on-line regimes.

KBSF is also a sound method from a theoretical point of view. As discussed, the distance between the value function computed by this algorithm and the one computed by KBRL is bounded by two factors: the quality and the level of stochasticity of the underlying stochastic factorization. We showed that both factors can be made arbitrarily small, which implies that, in theory, we can make KBSF's solution as close to KBRL's solution as desired.

But theoretical guarantees do not always translate into practical methods, either because they are built upon unrealistic assumptions or because they do not account for procedural difficulties that arise in practice. To ensure that this is not the case with our algorithm, we presented an extensive empirical study in which KBSF was successfully applied to different problems, some of them quite challenging. We also presented general guidelines on how to configure KBSF to solve a reinforcement learning problem.

For all the reasons listed above, we believe that KBSF has the potential of becoming a valuable resource in the solution of reinforcement learning problems. This is not to say that the subject has been exhausted. There are several possibilities for future research, some of which we now briefly discuss.

From an algorithmic perspective, perhaps the most pressing demand is for more principled methods to select the representative states. Incidentally, this also opens up the possibility of an automated procedure to set the kernel's widths $\bar{\tau}$ based solely on data. Taking the idea a bit further, one can think of having one distinct $\bar{\tau}_i$ associated with each kernel $\bar{\kappa}_{\bar{\tau}}(\cdot, \bar{s}_i)$. Another important advance would be to endow *i*KBSF with more elaborate exploration strategies, maybe following the line of research initiated by Jong and Stone (2006, 2009).

Regarding the integration of KBSF to its broader context, a subject that deserves further investigation is the possibility of building an approximation based on multiple models. Model averaging is not inherently linked to KBSF, and in principle it can be used

[5] This observation does not imply that Kveton and Theocharous's algorithm is not a principled method.

with virtually any reinforcement learning algorithm. However, KBSF's low computational cost makes it particularly amenable to this technique. Since our algorithm is orders of magnitude faster than any method whose complexity per iteration is a function of the number of sample transitions, we can afford to compute several approximations and still have a solution in comparable time (see Section ). Understanding to what extend this can improve the quality of the resulting decision policy is a matter of interest.

In this paper we emphasized the role of KBSF as a technique to reduce KBRL's computational cost. However, it is equally important to ask whether our algorithm provides benefits from a statistical point of view. Ormoneit and Sen (2002) showed that, in general, the number of sample transitions needed by KBRL to achieve a certain approximation accuracy grows exponentially with the dimension of the state space. As with other methods, the only way to avoid such an exponential dependency is to explore some sort of regularity in the problem's structure—paraphrasing the authors, one can only "break" the curse of dimensionality by incorporating prior knowledge into the approximation (Ormoneit and Sen, 2002). We think that KBSF may be cast as a strategy to do so. In particular, the definition of the representative states can be interpreted as a practical mechanism to incorporate knowledge into the approximation. Whether or not this will have an impact on the algorithm's sample complexity is an interesting question for future investigation.

We conclude by noting that KBSF represents one particular way in which the stochastic-factorization trick can be exploited in the context of reinforcement learning. In principle, any algorithm that builds a model based on sample transitions can resort to the same trick to leverage the use of the data. The basic idea remains the same: instead of estimating the transition probabilities between every pair of states, one focuses on a small set of representative states whose values are propagated throughout the state space based on some notion of similarity. We believe that this general framework can potentially be materialized into a multitude of useful reinforcement learning algorithms.

# A Theoretical Results

## A.1 Assumptions

We assume that KBSF's kernel $\bar{\phi}(x) : \mathbb{R}^+ \mapsto \mathbb{R}^+$ has the following properties:

(i) $\bar{\phi}(x) \geq \bar{\phi}(y)$ if $x < y$,

(ii) $\exists\, A_{\bar{\phi}} > 0, \lambda_{\bar{\phi}} \geq 1, B \geq 0$ such that $A_{\bar{\phi}} \exp(-x) \leq \bar{\phi}(x) \leq \lambda_{\bar{\phi}} A_{\bar{\phi}} \exp(-x)$ if $x \geq B$.

Given $\bar{\phi}$, we will denote by $B_{\bar{\phi}}$ the smallest $B$ that satisfies (ii). Assumption (ii) implies that the function $\bar{\phi}$ is positive and will eventually decay exponentially. Note that we assume that $\bar{\phi}$ is greater than zero everywhere in order to guarantee that $\bar{\kappa}_{\bar{\tau}}$ is well defined for any value of $\bar{\tau}$. It should be straightforward to generalize our results for the case in which $\bar{\phi}$ has finite support by ensuring that, given sets of sample transitions $S^a$ and a set of representative states $\bar{S}$, $\bar{\tau}$ is such that, for any $\hat{s}^a_i \in S^a$, with $a \in A$, there is a $\bar{s}_j \in \bar{S}$ for which $\bar{\mathrm{k}}_{\bar{\tau}}(\hat{s}^a_i, \bar{s}_j) > 0$ (note that this assumption is naturally satisfied by the "sparse kernels" used in some of the experiments).

## A.2 Proofs

**Lemma 1** *For any $s_i^a \in S^a$ and any $\epsilon > 0$, there is a $\delta > 0$ such that $|\kappa_\tau^a(s, s_i^a) - \kappa_\tau^a(s', s_i^a)| < \epsilon$ if $\|s - s'\| < \ \delta$.*

*Proof.* Define the function

$$\psi_{\tau,s}^{a,i}(s') = \left| \frac{\mathrm{k}_\tau(s, s_i^a)}{\sum_{j=1}^{n_a} \mathrm{k}_\tau(s, s_j^a)} - \frac{\mathrm{k}_\tau(s', s_i^a)}{\sum_{j=1}^{n_a} \mathrm{k}_\tau(s', s_j^a)} \right| = \left| \frac{\phi\left(\|s - s_i^a\|/\tau\right)}{\sum_{j=1}^{n_a} \phi\left(\|s - s_j^a\|/\tau\right)} - \frac{\phi\left(\|s' - s_i^a\|/\tau\right)}{\sum_{j=1}^{n_a} \phi\left(\|s' - s_j^a\|/\tau\right)} \right|.$$

Since $\phi$ is continuous, it is obvious that $\psi_{\tau,s}^{a,i}(s')$ is also continuous in $s'$. The property follows from the fact that $\lim_{s' \to s} \psi_{\tau,s}^{a,i}(s') = 0$. □

**Lemma 2**[†] *Let $s \in \mathbb{S}$, let $m > 1$, and assume there is a $w \in \{1, 2, ..., m-1\}$ such that $dist(s, w) < dist(s, w+1)$. Define $W \equiv \{k \mid \|s \ - \bar{s}_k\| \leq dist(s, w)\}$ and $\bar{W} \equiv \{1, 2, ..., m\} - W$. Then, for any $\alpha > 0$, we can guarantee that*

$$\sum_{k \in \bar{W}} \bar{\kappa}_{\bar{\tau}}(s, \bar{s}_k) < \alpha \sum_{k \in W} \bar{\kappa}_{\bar{\tau}}(s, \bar{s}_k) \tag{26}$$

*by making $\bar{\tau} < \varphi(s, w, m, \alpha)$, where*

$$\varphi(s, w, m, \alpha) = \min(\varphi_1(s, w), \varphi_2(s, w, m, \alpha)) \tag{27}$$

*and*

$$\varphi_1(s, w) = \begin{cases} \dfrac{dist(s, w)}{B_{\bar{\phi}}}, & \text{if } B_{\bar{\phi}} > 0, \\ \infty, & \text{otherwise}, \end{cases} \quad \varphi_2(s, w, m, \alpha) = \begin{cases} \dfrac{dist(s, w) - dist(s, w+1)}{\ln(\alpha w/(m-w)\lambda_{\bar{\phi}})}, & \text{if } \dfrac{\alpha w}{(m-w)\lambda_{\bar{\phi}}} < 1, \\ \infty, & \text{otherwise}. \end{cases}$$

*Proof.* Expression (26) can be rewritten as

$$\frac{\sum_{k \in \bar{W}} \bar{\mathrm{k}}_{\bar{\tau}}(s, \bar{s}_k)}{\sum_{i=1}^m \bar{\mathrm{k}}_{\bar{\tau}}(s, \bar{s}_i)} < \alpha \frac{\sum_{k \in W} \bar{\mathrm{k}}_{\bar{\tau}}(s, \bar{s}_k)}{\sum_{i=1}^m \bar{\mathrm{k}}_{\bar{\tau}}(s, \bar{s}_i)} \iff \sum_{k \in \bar{W}} \bar{\mathrm{k}}_{\bar{\tau}}(s, \bar{s}_k) < \alpha \sum_{k \in W} \bar{\mathrm{k}}_{\bar{\tau}}(s, \bar{s}_k),$$

which is equivalent to

$$\sum_{k \in \bar{W}} \bar{\phi}\left(\frac{\|s - \bar{s}_k\|}{\bar{\tau}}\right) < \alpha \sum_{k \in W} \bar{\phi}\left(\frac{\|s - \bar{s}_k\|}{\bar{\tau}}\right). \tag{28}$$

Based on Assumption (i), we know that a sufficient condition for (28) to hold is

[†] We restate the lemma here showing explicitly how to define $\bar{\tau}$. This detail was omitted in the main body of the text to improve clarity.

$$\bar{\phi}\left(\frac{dist(s,w+1)}{\bar{\tau}}\right) < \frac{\alpha w}{m-w}\bar{\phi}\left(\frac{dist(s,w)}{\bar{\tau}}\right). \tag{29}$$

Let $\beta = \alpha w/(m-w)$. If $\beta > 1$, then (29) is always true, regardless of the value of $\bar{\tau}$. We now show that, when $\beta \leq 1$, it is always possible to set $\bar{\tau}$ in order to guarantee that (29) holds. Let $z = dist(s,w)$ and let $\delta = dist(s,w+1) - z$. From Assumption (ii), we know that, if $B_{\bar{\phi}} = 0$ or $\bar{\tau} < z/B_{\bar{\phi}}$,

$$\frac{\bar{\phi}((z+\delta)/\bar{\tau})}{\bar{\phi}(z/\bar{\tau})} \leq \frac{\lambda_{\bar{\phi}} A_{\bar{\phi}} \exp(-(z+\delta)/\bar{\tau})}{A_{\bar{\phi}} \exp(-z/\bar{\tau})} = \frac{\lambda_{\bar{\phi}} \exp(-(z+\delta)/\bar{\tau})}{\exp(-z/\bar{\tau})}.$$

Thus, in order for the result to follow, it suffices to show that

$$\frac{\exp(-(z+\delta)/\bar{\tau})}{\exp(-z/\bar{\tau})} < \frac{\beta}{\lambda_{\bar{\phi}}}. \tag{30}$$

We know that, since $\delta > 0$, if $\beta/\lambda_{\bar{\phi}} = 1$ inequality (30) is true. Otherwise,

$$\frac{\exp(-(z+\delta)/\bar{\tau})}{\exp(-z/\bar{\tau})} < \frac{\beta}{\lambda_{\bar{\phi}}} \iff \ln\left(\frac{\exp(-(z+\delta)/\bar{\tau})}{\exp(-z/\bar{\tau})}\right) < \ln\left(\frac{\beta}{\lambda_{\bar{\phi}}}\right)$$
$$\iff -\frac{\delta}{\bar{\tau}} < \ln\left(\frac{\beta}{\lambda_{\bar{\phi}}}\right) \iff \bar{\tau} < -\frac{\delta}{\ln(\beta/\lambda_{\bar{\phi}})}.$$

Thus, by taking $\bar{\tau} < -\delta/\ln(\beta/\lambda_{\bar{\phi}})$ if $B_{\bar{\phi}} = 0$, or $\bar{\tau} < \min(-\delta/\ln(\beta/\lambda_{\bar{\phi}}), z/B_{\bar{\phi}})$ otherwise, the result follows. □

**Note:** We briefly provide some intuition on the functions $\varphi_1$ and $\varphi_2$. Since we know from Assumption (i) that $\bar{\phi}$ is non-increasing, we can control the magnitude of $\sum_{k\in\bar{W}} \bar{\kappa}_{\bar{\tau}}(s,\bar{s}_k)$ / $\sum_{k\in W} \bar{\kappa}_{\bar{\tau}}(s,\bar{s}_k)$ by controlling

$$\frac{\bar{\kappa}_{\bar{\tau}}(s, rs(s,w+1))}{\bar{\kappa}_{\bar{\tau}}(s, rs(s,w))} = \frac{\bar{\phi}(dist(s,w+1)/\bar{\tau})}{\bar{\phi}(dist(s,w)/\bar{\tau})}. \tag{31}$$

Function $\varphi_1$ imposes an upper bound on $\bar{\tau}$ in order to ensure that $dist(s,w)/\bar{\tau} \geq B_{\bar{\phi}}$. This implies that $\bar{\phi}(dist(s,w)/\bar{\tau})$ will be in the "exponential region" of $\bar{\phi}$, which makes it possible to control the magnitude of (31) by adjusting $\bar{\tau}$. In particular, because of Assumption (ii), we know that $\bar{\phi}(dist(s,w+1)/\bar{\tau})/\bar{\phi}(dist(s,w)/\bar{\tau}) \to 0$ as $\bar{\tau} \to 0$. Function $\varphi_2$ exploits this fact, decreasing the maximum allowed value for $\bar{\tau}$ according to two factors. The first one is the difference of magnitude of $dist(s,w+1)$ and $dist(s,w)$. This is easy to understand. Suppose we want to make (31) smaller than a given threshold. If $rs(s,w+1)$ is much farther from $s$ than $rs(s,w)$, the value of $\bar{\phi}(dist(s,w+1)/\bar{\tau})$ will be considerably smaller than the value of $\bar{\phi}(dist(s,w)/\bar{\tau})$ even if $\bar{\tau}$ is large. On the other hand, if the difference of magnitude of $dist(s,w+1)$ and $dist(s,w)$ is small, we have to decrease $\bar{\tau}$ to ensure that (31) is sufficiently small. Therefore, the upper bound for $\bar{\tau}$ set by $\varphi_2$

decreases with $|dist(s, w+1) - dist(s, w)|$. The second factor that influences this upper bound is $w/(m-w)$, the relative sizes of the sets $W$ and $\bar{W}$. Again, this is not hard to understand: as we reduce the size of $W$, we also decrease the number of terms in the sum $\sum_{k \in W} \bar{\kappa}_{\bar{\tau}}(s, \bar{s}_k)$, and thus we must decrease the ratio (31) to make sure that $\sum_{k \in \bar{W}} \bar{\kappa}_{\bar{\tau}}(s, \bar{s}_k) / \sum_{k \in W} \bar{\kappa}_{\bar{\tau}}(s, \bar{s}_k)$ is sufficiently small. Thus, the upper bound on $\bar{\tau}$ defined by $\varphi_2$ grows with $w/(m-w)$.

**Proposition 3** *For any $\epsilon > 0$, there is a $\delta > 0$ such that, if $\max_{a,i} dist(\hat{s}_i^a, 1) < \delta$, then we can guarantee that $\xi_v < \epsilon$ by making $\bar{\tau}$ sufficiently small.*

*Proof.* From (6) and (14), we know that

$$\|\hat{\mathbf{r}}^a - \mathbf{D}\bar{\mathbf{r}}^a\|_\infty = \|\hat{\mathbf{P}}^a \mathbf{r} - \mathbf{D}\mathbf{K}^a \mathbf{r}\|_\infty = \|(\hat{\mathbf{P}}^a - \mathbf{D}\mathbf{K}^a)\mathbf{r}\|_\infty \leq \|\hat{\mathbf{P}}^a - \mathbf{D}\mathbf{K}^a\|_\infty \|\mathbf{r}\|_\infty. \tag{32}$$

Thus, plugging (32) back into (8), it is clear that there is a $\nu > 0$ such that $\xi_v < \epsilon$ if

$$\max_a \|\hat{\mathbf{P}}^a - \mathbf{D}\mathbf{K}^a\|_\infty < \nu \tag{33}$$

and

$$\max_i \,(1 - \max_j d_{ij}) < \nu. \tag{34}$$

We start by showing that there is a $\delta > 0$ and a $\theta > 0$ such that expression (33) is true if $\max_{a,i} dist(\hat{s}_i^a, 1) < \delta$ and $\bar{\tau} < \theta$. Let $\check{\mathbf{P}}^a = \mathbf{D}\mathbf{K}^a$ and let $\hat{\mathbf{p}}_i^a \in \mathbb{R}^{1 \times n}$ and $\check{\mathbf{p}}_i^a \in \mathbb{R}^{1 \times n}$ be the $i^{\text{th}}$ rows of $\hat{\mathbf{P}}^a$ and $\check{\mathbf{P}}^a$, respectively. Then,

$$\begin{array}{ll} \|\hat{\mathbf{p}}_i^a - \check{\mathbf{p}}_i^a\|_\infty & = \sum_{j=1}^{n_a} |\hat{p}_{ij}^a - \sum_{k=1}^{m} \dot{d}_{ik}^a \dot{k}_{kj}^a| \\ & = \sum_{j=1}^{n_a} |\kappa_\tau^a(\hat{s}_i^a, s_j^a) - \sum_{k=1}^{m} \bar{\kappa}_{\bar{\tau}}(\hat{s}_i^a, \bar{s}_k) \kappa_\tau^a(\bar{s}_k, s_j^a)| \\ & = \sum_{j=1}^{n_a} |\sum_{k=1}^{m} \bar{\kappa}_{\bar{\tau}}(\hat{s}_i^a, \bar{s}_k) \kappa_\tau^a(\hat{s}_i^a, s_j^a) - \sum_{k=1}^{m} \bar{\kappa}_{\bar{\tau}}(\hat{s}_i^a, \bar{s}_k) \kappa_\tau^a(\bar{s}_k, s_j^a)| \\ & = \sum_{j=1}^{n_a} |\sum_{k=1}^{m} \bar{\kappa}_{\bar{\tau}}(\hat{s}_i^a, \bar{s}_k) [\kappa_\tau^a(\hat{s}_i^a, s_j^a) - \kappa_\tau^a(\bar{s}_k, s_j^a)]| \\ & \leq \sum_{j=1}^{n_a} \sum_{k=1}^{m} \bar{\kappa}_{\bar{\tau}}(\hat{s}_i^a, \bar{s}_k) \left| \kappa_\tau^a(\hat{s}_i^a, s_j^a) - \kappa_\tau^a(\bar{s}_k, s_j^a) \right|. \end{array} \tag{35}$$

Our strategy will be to show that, for any $a$, $i$, and $j$, there is a $\delta^{a,i,j} > 0$ and a $\theta^{a,i,j} > 0$ such that

$$\sum_{k=1}^{m} \bar{\kappa}_{\bar{\tau}}(\hat{s}_i^a, \bar{s}_k) |\kappa_\tau^a(\hat{s}_i^a, s_j^a) - \kappa_\tau^a(\bar{s}_k, s_j^a)| < \frac{\nu}{n_a} \tag{36}$$

if $dist(\hat{s}_i^a, 1) < \delta^{a,i,j}$ and $\bar{\tau} < \theta^{a,i,j}$. To simplify the notation, we will use the superscript '$z$' meaning '$a, i, j$'. Define $\varsigma_k^z \equiv |\kappa_\tau^a(\hat{s}_i^a, s_j^a) - \kappa_\tau^a(\bar{s}_k, s_j^a)|$. From Lemma 1 we know that there is a $\delta^z > 0$ such that $\varsigma^z_k < \nu/n_a$ if $\|\hat{s}^a_i - \bar{s}_k\| < \delta^z$. Let $W^z \equiv \{k \mid \|\hat{s}^a_i \ - \bar{s}_k\| < \delta^z\}$ and $\bar{W}^z \equiv \{1, 2, ..., m\} - W^z$. Since we are assuming that $dist(\hat{s}^a_i, 1) < \delta^z$, we know that $W^z \neq \emptyset$. In this case, we can write:

$$\sum_{k=1}^{m} \bar{\kappa}_{\bar{\tau}}(\hat{s}^a_i, \bar{s}_k)\varsigma^z_k = \sum_{k \in W^z} \bar{\kappa}_{\bar{\tau}}(\hat{s}^a_i, \bar{s}_k)\varsigma^z_k + \sum_{k \in \bar{W}^z} \bar{\kappa}_{\bar{\tau}}(\hat{s}^a_i, \bar{s}_k)\varsigma^z_k.$$

Let

$$\varsigma^z_{\min} = \begin{cases} \min\limits_{k \in W^z} \{\varsigma^z_k | \varsigma^z_k > 0\} \text{ if } \max\limits_{k \in W^z} \varsigma^z_k > 0, \\ 0 \text{ otherwise} \end{cases} \quad \text{and} \quad \bar{\varsigma}^z_{\max} = \begin{cases} \max\limits_{k \in \bar{W}^z} \varsigma^z_k \text{ if } |W^z| < m, \\ 0 \text{ otherwise.} \end{cases}$$

If $\bar{\varsigma}^z_{\max} = 0$, inequality (36) is necessarily true, since $\sum_{k \in W^z} \bar{\kappa}_{\bar{\tau}}(\hat{s}^a_i, \bar{s}_k)\varsigma^z_k \leq \max\limits_{k \in W^z} \varsigma^z_k < \nu/n_a$. We now turn to the case in which $\bar{\varsigma}^z_{\max} > 0$. Suppose first that $\varsigma^z_{\min} = 0$. In this case, we have to show that there is a $\bar{\tau}$ that yields

$$\textstyle\sum_{k \in \bar{W}^z} \bar{\kappa}_{\bar{\tau}}(\hat{s}^a_i, \bar{s}_k)\varsigma^z_k < \dfrac{\nu}{n_a}. \tag{37}$$

A sufficient condition for (37) to be true is

$$\sum_{k \in \bar{W}^z} \bar{\kappa}_{\bar{\tau}}(\hat{s}^a_i, \bar{s}_k) < \frac{\nu}{n_a \bar{\varsigma}^z_{\max}} \iff \frac{1}{\sum_{j=1}^{m} \bar{\mathrm{k}}_{\bar{\tau}}(\hat{s}^a_i, \bar{s}_j)} \sum_{k \in \bar{W}^z} \bar{\mathrm{k}}_{\bar{\tau}}(\hat{s}^a_i, \bar{s}_k) < \frac{\nu}{n_a \bar{\varsigma}^z_{\max}}. \tag{38}$$

Obviously, if $\bar{\varsigma}^z_{\max} \leq \nu/n_a$ inequality (38) is always true, regardless of the value of $\bar{\tau}$. Otherwise, we can rewrite (38) as

$$\sum_{k \in \bar{W}^z} \bar{\mathrm{k}}_{\bar{\tau}}(\hat{s}^a_i, \bar{s}_k) < \frac{\nu}{n_a \bar{\varsigma}^z_{\max}} \left( \sum_{j \in W^z} \bar{\mathrm{k}}_{\bar{\tau}}(\hat{s}^a_i, \bar{s}_j) + \sum_{k \in \bar{W}^z} \bar{\mathrm{k}}_{\bar{\tau}}(\hat{s}^a_i, \bar{s}_k) \right),$$

and, after a few algebraic manipulations, we obtain

$$\sum_{k \in \bar{W}^z} \bar{\mathrm{k}}_{\bar{\tau}}(\hat{s}^a_i, \bar{s}_k) < \frac{\nu}{n_a \bar{\varsigma}^z_{\max} - \nu} \sum_{k \in W^z} \bar{\mathrm{k}}_{\bar{\tau}}(\hat{s}^a_i, \bar{s}_k), \iff \sum_{k \in \bar{W}^z} \bar{\kappa}_{\bar{\tau}}(\hat{s}^a_i, \bar{s}_k) < \frac{\nu}{n_a \bar{\varsigma}^z_{\max} - \nu} \sum_{k \in W^z} \bar{\kappa}_{\bar{\tau}}(\hat{s}^a_i, \bar{s}_k). \tag{39}$$

We can guarantee that (39) is true by applying Lemma 2. Before doing so, though, lets analyze the case in which $\varsigma^z_{\min} > 0$. Define

$$\beta^z = \frac{\nu}{n_a \sum_{k \in W^z} \bar{\kappa}_{\bar{\tau}}(\hat{s}^a_i, \bar{s}_k)\varsigma^z_k} - 1 \tag{40}$$

(note that $\beta^z > 0$ because $\sum_{k \in W^z} \bar{\kappa}_{\bar{\tau}}(\hat{s}^a_i, \bar{s}_k)\varsigma^z_k < v/n_a$). In order for (36) to hold, we must show that there is a $\bar{\tau}$ that guarantees that

$$\textstyle\sum_{k\in\bar{W}^z} \bar{\kappa}_{\bar{\tau}}(\hat{s}_i^a, \bar{s}_k)\varsigma_k^z - \beta^z \sum_{k\in W^z} \bar{\kappa}_{\bar{\tau}}(\hat{s}_i^a, \bar{s}_k)\varsigma_k^z < 0. \tag{41}$$

A sufficient condition for (41) to hold is

$$\textstyle\sum_{k\in\bar{W}^z} \bar{\kappa}_{\bar{\tau}}(\hat{s}_i^a, \bar{s}_k) < \dfrac{\beta^z \varsigma^z_{\text{min}}}{\bar{\varsigma}^z_{\text{max}}} \sum_{k\in W^z} \bar{\kappa}_{\bar{\tau}}(\hat{s}_i^a, \bar{s}_k). \tag{42}$$

Observe that expressions (39) and (42) only differ in the coefficient multiplying the right-hand side of the inequalities. Let $\alpha^z < \min(\nu/(\bar{\varsigma}^z_{\text{max}} n_a - \nu), \beta^z \varsigma^z_{\text{min}}/\bar{\varsigma}^z_{\text{max}})$. Then, if we make $\theta^z = \varphi(\hat{s}_i^a, |W|, m, \alpha^z)$, with $\varphi$ defined in (27), we can apply Lemma 2 to guarantee that (36) holds. Finally, if we let $\delta = \min_z \delta^z = \min_{a,i,j} \delta^{a,i,j}$ and $\theta = \min_z \theta^z = \min_{a,i,j} \theta^{a,i,j}$, we can guarantee that (36) is true for all $a$, $i$, and $j$, which implies that (33) is also true (see (35)).

It remains to show that there is a $\omega > 0$ such that (34) is true if $\bar{\tau} < \omega$. Recalling that, for any $i$ and any $a$,

$$\max_j d^{a}_{ij} = \frac{\bar{\mathrm{k}}_{\bar{\tau}}(\hat{s}_i^a, rs(\hat{s}_i^a, 1))}{\sum_{k=1}^m \bar{\mathrm{k}}_{\bar{\tau}}(\hat{s}_i^a, \bar{s}_k)},$$

we want to show that

$$\bar{\mathrm{k}}_{\bar{\tau}}(\hat{s}_i^a, rs(\hat{s}_i^a, 1)) > (1-\nu)\left[\bar{\mathrm{k}}_{\bar{\tau}}(\hat{s}_i^a, rs(\hat{s}_i^a, 1)) + \sum_{k=2}^m \bar{\mathrm{k}}_{\bar{\tau}}(\hat{s}_i^a, rs(\hat{s}_i^a, k))\right],$$

which is equivalent to

$$(1-\nu)\sum_{k=2}^m \bar{\mathrm{k}}_{\bar{\tau}}(\hat{s}_i^a, rs(\hat{s}_i^a, k)) < \nu \bar{\mathrm{k}}_{\bar{\tau}}(\hat{s}_i^a, rs(\hat{s}_i^a, 1)). \tag{43}$$

If $\nu \geq 1$, inequality (43) is true regardless of the particular choice of $\bar{\tau}$. Otherwise, we can rewrite (43) as

$$\sum_{k=2}^m \bar{\mathrm{k}}_{\bar{\tau}}(\hat{s}_i^a, rs(\hat{s}_i^a, k)) < \frac{\nu}{1-\nu}\bar{\mathrm{k}}_{\bar{\tau}}(\hat{s}_i^a, rs(\hat{s}_i^a, 1)) \iff \sum_{k=2}^m \bar{\kappa}_{\bar{\tau}}(\hat{s}_i^a, rs(\hat{s}_i^a, k)) < \frac{\nu}{1-\nu}\bar{\kappa}_{\bar{\tau}}(\hat{s}_i^a, rs(\hat{s}_i^a, 1)). \tag{44}$$

Let $\alpha = \nu/(1-\nu)$. Then, if we make $\omega^{a,i} = \varphi(\hat{s}_i^a, 1, m, \alpha)$, with $\varphi$ defined in (27), we can resort to Lemma 2 to guarantee that (44) holds. As before, if we let $\omega = \min_{a,i} \omega^{a,i}$, we can guarantee that (34) is true. Finally, by making $\bar{\tau} = \min(\theta, \omega)$, the result follows. □

**Lemma 3** *Let $M \equiv (S, A, \mathbf{P}^a, \mathbf{r}^a, \gamma)$ and $\tilde{M} \equiv (S, A, \tilde{\mathbf{P}}^a, \tilde{\mathbf{r}}^a, \gamma)$ be two finite MDPs. Then, for any $s \in S$ and any $a \in A$,*

$$|Q^*(s,a) - \tilde{Q}^*(s,a)| \leq \frac{1}{1-\gamma}\max_a \|\mathbf{r}^a - \tilde{\mathbf{r}}^a\|_\infty + \frac{\gamma(2-\gamma)}{2(1-\gamma)^2} R_{\text{dif}} \max_a \|\mathbf{P}^a - \tilde{\mathbf{P}}^a\|_\infty, \tag{45}$$

*where* $R_{\text{dif}} = \max_{a,i} r_i^a - \min_{a,i} r_i^a$.

*Proof.* Let $\mathbf{q}_*^a, \tilde{\mathbf{q}}_*^a \in \mathbb{R}^{|S|}$ be the $a^{\text{th}}$ columns of $\mathbf{Q}^*$ and $\tilde{\mathbf{Q}}^*$, respectively. Then,

$$\begin{aligned}
\|\mathbf{q}_*^a - \tilde{\mathbf{q}}_*^a\|_\infty &= \left\|\mathbf{r}^a + \gamma\mathbf{P}^a\mathbf{v}^* - \tilde{\mathbf{r}}^a - \gamma\tilde{\mathbf{P}}^a\tilde{\mathbf{v}}^*\right\|_\infty \\
&\leq \|\mathbf{r}^a - \tilde{\mathbf{r}}^a\|_\infty + \gamma\left\|\mathbf{P}^a\mathbf{v}^* - \tilde{\mathbf{P}}^a\tilde{\mathbf{v}}^*\right\|_\infty \\
&= \|\mathbf{r}^a - \tilde{\mathbf{r}}^a\|_\infty + \gamma\left\|\mathbf{P}^a\mathbf{v}^* - \tilde{\mathbf{P}}^a\mathbf{v}^* + \tilde{\mathbf{P}}^a\mathbf{v}^* - \tilde{\mathbf{P}}^a\tilde{\mathbf{v}}^*\right\|_\infty \\
&\leq \|\mathbf{r}^a - \tilde{\mathbf{r}}^a\|_\infty + \gamma\left\|\mathbf{v}^*(\mathbf{P}^a - \tilde{\mathbf{P}}^a)\right\|_\infty + \gamma\left\|\tilde{\mathbf{P}}^a(\mathbf{v}^* - \tilde{\mathbf{v}}^*)\right\|_\infty \\
&\leq \|\mathbf{r}^a - \tilde{\mathbf{r}}^a\|_\infty + \gamma\left\|\mathbf{v}^*(\mathbf{P}^a - \tilde{\mathbf{P}}^a)\right\|_\infty + \gamma\left\|\mathbf{v}^* - \tilde{\mathbf{v}}^*\right\|_\infty, \qquad (46)
\end{aligned}$$

where in the last step we used the fact that $\tilde{\mathbf{P}}^a$ is stochastic, and thus $\|\tilde{\mathbf{P}}^a\mathbf{v}\|_\infty \leq \|\mathbf{v}\|_\infty$ for any $\mathbf{v}$. We now provide a bound for $\|\mathbf{v}^*(\mathbf{P}^a - \tilde{\mathbf{P}}^a)\|_\infty$. Let $\mathbf{A} = \mathbf{P}^a - \tilde{\mathbf{P}}^a$. Then, for any $i$, $\sum_j a_{ij} = \sum_j (p_{ij}^a - \tilde{p}_{ij}^a) = \sum_j p_{ij}^a - \sum_j \tilde{p}_{ij}^a = 0$, that is, the elements in each row of $\mathbf{A}$ sum to zero. Let $a_i^+$ be the sum of positive elements in the $i^{\text{th}}$ row of $\mathbf{A}$ and let $a_{\max}^+ = \max_i a_i^+$. It should be clear that $\|\mathbf{A}\|_\infty = 2a_{\max}^+$. Then, for any $i$,

$$\begin{aligned}
|\sum_j a_{ij}v_j^*| &\leq \sum_{(j:a_{ij}>0)} a_{ij}v_{\max}^* + \sum_{(j:a_{ij}<0)} a_{ij}v_{\min}^* = a_i^+ v_{\max}^* - a_i^+ v_{\min}^* \leq a_{\max}^+(v_{\max}^* - v_{\min}^*) \\
&\leq \frac{a_{\max}^+}{1-\gamma}(r_{\max}^a - r_{\min}^a) \leq \frac{a_{\max}^+ R_{\text{dif}}}{1-\gamma} = \frac{R_{\text{dif}}}{2(1-\gamma)}\|\mathbf{P}^a - \tilde{\mathbf{P}}^a\|_\infty, \qquad (47)
\end{aligned}$$

where we used the convention $v_{\max} = \max_i v_i$ (analogously for $v_{\min}$). As done in (10), we can resort to Whitt's (1978) Theorem 3.1 and Corollary (b) of his Theorem 6.1 to obtain a bound for $\|\mathbf{v}^* - \tilde{\mathbf{v}}^*\|_\infty$. Substituting such a bound and expression (47) in (46), we obtain

$$\begin{aligned}
\|\mathbf{q}_*^a - \tilde{\mathbf{q}}_*^a\|_\infty &\leq \|\mathbf{r}^a - \tilde{\mathbf{r}}^a\|_\infty + \frac{\gamma R_{\text{dif}}}{2(1-\gamma)}\|\mathbf{P}^a - \tilde{\mathbf{P}}^a\|_\infty + \frac{\gamma}{1-\gamma}\left(\max_a \|\mathbf{r}^a - \tilde{\mathbf{r}}^a\|_\infty + \frac{R_{\text{dif}}}{2(1-\gamma)}\max_a \|\mathbf{P}^a - \tilde{\mathbf{P}}^a\|_\infty\right) \\
&\leq \max_a \|\mathbf{r}^a - \tilde{\mathbf{r}}^a\|_\infty + \frac{R_{\text{dif}}}{2(1-\gamma)}\max_a \|\mathbf{P}^a - \tilde{\mathbf{P}}^a\|_\infty + \frac{\gamma}{1-\gamma}\left(\max_a \|\mathbf{r}^a - \tilde{\mathbf{r}}^a\|_\infty + \frac{\gamma R_{\text{dif}}}{2(1-\gamma)}\max_a \|\mathbf{P}^a - \tilde{\mathbf{P}}^a\|_\infty\right).
\end{aligned}$$

□

**Note:** From the proof of Lemma 3 we see that

$$|Q^*(s_i,a) - \tilde{Q}^*(s_i,a)| \leq |r_i^a - \tilde{r}_i^a| + \frac{\gamma R_{\text{dif}}}{2(1-\gamma)}\|\mathbf{P}^a - \tilde{\mathbf{P}}^a\|_\infty + \frac{\gamma}{1-\gamma}\left(\max_a \|\mathbf{r}^a - \tilde{\mathbf{r}}^a\|_\infty + \frac{R_{\text{dif}}}{2(1-\gamma)}\max_a \|\mathbf{P}^a - \tilde{\mathbf{P}}^a\|_\infty\right),$$

which is tighter than (45). Here we favor the more intelligible version of the bound, but of course Proposition 4 could also have been derived based on the expression above.

## A.3 Alternative error bound

In Section  we derived an upper bound for the approximation error introduced by the application of the stochastic-factorization trick. In this section we introduce another bound that has different properties. First, the bound is less applicable, because it depends on quantities that are usually unavailable in a practical situation (the fixed points of two contraction mappings). On the bright side, unlike the bound presented in Proposition 1, the new bound is valid for any norm. Also, it draws an interesting connection with an important class of approximators known as *averagers* (Gordon, 1995).

We start by deriving a theoretical result that only applies to stochastic factorizations of order $n$. We then generalize this result to the case in which the factorizations are of order $m < n$.

**Lemma 4.** *Let $M \equiv (S, A, \mathbf{P}^a, \mathbf{r}^a, \gamma)$ be a finite MDP with $|S| = n$ and $0 \le \gamma < 1$. Let $\mathbf{E}\mathbf{L}^a = \mathbf{P}^a$ be $|A|$ stochastic factorizations of order $n$ and let $\bar{\bar{\mathbf{r}}}^a$ be vectors in $\mathbb{R}^n$ such that $\mathbf{E}\bar{\bar{\mathbf{r}}}^a = \mathbf{r}^a$ for all $a \in A$. Define the MDPs $\check{M} \equiv (S, A, \mathbf{L}^a, \bar{\bar{\mathbf{r}}}^a, \gamma)$ and $\bar{\bar{M}} \equiv (S, A, \bar{\bar{\mathbf{P}}}^a, \bar{\bar{\mathbf{r}}}^a, \gamma)$, with $\bar{\bar{\mathbf{P}}}^a = \mathbf{L}^a\mathbf{E}$. Then,*

$$\|\mathbf{v}^* - T\mathbf{E}\bar{\bar{\mathbf{v}}}^*\| \le \xi'_v \equiv \frac{2\gamma}{1-\gamma}\|\mathbf{v}^* - \mathbf{u}\| + \frac{\gamma(1+\gamma)}{1-\gamma}\|\mathbf{v}^* - \check{\mathbf{v}}^*\|, \tag{48}$$

*where $\|\cdot\|$ is a norm in $R^n$ and $\mathbf{u}$ is a vector in $\mathbb{R}^n$ such that $\mathbf{E}\mathbf{u} = \mathbf{u}$.*

*Proof.* The Bellman operators of $M$, $\check{M}$, and $\bar{\bar{M}}$ are given by $T = \Gamma\Delta$, $\check{T} = \Gamma\check{\Delta}$, and $\bar{\bar{T}} = \Gamma\bar{\bar{\Delta}}$. Note that $\mathbf{q}^a = \mathbf{r}^a + \gamma\mathbf{P}^a\mathbf{v} = \mathbf{E}\bar{\bar{\mathbf{r}}}^a + \gamma\mathbf{E}\mathbf{L}^a\mathbf{v} = \mathbf{E}(\bar{\bar{\mathbf{r}}}^a + \gamma\mathbf{L}^a\mathbf{v})$, where $\mathbf{q}^a$ is the $a^{\text{th}}$ column of $\mathbf{Q}$. Thus, $\Delta = \mathbf{E}\check{\Delta}$. Since $\mathbf{E}$ is stochastic, we can think of it as one of Gordon's (1995) averagers given by $A(\mathbf{v}) = \mathbf{E}\mathbf{v}$, and then resort to Theorem 4.1 by the same author to conclude that $\bar{\bar{T}} = \mathbf{E}\check{T}$. Therefore,[6]

$$T\mathbf{v} = \Gamma\mathbf{E}\check{\Delta}\mathbf{v} \quad \text{and} \quad \bar{\bar{T}}\mathbf{v} = \mathbf{E}\Gamma\check{\Delta}\mathbf{v}. \tag{49}$$

Using (49), it is easy to obtain the desired upper bound by resorting to the triangle

[6] Interestingly, the effect of swapping matrices $\mathbf{E}$ and $\mathbf{L}^a$ is to also swap the operators $\Gamma$ and $\mathbf{E}$.

inequality, the definition of a contraction map, and Denardo's (1967) Theorem 1:

$$\begin{aligned}
\|\mathbf{v}^* - T\mathbf{E}\bar{\bar{\mathbf{v}}}^*\| \le {} & \gamma\|\mathbf{v}^* - \mathbf{E}\bar{\bar{\mathbf{v}}}^*\| \le \gamma(\|\mathbf{v}^* - \mathbf{u}\| + \|\mathbf{u} - \mathbf{E}\bar{\bar{\mathbf{v}}}^*\|) \le \gamma(\|\mathbf{v}^* - \mathbf{u}\| + \|\mathbf{u} - \bar{\bar{\mathbf{v}}}^*\|) \\
& \le \gamma\left(\|\mathbf{v}^* - \mathbf{u}\| + \frac{1}{1-\gamma}\|\mathbf{u} - \mathbf{E}\Gamma\check{\Delta}\mathbf{u}\|\right) \le \gamma\left(\|\mathbf{v}^* - \mathbf{u}\| + \frac{1}{1-\gamma}\|\mathbf{u} - \Gamma\check{\Delta}\mathbf{u}\|\right) \\
& \le \gamma\left[\|\mathbf{v}^* - \mathbf{u}\| + \frac{1}{1-\gamma}\left(\|\mathbf{u} - \check{\mathbf{v}}^*\| + \|\check{\mathbf{v}}^* - \Gamma\check{\Delta}\mathbf{u}\|\right)\right] \\
& \le \gamma\left[\|\mathbf{v}^* - \mathbf{u}\| + \frac{1}{1-\gamma}\left(\|\mathbf{u} - \check{\mathbf{v}}^*\| + \gamma\|\check{\mathbf{v}}^* - \mathbf{u}\|\right)\right] = \gamma\left[\|\mathbf{v}^* - \mathbf{u}\| + \frac{1+\gamma}{1-\gamma}\|\mathbf{u} - \check{\mathbf{v}}^*\|\right] \\
& \le \gamma\left[\|\mathbf{v}^* - \mathbf{u}\| + \frac{1+\gamma}{1-\gamma}\left(\|\mathbf{u} - \mathbf{v}^*\| + \|\mathbf{v}^* - \check{\mathbf{v}}^*\|\right)\right] \\
& = \gamma\|\mathbf{v}^* - \mathbf{u}\| + \frac{\gamma(1+\gamma)}{1-\gamma}\|\mathbf{v}^* - \mathbf{u}\| + \frac{\gamma(1+\gamma)}{1-\gamma}\|\mathbf{v}^* - \check{\mathbf{v}}^*\| \\
& = \frac{\gamma - \gamma^2 + \gamma + \gamma^2}{1-\gamma}\|\mathbf{v}^* - \mathbf{u}\| + \frac{\gamma(1+\gamma)}{1-\gamma}\|\mathbf{v}^* - \check{\mathbf{v}}^*\|.
\end{aligned}$$

□

The derived upper bound depends on two fixed points: $\mathbf{u}$, a fixed point of $\mathbf{E}$, and $\check{\mathbf{v}}^*$, the unique fixed point of $\check{T} = \Gamma\check{\Delta}$. Since the latter is defined by $\bar{\bar{\mathbf{r}}}^a$ and $\mathbf{L}^a$, the bound is essentially a function of the factorization terms, as expected. Notice that the bound is valid for any norm and any fixed point of $\mathbf{E}$ (we may think of $\mathbf{u}$ as the closest vector to $\mathbf{v}^*$ in $\mathbb{R}^n$ which satisfies this property). Notice also that the first term on the right-hand side of (48) is exactly the error bound derived in Gordon's (1995) Theorem 6.2. When $\mathbf{L}^a = \mathbf{P}^a$ and $\mathbf{r}^a = \bar{\bar{\mathbf{r}}}^a$ for all $a \in A$, the operators $T$ and $\check{T}$ coincide, and hence the second term of (48) vanishes. This makes sense, since in this case $\bar{\bar{T}} = \mathbf{E}T$, that is, the stochastic-factorization trick reduces to the averager $A(\mathbf{v}) = \mathbf{E}\mathbf{v}$.

As mentioned above, one of the assumptions of Lemma 4 is that the factorizations $\mathbf{E}\mathbf{L}^a = \mathbf{P}^a$ are of order $n$. This is unfortunate, since the whole motivation behind the stochastic-factorization trick is to create an MDP with $m < n$ states. One way to obtain such a reduction is to suppose that matrix $\mathbf{E}$ has $n - m$ columns with zeros only. Define $\mathcal{E} \subset \{1, 2, ..., n\}$ as the set of columns of $\mathbf{E}$ with at least one nonzero element and let $\mathbf{H}$ be a matrix in $\mathbb{R}^{m \times n}$ such that $h_{ij} = 1$ if $j$ is the $i^{\text{th}}$ smallest element in $\mathcal{E}$ and $h_{ij} = 0$ otherwise. The following proposition shows that, based on the action-value function of $\bar{M}$, it is possible to find an approximate solution for the original MDP whose distance to the optimal one is also bounded by (48).

**Proposition 5.** *Suppose the assumptions of Lemma 4 hold. Let* $\mathbf{D} = \mathbf{E}\mathbf{H}^\intercal$*,* $\mathbf{K}^a = \mathbf{H}\mathbf{L}^a$*, and* $\bar{\mathbf{r}}^a = \mathbf{H}\bar{\bar{\mathbf{r}}}^a$*, with* $\mathbf{H}$ *defined as described above. Define the MDP* $\bar{M} \equiv (\bar{S}, A, \bar{\mathbf{P}}^a, \bar{\mathbf{r}}^a, \gamma)$*, with* $|\bar{S}| = m$ *and* $\bar{\mathbf{P}}^a = \mathbf{K}^a\mathbf{D}$*. Then,* $\|\mathbf{v}^* - \Gamma\mathbf{D}\bar{\mathbf{Q}}^*\| \le \xi'_v$*, with* $\xi'_v$ *defined in (48).*

*Proof.* Let $\bar{\mathbf{q}}^a_* \in \mathbb{R}^m$ be the $a^{\text{th}}$ column of $\bar{\mathbf{Q}}^*$. Then,

$$\begin{aligned}\mathbf{D}\bar{\mathbf{q}}^a_* &= \mathbf{D}\left(\bar{\mathbf{r}}^a + \gamma\bar{\mathbf{P}}^a\bar{\mathbf{v}}^*\right) = \mathbf{D}\bar{\mathbf{r}}^a + \gamma\mathbf{D}\mathbf{K}^a\mathbf{D}\bar{\mathbf{v}}^* = \mathbf{E}\mathbf{H}^\intercal\mathbf{H}\bar{\bar{\mathbf{r}}}^a + \gamma\mathbf{E}\mathbf{H}^\intercal\mathbf{H}\mathbf{L}^a\mathbf{E}\mathbf{H}^\intercal\bar{\mathbf{v}}^* \\ &= \mathbf{E}\bar{\bar{\mathbf{r}}}^a + \gamma\mathbf{E}\bar{\bar{\mathbf{P}}}^a\mathbf{H}^\intercal\bar{\mathbf{v}}^* = \mathbf{E}\bar{\bar{\mathbf{r}}}^a + \gamma\mathbf{E}\bar{\bar{\mathbf{P}}}^a\bar{\bar{\mathbf{v}}}^* = \mathbf{E}\left(\bar{\bar{\mathbf{r}}}^a + \gamma\bar{\bar{\mathbf{P}}}^a\bar{\bar{\mathbf{v}}}^*\right) = \mathbf{E}\bar{\bar{\mathbf{q}}}^a_*,\end{aligned}$$

where the equality $\mathbf{E}\mathbf{H}^\intercal\mathbf{H} = \mathbf{E}$ follows from the definition of $\mathbf{H}$ and $\bar{\bar{\mathbf{P}}}^a\mathbf{H}^\intercal\bar{\mathbf{v}}^* = \bar{\bar{\mathbf{P}}}^a\bar{\bar{\mathbf{v}}}^*$ is a consequence of the fact that $s_i$ is transient if $i \notin \mathcal{E}$. Therefore, $\mathbf{D}\bar{\mathbf{Q}}^* = \mathbf{E}\bar{\bar{\mathbf{Q}}}^*$. Also, since $\mathbf{E}\bar{\bar{\mathbf{q}}}^a_* = \mathbf{E}\bar{\bar{\mathbf{r}}}^a + \gamma\mathbf{E}\mathbf{L}^a\mathbf{E}\bar{\bar{\mathbf{v}}}^* = \mathbf{r}^a + \gamma\mathbf{P}^a\mathbf{E}\bar{\bar{\mathbf{v}}}^*$, we know that $\mathbf{E}\bar{\bar{\mathbf{Q}}}^* = \Delta\mathbf{E}\bar{\bar{\mathbf{v}}}^*$. Putting these results together, we obtain $\|\mathbf{v}^* - \Gamma\mathbf{D}\bar{\mathbf{Q}}^*\| = \|\mathbf{v}^* - \Gamma\Delta\mathbf{E}\bar{\bar{\mathbf{v}}}^*\| = \|\mathbf{v}^* - T\mathbf{E}\bar{\bar{\mathbf{v}}}^*\|$, and Lemma 4 applies. □

The derived bound can be generalized to the case of approximate stochastic factorizations through the triangle inequality, as done in (9). However, if one resorts to Whitt's (1978) results to bound the distance between $\mathbf{v}^*$ and $\check{\mathbf{v}}^*$—where $\check{\mathbf{v}}^*$ is the optimal value function of $\check{M} \equiv (S, A, \mathbf{D}\mathbf{K}^a, \mathbf{D}\bar{\mathbf{r}}^a, \gamma)$—the compounded bound will no longer be valid for all norms, since (10) only holds for the infinity norm.

# B Details of the experiments

This appendix describes the details of the experiments omitted in the paper.

## B.1 Tasks

**Puddle World**: The puddle-world task was implemented as described by Sutton (1996), but here the task was modeled as a discounted problem with $\gamma = 0.99$. All the transitions were associated with a zero reward, except those leading to the goal, which resulted in a reward of $+5$, and those ending inside one of the puddles, which lead to a penalty of $-10$ times the distance to the puddle's nearest edge. If the agent did not reach the goal after 300 steps the episode was interrupted and considered as a failure. The algorithms were evaluated on two sets of states distributed over disjoint regions of the state space surrounding the puddles. The first set was a $3 \times 3$ grid defined over $[0.1, 0.3] \times [0.3, 0.5]$ and the second one was composed of four states: $\{0.1, 0.3\} \times \{0.9, 1.0\}$.

**Pole Balancing**: We implemented the simulator of the three versions of the pole-balancing task using the equations of motion and parameters given in the appendix of Gomez's (2003) PhD thesis. For the integration we used the $4^{th}$ order Runge-Kutta method with a time step of 0.01 seconds and actions chosen every 2 time steps. We considered the version of the task in which the angle between the pole and the vertical plane must be kept within $[-36^o, 36^o]$. The problem was modeled as a discounted task with $\gamma = 0.99$. In this formulation, an episode is interrupted and the agent gets a reward of $-1$ if the pole falls past a 36-degree angle or the cart reaches the boundaries of the track, located at 2.4m from its center. At all other steps the agent receives a reward of 0. In all versions of the problem an episode was considered a success if the pole(s) could be balanced for 3000 steps (one minute of simulated time). The test set was comprised of 81 states equally spaced in the region defined by $\pm[1.2\text{m}, 1.2/5\text{m}, 18^o, 75^o/s]$, for the single pole case, and by $\pm[1.2\text{m}, 1.2/5\text{m}, 18^o, 75^o/s, 18^o, 150^o/s]$ for the two pole version of the problem. These values correspond to a hypercube centered at the origin and covering 50% of the state-space axes in each dimension (since the velocity of the cart and the angular velocity of the poles are theoretically not bounded, we defined the limits of these variables based on samples generated in simple preliminary experiments). For the triple pole-balancing task we performed our simulations using the parameters usually adopted with the two pole version of the problem, but we added a third pole with the same length and mass as the longer pole (Gomez, 2003). In this case the decision policies were evaluated on a test set containing 256 states equally distributed in the region $\pm[1.2\text{m}, 1.2/5\text{m}, 18^o, 75^o/s, 18^o, 150^o/s, 18^o, 75^o/s]$.

**HIV drug schedule**: The HIV drug schedule task was implemented using the system of ordinary differential equations (ODEs) given by Adams et al. (2004). Integration was carried out by the Euler method using a step size of 0.001 with actions selected at each 5 000 steps (corresponding to 5 days of simulated time). As suggested by Ernst et al. (2006), the problem was modeled as a discounted task with $\gamma = 0.98$. All other parameters of the task, as well as the protocol used for the numerical simulations, also followed the suggestions of the same authors. In particular, we assumed the existence of 30 patients who were monitored for 1 000 days. During the monitoring period, the content of the drug cocktail administered to each patient could be changed at fixed intervals of 5 days. Thus, in a sample transition $(s_i^a, r_i^a, \hat{s}_i^a)$: $s_i^a$ is the initial patient condition, $a$ is one of the four types of cocktails to be administered for the next 5 days, $\hat{s}_i^a$ is the patient condition 5 days later, and $r_i^a$ is a reward computed based on the amount of drug in the selected cocktail $a$ and on the difference between the patient's condition from $s_i^a$ to $\hat{s}_i^a$ (Ernst et al., 2006). The results reported in Section  correspond to the performance of the greedy policy induced by the value function computed by the algorithms using all available sample transitions. The decision policies (in this case STI treatments) were evaluated for 5 000 days starting from an "unhealthy" state corresponding to a basin of attraction of the ODEs describing the problem's dynamics (see the papers by Adams et al. and Ernst et al.).

**Epilepsy suppression**: We used a generative model developed by Bush et al. (2009) to perform our experiments with the epilepsy suppression task. The model was generated based on labeled field potential recordings of five rat brain slices electrically stimulated at frequencies of 0.0 Hz, 0.5 Hz, 1.0 Hz, and 2.0 Hz. The data was used to construct a manifold embedding which in turn gave rise to the problem's state space. The objective is to minimize the occurrence of seizures using as little stimulation as possible, therefore there is a negative reward associated with both events (see Section ). Bush et al.'s generative model is public available as an environment for the RL-Glue package (Tanner and White, 2009). In our experiments the problem was modeled as a discounted task with $\gamma = 0.99$. The decision policies were evaluated on episodes of $10^5$ transitions starting from a fixed set of 10 test states drawn uniformly at random from the problem's state space.

**Helicopter hovering**: In the experiments with the helicopter hovering task we used the simulator developed by Abbeel et al. (2005), which is available as an environment for the RL-Glue package (Tanner and White, 2009). The simulator was built based on data collected from two separate flights of a XCell Tempest helicopter. The data was used to adjust the parameters of an "acceleration prediction model", which is more accurate than the linear model normally adopted by industry. The objective in the problem is to keep the helicopter hovering as close as possible to a specific location. Therefore, at each time step the agent gets a negative reward proportional to the distance from the target position. Since the problem's original action space is $A \equiv [-1,1]^4$, we discretized each dimension using 4 break points distributed unevenly over $[-1,1]$. We tried several possible discretizations and picked the one which resulted in the best performance of the SARSA agent (see Section ). After this process, the problem's action space was redefined as $A \equiv \{-0.25, -0.05, +0.05, +0.25\}^4$. The problem was modeled as a discounted task with $\gamma = 0.99$. The decision policies were evaluated in episodes starting from the target position and ending when the helicopter crashed.

### B.2 Algorithms

In all experiments, we used

$$\phi(z) \equiv \bar{\phi}(z) \equiv \exp(-z) \tag{50}$$

to define the kernels used by KBRL, LSPI, and KBSF. In the experiments involving a large number of sample transitions we used sparse kernels, that is, we only computed the $\mu$ largest values of $\mathrm{k}_\tau(\bar{s}_i, \cdot)$ and the $\bar{\mu}$ largest values of $\bar{\mathrm{k}}_{\bar{\tau}}(\hat{s}_i^a, \cdot)$. In order to implement this feature, we used a KD-tree to find the $\mu$ ($\bar{\mu}$) nearest neighbors of $\bar{s}_i$ ($\hat{s}_i^a$) and only computed $\mathrm{k}_\tau$ ($\bar{\mathrm{k}}_{\bar{\tau}}$) in these states (Bentley, 1975). The value of $\mathrm{k}_\tau$ and $\bar{\mathrm{k}}_{\bar{\tau}}$ outside this neighborhood was truncated to zero (we used specialized data structures to avoid storing those).

We now list a few details regarding the algorithms's implementations which were not described in the paper:

- **KBRL** and **KBSF**: We used modified policy iteration to compute $\hat{\mathbf{Q}}^*$ (Puterman and Shin, 1978). The value function of a fixed policy $\pi$ was approximated through value iteration using the stop criterion described by Puterman (1994, Proposition 6.6.5) with $\varepsilon = 10^{-6}$. Table 1 shows the parameters's values used by KBSF across the experiments.

- **LSPI**: As explained above, LSPI used the kernel derived from (50) as its basis function. Following Lagoudakis and Parr (2003), we adopted one block of basis functions for each action $a \in A$. Singular value decomposition was used to avoid eventual numerical instabilities in the system of linear equations constructed at each iteration of LSPI (Golub and Loan, 1993).

- **Fitted $Q$-iteration and extra trees**: FQIT has four main parameters: the number of iterations, the number of trees composing the ensemble, the number of candidate cut-points evaluated during the generation of the trees, and the minimum number of elements required to split a node, denoted here $\eta_{\min}$. In general, increasing the first three improves performance, while $\eta_{\min}$ has an inverse relation with the quality of the final value function approximation. Our experiments indicate that the following configuration of FQIT usually results in good performance on the tasks considered in this paper: 50 iterations (with the structure of the trees fixed after the 10th

| **Problem** | **Section** | $\bar{s}_i$ | $m$ | $\tau$ | $\bar{\tau}$ | $\mu$ | $\bar{\mu}$ |
|---|---|---|---|---|---|---|---|
| Puddle | | $k$-means | $\{10, 30, ..., 150\}$ | $\{0.01, 0.1, 0.1\}$ | $\{0.01, 0.1, 0.1\}$ | $\infty$ | $\infty$ |
| Puddle | | evenly | 100 | $\{0.01, 0.1, 0.1\}$ | $\{0.01, 0.1, 0.1\}$ | $\infty$ | $\infty$ |
| Single Pole | | $k$-means | $\{10, 30, ..., 150\}$ | 1 | $\{0.01, 0.1, 0.1\}$ | $\infty$ | $\infty$ |
| Two Poles | | $k$-means | $\{20, 40, ..., 200\}$ | 1 | $\{0.01, 0.1, 0.1\}$ | $\infty$ | $\infty$ |
| Triple Pole | | on-line | on-line | $100^*$ | $1^*$ | $50^*$ | $10^*$ |
| HIV | | random | $\{2000, 4000, ..., 10000\}$ | 1 | 1 | $2^*$ | $3^*$ |
| Epilepsy | | $k$-means | $50000^*$ | 1 | $\{0.01, 0.1, 0.1\}$ | $6^*$ | $6^*$ |
| Helicopter | | $k$-means | $500^*$ | 1 | 1 | $4^*$ | $4^*$ |

Table 1: Parameters used by KBSF on the computational experiments. The values marked with an asterisk (*) were determined by trial and error on preliminary tests. The remaining parameters were kept fixed from the start or were defined based on a very coarse search.

one), an ensemble of 30 trees, and $d_{\mathbb{S}}$ candidate cut points. The parameter $\eta_{\min}$ has a particularly strong effect on FQIT's performance and computational cost, and its correct value seems to be more problem-dependent. Therefore, in all of our experiments we fixed the parameters of FQIT as described above and only varied $\eta_{\min}$.

- **SARSA**: We adopted the implementation of SARSA($\lambda$) available in the RL-Glue package (Tanner and White, 2009). The algorithm uses gradient descent temporal-difference learning to configure a tile coding function approximator.

## Acknowledgments

Most of the work described in this technical report was done while André Barreto was a postdoctoral fellow in the School of Computer Science at McGill University. The authors would like to thank Yuri Grinberg and Amir-massoud Farahmand for valid discussions regarding KBSF and related subjects. We also thank Keith Bush for making the epilepsy simulator available, and Alicia Bendz and Ryan Primeau for helping in some of the computational experiments. Funding for this research was provided by the National Institutes of Health (grant R21 DA019800) and the NSERC Discovery Grant program.

## References

P. Abbeel, V. Ganapathi, and A. Ng. Learning vehicular dynamics, with application to modeling helicopters. In *Adv. in Neural Information Processing Systems (NIPS)*, 2005.

P. Abbeel, A. Coates, M. Quigley, and A. Ng. An application of reinforcement learning to aerobatic helicopter flight. In *Adv. in Neural Information Processing Systems (NIPS)*, 2007.

B. Adams, H. Banks, H. Kwon, and H. Tran. Dynamic multidrug therapies for HIV: optimal and STI control approaches. *Mathematical Biosciences and Engineering*, 1(2): 223–41, 2004.

C. Anderson. *Learning and Problem Solving with Multilayer Connectionist Systems*. PhD thesis, Computer and Information Science, University of Massachusetts, 1986.

A. Antos, R. Munos, and Cs. Szepesvári. Fitted Q-iteration in continuous action-space MDPs. In *Advances in Neural Information Processing Systems (NIPS)*, 2007.

C. Atkeson and J. Santamaria. A comparison of direct and model-based reinforcement learning. In *Proc. of the IEEE International Conference on Robotics and Automation*, 1997.

S. Bajaria, G. Webb, and D. Kirschner. Predicting differential responses to structured treatment interruptions during HAART. *Bulletin of Mathematical Biology*, 66(5):1093 – 1118, 2004.

A. Barreto and M. Fragoso. Computing the Stationary Distribution of a Finite Markov Chain Through Stochastic Factorization. *SIAM Journal on Matrix Analysis and Applications*, 32:1513–1523, 2011.

A. Barreto, J. Pineau, and D. Precup. Policy iteration based on stochastic factorization. Submitted, 2013.

A. Barreto, D. Precup, and J. Pineau. Reinforcement learning using kernel-based stochastic factorization. In *Advances in Neural Information Processing Systems (NIPS)*, 2011.

A. Barreto, D. Precup, and J. Pineau. On-line reinforcement learning using incremental kernel-based stochastic factorization. In *Advances in Neural Information Processing Systems (NIPS)*, 2012.

A. Barto, R. Sutton, and C. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13:834–846, 1983.

R. E. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

J. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

D. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2006.

N. Bhat, C. Moallemi, and V. Farias. Non-parametric approximate dynamic programming via the kernel method. In *Adv. in Neural Information Processing Systems (NIPS)*, 2012.

R. I. Brafman and M. Tennenholtz. R-MAX: a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231, 2003.

K. Bush, Pineau J., and M. Avoli Manifold embeddings for model-based reinforcement learning of neurostimulation policies. In *Proceedings of the ICML/UAI/COLT Workshop on Abstraction in Reinforcement Learning*, 2009.

K. Bush and J. Pineau. Manifold embeddings for model-based reinforcement learning under partial observability. In *Adv. in Neural Information Processing Systems (NIPS)*, 2009.

J. Cohen and U. Rothblum. Nonnegative ranks, decompositions and factorizations of nonnegative matrices. *Linear Algebra and its Applications*, 190:149–168, 1991.

A. Cutler and L. Breiman. Archetypal analysis. *Technometrics*, 36(4):338–347, 1994.

E. Denardo. Contraction mappings in the theory underlying dynamic programming. *SIAM Review*, 9(2):165–177, 1967.

T. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

D. Durand and M. Bikson. Suppression and control of epileptiform activity by electrical stimulation: a review. *Proceedings of the IEEE*, 89(7):1065 –1082, 2001.

Y. Engel, S. Mannor, and R. Meir. Reinforcement learning with Gaussian processes. In *Proceedings of the International Conference on Machine learning (ICML)*, 2005.

D. Ernst, G. Stan, J. Gonçalves, and L. Wehenkel. Clinical data based optimal STI strategies for HIV: a reinforcement learning approach. In *Proceedings of the IEEE Conference on Decision and Control (CDC)*, 2006.

D. Ernst, P. Geurts, and L. Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556, 2005.

A. Farahmand. *Regularization in reinforcement learning.* PhD thesis, Univ. of Alberta, 2011.

P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine Learning*, 36(1):3–42, 2006.

G. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins University Press, second edition, 1993.

F. Gomez, J. Schmidhuber, and R. Miikkulainen. Efficient non-linear control through neuroevolution. In *Proceedings of the European Conference on Machine Learning*, 2006.

F. Gomez. *Robust non-linear control through neuroevolution.* PhD thesis, The University of Texas at Austin, 2003. Technical Report AI-TR-03-303.

G. Gordon. Stable function approximation in dynamic programming. Technical Report CMU-CS-95-103, Computer Science Department, Carnegie Mellon University, 1995.

S. Grunewalder, G. Lever, L. Baldassarre, M. Pontil, and A. Gretton. Modelling transition dynamics in MDPs with RKHS embeddings. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2012.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2002.

K. Jerger and S. Schiff. Periodic pacing an in vitro epileptic focus. *Journal of Neurophysiology*, (2):876–879, 1995.

N. Jong and P. Stone. Kernel-based models for reinforcement learning in continuous state spaces. In *Proceedings of the International Conference on Machine Learning—Workshop on Kernel Machines and Reinforcement Learning*, 2006.

N. Jong and P. Stone. Compositional models for reinforcement learning. In *Proc. of the European Conference on Machine Learning and Knowledge Discovery in Databases*, 2009.

L. Kaufman and P. Rousseeuw. *Finding Groups in Data: an Introduction to Cluster Analysis*. John Wiley and Sons, 1990.

O. Kroemer and J. Peters. A non-parametric approach to dynamic programming. In *Advances in Neural Information Processing Systems (NIPS)*, 2011.

B. Kveton and G. Theocharous. Kernel-based reinforcement learning on representative states. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2012.

M. Lagoudakis and R. Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003.

D. Michie and R. Chambers. BOXES: An experiment on adaptive control. *Machine Intelligence 2*, pages 125–133, 1968.

A. Moore and C. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13:103–130, 1993.

R. Munos and Cs. Szepesvári. Finite-time bounds for fitted value iteration. *Journal of Machine Learning Research*, 9:815–857, 2008.

A. Ng, H. Kim, M. Jordan, and S. Sastry. Autonomous helicopter flight via reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2003.

D. Ormoneit and P. Glynn. Kernel-based reinforcement learning in average-cost problems. *IEEE Transactions on Automatic Control*, 47(10):1624–1636, October 2002.

D. Ormoneit and S. Sen. Kernel-based reinforcement learning. *Machine Learning*, 49 (2–3):161–178, 2002.

M. Puterman. *Markov Decision Processes—Discrete Stochastic Dynamic Programming.* John Wiley & Sons, Inc., 1994.

M. Puterman and M. Shin. Modified policy iteration algorithms for discounted Markov decision problems. *Management Science*, pages 1127–1137, 1978.

C. Rasmussen and M. Kuss. Gaussian processes in reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2004.

B. Ravindran. *An Algebraic Approach to Abstraction in Reinforcement Learning.* PhD thesis, University of Massachusetts, Amherst, MA, 2004.

G. Rummery and M. Niranjan. On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166, Cambridge University, 1994.

B. Schölkopf and A. Smola. *Learning with Kernels.* MIT Press, 2002.

J. Sorg and S. Singh. Transfer via soft homomorphisms. In *Autonomous Agents & Multiagent Systems/Agent Theories, Architectures, and Languages*, 2009.

A. Strehl and M. Littman. An analysis of model-based interval estimation for Markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331, 2008.

R. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems*, 1996.

R. Sutton and A. Barto. *Reinforcement Learning: An Introduction.* MIT Press, 1998.

Cs. Szepesvári. *Algorithms for Reinforcement Learning.* Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2010.

B. Tanner and A. White. RL-Glue: Language-independent software for reinforcement-learning experiments. *Journal of Machine Learning Research*, 10:2133–2136, 2009.

G. Taylor and R. Parr. Kernelized value function approximation for reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2009.

S. Vavasis. On the complexity of nonnegative matrix factorization. *SIAM Journal on Optimization*, 20:1364–1377, 2009.

W. Whitt. Approximations of dynamic programs, I. *Mathematics of Operations Research*, 3(3):231–243, 1978.

A. Wieland. Evolving neural network controllers for unstable systems. In *Proceedings of the International Joint Conference on Neural Networks*, 1991.

X. Xu, T. Xie, D. Hu, and X. Lu. Kernel Least-Squares Temporal Difference Learning. *Information Technology*, pages 54–63, 2005.

Y. Ye. The simplex and policy-iteration methods are strongly polynomial for the Markov decision problem with a fixed discount rate. *Mathematics of Operations Research*, 36 (4):593–603, 2011.