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 5×7 and a 10×10 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 θ 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 $\int_{\mathcal{M}} \max_\pi V_\mu^\pi d\beta(\mu \mid D)$. 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

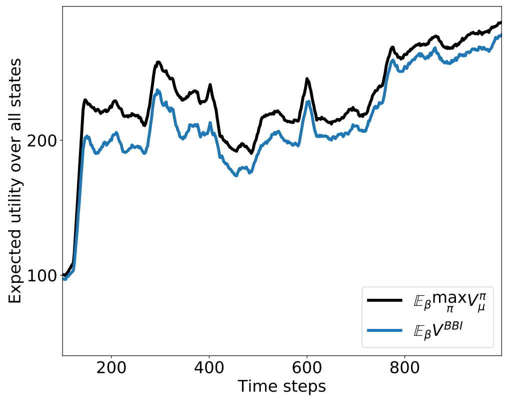
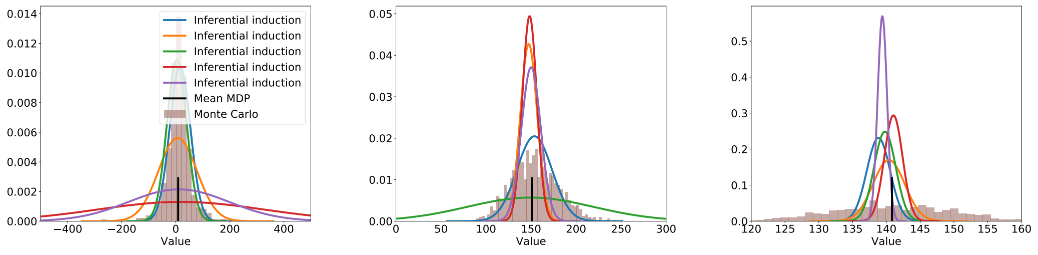


Figure 2: Comparisons of the achieved value functions of BBI with the upper bound on Bayes-optimal value functions. Upper bound and BBI are calculated from 100 MDPs and plotted for 10^5 time steps.

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.

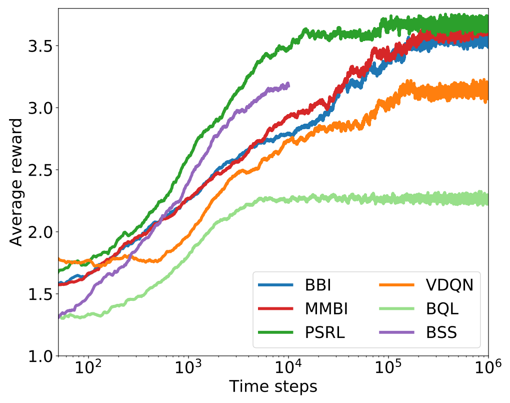
Table 2: Wasserstein distance to the true distribution of the value function, for Alg. 1 and the mean MDP model, for NChain. For Inferential Induction, the distances are averaged over 5 runs. The distances correspond to the plots in Figure 3.

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

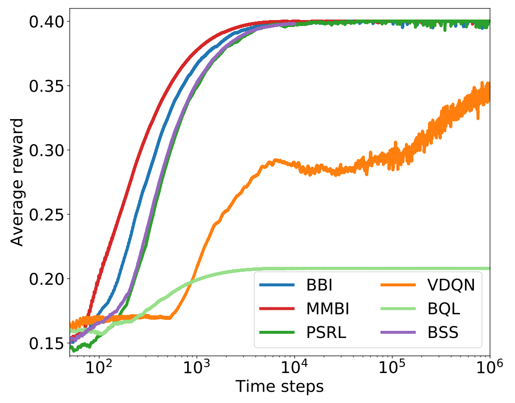
B.3 Further Comparisons on Discrete Environments: MMBI, BSS, BQL

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



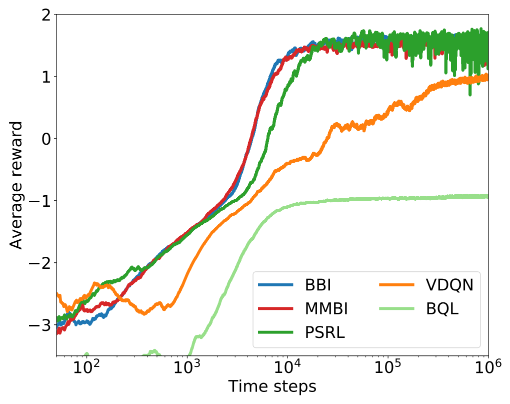
(a) NChain



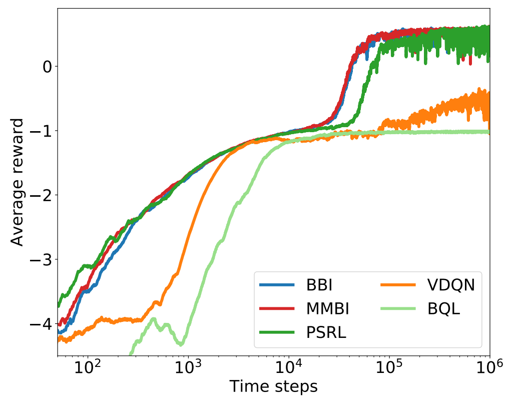
(b) DoubleLoop

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 5×7



(b) LavaLake 10×10

Figure 5: Evolution of average reward for 5×7 and 10×10 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_{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 10×10 and Maze). If the weights obtained in (5) are numerically unstable we attempt to resample the value functions and then double σ 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.

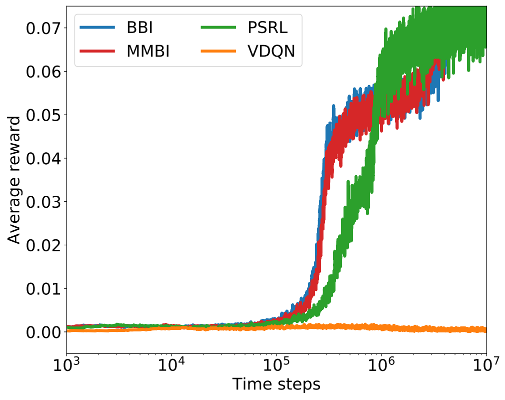
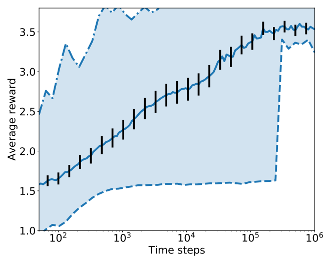
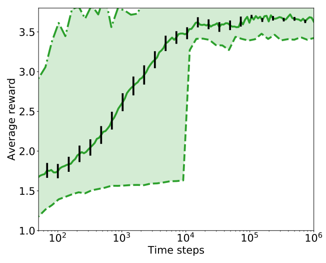


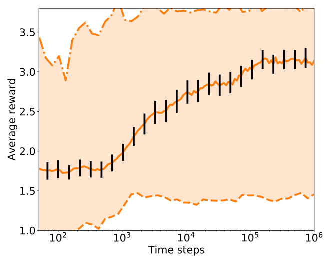
Figure 6: Evolution of average reward for the Maze environment. The results are averaged over 30 runs with a length of 10^6 for each algorithm. The runs are exponentially smoothened with a half-life 1000 before averaging.



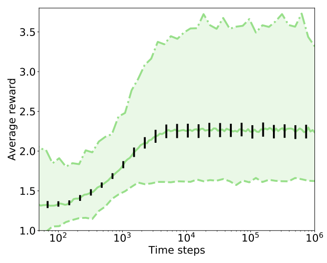
(a) BBI



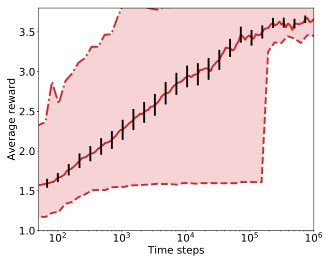
(b) PSRL



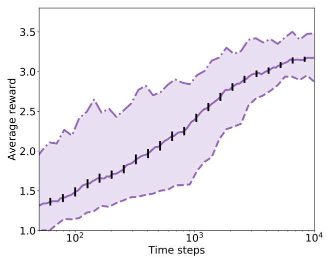
(c) VDQN



(d) BQL

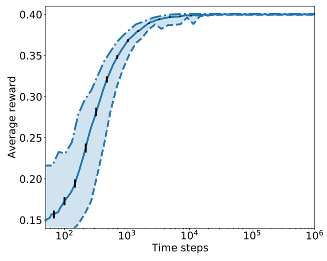


(e) MMBI

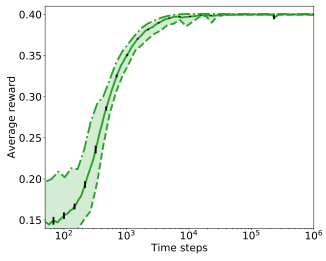


(f) BSS

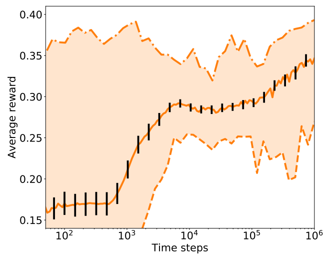
Figure 7: Evolution of average reward for NChain environment with runs of length 10^6 for each algorithm. For computational reasons BSS is only run for 10^4 steps. The runs are exponentially smoothened with a half-life 1000. The mean as well as the 5th and 95th percentile performance is shown for each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



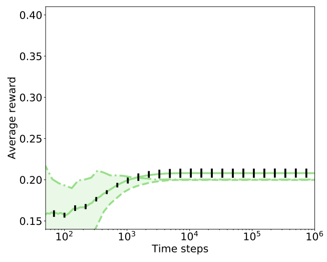
(a) BBI



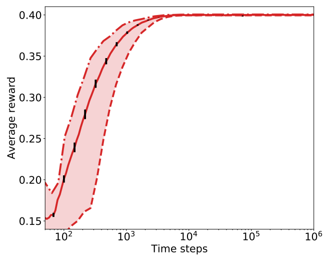
(b) PSRL



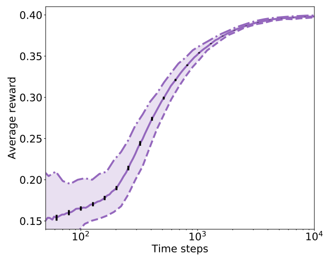
(c) VDQN



(d) BQL

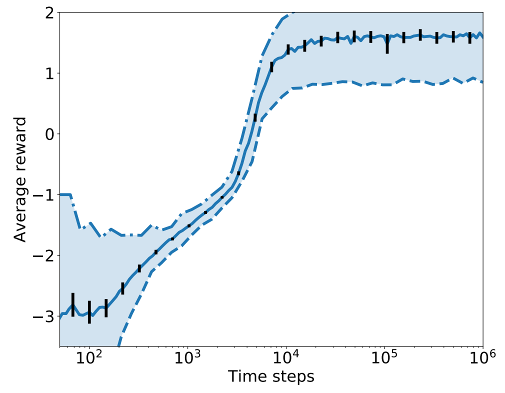


(e) MMBI

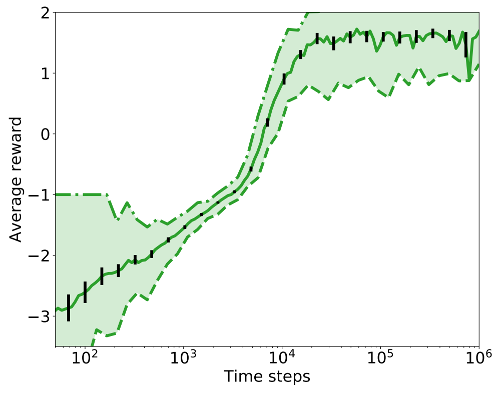


(f) BSS

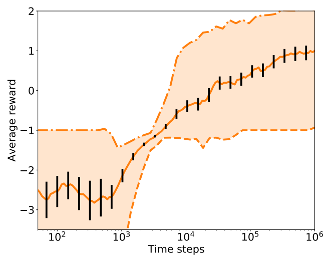
Figure 8: Evolution of average reward for DoubleLoop environment with runs of length 10^6 for each algorithm. For computational reasons BSS is only run for 10^4 steps. The runs are exponentially smoothened with a half-life 1000. The mean as well as the 5th and 95th percentile performance is shown for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



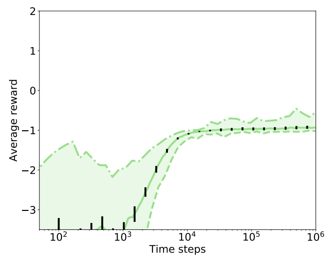
(a) BBI



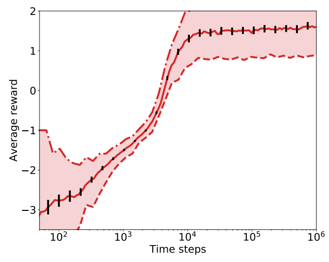
(b) PSRL



(c) VDQN

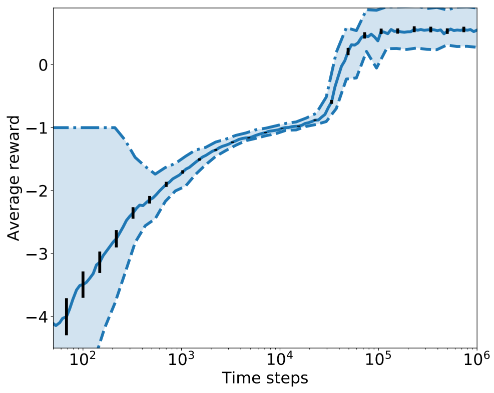


(d) BQL

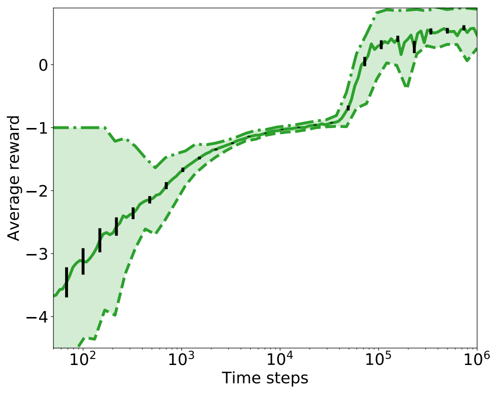


(e) MMBI

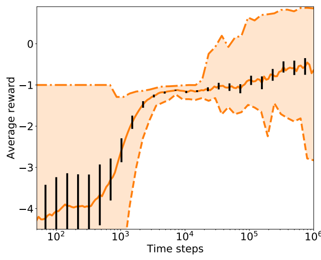
Figure 9: Evolution of average reward for LavaLake 5×7 environment with runs of length 10^6 for each algorithm. The runs are exponentially smoothened with a half-life 1000. The mean as well as the 5th and 95th percentile performance is shown for each algorithm and the standard error is illustrated with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



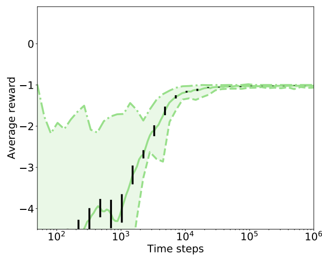
(a) BBI



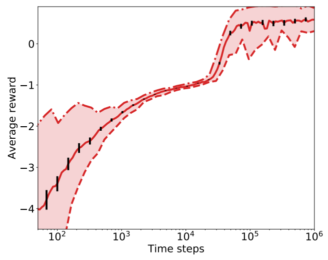
(b) PSRL



(c) VDQN

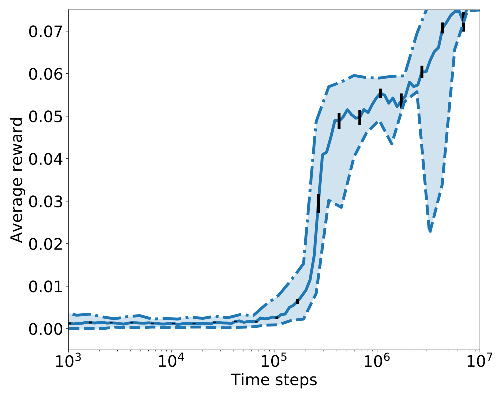


(d) BQL

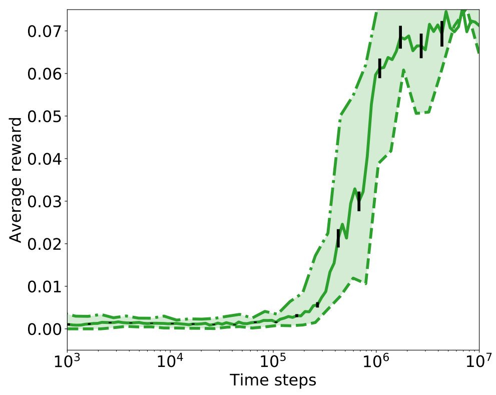


(e) MMBI

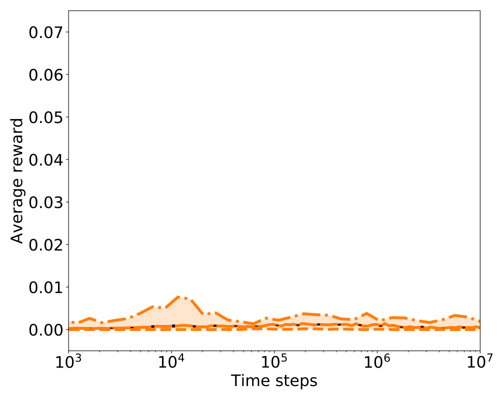
Figure 10: Evolution of average reward for LavaLake 10×10 environment with runs of length 10^6 for each algorithm. The runs are exponentially smoothened with a half-life 1000. The mean as well as the 5th and 95th percentile performance is shown for each algorithm and the standard error is illustrated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



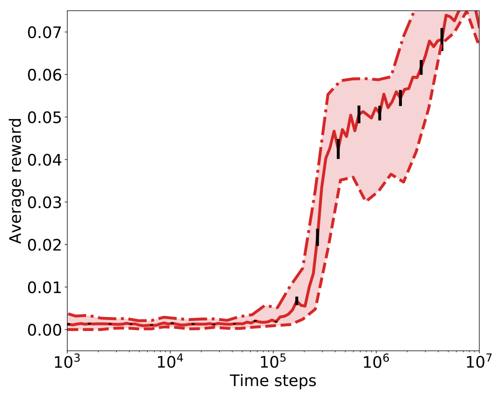
(a) BBI



(b) PSRL

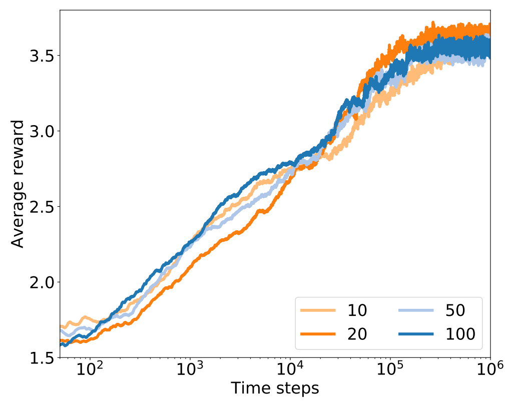


(c) VDQN

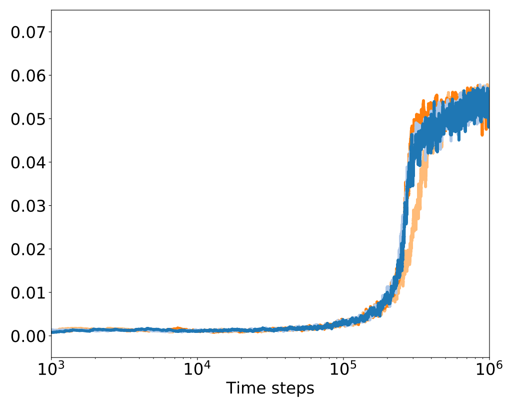


(d) MMBI

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.