Journal of Machine Learning Research 15 (2014) 2313-2335 Submitted 12/13; Revised 3/14; Published 6/14

Cover Tree Bayesian Reinforcement Learning

Nikolaos Tziortziotis NTZIORZI@GMAIL.COM
Department of Computer Science and Engineering
University of Ioannina
GR-45110, Greece

Christos Dimitrakakis CHRISTOS.DIMITRAKAKIS@GMAIL.COM
Department of Computer Science and Engineering
Chalmers University of Technology
SE-41296, Sweden

Konstantinos Blekas KBLEKAS@CS.UOI.GR
Department of Computer Science and Engineering
University of Ioannina
GR-45110, Greece

Editor: Laurent Orseau

Abstract

This paper proposes an online tree-based Bayesian approach for reinforcement learning. For inference, we employ a generalised context tree model. This defines a distribution on multivariate Gaussian piecewise-linear models, which can be updated in closed form. The tree structure itself is constructed using the cover tree method, which remains efficient in high dimensional spaces. We combine the model with Thompson sampling and approximate dynamic programming to obtain effective exploration policies in unknown environments. The flexibility and computational simplicity of the model render it suitable for many reinforcement learning problems in continuous state spaces. We demonstrate this in an experimental comparison with a Gaussian process model, a linear model and simple least squares policy iteration.

Keywords: Bayesian inference, non-parametric statistics, reinforcement learning

1. Introduction

In reinforcement learning, an agent must learn how to act in an unknown environment from limited feedback and delayed reinforcement. Efficient learning and planning requires models of the environment that are not only general, but can also be updated online with low computational cost. In addition, probabilistic models allow the use of a number of near-optimal algorithms for decision making under uncertainty. While it is easy to construct such models for small, discrete environments, models for the continuous case have so far been mainly limited to parametric models, which may not have the capacity to represent the environment (such as generalised linear models) and to non-parametric models, which do not scale very well (such as Gaussian processes).

In this paper, we propose a non-parametric family of tree models, with a data-dependent structure constructed through the cover tree algorithm, introduced by Beygelzimer et al. (2006). Cover trees are data structures that cover a metric space with a sequence of data-dependent partitions. They were initially proposed for the problem of k-nearest neighbour search, but they are in general a good method to generate fine partitions of a state space, due to their low complexity, and can be applied to any state space, with a suitable choice of metric. In addition, it is possible to create a statistical model using the cover tree as a basis. Due to the tree structure, online inference has low (logarithmic) complexity.

In this paper, we specifically investigate the case of a Euclidean state space. For this, we propose a model generalising the context tree weighting algorithm proposed by Willems et al. (1995), combined with Bayesian multivariate linear models. The overall prior can be interpreted as a distribution on piecewise-linear models. We then compare this model with a Gaussian process model, a single linear model, and the model-free method least-squares policy iteration in two well-known benchmark problems in combination with approximate dynamic programming and show that it consistently outperforms other approaches.

The remainder of the paper is organised as follows. Section 1.1 introduces the setting, Section 1.2 discusses related work and Section 1.3 explains our contribution. The model and algorithm are described in Section 2. Finally, comparative experiments are presented in Section 3 and we conclude with a discussion of the advantages of cover-tree Bayesian reinforcement learning and directions of future work in Section 4.

1.1 Setting

We assume that the agent acts within a fully observable discrete-time Markov decision process (MDP), with a metric state space $\mathcal{S}$, for example $\mathcal{S} \subset \mathbb{R}^m$. At time t, the agent observes the current environment state $\boldsymbol{s}_t \in \mathcal{S}$, takes an action a_t from a discrete set $\mathcal{A}$, and receives a reward $r_t \in \mathbb{R}$. The probability over next states is given in terms of a transition kernel $P_\mu(S \mid \boldsymbol{s}, a) \triangleq \mathbb{P}_\mu(\boldsymbol{s}_{t+1} \in S \mid \boldsymbol{s}_t = \boldsymbol{s}, a_t = a)$. The agent selects its actions using a *policy* $\pi \in \mathit{\Pi}$, which in general defines a conditional distribution $\mathbb{P}^\pi(a_t \mid \boldsymbol{s}_1, \ldots, \boldsymbol{s}_t, a_1, \ldots, a_{t-1}, r_1, \ldots, r_{t-1})$ over the actions, given the history of states and actions. This reflects the learning process that the agent undergoes, when the MDP μ is unknown.

The agent's *utility* is $U \triangleq \sum_{t=0}^{\infty} \gamma^t r_t$, the discounted sum of future rewards, with $\gamma \in (0, 1)$ a discount factor such that rewards further into the future are less important than immediate rewards. The goal of the agent is to maximise its expected utility:

$$\max_{\pi \in \mathit{\Pi}} \mathbb{E}^\pi_\mu U = \max_{\pi \in \mathit{\Pi}} \mathbb{E}^\pi_\mu \sum_{t=0}^{\infty} \gamma^t r_t,$$

where the value of the expectation depends on the agent's policy π and the environment μ. If the environment is known, well-known dynamic programming algorithms can be used to find the optimal policy in the discrete-state case (Puterman, 2005), while many approximate algorithms exist for continuous environments (Bertsekas and Tsitsiklis, 1996). In this case, optimal policies are memoryless and we let $\mathit{\Pi}_1$ denote the set of memoryless policies. Then MDP and policy define a Markov chain with kernel $P^\pi_\mu(S \mid \boldsymbol{s}, a) = \sum_{a \in A} P_\mu(S \mid \boldsymbol{s}, a) \pi(a \mid \boldsymbol{s})$.

However, since the environment μ is unknown, the above maximisation is ill-posed. In the Bayesian framework for reinforcement learning, this problem is alleviated by performing the maximisation conditioned on the agent's belief about the true environment μ. This converts the problem of reinforcement learning into a concrete, optimisation problem. However, this is generally extremely complex, as we must optimise over all history-dependent policies.

More specifically, the main assumption in Bayesian reinforcement learning is that the environment μ lies in a given set of environments $\mathcal{M}$. In addition, the agent must select a subjective prior distribution $p(\mu)$ which encodes its belief about which environments are most likely. The Bayes-optimal expected utility for p is:

$$U_p^* \triangleq \max_{\pi \in \Pi^D} \mathbb{E}_p^\pi U = \max_{\pi \in \Pi^D} \int_{\mathcal{M}} \left(\mathbb{E}_\mu^\pi U\right) \, \mathrm{d}p(\mu). \tag{1}$$

Unlike the known μ case, the optimal policy may not be memoryless, as our belief changes over time. This makes the optimisation over the policies significantly harder (Duff, 2002), as we have to consider the set of all history-dependent deterministic policies, which we denote by $\Pi^D \subset \Pi$. In this paper, we employ the simple, but effective, heuristic of Thompson sampling (Thompson, 1933; Wyatt, 1998; Dearden et al., 1998; Strens, 2000) for finding policies. This strategy is known by various other names, such as probability matching, stochastic dominance, sampling-greedy and posterior sampling. Very recently Osband et al. (2013) showed that it suffers small Bayes-regret relative to the Bayes-optimal policy for finite, discrete MDPs.

The second problem in Bayesian reinforcement learning is the choice of the prior distribution. This can be of critical importance for large or complex problems, for two reasons. Firstly, a well-chosen prior can lead to more efficient learning, especially in the finite-sample regime. Secondly, as reinforcement learning involves potentially unbounded interactions, the computational and space complexity of calculating posterior distributions, estimating marginals and performing sampling become extremely important. The choice of priors is the main focus of this paper. In particular, we introduce a prior over piecewise-linear multivariate Gaussian models. This is based on the construction of a context tree model, using a cover tree structure, which defines a conditional distribution on local linear Bayesian multivariate models. Since inference for the model can be done in closed form, the resulting algorithm is very efficient, in comparison with other non-parametric models such as Gaussian processes. The following section discusses how previous work is related to our model.

1.2 Related Work

One component in our model is the *context tree*. Context trees were introduced by Willems et al. (1995) for sequential prediction (see Begleiter et al., 2004, for an overview). In this model, a distribution of variable order Markov models for binary sequences is constructed, where the tree distribution is defined through context-dependent weights (for probability of a node being part of the tree) and Beta distributions (for predicting the next observation). A recent extension to switching time priors (van Erven et al., 2008) has been proposed by Veness et al. (2012). More related to this paper is an algorithm proposed by Kozat et al. (2007) for prediction. This asymptotically converges to the best univariate piecewise linear model in a class of trees with fixed structure.

Many reinforcement learning approaches based on such trees have been proposed, but have mainly focused on the discrete partially observable case (Daswani et al., 2012; Veness et al., 2011; Bellemare et al., 2013; Farias et al., 2010).[1] However, tree structures can generally be used to perform Bayesian inference in a number of other domains (Paddock et al., 2003; Meila and Jordan, 2001; Wong and Ma, 2010).

The core of our model is a generalised context tree structure that defines a distribution on multivariate piecewise-linear-Gaussian models. Consequently, a necessary component in our model is a multivariate linear model at each node of the tree. Such models were previously used for Bayesian reinforcement learning in Tziortziotis et al. (2013) and were shown to perform well relatively to least-square policy iteration (LSPI, Lagoudakis and Parr, 2003). Other approaches using linear models include Strehl and Littman (2008), which proves mistake bounds on reinforcement learning algorithms using online linear regression, and Abbeel and Ng (2005) who use separate linear models for each dimension. Another related approach in terms of structure is Brunskill et al. (2009), which partitions the space into *types* and estimates a simple additive model for each type.

Linear-Gaussian models are naturally generalised by Gaussian processes (GP). Some examples of GP in reinforcement learning include those of Rasmussen and Kuss (2004), Deisenroth et al. (2009) and Deisenroth and Rasmussen (2011), which focused on a model-predictive approach, while the work of Engel et al. (2005) employed GPs for expected utility estimation. GPs are computationally demanding, in contrast to our tree-structured prior. Another problem with the cited GP-RL approaches is that they employ the marginal distribution in the dynamic programming step. This heuristic ignores the uncertainty about the model (which is implicitly taken into account in Equations 1, 6). A notable exception to this is the policy gradient approach employed by Ghavamzadeh and Engel (2006) which uses full Bayesian quadrature. Finally, output dimensions are treated independently, which may not make good use of the data. Methods for efficient dependent GPs such as the one introduced by Alvarez et al. (2011) have not yet been applied to reinforcement learning.

For decision making, this paper uses the simple idea of Thompson sampling (Thompson, 1933; Wyatt, 1998; Dearden et al., 1998; Strens, 2000), which has been shown to be near-optimal in certain settings (Kaufmann et al., 2012; Agrawal and Goyal, 2012; Osband et al., 2013). This avoids the computational complexity of building augmented MDP models (Auer et al., 2008; Asmuth et al., 2009; Castro and Precup, 2010; Araya et al., 2012), Monte-Carlo tree search (Veness et al., 2011), sparse sampling (Wang et al., 2005), stochastic branch and bound (Dimitrakakis, 2010b) or creating lower bounds on the Bayes-optimal value function (Poupart et al., 2006; Dimitrakakis, 2011). Thus the approach is reasonable as long as sampling from the model is efficient.

1.3 Our Contribution

Our approach is based upon three ideas. The first idea is to employ a cover tree (Beygelzimer et al., 2006) to create a set of partitions of the state space. This avoids having to prespecify a structure for the tree. The second technical novelty is the introduction of an efficient non-parametric Bayesian conditional density estimator on the cover tree structure. This is a generalised context tree, endowed with a multivariate linear Bayesian model at each node. We use this to estimate the dynamics of the underlying environment. The multivariate

[1] We note that another important work in tree-based reinforcement learning, though not directly related to ours, is that of Ernst et al. (2005), which uses trees for expected utility rather than model estimation.

models allow for a sample-efficient estimation by capturing dependencies. Finally, we take a sample from the posterior to obtain a piecewise linear Gaussian model of the dynamics. This can be used to generate policies. In particular, from this, we obtain trajectories of simulated experience, to perform approximate dynamic programming (ADP) in order to select a policy. Although other methods could be used to calculate optimal actions, we leave them for future work.

The main advantage of our approach is its generality and efficiency. The posterior calculation and prediction is fully conjugate and can be performed online. At the t-th time step, inference takes $O(\ln t)$ time. Sampling from the tree, which need only be done infrequently, is $O(t)$. These properties are in contrast to other non-parametric approaches for reinforcement learning such as GPs. The most computationally heavy step of our algorithm is ADP. However, once a policy is calculated, the actions to be taken can be calculated in logarithmic time at each step. The specific ADP algorithm used is not integral to our approach and for some problems it might be more efficient to use an online algorithm.

2. Cover Tree Bayesian RL

The main idea of cover tree Bayesian reinforcement learning (CTBRL) is to construct a cover tree from the observations, simultaneously inferring a conditional probability density on the same structure, and to then use sampling to estimate a policy. We use a cover tree due to its efficiency compared with e.g., a fixed sequence of partitions or other dynamic partitioning methods such as KD-trees. The probabilistic model we use can be seen as a distribution over piecewise linear-Gaussian densities, with one local linear model for each set in each partition. Due to the tree structure, the posterior can be computed efficiently online. By taking a sample from the posterior, we acquire a specific piecewise linear Gaussian model. This is then used to find an approximately optimal policy using approximate dynamic programming.

An overview of CTBRL is given in pseudocode in Alg. 1. As presented, the algorithm works in an episodic manner.[2] When a new episode k starts at time t_k, we calculate a new stationary policy by sampling a tree μ_k from the current posterior $p_{t_k}(\mu)$. This tree corresponds to a piecewise-linear model. We draw a large number of rollout trajectories from μ_k using an arbitrary exploration policy. Since we have the model, we can use an initial state distribution that covers the space well. These trajectories are used to estimate a near-optimal policy π_k using approximate dynamic programming. During the episode, we take new observations using π_k, while growing the cover tree as necessary and updating the posterior parameters of the tree and the local model in each relevant tree node.

We now explain the algorithm in detail. First, we give an overview of the cover tree structure on which the context tree model is built. Then we show how to perform inference on the context tree, while Section 2.3 describes the multivariate model used in each node of the context tree. The sampling approach and the approximate dynamic method are described in Section 2.4, while the overall complexity of the algorithm is discussed in Section 2.5.

[2] An online version of the same algorithm (still employing Thompson sampling) would move line 6 to just before line 9. A fully Bayes online version would "simply" take an approximation of the Bayes-optimal action at every step.

Algorithm 1 CTBRL (Episodic, using Thompson sampling)

```
$k = 0$, $\pi_0 = \mathcal{U}nif(\mathcal{A})$, prior $p_0$ on $\mathcal{M}$.
for $t = 1, \ldots, T$ do
    if episode-end then
        $k := k + 1$.
        Sample model $\mu_k \sim p_t(\mu)$.
        Calculate policy $\pi_k \approx \arg\max_\pi \mathbb{E}^\pi_{\mu_k} U$.
    end if
    Observe state $\boldsymbol{s}_t$.
    Take action $a_t \sim \pi_k(\cdot \mid \boldsymbol{s}_t)$.
    Observe next state $\boldsymbol{s}_{t+1}$, reward $r_{t+1}$.
    Add a leaf node to the tree $\mathcal{T}_{a_t}$, containing $\boldsymbol{s}_t$.
    Update posterior: $p_{t+1}(\mu) = p_t(\mu \mid \boldsymbol{s}_{t+1}, \boldsymbol{s}_t, a_t)$ by updating the parameters of all
    nodes containing $\boldsymbol{s}_t$.
end for
```

2.1 The Cover Tree Structure

Cover trees are a data structure that can be applied to any metric space and are, among other things, an efficient method to perform nearest-neighbour search in high-dimensional spaces (Beygelzimer et al., 2006). In this paper, we use cover trees to automatically construct a sequence of partitions of the state space. Section 2.1.1 explains the properties of the constructed cover tree. As the formal construction duplicates nodes, in practice we use a reduced tree where every observed point corresponds to one node in the tree. This is explained in Section 2.1.2. An explanation of how nodes are added to the structure is given in Section 2.1.3.

2.1.1 Cover Tree Properties

To construct a cover tree $\mathcal{T}$ on a metric space $(\mathcal{Z}, \psi)$ we require a set of points $D_t = \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_t\}$, with $\boldsymbol{z}_i \in \mathcal{Z}$, a metric ψ, and a constant $\zeta > 1$. We introduce a mapping function $[\cdot]$ so that the i-th tree node corresponds to one point $\boldsymbol{z}_{[i]}$ in this set. The nodes are arranged in *levels*, with each point being replicated at nodes in multiple levels, i.e., we may have $[i] = [j]$ for some $i \neq j$. Thus, a point corresponds to multiple nodes in the tree, but to *at most one node* at any one level. Let G_n denote the set of points corresponding to the nodes at level n of the tree and $\mathfrak{C}(i) \subset G_{n-1}$ the corresponding set of children. If $i \in G_n$ then the level of i is $\ell(i) = n$. The tree has the following properties:

1. Refinement: $G_n \subset G_{n-1}$.

2. Siblings separation: $i, j \in G_n$, $\psi(\boldsymbol{z}_{[i]}, \boldsymbol{z}_{[j]}) > \zeta^n$.

3. Parent proximity: If $i \in G_{n-1}$ then $\exists$ a unique $j \in G_n$ such that $\psi(\boldsymbol{z}_{[i]}, \boldsymbol{z}_{[j]}) \leq \zeta^n$ and $i \in \mathfrak{C}(j)$.

These properties can be interpreted as follows. Firstly lower levels always contain more points. Secondly, siblings at a particular level are always well-separated. Finally, a child must be close to its parent. These properties directly give rise to the theoretical guarantees given by the cover tree structure, as well as methods for searching and adding points to the tree, as explained below.

2.1.2 The Reduced Tree

As formally the cover tree duplicates nodes, in practice we use the *explicit representation* (described in more detail in Section 2 of Beygelzimer et al., 2006). This only stores the top-most tree node i corresponding to a point $\boldsymbol{z}_{[i]}$. We denote this *reduced tree* by $\hat{\mathcal{T}}$. The *depth* $d(i)$ of node $i \in \hat{\mathcal{T}}$ is equal to its number of ancestors, with the root node having a depth of 0. After t observations, the set of nodes containing a point $\boldsymbol{z}$, is:

$$\hat{G}_t(\boldsymbol{z}) \triangleq \left\{ i \in \hat{\mathcal{T}} \;\middle|\; \boldsymbol{z} \in B_i \right\},$$

where $B_i = \left\{ \boldsymbol{z} \in \mathcal{Z} \mid \psi(\boldsymbol{z}_{[i]}, \boldsymbol{z}) \leq \zeta^{d(i)} \right\}$ is the neighbourhood of i. Then $\hat{G}_t(\boldsymbol{z})$ forms a path in the tree, as each node only has one parent, and can be discovered in logarithmic time through the `Find-Nearest` function (Beygelzimer et al., 2006, Theorem 5). This fact allows us to efficiently search the tree, insert new nodes, and perform inference.

2.1.3 Inserting Nodes in the Cover Tree

The cover tree insertion we use is only a minor adaptation of the `Insert` algorithm by Beygelzimer et al. (2006). For each action $a \in \mathcal{A}$, we create a different reduced tree $\hat{\mathcal{T}}_a$, over the state space, i.e., $\mathcal{Z} = \mathcal{S}$, and build the tree using the metric $\psi(\boldsymbol{s}, \boldsymbol{s}') = \|\boldsymbol{s} - \boldsymbol{s}'\|_1$.

At each point in time t, we obtain a new observation tuple $\boldsymbol{s}_t, a_t, \boldsymbol{s}_{t+1}$. We select the tree $\hat{\mathcal{T}}_{a_t}$ corresponding to the action. Then, we traverse the tree, decreasing d and keeping a set of nodes $Q_d \subset G_d$ that are ζ^d-close to $\boldsymbol{s}_t$. We stop whenever Q_d contains a node that would satisfy the *parent proximity* property if we insert the new point at $d-1$, while the children of all other nodes in Q_d would satisfy the *sibling separation* property. This means that we can now insert the new datum as a child of that node.[3] Finally, the next state $\boldsymbol{s}_{t+1}$ is only used during the inference process, explained below.

2.2 Generalised Context Tree Inference

In our model, each node $i \in \hat{\mathcal{T}}$ is associated with a particular Bayesian model. The main problem is how to update the individual models and how to combine them. Fortunately, a closed form solution exists due to the tree structure. We use this to define a *generalised context tree*, which can be used for inference.

As with other tree models (Willems et al., 1995; Ferguson, 1974), our model makes predictions by marginalising over a set of simpler models. Each node in the context tree is called a *context*, and each context is associated with a specific local model. At time t, given an observation $\boldsymbol{s}_t = \boldsymbol{s}$ and an action $a_t = a$, we calculate the marginal (predictive) density p_t of the next observation:

$$p_t(\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_t, a_t) = \sum_{c_t} p_t(\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_t, c_t) p_t(c_t \mid \boldsymbol{s}_t, a_t),$$

[3] The exact implementation is available in the CoverTree class in Dimitrakakis et al. (2007).

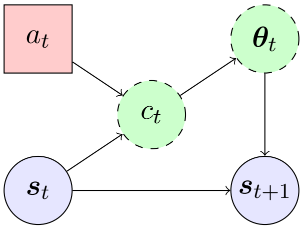


Figure 1: The generalised context tree graphical model. Blue circles indicate observed variables. Green dashed circles indicate latent variables. Red rectangles indicate choice variables. Arrows indicate dependencies. Thus, the context distribution at time t depends on both the state and action, while the parameters depend on the context. The next state depends on the action only indirectly.

where we use the symbol p_t throughout for notational simplicity to denote marginal distributions from our posterior at time t. Here, c_t is such that if $p_t(c_t = i \mid \boldsymbol{s}_t, a_t) > 0$, then the current state is within the neighbourhood of i-th node of the reduced cover tree $\hat{\mathcal{T}}_{a_t}$, i.e., $\boldsymbol{s}_t \in B_i$.

For Euclidean state spaces, the i-th component density $p_t(\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_t, c_t = i)$ employs a linear Bayesian model, which we describe in the next section. The graphical structure of the model is shown in simplified form in Fig. 1. The context at time t depends only on the current state $\boldsymbol{s}_t$ and action a_t. The context corresponds to a particular local model with parameter $\boldsymbol{\theta}_t$, which defines the conditional distribution.

The probability distribution $p_t(c_t \mid \boldsymbol{s}_t, a_t)$ is determined through *stopping probabilities.* More precisely, we set it be equal to the probability of stopping at the i-th context, when performing a walk from the leaf node containing the current observation towards the root, stopping at the j-th node with probability $w_{j,t}$ along the way:

$$p_t(c_t = i \mid \boldsymbol{s}_t, a_t) = w_{i,t} \prod_{j \in \mathfrak{D}_t(i)} (1 - w_{j,t}),$$

where $\mathfrak{D}_t(i)$ are the descendants of i that contain the observation $\boldsymbol{s}_t$. This forms a path from i to the leaf node containing $\boldsymbol{s}_t$. Note that $w_{0,t} = 1$, so we always stop whenever we reach the root. Due to the effectively linear structure of the relevant tree nodes, the stopping probability parameters w can be updated in closed form, as shown in Dimitrakakis (2010a, Theorem 1) via Bayes' theorem as follows:

$$w_{i,t+1} = \frac{p_t(\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_t, c_t = i) w_{i,t}}{p_t(\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_t, c_t \in \{i\} \cup \mathfrak{D}_t(i))}. \tag{2}$$

Since there is a different tree for each action, $c_t = i$ uniquely identifies a tree, the action does not need to enter in the conditional expressions above. Finally, it is easy to see, by marginalisation and the definition of the stopping probabilities, that the denominator in the above equation can be calculated recursively:

$$p_t(\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_t, c_t \in \{i\} \cup \mathfrak{D}_t(i)) = w_{i,t} p_t(\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_t, c_t = i) + (1 - w_{i,t}) p_t(\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_t, c_t \in \mathfrak{D}_t(i)).$$

Consequently, inference can be performed with a simple forward-backward sweep through a single tree path. In the forward stage, we compute the probabilities of the denominator, until we reach the point where we have to insert a new node. Whenever a new node is inserted in the tree, its weight parameter is initialised to $2^{-d(i)}$. We then go backwards to the root node, updating the weight parameters and the posterior of each model. The only remaining question is how to calculate the individual predictive marginal distributions for each context i in the forward sweep and how to calculate their posterior in the backward sweep. In this paper, we associate a linear Bayesian model with each context, which provides this distribution.

2.3 The Linear Bayesian Model

In our model we assume that, given $c_t = i$, the next state $\boldsymbol{s}_{t+1}$ is given by a linear transformation of the current state and additive noise $\boldsymbol{\varepsilon}_{i,t}$:

$$\boldsymbol{s}_{t+1} = \boldsymbol{A}_i \boldsymbol{x}_t + \boldsymbol{\varepsilon}_{i,t}, \qquad \boldsymbol{x}_t \triangleq \begin{pmatrix} \boldsymbol{s}_t \\ 1 \end{pmatrix}, \tag{3}$$

where $\boldsymbol{x}_t$ is the current state vector augmented by a unit basis.[4] In particular, each context models the dynamics via a Bayesian multivariate linear-Gaussian model. For the i-th context, there is a different (unknown) parameter pair $(\boldsymbol{A}_i, \boldsymbol{V}_i)$ where $\boldsymbol{A}_i$ is the *design* matrix and $\boldsymbol{V}_i$ is the *covariance* matrix. Then the next state distribution is:

$$\boldsymbol{s}_{t+1} \mid \boldsymbol{x}_t = \boldsymbol{x}, c_t = i \sim \mathcal{N}(\boldsymbol{A}_i \boldsymbol{x}, \boldsymbol{V}_i).$$

Thus, the parameters $\boldsymbol{\theta}_t$ which are abstractly shown in Fig. 1 correspond to the two matrices $\boldsymbol{A}, \boldsymbol{V}$. We now define the conditional distribution of these matrices given $c_t = i$.

We can model our uncertainty about these parameters with an appropriate prior distribution p_0. In fact, a conjugate prior exists in the form of the *matrix inverse-Wishart normal* distribution. In particular, given $\boldsymbol{V}_i = \boldsymbol{V}$, the distribution for $\boldsymbol{A}_i$ is matrix-normal, while the marginal distribution of $\boldsymbol{V}_i$ is inverse-Wishart:

$$\boldsymbol{A}_i \mid \boldsymbol{V}_i = \boldsymbol{V} \sim \mathcal{N}(\boldsymbol{A}_i \mid \underbrace{\boldsymbol{M}, \boldsymbol{C}}_{\text{prior parameters}}, \boldsymbol{V}) \tag{4}$$

$$\boldsymbol{V}_i \sim \mathcal{W}(\boldsymbol{V}_i \mid \overbrace{\boldsymbol{W}, n}). \tag{5}$$

Here $\mathcal{N}$ is the prior on design matrices, which has a matrix-normal distribution, conditional on the covariance and two prior parameters: $\boldsymbol{M}$, which is the prior mean and $\boldsymbol{C}$ which is the prior covariance of the dependent variable (i.e., the output). Finally, $\mathcal{W}$ is the marginal prior on covariance matrices, which has an inverse-Wishart distribution with $\boldsymbol{W}$ and n. More precisely, the distributions have the following forms:

$$\mathcal{N}(\boldsymbol{A}_i \mid \boldsymbol{M}, \boldsymbol{C}, \boldsymbol{V}) \propto e^{-\frac{1}{2}\operatorname{tr}\left[(\boldsymbol{A}_i - \boldsymbol{M})^\top \boldsymbol{V}^{-1} (\boldsymbol{A}_i - \boldsymbol{M}) \boldsymbol{C}\right]}$$

$$\mathcal{W}(\boldsymbol{V} \mid \boldsymbol{W}, n) \propto |\boldsymbol{V}^{-1} \boldsymbol{W} / 2|^{n/2} e^{-\frac{1}{2}\operatorname{tr}(\boldsymbol{V}^{-1} \boldsymbol{W})}.$$

[4] While other transformations of $\boldsymbol{s}_t$ are possible, we do not consider them in this paper.

Essentially, the model extends the classic Bayesian linear regression model (e.g., DeGroot, 1970) to the multivariate case via vectorisation of the mean matrix. Since the prior is conjugate, it is relatively simple to calculate the posterior after each observation. For simplicity, and to limit the total number of prior parameters we have to select, we use the same prior parameters $(\boldsymbol{M}_i, \boldsymbol{C}_i, \boldsymbol{W}_i, n_i)$ for all contexts in the tree.

To integrate this with inference in the tree, we must define the marginal distribution used in the nominator of (2). This is a multivariate Student-t distribution, so if the posterior parameters for context i at time t are $(\boldsymbol{M}_i^t, \boldsymbol{C}_i^t, \boldsymbol{W}_i^t, n_i^t)$, then this is:

$$p_t(\boldsymbol{s}_{t+1} \mid \boldsymbol{x}_t = \boldsymbol{x}, c_t = i) = \mathcal{S}tudent(\boldsymbol{M}_i^t, \boldsymbol{W}_i^t / z_i^t, 1 + n_i^t),$$

where $z_i^t = 1 - \boldsymbol{x}^\top (\boldsymbol{C}_i^t + \boldsymbol{x}\boldsymbol{x}^\top)^{-1} \boldsymbol{x}$.

2.3.1 Regression Illustration

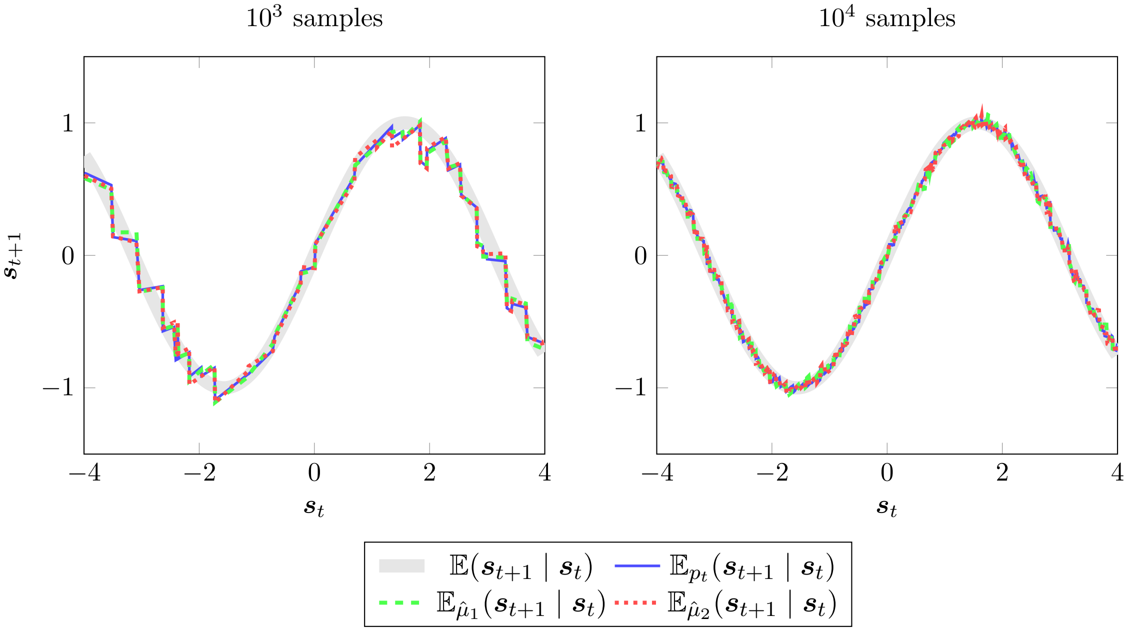


Figure 2: Regression illustration. We plot the expected value for the real distribution, the marginal, as well as two sampled models $\hat{\mu}_1, \hat{\mu}_2 \sim p_t(\mu)$.

An illustration of inference using the generalised context tree is given in Fig. 2, where the piecewise-linear structure is evident. The $\boldsymbol{s}_t$ variates are drawn uniformly in the displayed interval, while $\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_t = \boldsymbol{s} \sim \mathcal{N}(\sin(\boldsymbol{s}), 0.1)$, i.e., drawn a normal distribution with mean $\sin(\boldsymbol{s}_t)$ and variance 0.1. The plot shows the marginal expectation $\mathbb{E}_{p_t}$, as well as the expectation from two different models sampled from the posterior $p_t(\mu)$.

2.4 Approximating the Optimal Policy with Thompson Sampling

Many algorithms exist for finding the optimal policy for a specific MDP μ, or for calculating the expected utility of a given policy for that MDP. Consequently, a simple idea is to draw MDP samples μ_i from the current posterior distribution and then calculate the expected utility of each. This can be used to obtain approximate lower and upper bounds on the Bayes-optimal expected utility by maximising over the set of memoryless policies Π_1. Taking K samples, allows us to calculate the upper and lower bounds with accuracy $O(1/\sqrt{K})$.

$$\max_{\pi \in \Pi_1} \mathbb{E}_p^\pi U \approx \max_{\pi \in \Pi_1} \frac{1}{K} \sum_{i=1}^K \mathbb{E}_{\mu_i}^\pi U \le \frac{1}{K} \sum_{i=1}^K \max_{\pi \in \Pi_1} \mathbb{E}_{\mu_i}^\pi U, \qquad \mu_i \sim p_t(\mu). \tag{6}$$

We consider only the special case $K = 1$, i.e., when we only sample a single MDP. Then the two values are identical and we recover Thompson sampling. The main problems we have to solve now is how to sample a model and how to calculate a policy for the sampled model.

2.4.1 Sampling a Model from the Posterior

Each model μ sampled from the posterior corresponds to a particular choice of tree parameters. Sampling is done in two steps. The first generates a partition from the tree distribution and the second step generates a linear model for each context in the partition.

The first step is straightforward. We only need to sample a set of weights $\hat{w}_i \in \{0, 1\}$ such that $\mathbb{P}(\hat{w}_i = 1) = w_{i,t}$, as shown in Dimitrakakis (2010a, Rem. 2). This creates a *partition*, with one Bayesian multivariate linear model responsible for each context in the partition.

The second step is to sample a design and covariance matrix pair $(\hat{\boldsymbol{A}}_i, \hat{\boldsymbol{V}}_i)$ for each context i in the partition. This avoids sampling matrices for contexts not part of the sampled tree. As the model suggests, we can first sample the noise covariance by plugging the posterior parameters in (5) to obtain $\hat{\boldsymbol{V}}_i$. Sampling from this distribution can be done efficiently using the algorithm suggested by Smith and Hocking (1972). We then plug in $\hat{\boldsymbol{V}}_i$ into the conditional design matrix posterior (4) to obtain a design matrix $\hat{\boldsymbol{A}}_i$ by sampling from the resulting matrix-normal distribution.

The final MDP sample μ from the posterior has two elements. Firstly, a set of contexts $\hat{C}^\mu \subset \bigcup_{a \in \mathcal{A}} \hat{\mathcal{T}}_a$, from all action trees. This set is a partition with associated mapping $f^\mu : \mathcal{S} \times \mathcal{A} \to \hat{C}^\mu$. Secondly, a set of associated design and covariance matrices $\left\{ (A_i^\mu, V_i^\mu) \;\middle|\; i \in \hat{C}^\mu \right\}$ for each context. Then the prediction of the sampled MDP is:

$$\mathbb{P}_\mu(\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_t, a_t) = \mathcal{N}(A^\mu_{f(\boldsymbol{s}_t, a_t)} \boldsymbol{x}_t, V^\mu_{f(\boldsymbol{s}_t, a_t)}), \tag{7}$$

where $\boldsymbol{x}_t$ is given in (3).

2.4.2 Finding a Policy for a Sample via ADP

In order to calculate an optimal policy $\pi^*(\mu)$ for μ, we generate a large number of trajectories from μ using a uniform policy. After selecting an appropriate set of basis functions, we then employ a variant of the least-squares policy iteration (LSPI, Lagoudakis and Parr, 2003) algorithm, using least-squares temporal differences (LSTD, Bradtke and Barto, 1996) rather than LSTDQ. This is possible because since we have μ available, we have access to (7) and it makes LSPI slightly more efficient.

More precisely, consider the *value function* $V^\pi_\mu : \mathcal{S} \to \mathbb{R}$, defined as:

$$V^\pi_\mu(\boldsymbol{s}) \triangleq \mathbb{E}^\pi_\mu \left(U \mid \boldsymbol{s}_t = \boldsymbol{s}\right).$$

Unfortunately, for continuous $\mathcal{S}$ finding an optimal policy requires approximations. A common approach is to make use of the fact that:

$$V^\pi_\mu(\boldsymbol{s}) = \rho(\boldsymbol{s}) + \gamma \int_{\mathcal{S}} V^\pi_\mu(\boldsymbol{s}') \, \mathrm{d}P^\pi_\mu(\boldsymbol{s}' \mid \boldsymbol{s}),$$

where we assume for simplicity that $\rho(\boldsymbol{s})$ is the reward obtained at state $\boldsymbol{s}$. The conditional measure P^π_μ is the transition kernel on $\mathcal{S}$ induced by μ, π, introduced in Section 1.1. We then select a parametric family $v_\omega : \mathcal{S} \to \mathbb{R}$ with parameter $\omega \in \Omega$ and minimise:

$$h(\omega) + \int_{\mathcal{S}} \left\| v_\omega(\boldsymbol{s}) - \rho(\boldsymbol{s}) - \gamma \int_{\mathcal{S}} v_\omega(\boldsymbol{s}') \, \mathrm{d}\hat{P}^\pi_\mu(\boldsymbol{s}'|\boldsymbol{s}) \right\| \mathrm{d}\chi(\boldsymbol{s}), \tag{8}$$

where h is a regularisation term, χ is an appropriate measure on $\mathcal{S}$ and $\hat{P}^\pi_\mu$ is an empirical estimate of the transition kernel, used to approximate the respective integral that uses P^π_μ. As we can take an arbitrary number of trajectories from μ, π, this can be as accurate as our computational capacity allows.

In practice, we minimise (8) with a generalised linear model (defined on an appropriate basis) for v_ω while χ need only be positive on a set of representative states. Specifically, we employ a variant of the least-squares policy iteration (LSPI, Lagoudakis and Parr, 2003) algorithm, using the least-squares temporal differences (LSTD, Bradtke and Barto, 1996) for the minimisation of (8). Then the norm is the euclidean norm and the regularisation term is $h(\omega) = \lambda\|\omega\|$. In order to estimate the inner integral, we take $K_{\mathrm{L}} \geq 1$ samples from the model so that

$$\hat{P}^\pi_\mu(\boldsymbol{s}' \mid \boldsymbol{s}) \triangleq \frac{1}{K_{\mathrm{L}}} \sum_{i=1}^{K_{\mathrm{L}}} \mathbb{I}\left\{\boldsymbol{s}^i_{t+1} = \boldsymbol{s}' \mid \boldsymbol{s}^i_t = \boldsymbol{s}\right\}, \tag{9}$$
$$\boldsymbol{s}^i_{t+1} \mid \boldsymbol{s}^i_t = \boldsymbol{s} \sim P^\pi_\mu(\cdot \mid \boldsymbol{s}),$$

where $\mathbb{I}\{\cdot\}$ is an indicator function and P^π_μ is decomposable in known terms. Equation (9) is also used for action selection in order to calculate an approximate expected utility $q_\omega(\boldsymbol{s}, a)$ for each state-action pair $(\boldsymbol{s}, a)$:

$$q_\omega(\boldsymbol{s}, a) \triangleq \rho(s) + \gamma \int_{\mathcal{S}} v_\omega(\boldsymbol{s}') \, \mathrm{d}\hat{P}^\pi_\mu(\boldsymbol{s}'|\boldsymbol{s})$$

Effectively, this approximates the integral via sampling. This may add a small amount[5] of additional stochasticity to action selection, which can be reduced[6] by increasing K_{L}.

Finally, we optimise the policy by approximate policy iteration. At the j-th iteration we obtain an improved policy $\hat{\pi}_j(a \mid \boldsymbol{s}) \propto \mathbb{P}[a \in \arg\max_{a' \in \mathcal{A}} q_{\omega_{j-1}}(\boldsymbol{s}, a')]$ from ω_{j-1} and then estimate ω_j for the new policy.

[5] Generally, this error is bounded by $O(K_{\mathrm{L}}^{-1/2})$.

[6] We remind the reader that Thompson sampling itself results in considerable exploration by sampling an MDP from the posterior. Thus, additional randomness may be detrimental.

2.5 Complexity

We now analyse the computational complexity of our approach, including the online complexity of inference and decision making, and of the sampling and ADP taking place every episode. It is worthwhile to note two facts. Firstly, that the complexity bounds related to the cover tree depend on a constant c, which however depends on the distribution of samples in the state space. In the worst case (i.e., a uniform distribution), this is bounded exponentially in the dimensionality of the actual state space. While we do not expect this to be the case in practice, it is easy to construct a counterexample where this is the case. Secondly, that the complexity of the ADP step is largely independent of the model used, and mostly depends on the number of trajectories we take in the sampled model and the dimensionality of the feature space.

First, we examine the total computation time that is required to construct the tree.

Corollary 1 *Cover tree construction from t observations takes $O(t \ln t)$ operations.*

Proof In the cover tree, node insertion and query are $O(\ln t)$(Beygelzimer et al., 2006, Theorems 5, 6). Then note that $\sum_{k=1}^{t} \ln k \leq \sum_{k=1}^{t} \ln t = t \ln t$. ■

At every step of the process, we must update our posterior parameters. Fortunately, this also takes logarithmic time as we only need to perform calculations for a single path from the root to a leaf node.

Lemma 2 *If $\mathcal{S} \subset \mathbb{R}^m$, then inference at time step t has complexity $O(m^3 \ln t)$.*

Proof At every step, we must perform inference on a number of nodes equal to the length of the path containing the current observation. This is bounded by the depth of the tree, which is in turn bounded by $O(\ln t)$ from Beygelzimer et al. (2006, Lemma 4.3). Calculating (2) is linear in the depth. For each node, however, we must update the linear-Bayesian model, and calculate the marginal distribution. Each requires inverting an $m \times m$ matrix, which has complexity $O(m^3)$. ■

Finally, at every step we must choose an action through value function look-up. This again takes logarithmic time, but there is a scaling depending on the complexity of the value function representation.

Lemma 3 *If the LSTD basis has dimensionality m_L, then taking a decision at time t has complexity $O(K_L m_L \ln t)$.*

Proof To take a decision we merely need to search in each action tree to find a corresponding path. This takes $O(\ln t)$ time for each tree. After Thompson sampling, there will only be one linear model for each action tree. LSTD takes K_{L} operations, and requires the inner product of two m_L-dimensional vectors. ■

The above lemmas give the following result:

Theorem 4 *At time t, the online complexity of CTBRL is $O((m^3 + K_L m_L) \ln t)$.*

We now examine the complexity of finding a policy. Although this is the most computationally demanding part, its complexity is not dependent on the cover tree structure or the probabilistic inference method used. However, we include it here for completeness.

Lemma 5 *Thompson sampling at time t is $O(tm^3)$.*

Proof In the worst case, our sampled tree will contain all the leaf nodes of the reduced tree, which are $O(t)$. For each sampled node, the most complex operation is Wishart generation, which is $O(m^3)$ (Smith and Hocking, 1972). ■

Lemma 6 *If we use n_s samples for LSTD estimation and the basis dimensionality is m_L, this step has complexity $O(m_L^3 + n_s(m_L^2 + K_L m_L \ln t))$.*

Proof For each sample we must take a decision according to the last policy, which requires $O(K_{\mathrm{L}} m_L \ln t)$ as shown previously. We also need to update two matrices (see Boyan, 2002), which is $O(m_L^2)$. So, $O(n_s(m_L^2 + K_{\mathrm{L}} m_L \ln t))$ computations must be performed for the total number of the selected samples. Since LSTD requires an $m_L \times m_L$ matrix inversion, with complexity $O(m_L^3)$, we obtain the final result. ■

From Lemmas 3 and 6 it follows that:

Theorem 7 *If we employ API with K_A iterations, the total complexity of calculating a new policy is $O(tm^3 + K_A(m_L^3 + n_s(m_L^2 + K_L m_L \ln t)))$.*

Thus, while the online complexity of CTBRL is only logarithmic in t, there is a substantial cost when calculating a new policy. This is only partially due to the complexity of sampling a model, which is manageable when the state space has small dimensionality. Most of the computational effort is taken by the API procedure, at least as long as $t < (m_L/m)^3$. However, we think this is unavoidable no matter what the model used is.

The complexity of Gaussian process (GP) models is substantially higher. In the simplest model, where each output dimension is modelled independently, inference is $O(mt^3)$, while the fully multivariate tree model has complexity $O(m^3 t \ln t)$. Since there is no closed form method for sampling a function from the process, one must resort to iterative sampling of points. For n points, the cost is approximately $O(nmt^3)$, which makes sampling long trajectories prohibitive. For that reason, in our experiments we only use the mean of the GP.

3. Experiments

We conducted two sets of experiments to analyse the offline and the online performance. We compared CTBRL with the well-known LSPI algorithm (Lagoudakis and Parr, 2003) for the offline case, as well as an online variant (Buşoniu et al., 2010) for the online case. We also compared CTBRL with linear Bayesian reinforcement learning (LBRL, Tziortziotis et al., 2013) and finally GP-RL, where we simply replaced the tree model with a Gaussian process. For CTBRL and LBRL we use Thompson sampling. However, since Thompson sampling cannot be performed on GP models, we use the mean GP instead. In order to compute policies given a model, all model-based methods use the variant of LSPI explained in Section 2.4.2. Hence, the only significant difference between each approach is the model used, and whether or not they employ Thompson sampling.

A significant limitation of Gaussian processes is that their computational complexity becomes prohibitive as the number of samples becomes extremely large. In order to make the GP model computationally practical, the greedy approximation approach introduced by Engel et al. (2002) has been adopted. This is a kernel sparsification methodology which incrementally constructs a dictionary of the most representative states. More specifically, an *approximate linear dependence* analysis is performed in order to examine whether a state can be approximated sufficiently as a linear combination of current dictionary members or not.

We used one preliminary run and guidance from the literature to make an initial selection of possible hyper-parameters, such as the number of samples and the features used for LSTD and LSTD-Q. We subsequently used 10 runs to select a single hyper-parameter combination for each algorithm-domain pair. The final evaluation was done over an independent set of 100 runs.

For CTBRL and the GP model, we had the liberty to draw an arbitrary number of trajectories for the value function estimation. We drew 1-step transitions from a set of 3000 uniformly drawn states from the *sampled* model (the mean model in the GP case). We used 25 API iterations on this data.

For the offline performance evaluation, we first drew rollouts from $k = \{10, 20, \ldots, 50, 100, \ldots, 1000\}$ states drawn from the *true environment's* starting distribution, using a uniformly random policy. The maximum horizon of each rollout was set equal to 40. The collected data was then fed to each algorithm in order to produce a policy. This policy was evaluated over 1000 rollouts on the environment.

In the online case, we simply use the last policy calculated by each algorithm at the end of the last episode, so there is no separate learning and evaluation phase. This means that efficient exploration must be performed. For CTBRL, this is done using Thompson sampling. For online-LSPI, we followed the approach of Buşoniu et al. (2010), who adopts an ϵ-greedy exploration scheme with an exponentially decaying schedule $\epsilon_t = \epsilon_d^t$, with $\epsilon_0 = 1$. In preliminary experiments, we found $\epsilon_d = 0.997$ to be a reasonable compromise. We compared the algorithms online for 1000 episodes.

3.1 Domains

We consider two well-known continuous state, discrete-action, episodic domains. The first is the inverted pendulum domain and the second is the mountain car domain.

3.1.1 Inverted Pendulum

The goal in this domain, is to balance a pendulum by applying forces of a mixed magnitude (50 Newtons). The state space consists of two continuous variables, the vertical angle and the angular velocity of the pendulum. There are three actions: no force, left force or right force. A zero reward is received at each time step except in the case where the pendulum falls. In this case, a negative (-1) reward is given and a new episode begins. An episode also ends with 0 reward after 3000 steps, after which we consider that the pendulum is successfully balanced. Each episode starts by setting the pendulum in a perturbed state close to the equilibrium point. More information about the specific dynamics can be found at Lagoudakis and Parr (2003). The discount factor γ was 0.95. The basis we used for LSTD/LSPI, was equidistant 3×3 grid of RBFs over the state space following the suggestions of Lagoudakis and Parr (2003). This was replicated for each action for the LSTD-Q algorithm used in LSPI.

3.1.2 Mountain Car

The aim in this domain is to drive an underpowered car to the top of a hill. Two continuous variables characterise the vehicle state in the domain, its position and its velocity. The objective is to drive an underpowered vehicle up a steep valley from a randomly selected position to the right hilltop (at position > 0.5) within 1000 steps. There are three actions: forward, reverse and zero throttle. The received reward is -1 except in the case where the target is reached (zero reward). At the beginning of each rollout, the vehicle is positioned to a new state, with the position and the velocity uniformly randomly selected. The discount factor is set to $\gamma = 0.999$. An equidistant 4×4 grid of RBFs over the state space plus a constant term is selected for LSTD and LSPI.

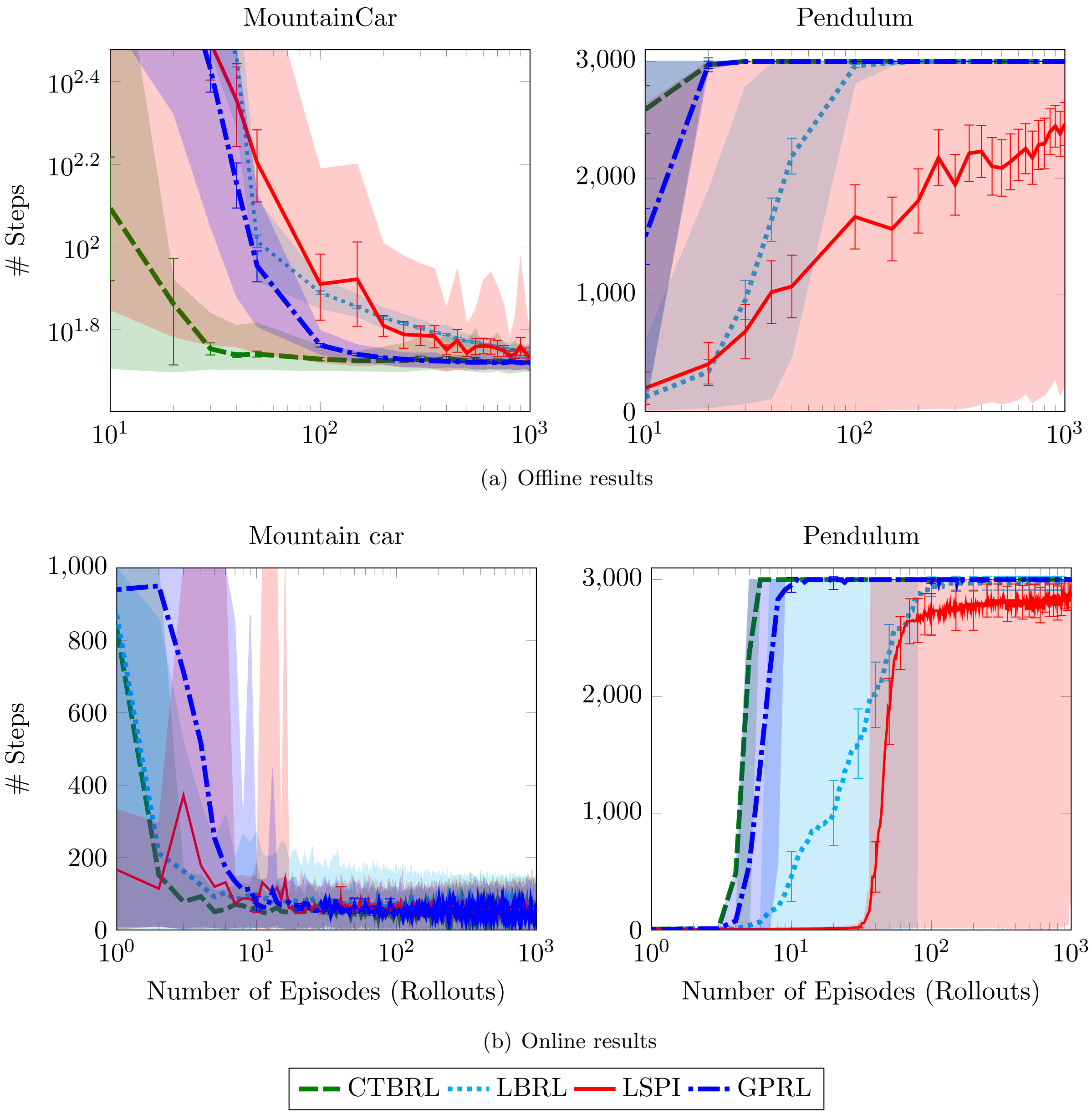
3.2 Results

In our results, we show the average performance in terms of number of steps of each method, averaged over 100 runs. For each average, we also plot the 95% confidence interval for the accuracy of the mean estimate with error bars. In addition, we show the 90% percentile region of the runs, in order to indicate inter-run variability in performance.

Figure 3(a) shows the results of the experiments in the *offline* case. For the *mountain car*, it is clear that CTBRL is significantly more stable compared to GPRL and LSPI. In contrast to the other two approaches, CTBRL needs only a small number of rollouts in order to discover the optimal policy. For the *pendulum* domain, the performance of both CTBRL and GPRL is almost perfect, as they need only about twenty rollouts in order to discover the optimal policy. On the other hand, LSPI despite the fact that manages to find the optimal policy frequently, around 5% of its runs fail.

Figure 3(b) shows the results of the experiments in the *online* case. For the *mountain car*, CTBRL managed to find an excellent policy in the vast majority of runs, while converging earlier than GPRL and LSPI. Moreover, CTBRL presents a more stable behaviour in contrast to the other two. In the *pendulum* domain, the performance difference relative to LSPI is even more impressive. It becomes apparent that both CTBRL and GPRL reach near optimal performances with an order of magnitude fewer episodes than LSPI, while the latter remains unstable. In this experiment, we see that CTBRL reaches an optimal policy slightly before GPRL. Although the difference is small, it is very consistent.

The success of CTBRL over the other approaches can be attributed to a number of reasons. Firstly, it could be a better model. Indeed, in the offline results for the mountain car domain, where the starting state distribution is uniform, and all methods have the same data, we can see that CTBRL has a far better performance than everything else. The second could be the more efficient exploration afforded by Thompson sampling. Indeed, in the mountain car online experiments we see that the LBRL performs quite well (Fig. 3(b)), even though its offline performance is not very good (Fig. 3(a)). However, Thompson sampling is not sufficient for obtaining a good performance, as seen by both the offline results and the performance in the pendulum domain.



(a) Offline results

(b) Online results

Figure 3: Experimental evaluation. The dashed line shows CTBRL, the dotted line shows LBRL, the solid line shows LSPI, while the dash-dotted line shows GPRL. The error bars denote 95% confidence intervals for the mean (i.e., statistical significance). The shaded regions denote 90% percentile performance (i.e., robustness) across runs. In all cases, CTBRL converges significantly quicker than the other approaches. In addition, as the percentile regions show, it is also much more stable than LBRL, GPRL and LSPI.

4. Conclusion

We proposed a computationally efficient, fully Bayesian approach for the exact inference of unknown dynamics in continuous state spaces. The total computation for inference after t steps is $O(t \ln t)$, in stark contrast to other non-parametric models such as Gaussian processes, which scale $O(t^3)$. In addition, inference is naturally performed online, with the computational cost at time t being $O(\ln t)$.

In practice, the computational complexity is orders of magnitude lower for cover trees than GP, even for these problems. We had to use a dictionary and a lot of tuning to make GP methods work, while cover trees worked out of the box. Another disadvantage of GP methods is that it is infeasible to implement Thompson sampling with them. This is because it is not possible to directly sample a function from the GP posterior. Although Thompson sampling confers no advantage in the offline experiments (as the data there were the same for all methods), we still see that the performance of CTBRL is significantly better on average and that it is much more stable.

Experimentally, we showed that cover trees are more efficient both in terms of computation and in terms of reward, relative to GP models that used the same ADP method to optimise the policy and to a linear Bayesian model which used both the same ADP method and the same exploration strategy. We can see that overall the linear model performs significantly worse than both GP-RL and CTBRL, though better than ϵ-greedy LSPI. This shows that the main reason for the success of CTBRL is the cover tree inference and not the linear model itself, or Thompson sampling.

CTBRL is particularly good in online settings, where the exact inference, combined with the efficient exploration provided by Thompson sampling give it an additional advantage. We thus believe that CTBRL is a method that is well-suited for exploration in unknown continuous state problems. Unfortunately, it is not possible to implement Thompson sampling in practice using GPs, as there is no reasonable way to sample a function from the GP posterior. Nevertheless, we found that in both online and offline experiments (where Thompson sampling should be at a disadvantage) the cover tree method achieved superior performance to Gaussian processes.

Although we have demonstrated the method in low dimensional problems, higher dimensions are not a problem for the cover tree inference itself. The bottleneck is the value function estimation and ADP. This is independent of the model used, however. For example, GP methods for estimating the value function (c.f. Deisenroth et al., 2009) typically have a large number of hyper-parameters for value function estimation, such as choice of representative states and trajectories, kernel parameters and method for updating the dictionary, to avoid problems with many observations.

While in practice ADP can be performed in the background while inference is taking place, and although we seed the ADP with the previous solution, one would ideally like to use a more incremental approach for that purpose. One interesting idea would be to employ a gradient approach in a similar vein to Deisenroth and Rasmussen (2011). An alternative approach would be to employ an online method, in order to avoid estimating a policy for the complete space.[7] Promising such approaches include running bandit-based tree search methods such as UCT (Kocsis and Szepesvári, 2006) on the sampled models.

[7] A suggestion made by the anonymous reviewers.

Another direction of future work is to consider more sophisticated exploration policies, particularly for larger problems. Due to the efficiency of the model, it should be possible to compute near-Bayes-optimal policies by applying the tree search method used by Veness et al. (2011). Finally, it would be interesting to examine continuous actions. These can be handled efficiently both by the cover tree and the local linear models by making the next state directly dependent on the action through an augmented linear model. While optimising over a continuous action space is challenging, more recent efficient tree search methods such as metric bandits (Bubeck et al., 2011) may alleviate that problem.

An interesting theoretical direction would be to obtain regret bounds for the problem. This could perhaps be done building upon the analyses of Kozat et al. (2007) for context tree prediction, and of Ortner and Ryabko (2012) for continuous MDPs. The statistical efficiency of the method could be improved by considering edge-based (rather than node-based) distributions on trees, as was suggested by Pereira and Singer (1999).

Finally, as the cover tree method only requires specifying an appropriate metric, the method could be applicable to many other problems. This includes both large discrete problems, and partially observable problems. It would be interesting to see if the approach also gives good results in those cases.

Acknowledgements

We would like to thank the anonymous reviewers, for their careful and detailed comments and suggestions, for this and previous versions of the paper, which have significantly improved the manuscript. We also want to thank Mikael Kågebäck for additional proofreading. This work was partially supported by the Marie Curie Project ESDEMUU "Efficient Sequential Decision Making Under Uncertainty" , Grant Number 237816 and by an ERASMUS exchange grant.

References

P. Abbeel and A. Y. Ng. Exploration and apprenticeship learning in reinforcement learning. In *International Conference on Machine learning (ICML)*, pages 1–8, 2005.

S. Agrawal and N. Goyal. Analysis of Thompson sampling for the multi-armed bandit problem. In *Conference on Learning Theory (COLT)*, pages 39.1–39.26, 2012.

M. Alvarez, D. Luengo-Garcia, M. Titsias, and N. Lawrence. Efficient multioutput Gaussian processes through variational inducing kernels. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 25–32, 2011.

M. Araya, V. Thomas, and O. Buffet. Near-optimal BRL using optimistic local transitions. In *International Conference on Machine Learning (ICML)*, pages 97–104, 2012.

J. Asmuth, L. Li, M. L. Littman, A. Nouri, and D. Wingate. A Bayesian sampling approach to exploration in reinforcement learning. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 19–26, 2009.

P. Auer, T. Jaksch, and R. Ortner. Near-optimal regret bounds for reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 89–96, 2008.

R. Begleiter, R. El-Yaniv, and G. Yona. On prediction using variable order Markov models. *Journal of Artificial Intelligence Research*, 22:385–421, 2004.

M. Bellemare, J. Veness, and M. Bowling. Bayesian learning of recursively factored environments. In *International Conference on Machine Learning (ICML)*, pages 1211–1219, 2013.

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

A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *International Conference on Machine Learning (ICML)*, pages 97–104, 2006.

J. A. Boyan. Technical update: Least-squares temporal difference learning. *Machine Learning*, 49(2):233–246, 2002.

S. J. Bradtke and A. G. Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22(1):33–57, 1996.

E. Brunskill, B. R. Leffler, L. Li, M. L. Littman, and N. Roy. Provably efficient learning with type parametric models. *Journal of Machine Learning Research*, 10:1955–1988, 2009.

S. Bubeck, R. Munos, G. Stoltz, and C. Szepesvári. X-armed bandits. *Journal of Machine Learning Research*, 12:1655–1695, 2011.

L. Buşoniu, D. Ernst, B. De Schutter, and R. Babuška. Online least-squares policy iteration for reinforcement learning control. In *American Control Conference (ACC)*, pages 486–491, 2010.

P. Castro and D. Precup. Smarter sampling in model-based Bayesian reinforcement learning. *Machine Learning and Knowledge Discovery in Databases*, pages 200–214, 2010.

M. Daswani, P. Sunehag, and M. Hutter. Feature reinforcement learning using looping suffix trees. In *European Workshop on Reinforcement Learning (EWRL)*, pages 11–24, 2012.

R. Dearden, N. Friedman, and S. J. Russell. Bayesian Q-learning. In *AAAI/IAAI*, pages 761–768, 1998. URL `citeseer.ist.psu.edu/dearden98bayesian.html`.

M. H. DeGroot. *Optimal Statistical Decisions.* John Wiley & Sons, 1970.

M. P. Deisenroth and C. E. Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *International Conference on Machine Learning (ICML)*, pages 465–472, 2011.

M. P. Deisenroth, C. E. Rasmussen, and J. Peters. Gaussian process dynamic programming. *Neurocomputing*, 72(7-9):1508–1524, 2009.

C. Dimitrakakis. Bayesian variable order Markov models. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 161–168, 2010a.

C. Dimitrakakis. Complexity of stochastic branch and bound methods for belief tree search in Bayesian reinforcement learning. In *International Conference on Agents and Artificial Intelligence (ICAART)*, pages 259–264, 2010b.

C. Dimitrakakis. Robust bayesian reinforcement learning through tight lower bounds. In *European Workshop on Reinforcement Learning (EWRL)*, pages 177–188, 2011.

C. Dimitrakakis, N. Tziortziotis, and A. Tossou. Beliefbox: A framework for statistical methods in sequential decision making. `http://code.google.com/p/beliefbox/`, 2007.

M. Duff. *Optimal Learning Computational Procedures for Bayes-adaptive Markov Decision Processes*. PhD thesis, University of Massachusetts at Amherst, 2002.

Y. Engel, S. Mannor, and R. Meir. Sparse online greedy support vector regression. In *European Conference on Machine Learning (ECML)*, pages 84–96, 2002.

Y. Engel, S. Mannor, and R. Meir. Reinforcement learning with Gaussian process. In *International Conference on Machine Learning (ICML)*, pages 201–208, 2005.

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

V. F. Farias, C. C. Moallemi, B. Van Roy, and T. Weissman. Universal reinforcement learning. *IEEE Transactions on Information Theory*, 56(5):2441–2454, 2010.

T. S. Ferguson. Prior distributions on spaces of probability measures. *The Annals of Statistics*, 2(4):615–629, 1974.

M. Ghavamzadeh and Y. Engel. Bayesian policy gradient algorithms. In *Advances in Neural Information Processing Systems (NIPS)*, pages 457–464, 2006.

E. Kaufmann, N. Korda, and R. Munos. Thompson sampling: An optimal finite time analysis. In *Algorithmic Learning Theory (ALT)*, pages 199–213, 2012.

L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. In *European Conference on Machine Learning (ECML)*, pages 282–293, 2006.

S. S. Kozat, A. C. Singer, and G. C. Zeitler. Universal piecewise linear prediction via context trees. *IEEE Transactions on Signal Processing*, 55(7):3730–3745, 2007.

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

M. Meila and M. I. Jordan. Learning with mixtures of trees. *The Journal of Machine Learning Research*, 1:1–48, 2001.

R. Ortner and D. Ryabko. Online regret bounds for undiscounted continuous reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1772–1780, 2012.

I. Osband, D. Russo, and B. Van Roy. (More) efficient reinforcement learning via posterior sampling. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3003–3011, 2013.

S. M. Paddock, F. Ruggeri, M. Lavine, and M. West. Randomized Polya tree models for nonparametric Bayesian inference. *Statistica Sinica*, 13(2):443–460, 2003.

F. C. Pereira and Y. Singer. An efficient extension to mixture techniques for prediction and decision trees. *Machine Learning*, 36(3):183–199, 1999.

P. Poupart, N. Vlassis, J. Hoey, and K. Regan. An analytic solution to discrete Bayesian reinforcement learning. In *International Conference on Machine Learning (ICML)*, pages 697–704, 2006.

M. L. Puterman. *Markov Decision Processes : Discrete Stochastic Dynamic Programming.* John Wiley & Sons, New Jersey, US, 2005.

C. E. Rasmussen and M. Kuss. Gaussian processes in reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 751–759, 2004.

W. B. Smith and R. R. Hocking. Wishart variates generator, algorithm AS 53. *Applied Statistics*, 21:341–345, 1972.

A. L. Strehl and M. L. Littman. Online linear regression and its application to model-based reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1417–1424, 2008.

M. Strens. A Bayesian framework for reinforcement learning. In *International Conference on Machine Learning (ICML)*, pages 943–950, 2000.

W. R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3-4):285–294, 1933.

N. Tziortziotis, C. Dimitrakakis, and K. Blekas. Linear Bayesian reinforcement learning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1721–1728, 2013.

T. van Erven, P. D. Grünwald, and S. de Rooij. Catching up faster by switching sooner: a prequential solution to the AIC-BIC dilemma. *arXiv*, 2008. A preliminary version appeared in NIPS 2007.

J. Veness, K. S. Ng, M. Hutter, W. Uther, and D. Silver. A Monte-Carlo AIXI approximation. *Journal of Artificial Intelligence Research*, 40:95–142, 2011.

J. Veness, K. S. Ng, M. Hutter, and M. Bowling. Context tree switching. In *Data Compression Conference (DCC)*, pages 327–336, 2012.

T. Wang, D. Lizotte, M. Bowling, and D. Schuurmans. Bayesian sparse sampling for on-line reward optimization. In *International Conference on Machine Learning (ICML)*, pages 956–963, 2005.

F. M. J. Willems, Y. M. Shtarkov, and T. J. Tjalkens. The context tree weighting method: basic properties. *IEEE Transactions on Information Theory*, 41(3):653–664, 1995.

W. H. Wong and L. Ma. Optional Pólya tree and Bayesian inference. *The Annals of Statistics*, 38(3):1433–1459, 2010.

J. Wyatt. *Exploration and inference in learning from reinforcement.* PhD thesis, University of Edinburgh. College of Science and Engineering. School of Informatics, 1998.