# Inferential Induction: A Novel Framework for Bayesian Reinforcement Learning

Emilio Jorge, Hannes Eriksson, Christos Dimitrakakis, Debabrota Basu, Divya Grover

# Inferential Induction: A Novel Framework for Bayesian Reinforcement Learning

**Emilio Jorge**[*]
Chalmers University of Technology
emilio.jorge@chalmers.se

**Hannes Eriksson**[*]
Chalmers University of Technology
Zenseact

**Christos Dimitrakakis**[*]
University of Oslo
Chalmers University of Technology

**Debabrota Basu**
Scool, Inria Lille-Nord Europe
Chalmers University of Technology

**Divya Grover**
Chalmers University of Technology

## Abstract

Bayesian Reinforcement Learning (BRL) offers a decision-theoretic solution to the reinforcement learning problem. While "model-based" BRL algorithms have focused either on maintaining a posterior distribution on models, BRL "model-free" methods try to estimate value function distributions but make strong implicit assumptions or approximations. We describe a novel Bayesian framework, *inferential induction*, for correctly inferring value function distributions from data, which leads to a new family of BRL algorithms. We design an algorithm, Bayesian Backwards Induction (BBI), with this framework. We experimentally demonstrate that BBI is competitive with the state of the art. However, its advantage relative to existing BRL model-free methods is not as great as we have expected, particularly when the additional computational burden is taken into account.

## 1 Introduction

Many Reinforcement Learning (RL) algorithms are grounded on the application of dynamic programming to a Markov Decision Process (MDP) [Sutton and Barto, 2018]. When the underlying MDP $\mu$ is known, efficient algorithms for finding an optimal policy exist that exploit the Markov property by calculating value functions. Such algorithms can be applied to RL, where the learning agent simultaneously acts in and learns about the MDP, through e.g. stochastic approximations, without explicitly reasoning about the underlying MDP. Hence, these algorithms are called model-free.

In Bayesian RL (BRL) [Ghavamzadeh et al., 2015], we represent our knowledge about the underlying MDP $\mu$ through some prior distribution $\beta$ over a set $\mathcal{M}$ of possible MDPs. This explicit representation of uncertainty admits algorithms that can perform near-optimal exploration, in contrast to methods that only rely on a single empirical model of the MDP, which require exploration heuristics.[c.f. Vlassis et al., 2012] While model-based BRL is well-understood, many BRL algorithms try to become model-free by calculating distributions on value functions. Unfortunately, these methods typically make implicit assumptions or approximations about the underlying MDP distribution.

[*]Equal contribution

1st I Can't Believe It's Not Better Workshop (ICBINB@NeurIPS 2020), Vancouver, Canada.

In the rest of this section, we provide background in terms of setting and related work. In Section 2, we explain our Inferential Induction framework and three different inference methods that emerge from it, before instantiating one of them into a concrete procedure. Based on this, Section 3 describes the BBI algorithm. In Section 4, we experimentally compare BBI with state-of-the art BRL algorithms. In the appendix additional experimental results are available in Appendix B and some implementation details in Appendix C.

### 1.1 Setting and Notation

In this paper, we generally use $\mathbb{P}$ and $\mathbb{E}$ to refer to probability (measures) and expectations while allowing some abuse of notation for compactness.

**Reinforcement Learning (RL)** is a sequential learning problem faced by agents acting in an unknown environment $\mu$, typically modelled as a Markov decision process [c.f. Puterman, 2005].

**Definition 1.1** (Markov Decision Process (MDP))**.** An MDP $\mu$ with state space $\mathcal{S}$ and action space $\mathcal{A}$ is equipped with a reward distribution $\mathbb{P}_\mu(r \mid s)$ with corresponding expectation $\rho_\mu(s)$ and a transition kernel $\mathbb{P}_\mu(s'|s,a)$ for states $s, s' \in \mathcal{S}$ and actions $a \in \mathcal{A}$.

At time $t$, the agent observes[1] the environment state $s_t$, and then selects an action $a_t$. Then, it receives and observes a reward $r_t$ and a next state $s_{t+1}$. The agent is interested in the utility $U_t \triangleq \sum_{k=t}^{T} \gamma^{k-t} r_k$, i.e. the sum of future rewards $r_t$. Here, $\gamma \in (0,1]$ is the discount factor and $T \in [1,\infty]$ is the problem horizon. Typically, the agent wishes to maximise the *expected utility*, but other objectives are possible.

The agent acts in the environment using a policy $\pi = (\pi_1, \dots, \pi_t, \dots)$ that takes an action $a_t$ at time $t$ with probability $\pi_t(a_t \mid s_t, r_{t-1}, a_{t-1}, s_{t-1}, \dots, r_1, a_1, s_1)$. Dependence on the complete observation history is necessary, if the agent is learning from experience. However, when $\mu$ is known, the policy $\pi^*_\mu$ maximising expected utility over finite horizon is Markovian[2] of the form $\pi_t(a_t \mid s_t)$ and is computable using dynamic programming. A useful algorithmic tool for achieving this is the value function, i.e. the expected utility of a policy $\pi$ from different starting states and action:

**Definition 1.2** (Value Function)**.** The state value function of policy $\pi$ in MDP $\mu$ is $V^\pi_{\mu,t}(s) \triangleq \mathbb{E}^\pi_\mu(U_t \mid s_t = s)$ and the corresponding state-action (or Q-)value function is $Q^\pi_{\mu,t}(s,a) \triangleq \mathbb{E}^\pi_\mu(U_t \mid s_t = s, a_t = a)$. $\mathbb{P}^\pi_\mu$ and $\mathbb{E}^\pi_\mu$ denote probabilities and expectations under the process induced by $\pi$ and $\mu$.

Finally, the *Bellman operator* $\mathscr{B}^\pi_\mu V(s) \triangleq \rho_\mu(s) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}^\pi_\mu(s' \mid s) V(s')$ allows us to compute the value function recursively through $V^\pi_{\mu,t} = \mathscr{B}^\pi_\mu V^\pi_{\mu,t+1}$.[3]

**Bayesian RL (BRL).** In BRL, our subjective belief is represented as a probability measure $\beta$ over possible MDPs. We refer to the initial belief $\beta$ as the *prior distribution*. By interacting with the environment until time $t$, the agent obtains *data* $D_t = (s_1, a_1, r_1, \dots, s_t)$. This data is used to calculate a *posterior distribution* $\beta(\mu \mid D_t)$ that represents agent's current knowledge about the MDP.[4] For a given belief and an adaptive policy$\pi_\beta(s)$, we define the *Bayesian value function* to be:

$$\mathscr{V}^{\pi_\beta}_{\beta,t}(s) \triangleq \int_{\mathcal{M}} V^{\pi_\beta}_{\mu,t}(s) \, \mathrm{d}\beta(\mu). \tag{1}$$

The Bayesian value function is the expected value function under the distribution $\beta$. The *Bayes-optimal policy* achieves the Bayes-optimal value function $\mathscr{V}^*_{\beta,t}(s) = \sup_\pi \mathscr{V}^\pi_{\beta,t}$. Calculating $\mathscr{V}^\pi_{\beta,t}$ involves integrating $V^\pi_{\mu,t}$ for all $\mu$, while $\mathscr{V}^*_{\beta,t}$ typically requires exponential time. Information about the value function distribution can be a useful tool for constructing near-optimal policies, as well a way to compute risk-sensitive policies.

[1] In the partially-observable setting, the agent instead observes another variable dependent on the state.

[2] For infinite horizon problems this policy is also stationary.

[3] In the discounted setting, the value function converges to $V^\pi_\mu \equiv V^\pi_{\mu,1}$ as $T \to \infty$.

[4] This is expressible in closed form. When the MDP is discrete, a Dirichlet-product prior can be used, or when the MDP is continuous and the dynamics are assumed to be linear, a Gaussian-Wishart prior can be used [DeGroot, 1970]. Gaussian process inference can also be expressed in a closed-form but inference becomes approximate because the computational complexity scales quadratically with time.

### 1.2 Related Work and Our Contribution

We arrange related work in the following areas. Firstly, "model-free" methods that do not explicitly try to take into account the uncertainty about the underlying MDP. Secondly, "model-based" methods, that explicitly maintain a distribution on MDP models. Finally, we discuss approximations to the Bayes-optimal solution, before we outline our contributions.

**Model-free Bayesian Value Functions.** Bayesian value function distributions have been considered extensively in model-free Bayesian Reinforcement Learning (BRL). One of the first methods was Bayesian Q-learning [Dearden et al., 1998], which used a normal-gamma prior on the utility distribution. However, as i.i.d. utility samples cannot be obtained by bootstrapping from value function estimates, this idea had inherent flaws. Engel et al. [2003] developed a more sophisticated approach, the Gaussian Process Temporal Difference (GPTD) algorithm, which has a Gaussian process (GP) prior $\beta(V)$ on value functions. It then combines this with the likelihood function $\mathbb{P}(D \mid V) \propto \prod_{i=1}^{t} \exp\{-|V(s_i) - r_i - \gamma V(s_{i+1})|^2\}$. However, this makes the implicit assumption that the deterministic empirical MDP model is correct. Engel et al. [2005] tried to relax this assumption by allowing for correlation between sequentially visited states. Deisenroth et al. [2009] developed a dynamic programming algorithm with a GP prior on value functions and an explicit GP model of the MDP. Finally, Tang and Agrawal [2018] introduced VDQN, generalising such methods to Bayesian neural networks. The assumptions that these model-free Bayesian methods implicitly make about the MDP are hard to interpret, and we find the use of an MDP model independently of the value function distribution unsatisfactory. We argue that explicitly reasoning about the joint value function and MDP distribution is necessary to obtain a coherent Bayesian procedure. Unlike the above methods, we calculate a value function posterior $\mathbb{P}(V|D)$ while simultaneously taking into account uncertainty about the MDP.

**Model-based Bayesian Value Functions.** If a posterior over MDPs is available, we can calculate a distribution over value functions in two steps: a) sample from the MDP posterior and b) calculate the value function of each MDP. Essentially, this amounts to performing posterior sampling (PSRL, Strens [2000], Thompson [1933]) followed by policy evaluation. Dearden et al. [1999] suggested an early version of this approach that obtained approximate upper bounds on the Bayesian value function and sketched a Bellman-style update for performing it online. Posterior sampling approach was later used to obtain value function distributions in the discrete case by Dimitrakakis [2011] and in the continuous case by Osband et al. [2016]. Finally, O'Donoghue et al. [2018] derive bounds on the variance of the value function posterior. We instead focus on whether it is possible to compute complete value function distributions exactly or approximately through a *backwards induction* procedure. In particular, how can we obtain $\mathbb{P}(V_i|D)$ from $\mathbb{P}(V_{i+1}|D)$?

**Our Contribution.** We introduce *Inferential Induction*, a new Bayesian Reinforcement Learning (BRL) framework, which leads to a Bayesian form of backwards induction. Our framework allows Bayesian inference over value functions without any implicit assumption or approximation unlike its predecessors. The main idea is to calculate the conditional value function distribution at step $i$ from the value function distribution at step $i + 1$ analogouslto backwards induction for the expectation (Eq. (2)). Following this, we propose a simple marginalisation techniques and design an appropriate Monte Carlo approximation for it. We can combine this procedure with a policy optimisation mechanism. We use a Bayesian adaptation of dynamic programming for this and propose the *Bayesian backwards induction (BBI)* algorithm. Our experimental evaluation shows that BBI is competitive to the current state of the art. Inferential Induction framework provides the opportunity to further design more efficient algorithms of this family.

## 2 Inferential Induction

Given a prior belief $\beta$, and data $D_t = (s_1, a_1, r_1, \ldots, s_t)$ obtained by interaction of the agent with the MDP, we wish to calculate the value function distribution from step $t$ onwards, i.e. $\mathbb{P}^\pi_\beta(V_t \mid D_t)$ for a policy $\pi$ under the belief $\beta$, conditioned on the data $D_t$. The main idea for doing this to inductively calculate $\mathbb{P}^\pi_\beta(V_i \mid D_t)$ from $\mathbb{P}^\pi_\beta(V_{i+1} \mid D_t)$ for $i \in \{t, \ldots, T\}$ as follows:

$$\mathbb{P}^\pi_\beta(V_i \mid D_t) = \int_{\mathcal{V}} \mathbb{P}^\pi_\beta(V_i \mid V_{i+1}, D_t) \, \mathrm{d}\, \mathbb{P}^\pi_\beta(V_{i+1} \mid D_t). \tag{2}$$

Let $\psi_{i+1}$ be a (possibly approximate) representation of $\mathbb{P}^\pi_\beta(V_{i+1} \mid D_t)$. If we can calculate the above integral, then we can also obtain $\psi_i \approx \mathbb{P}^\pi_\beta(V_i \mid D_t)$ recursively, from time $T$ up to the current time step $t$. Then the estimation problem reduces to defining the term $\mathbb{P}^\pi_\beta(V_i \mid V_{i+1}, D_t)$ appropriately. In this paper, we describe a simple Monte Carlo method for solving the estimation problem and an approximate dynamic programming algorithm for optimising the policy within this framework.

**Integrating over $\mathbb{P}^\pi_\beta(\mu \mid V_{i+1}, D_t)$.** A simple idea for dealing with the term linking the two value functions is to directly marginalise over the MDP as follows:

$$\mathbb{P}^\pi_\beta(V_i \mid V_{i+1}, D_t) = \int_{\mathcal{M}} \mathbb{P}^\pi_\mu(V_i \mid V_{i+1}) \,\mathrm{d}\, \mathbb{P}^\pi_\beta(\mu \mid V_{i+1}, D_t). \tag{3}$$

This equality holds because given $\mu$, $V_i$ is uniquely determined by the policy $\pi$ and $V_{i+1}$ through the Bellman operator. However, it is crucial to note that $\mathbb{P}^\pi_\beta(\mu \mid V_{i+1}, D_t) \neq \mathbb{P}_\beta(\mu \mid D_t)$, as knowing the value function gives information about the MDP.[5] This is a crucial difference with "model-free" Bayesian value function methods, which effectively hide strong assumptions about the MDP when they perform value function inference. We expect that this approach would give superior results. The remaining computations are rather straightforward Monte Carlo approximations.

**The backwards induction step.** To calculate the previous-step distribution from next-step distribution, let us now combine the induction step in (2) with the marginalisation in (3). We also substitute an approximate representation $\psi_{i+1}$ for the next-step belief $\mathbb{P}(V_{i+1} \mid D_t)$, to obtain the following conditional probability measure on value functions:

$$\psi_i(B) \triangleq \mathbb{P}^\pi_\beta(V_i \in B | D_t) = \int_{\mathcal{V}} \int_{\mathcal{M}} \mathbb{1}\left\{\mathscr{B}^\pi_\mu V_{i+1} \in B\right\} \,\mathrm{d}\, \mathbb{P}^\pi_\beta(\mu | V_{i+1}, D_t) \,\mathrm{d}\psi_{i+1}(V_{i+1})$$

**Monte-Carlo approximation of the inner integral.** We now have to leave generalities and commit to some hard choices for defining and calculating $\mathbb{P}^\pi_\beta(\mu \mid V_{i+1}, D_t)$. Expanding this term we obtain, for subsets of MDPs $A \subseteq \mathcal{M}$, the following measure:

$$\mathbb{P}^\pi_\beta(\mu \in A \mid V_{i+1}, D_t) = \frac{\int_A \mathbb{P}^\pi_\mu(V_{i+1}) \,\mathrm{d}\beta(\mu \mid D_t)}{\int_{\mathcal{M}} \mathbb{P}^\pi_\mu(V_{i+1}) \,\mathrm{d}\beta(\mu \mid D_t)}, \tag{4}$$

since $\mathbb{P}^\pi_\mu(V_{i+1} \mid D_t) = \mathbb{P}^\pi_\mu(V_{i+1})$, as $\mu, \pi$ are sufficient for calculating $V_{i+1}$.

To compute $\mathbb{P}^\pi_\mu(V_{i+1})$, we can sample rollouts from an arbitrary starting state distribution $q$ and the policy $\pi$ for each sampled MDP $\mu$ and then marginalise over the resulting utility values $U$. In exact form, this is done through the integral:

$$\mathbb{P}^\pi_\mu(V_{i+1}) = \int_{\mathcal{S}} \mathrm{d}q(s) \int_{-\infty}^{\infty} \mathbb{P}^\pi_\mu(V_{i+1} \mid U, s) \,\mathbb{P}^\pi_\mu(U|s) \,\mathrm{d}U.$$

In order to understand the meaning of the term $\mathbb{P}^\pi_\mu(V_{i+1} \mid U, s)$, note that $V^\pi_{i+1,\mu}(s) = \mathbb{E}^\pi_\mu[U \mid s_{i+1} = s]$. Thus, a rollout $u$ from state $s$ gives us partial information about the value function. Finally, the starting state distribution $q$ is used to measure the goodness-of-fit, similarly to e.g. fitted Q-iteration[6].

As a design choice, we define the density of $V_{i+1}$ given a rollout sample $u_m$ in MDP $\mu$ from state $s_m \sim q$ to be a Gaussian with variance $\sigma^2$ . If we generate $N_\mu$ number of MDPs $\mu^{(j)} \sim \beta(\mu \mid D_t)$ and set:

$$w_{jk} \triangleq \frac{\sum_{m=1}^{n} e^{-\frac{\left|V^{(k)}_{i+1}(s_m) - u^j_m\right|^2}{2\sigma^2}}}{\sum_{j'=1}^{N_\mu} \sum_{m=1}^{n} e^{-\frac{\left|V^{(k)}_{i+1}(s_m) - u^{j'}_m\right|^2}{2\sigma^2}}}, \tag{5}$$

we get $\mathbb{E}[w_{jk}] = \mathbb{P}^\pi_\beta(\mu \in M \mid V_{i+1}, D_t)$. The weight $w_{jk}$ can be interpreted as a measure of how well the value function $V^{(k)}_{i+1}$ matches the rollouts obtained from $\mu^{(j)}$. This allows us to obtain value function samples for step $i$,

$$V^{(j,k)}_i \triangleq \mathscr{B}^\pi_{\mu^{(j)}} V^{(k)}_{i+1}, \tag{6}$$

[5] Assuming otherwise results in a mean-field approximation. See Sec. 2.2. in the arXiv version of the paper.

[6] As long as $q$ has full support over the state space, any choice should be fine. For discrete MDPs, we use a uniform distribution $q$ over states and sum over all of them, while we sample from $q$ in the continuous case.

each weighted by $w_{jk}$, leading to the following Monte Carlo estimate of the value function distribution at step $i$

$$\psi_i(B) = \frac{1}{N_V N_\mu} \sum_{k=1}^{N_V} \sum_{j=1}^{N_\mu} \mathbb{1}\left\{V_i^{(j,k)} \in B\right\} w_{jk}. \tag{7}$$

Here, $N_V$ is the number value function samples $V_{i+1}^{(k)}$. This ends the general description of the Monte Carlo method. Detailed design of an algorithm depends on the representation that we use for $\psi_i$ and whether the MDP is discrete or continuous.

Section 3 gives algorithmic details, while specifications of hyperparameters and distributions are given in Section 4.

## 3 Algorithms

Algorithm 1 is a concise description of the Monte Carlo procedure that we develop. At each time step $t$, the algorithm is called with the prior and data $D$ collected so far, and it looks ahead up to some lookahead factor $H$. [7] We instantiate it below for discrete and continuous state spaces.

**Algorithm 1** Policy Evaluation with Method 1

**Input:** Prior $\beta$, data $D$, lookahead $H$, discount $\gamma$, policy $\pi$, $N_\mu, N_V$.
Initialise $\psi_H$.
Sample $\hat{M} \triangleq \left\{\mu^{(j)} \;\middle|\; j \in [N_\mu]\right\}$ from $\beta(\mu \mid D)$.
**for** $i = H-1, \ldots, 1$ **do**
    Sample $V^{(k)} \sim \psi_{i+1}(\boldsymbol{v})$ for $k \in [N_V]$.
    Generate $n$ utility samples $u_m$
    Calculate $w_{jk}$ from (5) and $V_i^{(j,k)}$ from (6).
    Calculate $\psi_i$ from (7).
**end for**
**return** $\{\psi_i \mid i = 1, \ldots, H\}$

**Discrete MDPs.** When the MDPs are discrete, the algorithm is straightforward. Then the belief $\beta(\mu \mid D)$ admits a conjugate prior in the form of a Dirichlet-product for the transitions. In that case, it is also possible to use a histogram representation for $\psi_i$, so that it can be calculated by simply adding weights to bins according to (7). However, as a histogram representation is not convenient for a large number of states, we model using a Gaussian $\psi_t$. In order to do this, we use the sample mean and covariance of the weighted value function samples $V_i^{(j,k)}$:

$$\boldsymbol{m}_i = \frac{1}{N_V N_\mu} \sum_{k=1}^{N_V} \sum_{j=1}^{N_\mu} V_i^{(j,k)} w_{jk}, \boldsymbol{\Sigma}_i = \frac{1}{N_V N_\mu} \sum_{k=1}^{N_V} \sum_{j=1}^{N_\mu} (V_i^{(j,k)} - \boldsymbol{m}_i)(V_i^{(j,k)} - \boldsymbol{m}_i)^\top w_{jk}. \tag{8}$$

such that $\psi_i = \mathcal{N}(\boldsymbol{m}_i, \boldsymbol{\Sigma}_i)$ is a multivariate normal distribution.

**Continuous MDPs.** In the continuous state case, we obtain $\psi$ through fitted Q-iteration [c.f. Ernst et al., 2005]. For each action $a$ in a finite set, we fit a weighted linear model $Q_i(s_i, a) = s_i^T \omega_a + \epsilon_a$, where $s_i, \omega_a \in \mathbb{R}^d$ and $\epsilon_a \sim \mathcal{N}(0, \sigma_a^2)$. Thus, we obtain the sample mean and covariance to be:

$$\boldsymbol{m}_i^a = \frac{1}{N_V N_\mu} \sum_{k=1}^{N_V} \sum_{j=1}^{N_\mu} (s_i^{(j,k)})^T \omega_a, \boldsymbol{\Sigma}_i^a = \frac{\sigma_a^2}{N_V N_\mu} \sum_{k=1}^{N_V} \sum_{j=1}^{N_\mu} (Q_i^{(j,k)}(a) - \boldsymbol{m}_i^a)(Q_i^{(j,k)}(a) - \boldsymbol{m}_i^a)^\top w_{jk}. \tag{9}$$

In practice, we often use a feature map $\phi : \mathbb{R}^d \to \mathbb{R}^f$ for states.

[7] When the horizon $T$ is small, we can set $H = T - t$.

### 3.1 Bayesian Backwards Induction

We now construct a policy optimisation component to use with the inferential induction based policy evaluation and the aforementioned two approximation techniques. We use a dynamic programming algorithm that looks ahead $H$ steps, and at each step $i$ calculates a policy maximising the Bayesian expected utility in the next $i+1$ steps. We describe the corresponding pseudocode in Algorithm 2.

**Algorithm 2 Line 8: Discrete MDPs.** Just as in standard backwards induction, at each step, we can calculate $\pi_i$ by keeping $\pi_{i+1}, \ldots, \pi_H$ fixed:

$$\mathscr{Q}_i(s,a) \triangleq \mathbb{E}_\beta(U \mid s_i = s, a_i = a, D) = \int_{\mathcal{M}} \rho_\mu(s,a) + \sum_{s'} \mathbb{P}_\mu^{(j)}(s'|s,a) V_{\mu,i}^{\pi_{i+1},\ldots,\pi_H}(s')$$

$$\approx \sum_{j,k} \left[ \rho_{\mu^{(j)}}(s,a) + \sum_{s'} \mathbb{P}_\mu^{(j)}(s'|s,a) V_{i+1}^{(k)}(s') \right] \frac{w_{jk}}{N_\mu N_V}. \quad (10)$$

**Algorithm 2 Line 8: Continuous MDPs.** As we are using fitted Q-iteration, we can directly use the state-action value estimates. So we simply set $\mathscr{Q}_i(s,a) = \hat{Q}_i(s,a)$.

The $\mathscr{Q}_i$ estimate is then used to select actions for every state. We set $\pi_i(a|s) = 1$ for $a = \arg\max \mathscr{Q}_i(s,a)$ (Line 2.9) and calculate the value function distribution (Lines 2.10 and 2.11) for the partial policy $(\pi_i, \pi_{i+1}, \ldots, \pi_H)$.

**Algorithm 2** Bayesian Backwards Induction (BBI) with Method 1

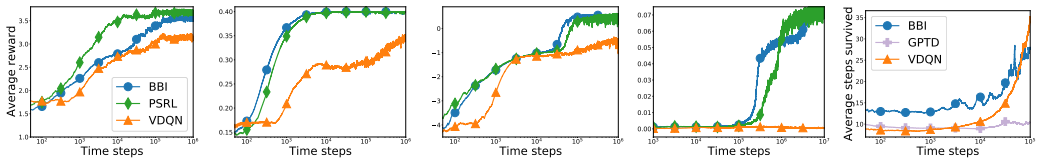
1: **Input:** Prior $\beta$, data $D$, lookahead $H$, discount $\gamma$, $N_\mu$, $N_V$.
2: Initialise $\psi_i$.
3: Sample $\hat{M} \triangleq \left\{ \mu^{(j)} \;\middle|\; j \in [N_\mu] \right\}$ from $\beta(\mu \mid D)$.
4: **for** $i = H-1, \ldots, 1$ **do**
5: Sample $V^{(k)} \sim \psi_{i+1}(\boldsymbol{v})$ for $k \in [N_V]$.
6: Generate $n$ utility samples $u_i$
7: Calculate $w_{jk}$ from (5).
8: Calculate $\mathscr{Q}_i$ from (10) or fitted Q-iteration.
9: Set $\pi_i(a|s) = 1$ for $a \in \arg\max \mathscr{Q}_i(s,a)$.
10: Calculate $w_{jk}$ from (5) and $V_i^{(j,k)}$ from (6) with policy $\pi_i$: $V_i^{(j,k)} \triangleq \mathscr{B}_{\mu^{(j)}}^{\pi_i} V_{i+1}^{(k)}$.
11: Calculate $\psi_i$ from (8) or (9).
12: **end for**
13: **return** $\pi = (\pi_1, \ldots, \pi_H)$.

## 4 Experimental Analysis

For performance evaluation, we compare Bayesian Backwards Induction (BBI, Alg. 2) with exploration by distributional reinforcement learning [VDQN, Tang and Agrawal, 2018]. We also compare BBI with posterior sampling [PSRL, Strens, 2000, Thompson, 1933] for the discrete MDPs and with Gaussian process temporal difference [GPTD, Engel et al., 2003] for the continuous MDPs. First, we describe the experimental setup and the priors used for implementation. In Section 4.1, we analyse the results obtained for different environments in terms of average reward obtained over time. Comparisons with Multi-MDP Backwards Induction [Dimitrakakis, 2011, MMBI], Bayesian Sparse Sampling [Wang et al., 2005, BSS] and Bayesian Q-Learning [Dearden et al., 1998, BQL] are available in the appendix.

**Parameters.** We run the algorithms for the infinite-horizon formulation of value function with discount factor $\gamma = 0.99$. Each algorithm updates its policy at steps $t_k = \frac{k(k+1)}{2}$, where $k$ increases as $1, 2, \ldots$. This sequence of $t_k$'s causes a total of $1413$ updates in $10^6$ steps. We set the lookahead $H$ to 100 and 20 for discrete and continuous MDPs respectively. More implementation details can be found in the appendix.

**Prior.** For discrete MDPs, we use Dirichlet $Dir(\alpha)$ priors over each of the transition probabilities $\mathbb{P}(s'|s,a)$. The prior parameter $\alpha$ for each transition is set to $0.5$. We use separate NormalGamma




(a) NChain (b) DoubleLoop (c) LavaLake $10 \times 10$ (d) Maze (e) Inverted Pendulum

Figure 1: Evolution of average reward for NChain, DoubleLoop, LavaLake $10 \times 10$, Maze and InvertedPendulum. The results are averaged over 30 runs of length $10^5$ for the continuous Inverted Pendulum. The discrete environment runs are of length $10^6$ ($10^6$ for Maze) with 50 runs for NChain and DoubleLoop and 30 runs for Maze and LavaLake $10 \times 10$. The runs are exponentially smoothed with a half-life 1000 and 2500 before averaging for the discrete and continuous runs respectively.

$\mathcal{NG}(\mu, \kappa, \alpha, \beta)$ priors for each of the reward distributions $\mathbb{P}(r|s, a)$. We set the prior parameters to $[\mu_0, \kappa_0, \alpha_0, \beta_0] = [0, 1, 1, 1]$.

For continuous MDPs, we use factored Bayesian Multivariate Regression [Minka, 2001] models as priors over transition kernels and reward functions for the continuous environments. This implies that the transition kernel $\mathbb{P}(s'|s, a)$ and reward kernel $\mathbb{P}(r|s, a)$ modelled as $\mathcal{N}(A_a^{\text{Trans}} s, \Sigma)$ and $\mathcal{N}(A_a^{\text{Reward}} s, \sigma^2)$. $\Sigma$ is sampled from inverse Wishart distribution with corresponding $d \times d$ dimensional scale matrix, while $\sigma$ is sampled from inverse Gamma with prior parameters $(\frac{1}{2}, \frac{1}{2})$. For transitions, we set the prior parameters to $\boldsymbol{\Psi}_0 = 0.001\boldsymbol{I}$ and degrees of freedom $\nu_0 = \text{rank}(\boldsymbol{\Psi}_0)$.

**Feature Map.** For InvertedPendulum we use the 10 dimensional feature map $\phi : \mathbb{R}^d \to \mathbb{R}^f$ used by Lagoudakis and Parr [2003] for BBI and GPTD such that the modelling becomes $\mathbb{P}(s' \mid \phi(s), a)$ and $\mathbb{P}(r \mid \phi(s), a)$. VDQN works directly on the underlying state $s$ as it uses a neural network. We also add a regularizing term, $\lambda \boldsymbol{I}$, $\lambda = 0.01$ to the diagonal of the fitted Q-iteration fit. The choice of state distribution $q(s)$ is of utmost importance in continuous environments. In the continuous environment, we experimented with a few options, trading of sampling states from our history, sampling from the starting configuration of the environment and sampling from the full support of the state space.

**Description of Environments** We evaluate the algorithms on four discrete (NChain [Strens, 2000], DoubleLoop [Strens, 2000], LavaLake [Leike et al., 2017] and Maze [Strens, 2000]) and one continuous environment (InvertedPendulum Lagoudakis and Parr [2003]). See the appendix for more details.

### 4.1 Experimental Results

The following experiments are intended to show that the general methodological idea is indeed sound, and can potentially lead to high performance algorithms. More results, comparative experiments, as well as tests of convergence, are provided in the appendix.

Figures 1a, 1b, 1c, 1d illustrate the evolution of average reward for BBI, PSRL and VDQN on the discrete MDPs.[8] BBI is competitive with PSRL, which has good exploration properties, while VDQN generally performs worse. Figure 1e shows a comparison with state-of-the art algorithms, such as VDQN and GPTD. Our algorithm is competitive, and in particular performs much better than GPTD, while it performs similarly to VDQN, which is slightly worse initially and slightly better later in terms of average steps survived. This performance could partially be explained by the use of a linear value function $Q(\phi(s), a)$, in contrast to VDQN which uses a neural network.

### 4.2 Comparison of Computation Times

In Table 1 we can see the timing results for the different algorithms when run on Intel(R) Core(TM) i9-9900K CPU @ 3.60GHz. Here we can see that PSRL is faster and scales better due to the simplicity of using a single MDP. It is probable that BBI could obtain speedups through improve sampling

[8] More detailed figures with error bars and percentiles as well as comparisons with MMBI, BSS and BQL are available in the appendix.

Table 1: CPU-time (in seconds) used for each algorithm. Chain has 5 states and is run for 5000 steps (100 policy updates) while Maze has 264 states and is run for 20 steps (5 policy updates).

| | BBI | PSRL | MMBI |
|---|---|---|---|
| Chain | 14 | 5 | 19 |
| Maze | 921 | 6 | 256 |

methods, alternative kernels, or a different marginalisation. We also compared it with MMBI, which samples of multiple MDP models from the posterior and performs approximate dynamic programming to obtain a policy, with the main difference being that MDPs are sampled only once, instead of at every backwards induction step.

## 5 Discussion and Future Work

**New Insights.** We offered a new perspective on Bayesian value function estimation. The central idea is to calculate *the conditional value function distribution* $\mathbb{P}^\pi_\beta(V_i \mid V_{i+1}, D)$ using the data and to apply it inductively for computing *the marginal value function distribution* $\mathbb{P}^\pi_\beta(V_i \mid D)$. We then designed a straightforward Monte Carlo approximation and combined it with a suitable policy optimisation mechanism and showed that it can be competitive with the state of the art.

In order to place it in context, standard backwards induction (i.e. value iteration) calculates $V_i = \mathscr{B}^\pi_\mu V_{i+1}$ and distributional reinforcement learning methods calculate $\mathbb{P}^\pi_\mu(U_i \mid U_{i+1})$, both for a given, underlying MDP. In an RL setting, this can be replaced either through stochastic approximation, such as Q-learning, or through an explicit empirical model.

Inferential Induction differs from existing Bayesian value function methods, which essentially cast the problem into regression. For example, GPTD [Engel et al., 2003] can be written as Bayesian inference with a GP prior over value functions and a data likelihood that uses a deterministic empirical model of the MDP. While this can be relaxed by using temporal correlations as in [Engel et al., 2005], the fundamental problem remains. Even though such methods have practical value, we show that Bayesian estimation of value functions requires us to explicitly think about the MDP distribution as well.

**Algorithm Design.** We use specific approximations for discrete and continuous MDPs to propose the Bayesian Backwards Induction (BBI) algorithm. The algorithm we developed from this family appears promising, as we are able to outperform methods that implicitly use the empirical model of the MDP, such as GPTD. This shows that the inference is inherently sound. We see that BBI is also competitive with state-of-the art methods like PSRL, and it generally outperforms algorithms relying on approximate inference, such as VDQN.

**Limitations.** We have observed that the particular method, BBI, that we have developed requires a significant number of samples for it to obtain a good performance, which makes it inherently slow. Our timing analysis in Table 1 shows that methods like PSRL may work significantly faster as they have to keep track of only one MDP and are also simpler to implement. It is probable that BBI could obtain speedups from using sparse posteriors or sampling from state space in the importance sampling step of the algorithm.

For the continuous environments, we have used weighted linear model and linear priors for updating and tracking the value function distribution. Thus, we rely on pre-defined features (e.g. the features from Lagoudakis and Parr [2003]) which are problem specific, not always efficient, and can be replaced more efficient functional approximators, such as neural networks (similar to Tang and Agrawal [2018]).

Since our aim has been to propose the framework, we have emphasised one policy evaluation and used simpler methods, such as Q-update and fitted Q-iteration, for policy update. It is possible to incorporate this method with policy gradient methods to design more efficient algorithms. This point shows the flexibility of our approach as a framework and weakness as a specifically designed algorithm, which is a topic of future investigation.

## Acknowledgments and Disclosure of Funding

Thank you to Nikolaos Tziortziotis for his useful discussions. This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. The experiments were partly performed on resources at Chalmers Centre for Computational Science and Engineering (C3SE) provided by the Swedish National Infrastructure for Computing (SNIC).

## References

Richard Dearden, Nir Friedman, and Stuart Russell. Bayesian Q-learning. In *Aaai/iaai*, pages 761–768, 1998.

Richard Dearden, Nir Friedman, and David Andre. Model based Bayesian exploration. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 150–159, 1999.

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

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

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

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

Yaakov Engel, Shie Mannor, and Ron Meir. Bayes meets Bellman: The Gaussian process approach to temporal difference learning. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 154–161, 2003.

Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(Apr):503–556, 2005.

Nicolas Fournier and Arnaud Guillin. On the rate of convergence in Wasserstein distance of the empirical measure. *Probability Theory and Related Fields*, 162(3-4):707–738, 2015.

Mohammad Ghavamzadeh, Shie Mannor, Joelle Pineau, and Aviv Tamar. Bayesian reinforcement learning: A survey. *Foundations and Trends in Machine Learning*, 8(5-6):359–483, 2015.

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

Jan Leike, Miljan Martic, Victoria Krakovna, Pedro A Ortega, Tom Everitt, Andrew Lefrancq, Laurent Orseau, and Shane Legg. Ai safety gridworlds. *arXiv preprint arXiv:1711.09883*, 2017.

Thomas P. Minka. Bayesian linear regression. Technical report, Microsoft research, 2001.

Brendan O'Donoghue, Ian Osband, Remi Munos, and Volodymyr Mnih. The uncertainty bellman equation and exploration. In *International Conference on Machine Learning*, pages 3836–3845, 2018.

Ian Osband, Benjamin Van Roy, and Zheng Wen. Generalization and exploration via randomized value functions. In *ICML*, 2016.

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

Malcolm Strens. A Bayesian framework for reinforcement learning. In *ICML 2000*, pages 943–950, 2000.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Yunhao Tang and Shipra Agrawal. Exploration by distributional reinforcement learning. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 2710–2716. AAAI Press, 2018.

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. Vlassis, M. Ghavamzadeh, S. Mannor, and P. Poupart. *Reinforcement Learning*, chapter Bayesian Reinforcement Learning, pages 359–386. Springer, 2012.

Tao Wang, Daniel Lizotte, Michael Bowling, and Dale Schuurmans. Bayesian sparse sampling for on-line reward optimization. In *Proceedings of the 22nd international conference on Machine learning*, pages 956–963, 2005.

# A Description of Environments

**NChain.** This is a discrete stochastic MDP with 5 states, 2 actions [Strens, 2000]. Taking the first action returns a reward 2 for all states and transitioning to the first state. Taking the second action returns 0 reward in the first four states (and the state increases by one) but returns 10 for the fifth state and the state remains unchained. There is a probability of slipping of $0.2$ with which its action has the opposite effect. This environment requires both exploration and planning to be solved effectively and thus acts as an evaluator of posterior estimation, efficient performance and effective exploration.

**DoubleLoop.** This is a slightly more complex discrete deterministic MDP with with two loops of states [Strens, 2000]. Taking the first action yields traversal of the right loop and a reward 1 for every 5 state traversal. Taking the second action yields traversal of the left loop and a reward 2 for every 5 state traversal. This environment acts as an evaluator of efficient performance and effective exploration.

**LavaLake.** This is a stochastic grid world [Leike et al., 2017] where every state gives a reward of -1, unless you reach the goal, in which case you get 50, or fall into lava, where you get -50. We tested on the $5 \times 7$ and a $10 \times 10$ versions of the environment. The agent moves in the direction of the action (up,down,left,right) with probability 0.8 and with probability 0.2 in a direction perpendicular to the action.

**Maze.** This is a grid world with four actions (ref. Fig. 3 in [Strens, 2000]). The agent must obtain 3 flags and reach a goal.There are 3 flags throughout the maze and upon reaching the goal state the agent obtains a reward of 1 for each flag it has collected and the environment is reset. Similar to LavaLake, the agent moves with probability 0.9 in the desired direction and 0.1 in one of the perpendicular directions. The maze has 33 reachable locations and 8 combination of obtained flags for a total of 264 states.

**InvertedPendulum.** To extend our results for the continuous domain we evaluated our algorithm in a classical environment described in [Lagoudakis and Parr, 2003]. The goal of the environment is to stabilize a pendulum and to keep it from falling. If the pendulum angle $\theta$ falls outside $[\frac{-\pi}{2}, \frac{\pi}{2}]$ then the episode is terminated and the pendulum returned to its starting configuration. The state dimensionality is a tuple of the pendulum angle as well as its angular velocity, $\dot{\theta}$, $s = (\theta, \dot{\theta})$. The environment is considered to be completed when the pendulum has been kept within the accepted range for $3000$ steps. For further environment details, see [Lagoudakis and Parr, 2003].

# B Additional Results

Here we present some experiments that examine the performance of inferential induction in terms of value function estimation, inference and utility obtained. For the latter we present additional results against other algorithms, as well as one new continuous environment.

## B.1 Bayesian Value Function Estimation

In this experiment, we evaluate the Bayesian (i.e. mean) value function of the proposed algorithm (BBI) with respect to the upper bound on the Bayes-optimal value function. The upper bound is calculated from $\int_{\mathcal{M}} \max_\pi V^\pi_\mu d\beta(\mu \mid D)$. We estimate this bound through $100$ MDP samples for NChain. We plot the time evolution of our value function and the simulated Bayes bound in Figure 2 for $10^5$ steps. We observe that this is becomes closer to the upper bound as we obtain more data.

## B.2 Value Function Distribution Estimation

Here we evaluate whether inferential induction based policy evaluation (Alg. 1) results in a good approximation of the actual value function posterior. In order to evaluate the effectiveness of estimating the value function distribution using inferential induction (Alg. 1), we compare it with the *Monte Carlo* distribution and the mean MDP. We compare this for posteriors after 10, 100 and 1000 time steps, obtained with a fixed policy in NChain that visits all the states, in Figure 3 for 5 runs of Alg. 1. The fixed policy selects the first action with probability 0.8 and the second action with probability 0.2. The Monte Carlo estimate is done through $1000$ samples of the value function vector ($\gamma = 0.99$). This shows that the estimate of Alg. 1 reasonably captures the uncertainty in the true

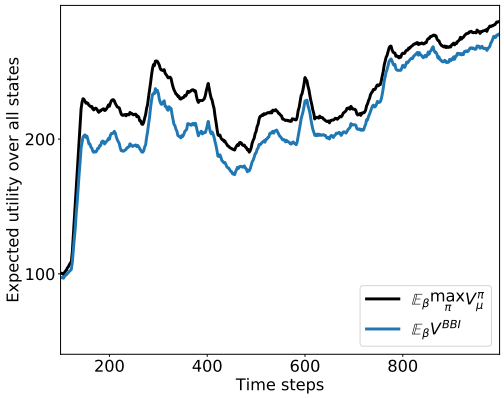
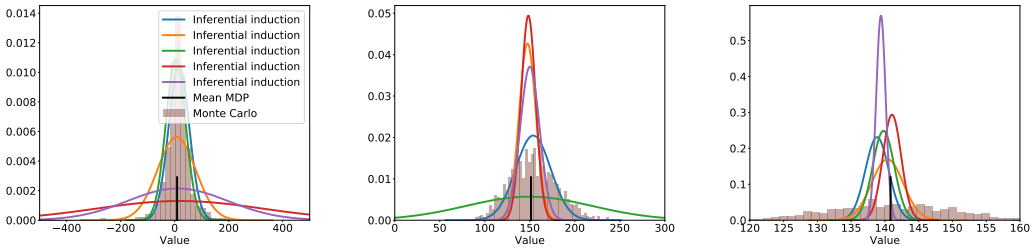



Figure 2: Comparisons of the achieved value functions of BBI with the upper bound on Bayes-optimal value functions. Upper bound and BBI are calculated from 100 MDPs and plotted for $10^5$ time steps.

distribution. For this data, we also compute the Wasserstein distance [Fournier and Guillin, 2015] between the true and the estimated distributions at the different time steps as can be found in Table 2. There we can see that the distance to the true distribution decreases over time.




(a) $T = 10$ (b) $T = 100$ (c) $T = 1000$

Figure 3: Comparison of value function posteriors obtained by inferential induction and Monte Carlo evaluations at different time steps for a fixed policy. We plot for five runs of inferential induction at each time step. The value of the mean MDP is shown by a vertical line.

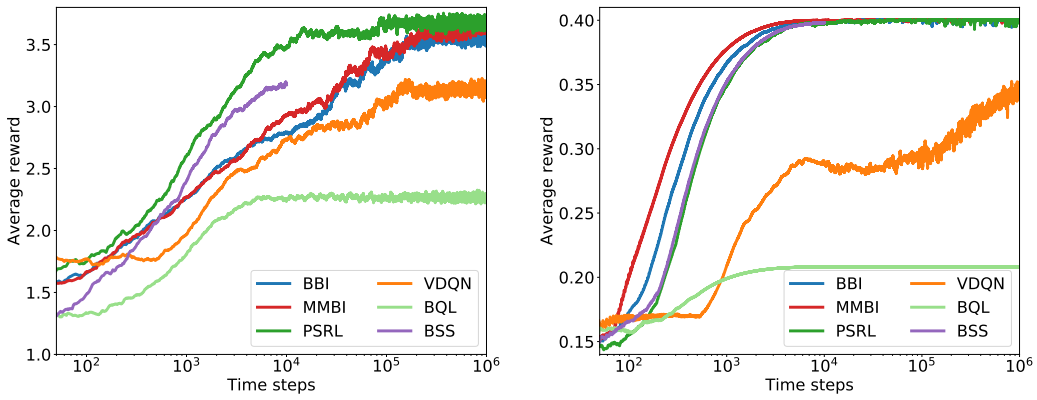
Table 2: Wasserstein distance to the true distribution of the value function, for Alg. 1 and the mean MDP model, for NChain. For Inferential Induction, the distances are averaged over 5 runs. The distances correspond to the plots in Figure 3.

| Time steps | Inf. Induction | Mean MDP |
|---|---|---|
| 10 | 22.80 | 30.69 |
| 100 | 16.41 | 17.90 |
| 1000 | 4.18 | 4.27 |

### B.3 Further Comparisons on Discrete Environments: MMBI, BSS, BQL

We have also run additional experiments with other Bayesian algorithms. In particular, here we show comparisons with MMBI [Dimitrakakis, 2011], BSS [Wang et al., 2005] and BQL Dearden et al. [1998].

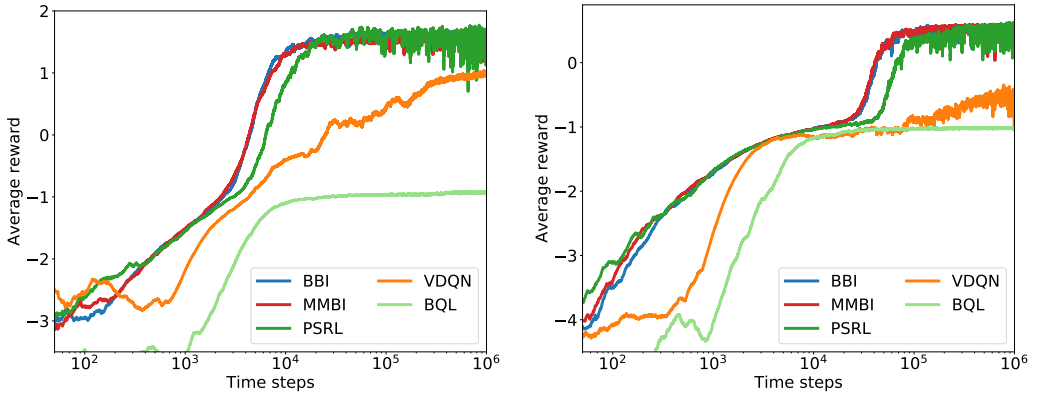
As can be seen in Figures 4 to 6, BBI performs similarly to to MMBI and PSRL. This is to be expected, as the optimisation algorithm used in MMBI is close in spirit to BBI, with only the inference being different. In particular, this algorithm takes $k$ MDP samples from the posterior, and




(a) NChain (b) DoubleLoop

Figure 4: Evolution of average reward for NChain and DoubleLoop environments, averaged over 50 runs of length $10^6$ for each algorithm. For computational reasons BSS is only run for $10^4$ steps. The runs are exponentially smoothened with a half-life 1000 before averaging.

then performs backward induction in all the MDP simultaneously to obtain a Markov policy. In turn, PSRL can be seen as a special case of MMBI with just one sample. This indicates that the BBI inference procedure is sound. The near-optimal Bayesian approximation performs slightly worse in this setting, perhaps because it was not feasible to increase the planning horizon sufficiently.[9] Finally, the less principled approximations, like VDQN and BQL do not manage to have a satisfactory performance in these environments.




(a) LavaLake $5 \times 7$ (b) LavaLake $10 \times 10$

Figure 5: Evolution of average reward for $5 \times 7$ and $10 \times 10$ LavaLake environments. The results are averaged over 20 and 30 runs respectively with a length of $10^6$ for each algorithm. The runs are exponentially smoothened with a half-life 1000 before averaging.

### B.4 Analysis of Variation in Performance

In Figures 7 to 11, we illustrate the variation in performance of the different algorithms for each environment. The black lines illustrate the standard error and the 5th and 95th percentile performance is highlighted. The results indicate that BSS is the most stable algorithm, followed by BBI, MMBI and PSRL, which nevertheless have better mean performance. VDQN is quite unstable, however.

[9] For computational reasons we used a planning horizon of two with four next state samples and two reward samples in each branching step. We hope to be able to run further experiments with BSS at a later point.

# C Implementation Details

In this section we discuss some additional implementation details, in particular how exactly we performed the rollouts and the selection of some algorithm hyperparameters, as well as some sensitivity analysis.

## C.1 Computational Details of Rollouts

To speed up the computation of rollouts, we have used three possible methods that essentially bootstrap previous rollouts or use value function samples:

$$u_t^{\mu,\pi_t}(s) = r(s,a) + \gamma u_{t+1}^{\mu,\pi_{t+1}}(s') \tag{11}$$

$$u_t^{\mu,\pi_t}(s) = \sum_{s'} r(s,a) + \gamma P(s'|s,a) u_{t+1}^{\mu,\pi_{t+1}}(s') \tag{12}$$

$$u_t^{\mu,\pi_t}(s) = \sum_{s'} r(s,a) + \gamma P(s'|s,a) V_{t+1}(s') \tag{13}$$

where $V_{t+1} \sim \psi_{t+1}$. In experiments, we have found no significant difference between them. All results in the paper use the formulation in (13).

## C.2 Hyperparameters

For the experiments, we use the following hyperparameters.

We use 10 MDP samples, a planning horizon T of 100, $\gamma = 0.99$ and we set the variance of the Gaussian to be $\sigma^2 = V_{\text{span}}^2 10^{-4}$, where $V_{\text{span}}$ is the span of possible values for each environment (obtained assuming maximum and minimum reward). We use Eq. 13 for rollout computation with 10 samples from $V_{t+1}$ and 50 samples from $V_t$ (20 for LavaLake $10 \times 10$ and Maze). If the weights obtained in (5) are numerically unstable we attempt to resample the value functions and then double $\sigma$ until it works (but is reset to original value when new data is obtained). This is usually only a problem when very little data has been obtained.

**Varying Horizon $T$.** In order to check the sensitivity on the choice of horizon $T$, we perform a sensitivity analysis with $T = 10, 20, 50, 100$. In Figure 12, we can see that varying the horizon has a very small impact for NChain and Maze environments.

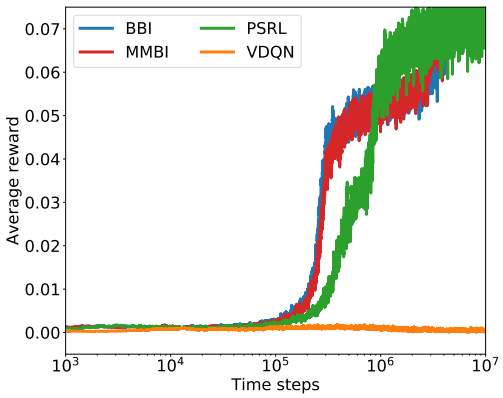
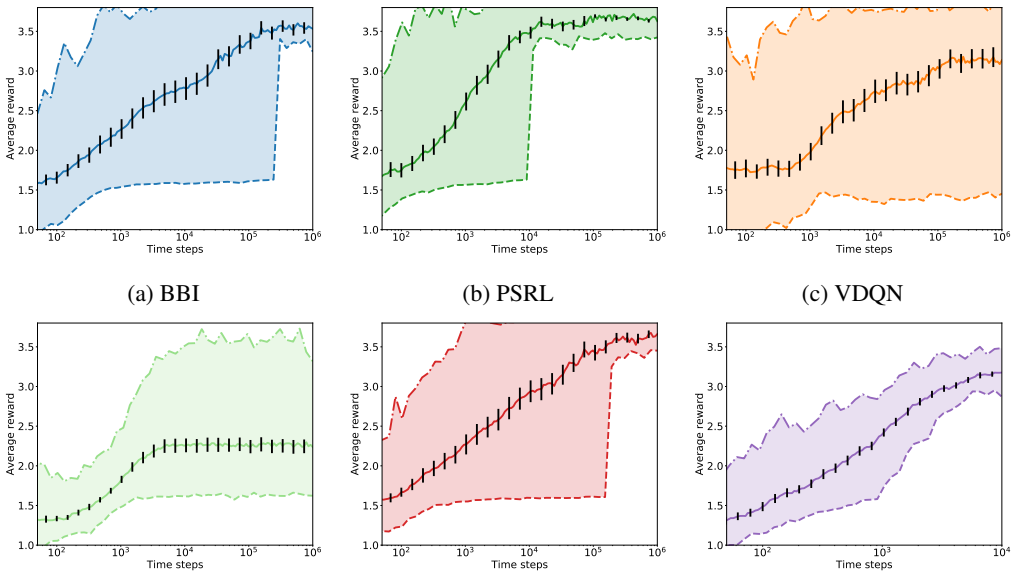


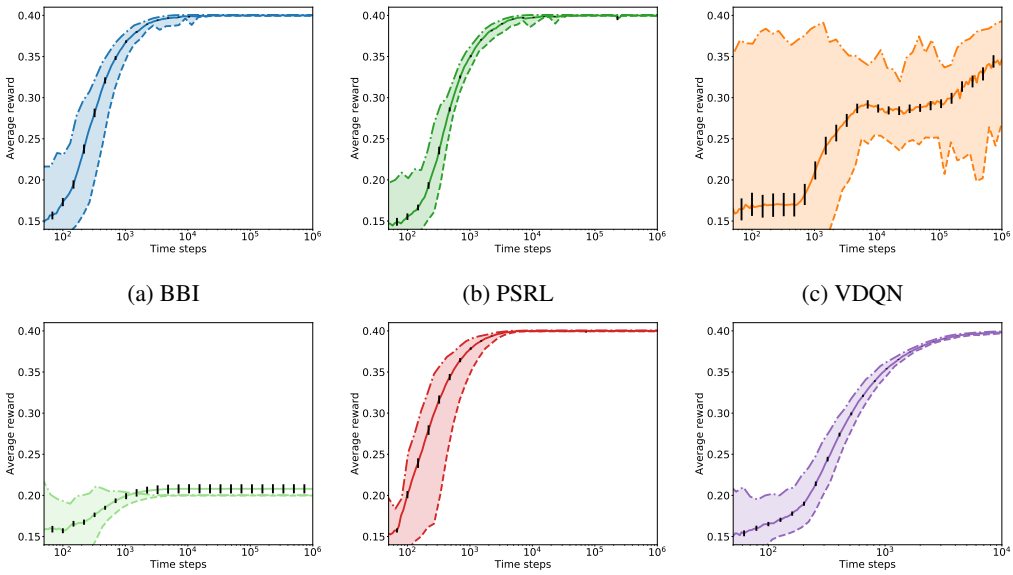

Figure 6: Evolution of average reward for the Maze environment. The results are averaged over 30 runs with a length of $10^6$ for each algorithm. The runs are exponentially smoothened with a half-life 1000 before averaging.




(a) BBI (b) PSRL (c) VDQN

(d) BQL (e) MMBI (f) BSS

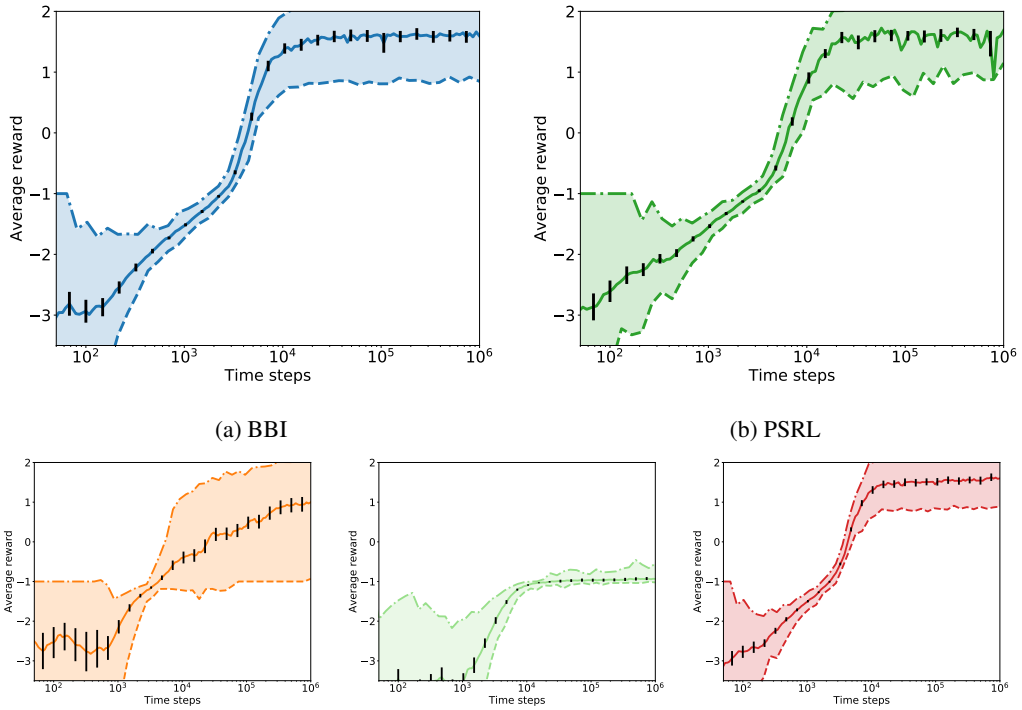
Figure 7: Evolution of average reward for NChain environment with runs of length $10^6$ for each algorithm. For computational reasons BSS is only run for $10^4$ steps. The runs are exponentially smoothened with a half-life 1000. The mean as well as the 5th and 95th percentile performance is shown for each algorithm and the standard error is illustrated with black lines.




(a) BBI (b) PSRL (c) VDQN

(d) BQL (e) MMBI (f) BSS

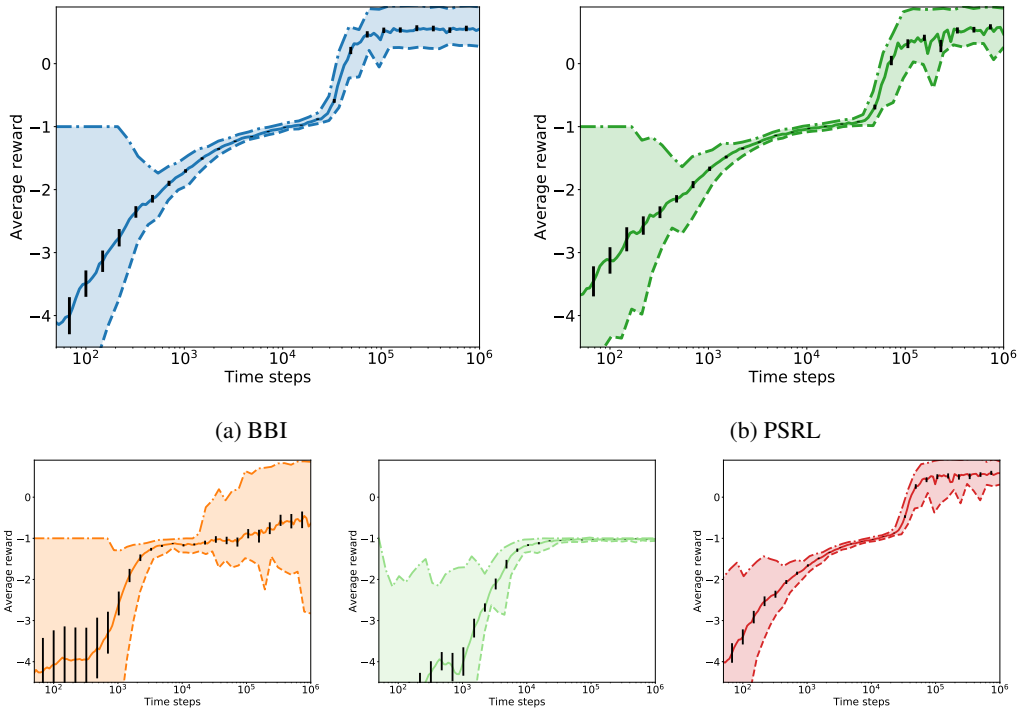
Figure 8: Evolution of average reward for DoubleLoop environment with runs of length $10^6$ for each algorithm. For computational reasons BSS is only run for $10^4$ steps. The runs are exponentially smoothened with a half-life 1000. The mean as well as the 5th and 95th percentile performance is shown for each algorithm and the standard error is illustrated with black lines.




(a) BBI (b) PSRL

(c) VDQN (d) BQL (e) MMBI

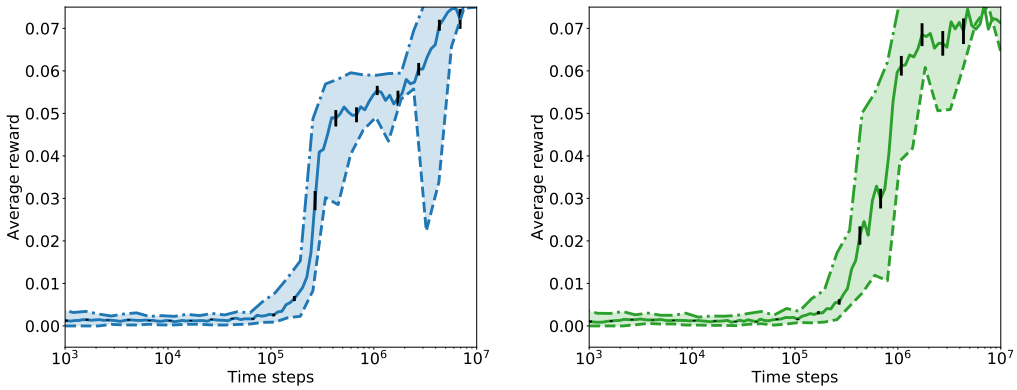
Figure 9: Evolution of average reward for LavaLake $5 \times 7$ environment with runs of length $10^6$ for each algorithm. The runs are exponentially smoothened with a half-life $1000$. The mean as well as the 5th and 95th percentile performance is shown for each algorithm and the standard error is illustrated with black lines.




(a) BBI (b) PSRL

(c) VDQN (d) BQL (e) MMBI

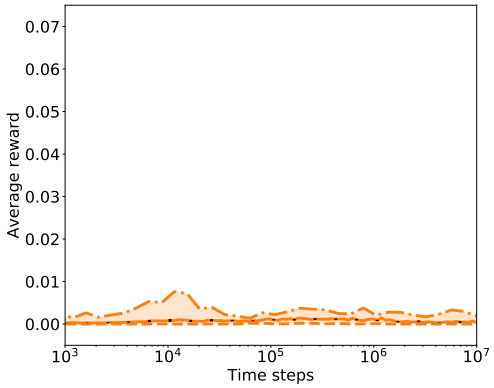
Figure 10: Evolution of average reward for LavaLake $10 \times 10$ environment with runs of length $10^6$ for each algorithm. The runs are exponentially smoothened with a half-life 1000. The mean as well as the 5th and 95th percentile performance is shown for each algorithm and the standard error is illustrated with black lines.



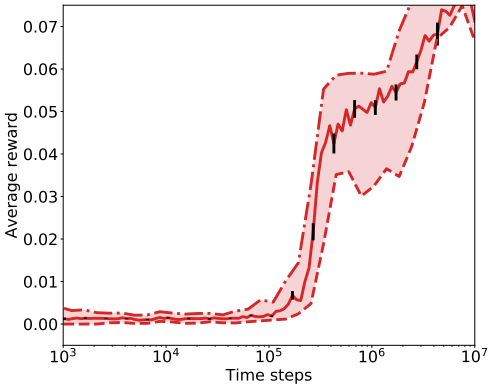

(a) BBI




(b) PSRL

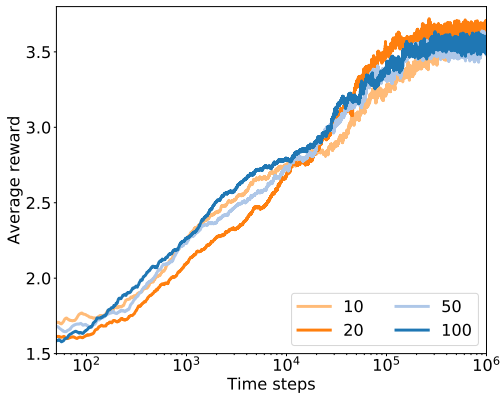



(c) VDQN

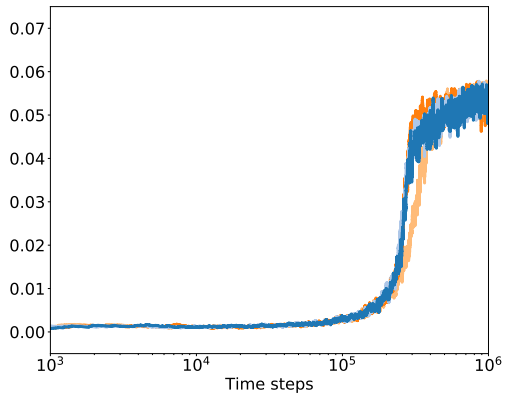



(d) MMBI

Figure 11: Evolution of average reward for Maze environment with runs of length $10^7$ for each algorithm. The runs are exponentially smoothened with a half-life 1000. The mean as well as the 5th and 95th percentile performance is shown for each algorithm and the standard error is illustrated with black lines.




(a) NChain




(b) Maze

Figure 12: Illustration of the impact of varying the horizon T in BBI. The results are averaged over 50 and 30 runs respectively with a length of $10^6$ for each algorithm. The runs are exponentially smoothened with a half-life 1000 before averaging.