Inferential Induction: A Novel Framework for Bayesian Reinforcement Learning

Downloaded from: https://research.chalmers.se, 2021-11-13 14:00 UTC

Citation for the original published paper (version of record):
Inferential Induction: A Novel Framework for Bayesian Reinforcement Learning
Proceedings of Machine Learning Research, 137: 43-52

N.B. When citing this work, cite the original published paper.
Inferential Induction: A Novel Framework for Bayesian Reinforcement Learning

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) [Ghaemzadeh 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
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 $P$ and $E$ 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 $S$ and action space $A$ is equipped with a reward distribution $P_\mu(r \mid s)$ with corresponding expectation $\mu(s)$ and a transition kernel $P_\mu(s' \mid s, a)$ for states $s, s' \in S$ and actions $a \in A$.

At time $t$, the agent observes 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 $U_t \equiv \sum_{k=t}^{T} \gamma^{k-t} r_k$, i.e. the sum of future rewards $r_k$. 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 $\pi = (\pi_1, \ldots, \pi_t, \ldots)$ that takes an action $a_t$ at time $t$ with probability $\pi_t(a_t \mid s_t, r_{t-1}, a_{t-1}, s_{t-1}, \ldots, r_1, a_1, s_1)$. 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 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 $V_{\mu, t}^\pi(s) \equiv E_\mu(U_t \mid s_t = s)$ and the corresponding state-action (or Q-)value function is $Q_{\mu, t}^\pi(s, a) \equiv E_\mu(U_t \mid s_t = s, a_t = a)$. $P_\mu$ and $E_\mu$ denote probabilities and expectations under the process induced by $\pi$ and $\mu$.

Finally, the Bellman operator $B_t^\pi V(s) \equiv \mu(s) + \gamma \sum_{s' \in S} P_\mu(s' \mid s)V(s')$ allows us to compute the value function recursively through $V_{\mu, t}^\pi \approx B_t^\pi V_{\mu, t+1}^\pi$.

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 $D_t = (s_t, a_1, r_1, \ldots, s_t)$. This data is used to calculate a posterior distribution $\beta(\mu \mid D_t)$ that represents agent’s current knowledge about the MDP.

For a given belief and an adaptive policy $\mu'_\beta$, we define the Bayesian value function to be:

$$V_{\beta, t}^\pi(s) \equiv \int_{M} V_{\mu, t}^\pi(s) d\beta(\mu).$$

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

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

2For infinite horizon problems this policy is also stationary.

3In the discounted setting, the value function converges to $V_{\mu, 1}^\pi \equiv V_{\mu, 1}^\pi$ as $T \to \infty$.

4This 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. [2005] 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

\[
\mathbb{P}(D \mid V) \propto \prod_{t=1}^T \exp\left\{-|V(s_t) - r_t - \gamma V(s_{t+1})|^2\right\}.
\]

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 [2016] 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 \(\mathbb{P}(V \mid D)\) 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 [1993]) 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 \(\mathbb{P}(V_t \mid D)\) from \(\mathbb{P}(V_{t+1} \mid D)\)?

**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 \(i+1\) analogous to 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. \(\mathbb{P}_\beta(V_t \mid D_t)\) for a policy \(\pi\) under the belief \(\beta\), conditioned on the data \(D_t\). The main idea for doing this is to inductively calculate \(\mathbb{P}_\beta(V_t \mid D_t)\) from \(\mathbb{P}_\beta(V_{t+1} \mid D_t)\) for \(i \in \{t, \ldots, T\}\) as follows:

\[
\mathbb{P}_\beta(V_t \mid D_t) = \int_{V_t} \mathbb{P}_\beta(V_t \mid V_{t+1}, D_t) \mathbb{d}\mathbb{P}_\beta(V_{t+1} \mid D_t).
\]  

(2)
Let $\psi_{i+1}$ be a (possibly approximate) representation of $P_\beta^i(V_{i+1} \mid D_t)$. If we can calculate the above integral, then we can also obtain $\psi_i \approx P_\beta^i(V_i \mid D_t)$ recursively, from time $T$ up to the current time step $t$. Then the estimation problem reduces to defining the term $P_\beta^i(V_i \mid V_{i+1}, D_t)$ 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 $P_\beta^i(\mu \mid V_{i+1}, D_t)$.** A simple idea for dealing with the term linking the two value functions is to directly marginalise over the MDP as follows:

$$P_\beta^i(V_i \mid V_{i+1}, D_t) = \int_M P_\mu^i(V_i \mid V_{i+1}) d P_\beta^i(\mu \mid V_{i+1}, D_t).$$

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 $P_\beta^i(\mu \mid V_{i+1}, D_t) \neq P_\beta(\mu \mid D_t)$, as knowing the value function gives information about the MDP. 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 $P(V_{i+1} \mid D_t)$, to obtain the following conditional probability measure on value functions:

$$\psi_i(B) \triangleq P_\beta^i(V_i \in B \mid D_t) = \int_{\mathcal{V}} \int_M 1 \{ \psi_{i+1} \in B \} d P_\beta^i(\mu|V_{i+1}, D_t) d\psi_{i+1}(V_{i+1})$$

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

$$P_\beta^i(\mu \in A \mid V_{i+1}, D_t) = \frac{\int_{A} P_\mu^i(V_{i+1}) dP_\beta^i(\mu \mid D_t)}{\int_{\mathcal{M}} P_\mu^i(V_{i+1}) dP_\beta^i(\mu \mid D_t)},$$

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

To compute $P_\beta^i(V_{i+1})$, 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:

$$P_\mu^i(V_{i+1}) = \int_S dq(s) \int_{-\infty}^{\infty} P_\mu^i(V_{i+1} \mid U, s) P_\mu^i(U \mid s) dU.$$

In order to understand the meaning of the term $P_\mu^i(V_{i+1} \mid U, s)$, note that $V_{i+1, \mu}(s) = \mathbb{E}_u^\mu[U \mid s_{i+1} = s]$. 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-learning.

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:

$$w_{jk} \triangleq \frac{\sum_{m=1}^{n} e^{-\frac{|V_{i+1}^{(j)}(s_m) - v_m|^2}{2\sigma^2}}}{\sum_{j'=1}^{N_\mu} \sum_{m=1}^{n} e^{-\frac{|V_{i+1}^{(j')} (s_m) - v_m|^2}{2\sigma^2}}}$$

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

$$V_i^{(j,k)} \triangleq \mathbb{E}_{\mu^{(j)}}^{\beta} V_{i+1}^{(k)}$$

---

^3Assuming otherwise results in a mean-field approximation. See Sec. 2.2. in the arXiv version of the paper.

^4As 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 \( i \)

\[
\psi_i(B) = \frac{1}{N_V N_\mu} \sum_{k=1}^{N_V} \sum_{j=1}^{N_\mu} \mathbb{1} \{ V_i^{(j,k)} \in B \} w_{jk},
\]

(7)

Here, \( N_V \) is the number value function samples \( V_i^{(k)} \). 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 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 \( H \). We instantiate it below for discrete and continuous state spaces.

**Algorithm 1** Policy Evaluation with Method 1

1: **Input**: Prior \( \beta \), data \( D \), lookahead \( H \), discount \( \gamma \), policy \( \pi \), \( N_\mu, N_V \).
2: Initialise \( \psi_H \).
3: Sample \( M \triangleq \{ \mu^{(j)} \mid j \in [N_\mu] \} \) from \( \beta(\mu \mid D) \).
4: for \( i = H - 1, \ldots, 1 \) do
5: Sample \( V_i^{(0)} \sim \psi_{i+1}(u) \) for \( k \in [N_V] \).
6: Generate \( n \) utility samples \( u_m \).
7: Calculate \( w_{jk} \) from (9) and \( V_i^{(j,k)} \) from (5).
8: Calculate \( \psi_i \) from (5).
9: end for
10: 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 (9). However, as a histogram representation is not convenient for a large number of states, we model using a Gaussian \( \psi_i \). In order to do this, we use the sample mean and covariance of the weighted value function samples \( V_i^{(j,k)} \):

\[
m_i = \frac{1}{N_V N_\mu} \sum_{k=1}^{N_V} \sum_{j=1}^{N_\mu} \mu_{i} V_i^{(j,k)} w_{jk}, \quad \Sigma_i = \frac{1}{N_V N_\mu} \sum_{k=1}^{N_V} \sum_{j=1}^{N_\mu} (V_i^{(j,k)} - m_i)(V_i^{(j,k)} - m_i) \Sigma_{w_{jk}}^{-1}
\]

(8)

such that \( \psi_i \sim \mathcal{N}(m_i, \Sigma_i) \) 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 \( Q_i(s_i, a) = s_i^\top \omega_i + \epsilon_i \), where \( s_i, \omega_i \in \mathbb{R}^d \) and \( \epsilon_i \sim \mathcal{N}(0, \sigma_a^2) \). Thus, we obtain the sample mean and covariance to be:

\[
m_i^a = \frac{1}{N_V N_\mu} \sum_{k=1}^{N_V} \sum_{j=1}^{N_\mu} (s_i^{(j,k)})^\top \omega_i, \quad \Sigma_i^a = \frac{\sigma_a^2}{N_V N_\mu} \sum_{k=1}^{N_V} \sum_{j=1}^{N_\mu} (Q_i^{(j,k)}(a) - m_i^a)(Q_i^{(j,k)}(a) - m_i^a) \Sigma_{w_{jk}}^{-1}
\]

(9)

In practice, we often use a feature map \( \phi : \mathbb{R}^d \rightarrow \mathbb{R}^l \) for states.

---

3When 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:

\[
\mathcal{Q}_i(s,a) \triangleq \mathbb{E}_\beta(U | s_i = s, a_i = a, D) = \int_\mathcal{M} \rho_\mu(s,a) + \sum_{s'} \mathbb{P}_\mu^{(j)}(s'|s,a)V_{\pi_{i+1}}(s') \approx \sum_{j,k} \left[ \rho_{\mu(j)}(s,a) + \sum_{s'} \mathbb{P}_\mu^{(j)}(s'|s,a)V_{\pi_{i+1}}^{(k)}(s') \right] \frac{w_{jk}}{N_\mu N_V}. \tag{10}
\]

**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 \( \mathcal{Q}_i(s,a) = \hat{Q}_i(s,a) \).

The \( \mathcal{Q}_i \) estimate is then used to select actions for every state. We set \( \pi_i(a|s) = 1 \) for \( a = \arg \max \mathcal{Q}_i(s,a) \) (Line 19) and calculate the value function distribution (Lines 19 and 20) for the partial policy \( (\pi_i, \pi_{i+1}, \ldots, \pi_H) \).

---

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 [Dimararakis, 2011, MMBI], Bayesian Sparse Sampling [Wang et al., 2008, 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 \( \text{Dir}(\alpha) \) 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
\(N(\mu, \kappa, \alpha, \beta)\) priors for each of the reward distributions \(P(r|s, a)\). 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 \(P(s'|s, a)\) and reward kernel \(P(r|s, a)\) modelled as \(\mathcal{N}(A^{\text{trans}}_s, \Sigma)\) and \(\mathcal{N}(A^{\text{reward}}_s, \sigma^2)\). \(\Sigma\) 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 \(\Psi_0 = 0.001I\) and degrees of freedom \(v_0 = \text{rank}([\Psi_0])\).

**Feature Map.** For InvertedPendulum we use the 10 dimensional feature map \(\phi: \mathbb{R}^d \rightarrow \mathbb{R}^I\) used by [Lagoudakis and Parr 2003] for BBI and GPTD such that the modelling becomes \(P(s'|\phi(s), a)\) and \(P(r|\phi(s), a)\). VDQN works directly on the underlying state \(s\) as it uses a neural network. We also add a regularizing term, \(\lambda I, \lambda = 0.01\) 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. 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

---

8More 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).

<table>
<thead>
<tr>
<th></th>
<th>BBI</th>
<th>PSRL</th>
<th>MMBI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chain</td>
<td>14</td>
<td>5</td>
<td>19</td>
</tr>
<tr>
<td>Maze</td>
<td>921</td>
<td>6</td>
<td>256</td>
</tr>
</tbody>
</table>

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 \( P_{\pi}^\beta(V_i | V_{i+1}, D) \) using the data and to apply it inductively for computing the marginal value function distribution \( P_{\pi}^\beta(V_i | D) \). 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 \( V_i = B_{\pi} \mu V_{i+1} \) and distributional reinforcement learning methods calculate \( P_{\pi}^\mu(U_i | U_{i+1}) \), 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 II 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


