Safe Policy Improvement with Baseline Bootstrapping

Romain Laroche [1] **Paul Trichelair** [1] **Remi Tachet des Combes** [1]

Abstract

This paper considers Safe Policy Improvement (SPI) in Batch Reinforcement Learning (Batch RL): from a fixed dataset and without direct access to the true environment, train a policy that is guaranteed to perform at least as well as the baseline policy used to collect the data. Our approach, called SPI with Baseline Bootstrapping (SPIBB), is inspired by the knows-what-it-knows paradigm: it bootstraps the trained policy with the baseline when the uncertainty is high. Our first algorithm, Π_b-SPIBB, comes with SPI theoretical guarantees. We also implement a variant, $\Pi_{\leq b}$-SPIBB, that is even more efficient in practice. We apply our algorithms to a motivational stochastic gridworld domain and further demonstrate on randomly generated MDPs the superiority of SPIBB with respect to existing algorithms, not only in safety but also in mean performance. Finally, we implement a model-free version of SPIBB and show its benefits on a navigation task with deep RL implementation called SPIBB-DQN, which is, to the best of our knowledge, the first RL algorithm relying on a neural network representation able to train efficiently and reliably from batch data, without any interaction with the environment.

1. Introduction

Most real-world Reinforcement Learning agents (Sutton & Barto, 1998, RL) are to be deployed simultaneously on numerous independent devices and cannot be patched quickly. In other practical applications, such as crop management or clinical tests, the outcome of a treatment can only be assessed after several years. Consequently, a bad update could be in effect for a long time, potentially hurting the user's trust and/or causing irreversible damages. Devising safe algorithms with guarantees on the policy performance is a key challenge of modern RL that needs to be tackled before any wide-scale adoption.

Batch RL is an existing approach to such offline settings and consists in training a policy on a fixed set of observations without access to the true environment (Lange et al., 2012). It should not be mistaken with the multi-batch setting where the learner trains successive policies from small batches of interactions with the environment (Duan et al., 2016). Current Batch RL algorithms are however either unsafe or too costly computationally to be used in real-world applications. Safety in RL (García & Fernández, 2015) is an overloaded term, as it may be considered with respect to parametric uncertainty (Thomas et al., 2015a; Petrik et al., 2016), internal uncertainty (Altman, 1999; Carrara et al., 2019), interruptibility (Orseau & Armstrong, 2016; Guerraoui et al., 2017), or as exploration in a hazardous environment (Schulman et al., 2015; 2017; Fatemi et al., 2019). We focus on the former.

In this paper, we develop novel *safe and efficient* Batch RL algorithms. Our methodology for Safe Policy Improvement (SPI), called SPI with Baseline Bootstrapping (SPIBB), is introduced in Section 2. It consists in bootstrapping the trained policy with the behavioral policy, called *baseline*, in the state-action pair transitions that were not probed enough in the dataset. It therefore assumes access to the baseline, an assumption already made in the SPI literature (Petrik et al., 2016). Other SPI algorithms assume knowledge of the baseline performance instead (Thomas et al., 2015a;b). We argue that our assumption is more natural since SPI aims to improve an existing policy. This scenario is typically encountered when a policy is trained in a simulator and then run in its real environment, for instance in Transfer RL (Taylor & Stone, 2009); or when a system is designed with expert knowledge and then optimized, for example in Dialogue applications (Laroche et al., 2010).

Still in Section 2, we implement a computationally efficient algorithm, Π_b-SPIBB, that provably approximately outperforms the baseline with high confidence. At the expense of theoretical guarantees, we design a variant, $\Pi_{\leq b}$-SPIBB, that is even more efficient in practice. Moreover, we implement an equivalent model-free version. Coupled with a pseudo-count implementation (Bellemare et al., 2016), it allows applying SPIBB algorithms to tasks requiring a

[1] Microsoft Research, Montréal, Canada. Correspondence to: Romain Laroche <romain.laroche@microsoft.com>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

neural network representation. Finally, we position our algorithm with respect to competing SPI algorithms found in the literature.

Then, in Section 3, we motivate our approach on a small stochastic gridworld domain and further demonstrate on randomly generated MDPs the superiority of SPIBB compared to existing algorithms, not only in safety but also in mean performance. Furthermore, we apply the model-free version to a continuous navigation task. It is, to the best of our knowledge, the first RL algorithm relying on a neural network representation able to train efficiently and reliably from batch data, without any interaction with the environment (Duan et al., 2016).

Finally, Section 4 concludes the paper. The appendix includes the proofs, thorough experiment details, and the complete results of experiments. The code may be found at `https://github.com/RomainLaroche/SPIBB` and `https://github.com/rems75/SPIBB-DQN`.

2. SPI with Baseline Bootstrapping

A proper introduction to Markov Decision Processes (Bellman, 1957, MDPs) and Reinforcement Learning (Sutton & Barto, 1998, RL) is available in Appendix A.1. Due to space constraint, we only define our notations here.

An MDP is denoted by $M = \langle \mathcal{X}, \mathcal{A}, R, P, \gamma \rangle$, where $\mathcal{X}$ is the state space, $\mathcal{A}$ is the action space, $R^*(x,a) \in [-R_{max}, R_{max}]$ is the bounded stochastic reward function, $P^*(\cdot|x,a)$ is the transition distribution, and $\gamma \in [0,1)$ is the discount factor. The true environment is modelled as an unknown finite MDP $M^* = \langle \mathcal{X}, \mathcal{A}, R^*, P^*, \gamma \rangle$ with $R^*(x,a) \in [-R_{max}, R_{max}]$. $\Pi = \{\pi : \mathcal{X} \to \Delta_{\mathcal{A}}\}$ is the set of stochastic policies, where $\Delta_{\mathcal{A}}$ denotes the set of probability distributions over the set of actions $\mathcal{A}$.

The state and state-action value functions are respectively denoted by $V_M^\pi(x)$ and $Q_M^\pi(x,a)$. We define the performance of a policy by its expected return, starting from the initial state x_0: $\rho(\pi, M) = V_M^\pi(x_0)$. Given a policy subset $\Pi' \subseteq \Pi$, a policy π' is said to be Π'-optimal for an MDP M when it maximizes its performance on Π': $\rho(\pi', M) = \max_{\pi \in \Pi'} \rho(\pi, M)$. We will also make use of the notation V_{max} as a known upper bound of the return's absolute value: $V_{max} \leq \frac{R_{max}}{1-\gamma}$.

In this paper, we focus on the batch RL setting where the algorithm does its best at learning a policy from a fixed set of experience. Given a dataset of transitions $\mathcal{D} = \langle x_j, a_j, r_j, x'_j \rangle_{j \in [\![1,N]\!]}$, we denote by $N_\mathcal{D}(x,a)$ the state-action pair counts; and by $\widehat{M} = \langle \mathcal{X}, \mathcal{A}, \widehat{R}, \widehat{P}, \gamma \rangle$ the Maximum Likelihood Estimation (MLE) MDP of the environment, where $\widehat{R}$ is the reward mean and $\widehat{P}$ is the transition statistics observed in the dataset. Vanilla batch RL, referred hereinafter as Basic RL, looks for the optimal policy in $\widehat{M}$. This policy may be found indifferently using dynamic programming on the explicitly modelled MDP $\widehat{M}$, Q-learning with experience replay until convergence (Sutton & Barto, 1998), or Fitted-Q Iteration with a one-hot vector representation of the state space (Ernst et al., 2005).

2.1. Percentile criterion and Robust MDPs

We start from the *percentile criterion* (Delage & Mannor, 2010) on the safe policy improvement over the baseline π_b:

$$\pi_C = \operatorname*{argmax}_{\pi \in \Pi} \mathbb{E}\left[\rho(\pi, M) \,|\, M \sim \mathbb{P}_{\text{MDP}}(\cdot|\mathcal{D})\right], \qquad (1)$$

$$\text{s.t. } \mathbb{P}\left(\rho(\pi, M) \geq \rho(\pi_b, M) - \zeta \,|\, M \sim \mathbb{P}_{\text{MDP}}(\cdot|\mathcal{D})\right) \geq 1 - \delta,$$

where $\mathbb{P}_{\text{MDP}}(\cdot|\mathcal{D})$ is the posterior probability of the MDP parameters, $1-\delta$ is the high probability meta-parameter, and ζ is the approximation meta-parameter. (Petrik et al., 2016) use Robust MDP (Iyengar, 2005; Nilim & El Ghaoui, 2005) to bound from below the constraint in (1) by considering a set of admissible MDPs $\Xi = \Xi(\widehat{M}, e)$ defined as:

$$\Xi(\widehat{M}, e) := \{M = \langle \mathcal{X}, \mathcal{A}, R, P, \gamma \rangle \quad \text{s.t. } \forall (x,a) \in \mathcal{X} \times \mathcal{A}, \left. \begin{array}{r} ||P(\cdot|x,a) - \widehat{P}(\cdot|x,a)||_1 \leq e(x,a), \\ |R(x,a) - \widehat{R}(x,a)| \leq e(x,a) R_{max} \end{array} \right\} \qquad (2)$$

where $e : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$ is an error function depending on $\mathcal{D}$ and δ. In place of the intractable expectation in Equation (1), Robust MDP classically consider optimizing the policy performance $\rho(\pi, M)$ of the worst-case scenario in Ξ:

$$\pi_R = \operatorname*{argmax}_{\pi \in \Pi} \min_{M \in \Xi} \rho(\pi, M). \qquad (3)$$

In our benchmarks, we use the Robust MDP solver described in Petrik et al. (2016). Petrik et al. (2016) also contemplate the policy improvement worst-case scenario:

$$\pi_S = \operatorname*{argmax}_{\pi \in \Pi} \min_{M \in \Xi} \left(\rho(\pi, M) - \rho(\pi_b, M)\right). \qquad (4)$$

They prove that this optimization is an NP-hard problem and propose an algorithm approximating the solution without any formal proof: Approximate Robust Baseline Regret Minimization (ARBRM). There are three problems with ARBRM. First, it assumes that there is no error in the transition probabilities of the baseline, which is very restrictive and amounts to Basic RL when the support of the baseline is the full action space in each state (as is the case in all our experiments). Second, given its high complexity, it is difficult to empirically assess its percentile criterion safety except on simple tasks. Third, in order to retain safety guarantees, ARBRM requires a conservative safety test that consistently fails in our experiments. These are the reasons why our benchmarks do not include ARBRM.

2.2. SPIBB methodology

In this section, we reformulate the percentile criterion to make searching for an efficient and provably-safe policy tractable in terms of computer time. Our new criterion consists in optimizing the policy with respect to its performance in the MDP estimate $\widehat{M}$, while guaranteeing it to be ζ-approximately at least as good as π_b in the admissible MDP set Ξ. Formally, we write it as follows:

$$\max_{\pi\in\Pi} \rho(\pi, \widehat{M}), \text{ s.t. } \forall M \in \Xi, \rho(\pi, M) \geq \rho(\pi_b, M) - \zeta. \tag{5}$$

From Theorem 8 of Petrik et al. (2016), it is direct to guarantee that, if all the state-action pair counts satisfy:

$$N_\mathcal{D}(x,a) \geq N_\wedge = \frac{8V_{max}^2}{\zeta^2(1-\gamma)^2} \log \frac{2|\mathcal{X}||\mathcal{A}|2^{|\mathcal{X}|}}{\delta}, \tag{6}$$

and if $\widehat{M}$ is the Maximum Likelihood Estimation (MLE) MDP, then, with high probability $1-\delta$, the optimal policy $\pi^\odot = \operatorname{argmax}_{\pi\in\Pi} \rho(\pi, \widehat{M})$ in $\widehat{M}$ is ζ-approximately safe with respect to the true environment M^*:

$$\rho(\pi^\odot, M^*) \geq \rho(\pi^*, M^*) - \zeta \geq \rho(\pi_b, M^*) - \zeta. \tag{7}$$

In the following, we extend this result by allowing constraint (6) to be violated on a subset of the state-action pairs $\mathcal{X} \times \mathcal{A}$, called the bootstrapped set and denoted by $\mathfrak{B}$. $\mathfrak{B}$ is the set of state-action pairs with counts smaller than $N_\wedge$.

2.3. Π_b-SPIBB

In this section, we develop two novel algorithms based on policy bootstrapping and prove associated SPI bounds. More precisely, when a state-action pair (x,a) is rarely seen in the dataset, we propose to rely on the baseline by copying its probability to take action a:

$$\pi^\odot_{spibb}(a|x) = \pi_b(a|x) \text{ if } (x,a) \in \mathfrak{B}. \tag{8}$$

We let Π_b denote the set of policies that verify (8) for all state-action pairs. Our first algorithm, coined Π_b-SPIBB, consists in the usual policy optimization of the expected return $\rho(\pi, \widehat{M})$ under constraint (8). In practice, it may be achieved in a model-based manner by explicitly computing the MDP model $\widehat{M}$, constructing the set of allowed policies Π_b and finally searching for the Π_b-optimal policy $\pi^\odot_{spibb}$ in $\widehat{M}$ using policy iteration over Π_b (Howard, 1966; Puterman & Brumelle, 1979). In the policy evaluation step, the current policy $\pi^{(i)}$ is evaluated as $Q^{(i)}_{\widehat{M}}$. In the policy improvement step, $\pi^{(i+1)}$ is defined as the greedy policy with respect to $Q^{(i)}$ under the constraint of belonging to Π_b (Algorithm 1 describes how to enforce this constraint in linear time).

Algorithm 1 Greedy projection of $Q^{(i)}$ on Π_b

```
Input: Baseline policy $\pi_b$
Input: Last iteration value function $Q^{(i)}$
Input: Set of bootstrapped state-action pairs $\mathfrak{B}$
Input: Current state $x$ and action set $\mathcal{A}$
Initialize $\pi^{(i)}_{spibb} = 0$
for $(x,a) \in \mathfrak{B}$ do $\pi^{(i)}_{spibb}(a|x) = \pi_b(a|x)$ ;
$\pi^{(i)}_{spibb}\left(x, \operatorname{argmax}_{a|(x,a)\notin\mathfrak{B}} Q^{(i)}(x,a)\right) = \sum_{a|(x,a)\notin\mathfrak{B}} \pi_b(a|x)$
return $\pi^{(i)}_{spibb}$
```

The following theorems prove that Π_b-SPIBB converges to a Π_b-optimal policy $\pi^\odot_{spibb}$, and that $\pi^\odot_{spibb}$ is a safe policy improvement over the baseline in the true MDP M^*.

Theorem 1 (Convergence)**.** *Π_b-SPIBB converges to a policy $\pi^\odot_{spibb}$ that is Π_b-optimal in the MLE MDP $\widehat{M}$.*

Theorem 2 (Safe policy improvement)**.** *Let Π_b be the set of policies under the constraint of following π_b when $(x,a) \in \mathfrak{B}$. Then, $\pi^\odot_{spibb}$ is a ζ-approximate safe policy improvement over the baseline π_b with high probability $1-\delta$, where:*

$$\zeta = \frac{4V_{max}}{1-\gamma}\sqrt{\frac{2}{N_\wedge}\log\frac{2|\mathcal{X}||\mathcal{A}|2^{|\mathcal{X}|}}{\delta}} - \rho(\pi^\odot_{spibb}, \widehat{M}) + \rho(\pi_b, \widehat{M})$$

Proofs of both theorems are available in Appendix A.3. Theorem 1 is a direct application of the classical policy iteration theorem. Theorem 2 is a generalization of Theorem 8 in Petrik et al. (2016). The resulting bounds may look very similar at first. The crucial difference is that, in our case, $N_\wedge$ is not a property of the dataset, but a hyper-parameter of the algorithm. In all our experiments, $\|e\|_\infty$ from Theorem 8 would be equal to 2, leading to a trivial bound. In comparison, Π_b-SPIBB allows safe improvement if $N_\wedge$ is chosen large enough to ensure safety and small enough to ensure improvement.

SPIBB takes inspiration from Petrik et al. (2016)'s idea of finding a policy that is guaranteed to be an improvement for any realization of the uncertain parameters, and similarly estimates the error on those parameters, as a function of the state-action pair counts. But instead of searching for the analytic optimum, SPIBB looks for a solution that improves the baseline when it can guarantee improvement and falls back on the baseline when the uncertainty is too high. One can see it as a *knows-what-it-knows* algorithm, asking for help from the baseline when it *does not know whether it knows* (Li et al., 2008). As such, our algorithms can be seen as pessimistic, the flip side of *optimism in the face of uncertainty* (Szita & Lőrincz, 2008). As a consequence, Π_b-

SPIBB is not optimal with respect to the criterion in Equation (5). But in return, it is inherently safe as it only allows to search in a set of policies for which the improvement over the baseline can be safely evaluated (Thomas et al., 2017). It is also worth mentioning that SPIBB is computationally simple, which allows us to develop the SPIBB-DQN algorithm in the next section.

2.4. Model-free Π_b-SPIBB and SPIBB-DQN

The Π_b-SPIBB policy optimization may indifferently be achieved in a model-free manner by fitting the Q-function to the following target $y_j^{(i)}$ over the transition samples in the dataset $\mathcal{D} = \langle x_j, a_j, r_j, x'_j \rangle_{j \in [\![1,N]\!]}$:

$$y_j^{(i)} = r_j + \gamma \sum_{a' | (x'_j, a') \in \mathfrak{B}} \pi_b(a'|x'_j) Q^{(i)}(x'_j, a') \qquad (9)$$
$$+ \gamma \left(\sum_{a' | (x'_j, a') \notin \mathfrak{B}} \pi_b(a'|x'_j) \right) \max_{a' | (x'_j, a') \notin \mathfrak{B}} Q^{(i)}(x'_j, a')$$

The first term r_j is the immediate reward observed during the recorded transition, the second term is the return estimate of the bootstrapped actions (where the trained policy is constrained to the baseline policy), and the third term is the return estimate maximized over the non-bootstrapped actions. SPIBB-DQN is the DQN algorithm fitted to these targets $y_j^{(i)}$ (Mnih et al., 2015). Note that computing the SPIBB targets requires determining the bootstrapped set $\mathfrak{B}$, which relies on an estimate of the state-action counts $\widetilde{N}_\mathcal{D}(x, a)$, also called pseudo-counts (Bellemare et al., 2016; Fox et al., 2018; Burda et al., 2019).

Theorem 3. *In finite MDPs, Equation 9 admits a unique fixed point that coincides with the Q-value of the policy trained with model-based* Π_b*-SPIBB.*

2.5. $\Pi_{\leq b}$-SPIBB

In our empirical evaluation, we consider a variant of Π_b-SPIBB: the space of policies to search is relaxed to $\Pi_{\leq b}$, the set of policies that do not to give more weight than π_b to bootstrapped actions. As a consequence, in comparison with Π_b-SPIBB, it allows to cut off bad performing actions even when their estimate is imprecise:

$$\Pi_{\leq b} = \{\pi \in \Pi \,|\, \pi(a|x) \leq \pi_b(a|x) \text{ if } (x, a) \in \mathfrak{B}\} \quad (10)$$

The resulting algorithm is referred as $\Pi_{\leq b}$-SPIBB and amounts, as for Π_b-SPIBB, to perform a policy iteration under the policy constraint to belong to $\Pi_{\leq b}$. The convergence guarantees of Theorem 1 still apply to $\Pi_{\leq b}$-SPIBB, but we lose the SPI ones.

Algorithm 2 in Appendix A.4, describes the greedy projection of $Q^{(i)}$ on $\Pi_{\leq b}$. Appendix A.5 also includes a comprehensive example that illustrates the difference between the Π_b-SPIBB and $\Pi_{\leq b}$-SPIBB policy improvement steps. Despite the lack of safety guarantees, our experiments show $\Pi_{\leq b}$-SPIBB to be even safer than Π_b-SPIBB while outperforming it in most scenarios. Multi-batch settings – where it may be better to keep exploring the bootstrapped pairs – might be an exception (Lange et al., 2012).

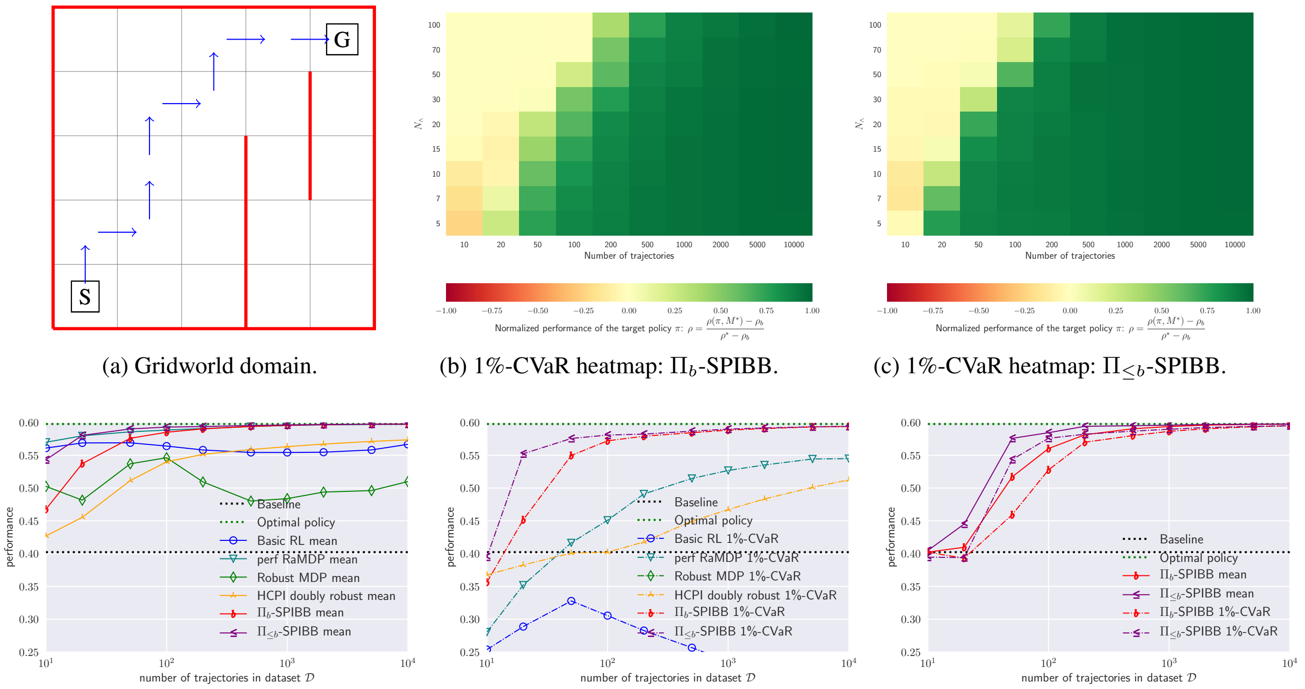
2.6. Other related works

High-Confidence PI refers to the family of algorithms introduced in Paduraru (2013); Mandel et al. (2014); Thomas et al. (2015a), which rely on the ability to produce high-confidence lower bound on the Importance Sampling (IS) estimate of the trained policy performance. IS and SPIBB approaches are very different in nature: IS provides frequentist bounds, while SPIBB provides Bayesian bounds. In comparison to SPIBB, IS has the advantage of not depending on the MDP model and as a consequence may be applied to infinite MDPs with guarantees. However, the IS estimates are known to be high variance. Another drawback of the IS approach is that it fails for long horizon problem. Indeed, Guo et al. (2017) show that the amount of data required by IS-based SPI algorithms scales exponentially with the horizon of the MDP. Regarding the dependency in the horizon of SPIBB algorithms, the discount factor γ is often translated as a planning horizon: $H = \frac{1}{1-\gamma}$. This is the case in UCT for instance (Kocsis & Szepesvári, 2006). As a consequence, Theorem 2 tells us that the safety is linear in the horizon (given a fixed V_{max}).

In Kakade & Langford (2002), Conservative Policy Iteration (CPI) not only assumes access to the environment, but also to a μ-restart mechanism which can basically sample at will from the environment according to a distribution μ. This is used in step (2) of the CPI algorithm to build an estimate of the advantage function precise enough to ensure policy improvement with high probability. SPIBB does not have access to the true environment: all it sees are the finite samples from the batch. Similarly, Pirotta et al. (2013a;b) consider a single safe policy improvement in order to speed up training of policy gradients (use less policy iterations). These are however not safe in the sense of finding a policy that improves a previous policy with high confidence: they will converge to the same policy asymptotically, the optimal one in the MLE MDP. Additionally, they are not considering the batch setting.

3. SPIBB Empirical Evaluation

The performance of Batch RL algorithms can vary greatly from one dataset to another. To properly assess existing and SPIBB algorithms, we evaluate their ability to generate policies that consistently outperform the baseline. Practically, we repeated 100k times the following procedure on



(a) Gridworld domain. (b) 1%-CVaR heatmap: Π_b-SPIBB. (c) 1%-CVaR heatmap: $\Pi_{\leq b}$-SPIBB.

(d) Mean: benchmark with $N_\wedge = 5$. (e) 1%-CVaR: benchmark with $N_\wedge = 5$. (f) Mean & 1%-CVaR: SPIBB w. $N_\wedge = 20$.

Figure 1. Gridworld experiment: Figure (a) illustrates the domain with an optimal trajectory. Figures (b-c) are heatmaps of the 1%-CVaR normalized performance of the SPIBB algorithms as a function of $N_\wedge$. Figures (d-e) show the benchmark for the mean and 1%-CVaR performance. Figure (f) displays additional curves for another value of $N_\wedge$.

various environments: randomly generate a dataset, train a policy on that dataset using each algorithm and each hyper-parameter in the benchmark and compute the performance of the trained policy (with $\gamma = 0.95$). We formalize this experimental protocol in Appendix B.1.1. The algorithms are then evaluated using the mean performance and conditional value at risk performance (CVaR, also called expected shortfall) of the policies they produced. The X%-CVaR is the mean performance over the X% worst runs. Given the high number of runs, all the results that are visible to the naked eye are significant.

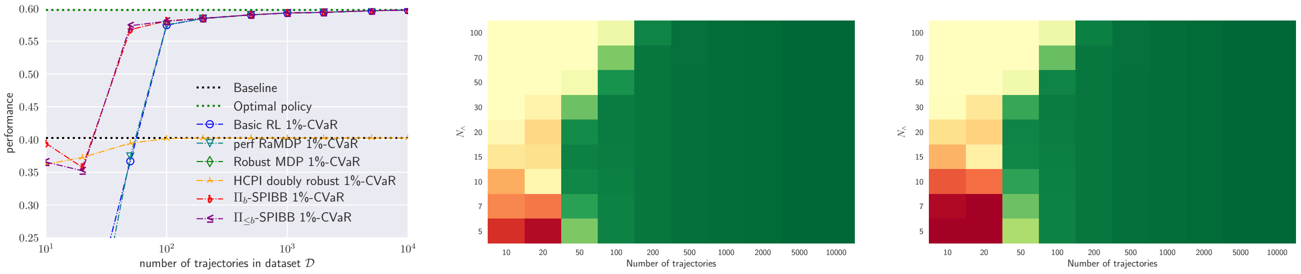
In addition to the SPIBB algorithms, our finite MDP benchmark contains four algorithms: Basic RL, HCPI (Thomas et al., 2015a), Robust MDP, and RaMDP (Petrik et al., 2016). RaMDP stands for Reward-adjusted MDP and applies an exploration penalty when performing actions rarely observed in the dataset. At the exception of Basic RL, they all rely on one hyper-parameter: δ_{hcpi}, δ_{rob} and κ_{adj} respectively. We performed a grid search on those parameters and for HCPI compared 3 versions. In the main text, we only report the best performance we found ($\delta_{hcpi} = 0.9$, $\delta_{rob} = 0.1$, and $\kappa_{adj} = 0.003$), the full results can be found in Appendix B.2. Additionally, Robust MDP and RaMDP depend on a safety test that always failed in our experiments. We still report their performance.

3.1. Does SPIBB outperform existing algorithms?

Our first domain is a discrete, stochastic, 5×5 gridworld (see Figure 1(a)), with 4 actions: up, down, left and right. The transitions are stochastic: the agent moves in the requested direction with 75% chance, in the opposite one with 5% chance and to either side with 10% chance each. The initial and final states are respectively the bottom left and top right corners. The reward function is $+1$ when the final state is reached and 0 everywhere else. The baseline we use in this experiment is a fixed stochastic policy with a 0.4 performance, the optimal policy has a 0.6 performance.

We start by analysing the sensitivity of Π_b-SPIBB and $\Pi_{\leq b}$-SPIBB with respect to $N_\wedge$. We visually represent the results as two 1%-CVaR heatmaps: Figures 1(b) and 1(c) for Π_b-SPIBB and $\Pi_{\leq b}$-SPIBB. They read as follows: the colour of a cell indicates the improvement over the baseline normalized with respect to the optimal performance: red, yellow, and green respectively mean below, equal to, and above baseline performance. We observe for SPIBB algorithms that the policy improvement is safe (at the slight exception of Π_b-SPIBB with a low $N_\wedge$ on 10-trajectory datasets), that the bigger the $N_\wedge$, the more conservative SPIBB gets, and that $\Pi_{\leq b}$-SPIBB outperforms Π_b-SPIBB.

In Figure 1(d), we see that Basic RL improves the base-



(a) 1%-CVaR benchmark with $N_\wedge = 20$.

(b) 1%-CVaR heatmap: Π_b-SPIBB.

(c) 1%-CVaR heatmap: $\Pi_{\leq b}$-SPIBB.

Figure 2. Gridworld experiment with random behavioural policy: Figure (a) shows the benchmark for the 1%-CVaR performance, with SPIBB using $N_\wedge = 20$. Figures (b-c) are the heatmaps of the 1%-CVaR normalized performance of the SPIBB algorithms as a function of $N_\wedge$ (same heat colours as in Figures 1(b) and 1(c)).

line on average, but not monotonically with the size of the dataset, and remains quite far from optimal. That fact is explained by the fairly frequent learning of catastrophic policies and will be analyzed in details with the 1%-CVaR results. HCPI is more conservative for small datasets but slightly outperforms Basic RL for bigger ones, still remaining away from optimal. We also observe that Robust MDPs do even worse than Basic RL; in fact, they learn policies that remain at the center of the grid where the dataset contains a maximum of transitions and therefore where the Robust MDPs have a minimal estimate error, and completely ignore the goal. A similar behaviour is observed with RaMDP when its hyper-parameter is set too high (≥ 0.004). Inversely, when it is set too low (≤ 0.002), RaMDP behaves like Basic RL. But in the tight spot of 0.003, RaMDP is very efficient. We refer the interested reader to Appendix B.2 for the analysis of hyper-parameter search for the benchmark algorithms. Overall, RaMDP and $\Pi_{\leq b}$-SPIBB win this benchmark based on mean performance, with Π_b-SPIBB not far behind.

Figure 1(e) displays the 1%-CVaR performance of the algorithms. We observe that the very good mean performance of RaMDP hides some catastrophic runs where the trained policy under-performs for small datasets. In contrast, $\Pi_{\leq b}$-SPIBB's curve remains over the baseline. Π_b-SPIBB is again a bit behind. HCPI also proves to be near safe. We explained in the previous paragraph why Robust MDP often generates bad policies. It actually does it so often, and the policies are so bad, that its curve does not even show on the graph. Let us now consider Basic RL and explain why it does so poorly, even at times on very large datasets (considering that the MDP has 25 states and 4 actions). The dataset is collected using a baseline that performs some actions only very rarely. As a consequence, even in big datasets, some state-action pairs are observed only once or twice. Given the stochasticity of the environment, the MLE MDP might be quite different from the true MDP in those states, leading to policies falsely taking advantage of those chimaeras. SPIBB algorithms are not allowed to jump to conclusions without sufficient proof and have to conservatively reproduce the baseline policy in those configurations.

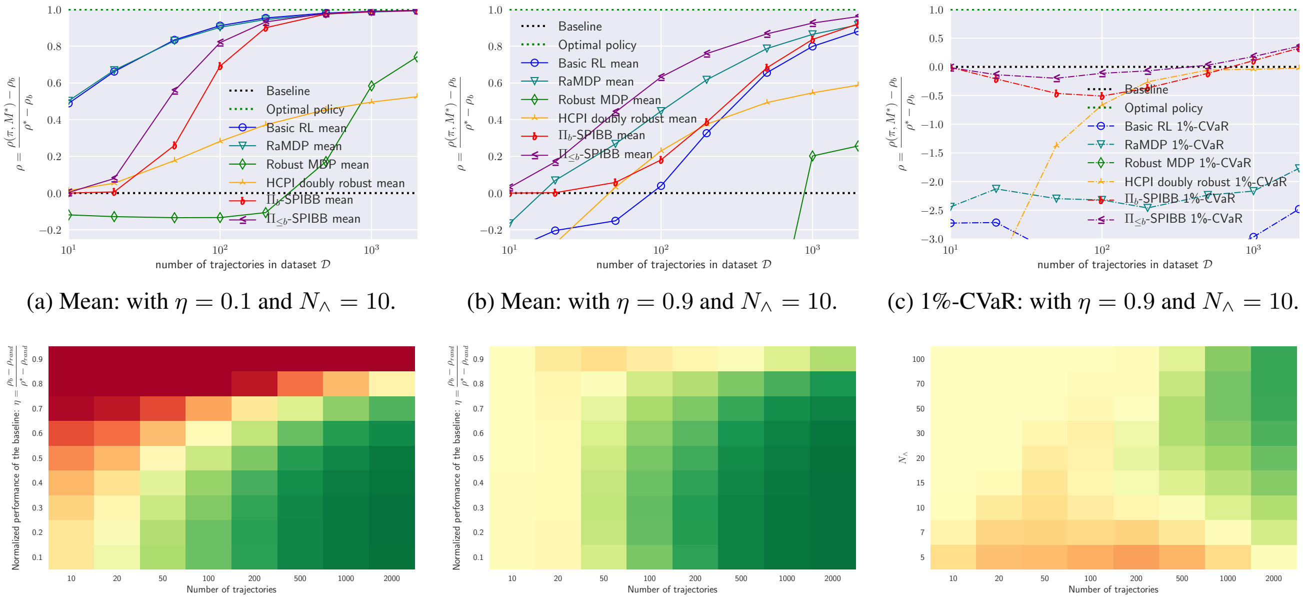
Figure 1(f) shows the SPIBB curves for a higher value of $N_\wedge = 20$. There, the algorithms are more conservative and therefore safe, while still achieving near optimality on big datasets. Full results may be found in Appendix C.1.

3.2. Must the dataset be collected with the baseline?

SPIBB theory relies on the assumption that the baseline was used for the data collection, which is a limiting factor of the method. In practice, this assumption simply ensures that the preferential trajectories of the baseline are experienced in the batch of trajectories used for training. We modify the previous experiment by producing datasets using a uniform random policy, while keeping the same Gridworld environment and the same baseline for bootstrapping. In this setting, Basic RL does not have its non-monotonic behaviour anymore, but both our algorithms, Π_b-SPIBB and $\Pi_{\leq b}$-SPIBB, still significantly outperform their competitors (see Figure 2(a)). Note however the following differences: Basic RL becomes safe with 100 trajectories, RaMDP does not improve Basic RL anymore, and HCPI has more difficulty improving the baseline. Robust MDP still does not show on the 1%-CVaR figure. Focusing more specifically on the SPIBB algorithms and their $N_\wedge$ sensitivity, Figures 2(b) and 2(c) show that they fail to be completely safe when $N_\wedge \leq 10$ and $|\mathcal{D}| \leq 20$; and that Π_b-SPIBB slightly outperforms $\Pi_{\leq b}$-SPIBB. Indeed, $\Pi_{\leq b}$-SPIBB cannot take advantage anymore of the bias that the behavioural policy tends to take actions that are better than average. Full results may be found in Appendix C.2.

3.3. Does SPIBB achieve SPI in most domains?

In this section, we study the conditions required on the environment and on the baseline for SPIBB to be helpful. To do so, we use a generator of Random MDPs where the



(a) Mean: with $\eta = 0.1$ and $N_\wedge = 10$.

(b) Mean: with $\eta = 0.9$ and $N_\wedge = 10$.

(c) 1%-CVaR: with $\eta = 0.9$ and $N_\wedge = 10$.

(d) 1%-CVaR: RaMDP with $\delta_{adj} = 0.003$.

(e) 1%-CVaR: $\Pi_{\leq b}$-SPIBB with $N_\wedge = 10$.

(f) 1%-CVaR: $\Pi_{\leq b}$-SPIBB with $\eta = 0.9$.

Figure 3. Random MDPs domain: Figures (a-c) show the mean and 1%-CVaR performances for η values of 0.1 and 0.9 and SPIBB with $N_\wedge = 10$. Figures (d-e) are the 1%-CVaR as a function of η for RaMDP and $\Pi_{\leq b}$-SPIBB respectively. Figure (f) is the 1%-CVaR heatmap for $\Pi_{\leq b}$-SPIBB as a function of $N_\wedge$ with $\eta = 0.9$.

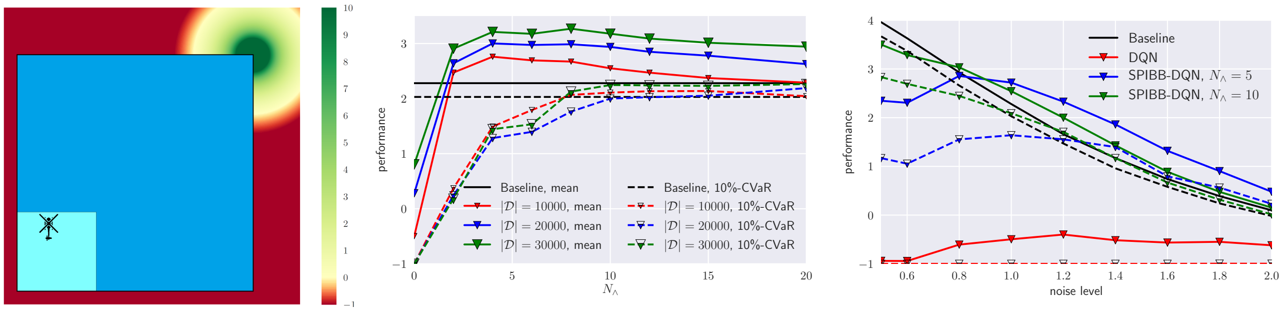
number of states has been fixed to $|\mathcal{X}| = 50$, the number of actions to $|\mathcal{A}| = 4$ and the connectivity of the transition function to 4. This means that for a given state-action pair (x, a), its transition function $P(x'|x, a)$ is non-zero on four states x' only. The initial state is fixed at x_0. The reward function is 0 everywhere except when entering the terminal state, where it equals 1. The terminal state is chosen in such a way that the optimal value function is minimal. It coarsely amounts to choosing the state that is the hardest to reach/farthest from x_0. For a randomly generated MDP M, we generate baselines with different levels of performance (the process is detailed in Appendix B.1.4). Specifically, we set a target performance for the baseline based on a hyper-parameter $\eta \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$: $\rho(\pi_b, M) = \eta\rho(\pi^*, M) + (1 - \eta)\rho(\tilde{\pi}, M)$, where π^* and $\tilde{\pi}$ are respectively the optimal and the uniform policies.

Figure 3(a) shows the mean results with a bad, highly stochastic baseline ($\eta = 0.1$). Since, the baseline is bad, it is an easy task to safely improve it. Basic RL and RaMDP dominate the benchmark in mean, but also in safety (not shown). SPIBB algorithms are too conservative for small datasets but catch up on the bigger ones. Figure 3(b) shows the mean results with a very good baseline, therefore very hard task to safely improve. On average, the podium is composed by $\Pi_{\leq b}$-SPIBB, RaMDP, Π_b-SPIBB, followed closely by Basic RL. But, when one considers more specifically the 1%-CVaR performance, all fail to be safe but the SPIBB algorithms. Note that a -0.5 normalized performance is still a good performance, and that this loss is actually predicted by the theory: Theorem 2 proves a ζ-approximate safe policy improvement.

The heatmaps shown in Figures 3(d) and 3(e) allow us to compare more globally the 1%-CVaR performance of RaMDP and $\Pi_{\leq b}$-SPIBB. One observes that the former is unsafe in a large area of the map (where it is red, for high η or small datasets), while the latter is safe everywhere. Figure 3(f) displays a heatmap of the $\Pi_{\leq b}$-SPIBB 1%-CVaR performance in the hardest scenario ($\eta = 0.9$) in function of its $N_\wedge$ hyper-parameter. Unsurprisingly, the algorithm becomes slightly unsafe when $N_\wedge$ gets too low. As it increases, the red stains disappear meaning that it becomes completely safe. The green sections show that it still allows for some policy improvement. Full results may be found in Appendix C.3.

3.4. Does SPIBB scale to larger tasks?

For the sake of simplicity and to be able to repeat several runs of each experiment efficiently, instead of applying pseudo-count methods from the literature (Bellemare et al., 2016; Fox et al., 2018; Burda et al., 2019), we consider here a pseudo-count heuristic based on the Euclidean state-distance, and a task where it makes sense to do so. The pseudo-count of a state-action (x, a) is defined as the sum of its similarity with the state-action pairs (x_i, a_i) found



(a) Helicopter environment. (b) Mean and 10%-CVaR in function of $N_\wedge$. (c) Performance in function of the noise factor.

Figure 4. SPIBB-DQN experiments: Figure (a) is an illustration of the environment. Figure (b) displays the mean and 10%-CVaR performance as a function of $N_\wedge$ for three dataset sizes. Figure (c) displays the mean and 10%-CVaR performance for the baseline, vanilla DQN, RaMDP with $\kappa_{adj} = 0.01$, SPIBB-DQN with $N_\wedge = 5$, and with $N_\wedge = 10$, as a function of the transition noise factor.

in the dataset. The similarity between (x, a) and (x_i, a_i) is equal to 0 if $a_i \neq a$, and to $\max(0, 1 - d(x, x_i))$ otherwise, where $d(\cdot, \cdot)$ is the Euclidean distance between two states.

We consider a helicopter navigation task (see Figure 4(a)). The helicopter starts from a random position in the teal area, with a random initial velocity. The 9 available actions consist in applying thrust: backward, no, or forward acceleration, along the two dimensions. The episode terminates when the velocity exceeds some maximal value, in which case it gets a -1 reward, or when the helicopter leaves the blue area, in which case it gets a reward as chromatically indicated on Figure 4(a). The dynamics of the domain follow the basic laws of physics with a Gaussian centered additive noise both on the position and the velocity, see Appendix D.1 for full details of the domain. To train our algorithms, we use a discount factor $\gamma = 0.9$, but we report in our results the undiscounted final reward. The baseline is generated as follows: we first train a policy with online DQN, stop before full convergence and then apply a softmax on the obtained Q-network. Our experiments consist in 300 runs on SPIBB-DQN with a range of $N_\wedge$ values and for different dataset sizes. SPIBB-DQN with $N_\wedge = 0$ is equivalent to vanilla DQN. We also tried RaMDP with several values of $\kappa_{adj} \in [0.001, 0.1]$ without any success. For figure clarity, we do not report RaMDP in the Main Document figures. The set of used parameters and the results of the preliminary experiments are reported in Appendices D.3 and D.4.

Figure 4(b) displays the mean and 10%-CVaR performances in function of $N_\wedge$ for three dataset sizes (10k, 20k, and 30k). We observe that vanilla DQN ($N_\wedge = 0$) significantly worsens the baseline in mean and achieves the worst possible 10%-CVaR performance. SPIBB-DQN not only significantly improves the baseline in mean performance for $N_\wedge \geq 1$, but also in 10%-CVaR when $N_\wedge \geq 8$. The discerning reader might wonder about the CVaR curve for the baseline. It is explained by the fact that the evaluation of the policies are not exact. The curve accounts for the evaluation errors, errors also obviously encountered with the trained policies.

We performed an additional experiment. Keeping the baseline identical, we trained on 10k-transitions datasets obtained from environments with a different transition noise. Figure 4(c) shows the mean and 10%-CVaR performances for the baseline, vanilla DQN, and SPIBB-DQN with $N_\wedge \in \{5, 10\}$. First, we observe that vanilla DQN performs abysmally. Second, we see that the baseline quickly gets more efficient when the noise is removed making the safe policy improvement task harder for SPIBB-DQN. SPIBB is efficient at dealing with stochasticity, the noise attenuation reduces its usefulness. Third, as we get to higher noise factors, the stochasticity becomes too high to efficiently aim at the goal, but SPIBB algorithms still succeed at safely improving the baseline.

4. Conclusion and Future Work

In this paper, we tackle the problem of safe Batch Reinforcement Learning. We reformulate the percentile criterion without compromising its safety. We lose optimality that way but keep a PAC-style guarantee of policy improvement. It allows the implementation of an algorithm Π_b-SPIBB that run as fast as a vanilla model-based RL algorithm, while generating a provably safe policy improvement over a known baseline π_b. A variant algorithm $\Pi_{\leq b}$-SPIBB is shown to perform better and safer on a wide range of domains, but does not come with safety guarantees. Basic Batch RL and the other benchmark competitors are shown to fall short on at least one, and generally two, of the following criteria: mean performance, safety, or domain-dependent hyper-parameter sensitivity. Finally, we implement a DQN version of SPIBB that is the first deep batch algorithm allowing policy improvement in a safe manner.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

Altman, E. *Constrained Markov Decision Processes*. CRC Press, 1999.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. Unifying count-based exploration and intrinsic motivation. In *Proceedings of the 29th Advances in Neural Information Processing Systems (NIPS)*, 2016.

Bellman, R. A markovian decision process. *Journal of Mathematics and Mechanics*, 1957.

Burda, Y., Edwards, H., Storkey, A., and Klimov, O. Exploration by random network distillation. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019.

Carrara, N., Leurent, E., Laroche, R., Urvoy, T., Maillard, O., and Pietquin, O. Scaling up budgeted reinforcement learning. *CoRR*, abs/1903.01004, 2019. URL `http://arxiv.org/abs/1903.01004`.

Chollet, F. et al. Keras. `https://keras.io`, 2015.

Delage, E. and Mannor, S. Percentile optimization for markov decision processes with parameter uncertainty. *Operations research*, 2010.

Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. Benchmarking deep reinforcement learning for continuous control. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, pp. 1329–1338, 2016.

Efron, B. Better bootstrap confidence intervals. *Journal of the American statistical Association*, 82(397):171–185, 1987.

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

Fatemi, M., Sharma, Shikharand van Seijen, H., and Ebrahimi Kahou, S. Dead-ends and secure exploration in reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, 2019.

Fox, L., Choshen, L., and Loewenstein, Y. Dora the explorer: Directed outreaching reinforcement action-selection. In *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, 2018.

García, J. and Fernández, F. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 2015.

Gosavi, A. A reinforcement learning algorithm based on policy iteration for average reward: Empirical results with yield management and convergence analysis. *Machine Learning*, 2004.

Guerraoui, R., Hendrikx, H., Maurer, A., et al. Dynamic safe interruptibility for decentralized multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 130–140, 2017.

Guo, Z., Thomas, P. S., and Brunskill, E. Using options and covariance testing for long horizon off-policy policy evaluation. In *Proceedings of the 30th Advances in Neural Information Processing Systems (NIPS)*, pp. 2492–2501, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *arXiv preprint arXiv:1502.01852*, 2015.

Howard, R. A. Dynamic programming. *Management Science*, 1966.

Iyengar, G. N. Robust dynamic programming. *Mathematics of Operations Research*, 2005.

Jiang, N. and Li, L. Doubly robust off-policy value evaluation for reinforcement learning. *arXiv preprint arXiv:1511.03722*, 2015.

Kakade, S. and Langford, J. Approximately optimal approximate reinforcement learning. In *Proceedings of the 19th International Conference on Machine Learning (ICML)*, volume 2, pp. 267–274, 2002.

Kocsis, L. and Szepesvári, C. Bandit based monte-carlo planning. In *European conference on machine learning*, pp. 282–293. Springer, 2006.

Lange, S., Gabel, T., and Riedmiller, M. Batch reinforcement learning. In *Reinforcement learning*. Springer, 2012.

Laroche, R., Putois, G., and Bretier, P. Optimising a handcrafted dialogue system design. In *Proceedings of the 11th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, 2010.

Li, L., Littman, M. L., and Walsh, T. J. Knows what it knows: a framework for self-aware learning. In *Proceedings of the 25th International Conference on Machine Learning (ICML)*, 2008.

Mandel, T., Liu, Y.-E., Levine, S., Brunskill, E., and Popovic, Z. Offline policy evaluation across representations with applications to educational games. In *Proceedings of the 13th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2014.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature*, 2015.

Nilim, A. and El Ghaoui, L. Robust control of markov decision processes with uncertain transition matrices. *Operations Research*, 2005.

Orseau, L. and Armstrong, S. Safely interruptible agents. In *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence (UAI)*, UAI'16, pp. 557–566, Arlington, Virginia, United States, 2016. AUAI Press. ISBN 978-0-9966431-1-5. URL http://dl.acm.org/citation.cfm?id=3020948.3021006.

Paduraru, C. *Off-policy Evaluation in Markov Decision Processes*. PhD thesis, PhD thesis, McGill University, 2013.

Parr, R. E. and Russell, S. *Hierarchical control and learning for Markov decision processes*. University of California, Berkeley, CA, 1998.

Petrik, M., Ghavamzadeh, M., and Chow, Y. Safe policy improvement by minimizing robust baseline regret. In *Proceedings of the 29th Advances in Neural Information Processing Systems (NIPS)*, 2016.

Pirotta, M., Restelli, M., and Bascetta, L. Adaptive step-size for policy gradient methods. In *Proceedings of the 26th Advances in Neural Information Processing Systems (NIPS)*, pp. 1394–1402, 2013a.

Pirotta, M., Restelli, M., Pecorino, A., and Calandriello, D. Safe policy iteration. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, pp. 307–315, 2013b.

Puterman, M. L. and Brumelle, S. L. On the convergence of policy iteration in stationary dynamic programming. *Mathematics of Operations Research*, 1979.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, 2015.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

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

Sutton, R. S., Precup, D., and Singh, S. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 1999.

Szita, I. and Lőrincz, A. The many faces of optimism: a unifying approach. In *Proceedings of the 25th International Conference on Machine Learning (ICML)*, 2008.

Taylor, M. E. and Stone, P. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685, 2009.

Thomas, P. S., Theocharous, G., and Ghavamzadeh, M. High confidence policy improvement. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, 2015a.

Thomas, P. S., Theocharous, G., and Ghavamzadeh, M. High-confidence off-policy evaluation. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, 2015b.

Thomas, P. S., da Silva, B. C., Barto, A. G., and Brunskill, E. On ensuring that intelligent machines are well-behaved. *arXiv preprint arXiv:1708.05448*, 2017.

Tieleman, T. and Hinton, G. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015. URL http://dblp.uni-trier.de/db/journals/corr/corr1509.html#HasseltGS15.

A. SPIBB Theory Complements

A.1. The MDP framework

Markov Decision Processes (Bellman, 1957, MDPs) are a widely used framework to address the problem of optimizing a sequential decision making problem. In this work, we assume that the true environment is modelled as an unknown finite MDP $M^* = \langle \mathcal{X}, \mathcal{A}, R^*, P^*, \gamma \rangle$, where $\mathcal{X}$ is the finite state space, $\mathcal{A}$ is the finite action space, $R^*(x,a) \in [-R_{max}, R_{max}]$ is the true bounded stochastic reward function, $P^*(\cdot|x,a)$ is the true transition distribution, and $\gamma \in [0,1)$ is the discount factor. Without loss of generality, we assume that the process deterministically begins in state x_0. The agent then makes a decision about which action a_0 to select. This action leads to a new state that depends on the transition probability and the agent receives a reward $R^*(x_0, a_0)$. This process is then repeated until the end of the episode. We denote by π the policy which corresponds to the decision making mechanism that assigns actions to states. $\Pi = \{\pi : \mathcal{X} \rightarrow \Delta_{\mathcal{A}}\}$ denotes the set of stochastic policies, and $\Delta_{\mathcal{A}}$ denotes the set of probability distributions over the set of actions $\mathcal{A}$.

The state value function $V_M^\pi(x)$ (resp. state-action value function $Q_M^\pi(x,a)$) is the expectation of the discounted sum of rewards when following $\pi \in \Pi$, starting from state $x \in \mathcal{X}$ (resp. performing action $a \in \mathcal{A}$ in state $x \in \mathcal{X}$) in the MDP $M = \langle \mathcal{X}, \mathcal{A}, R, P, \gamma \rangle$:

$$V_M^\pi(x) = \sum_{a \in \mathcal{A}} \pi(a|x) Q_M^\pi(x,a) = \mathbb{E}_{M,\pi,x}\left[\sum_t \gamma^t r_t\right]. \tag{11}$$

The goal of a reinforcement learning algorithm is to discover the unique optimal state value function V_M^* (resp. action-state value function Q_M^*) and/or a policy that implements it. We define the performance of a policy by its expected return, starting from the initial state: $\rho(\pi, M) = V_M^\pi(x_0)$. Given a policy subset $\Pi' \subseteq \Pi$, a policy π' is said to be Π'-optimal for an MDP M when it maximizes its performance on Π': $\rho(\pi', M) = \max_{\pi \in \Pi'} \rho(\pi, M)$. We will also make use of the notation V_{max} as a known upper bound of the return's absolute value: $V_{max} \leq \frac{R_{max}}{1-\gamma}$.

In this paper, we focus on the batch RL setting where the algorithm does its best at learning a policy from a fixed set of experience. Given a dataset of transitions $\mathcal{D} = \langle x_j, a_j, r_j, x'_j \rangle_{j \in [\![1,N]\!]}$, we denote by $N_\mathcal{D}(x,a)$ the state-action pair counts; and by $\widehat{M} = \langle \mathcal{X}, \mathcal{A}, \widehat{R}, \widehat{P}, \gamma \rangle$ the Maximum Likelihood Estimation (MLE) MDP of the environment, where $\widehat{R}$ is the reward mean and $\widehat{P}$ is the transition statistics observed in the dataset. Vanilla batch RL, referred hereinafter as Basic RL, looks for the optimal policy in $\widehat{M}$. This policy may be found indifferently using dynamic programming on the explicitly modelled MDP $\widehat{M}$, Q-learning with experience replay until convergence (Sutton & Barto, 1998), or Fitted-Q Iteration with a one-hot vector representation of the state space (Ernst et al., 2005).

A.2. Matrix notations for the proofs

The proofs make use of the matrix representation for the Q-function, V-function, the policy, the transition, the reward and the discount rate (when dealing with semi-MDPs) functions.

The Q-functions matrices have 1 row and $|\mathcal{X}||\mathcal{A}|$ columns.

The V-functions matrices have 1 row and $|\mathcal{X}|$ columns.

The policy matrices π have $|\mathcal{X}||\mathcal{A}|$ row and $|\mathcal{X}|$ columns. Even though a policy is generally defined as a function from $\mathcal{X}$ to $\mathcal{A}$ and should be represented by a compact matrix with $|\mathcal{A}|$ rows and $|\mathcal{X}|$ columns, in order to use simple matrix operators, we need the policy matrix to output a distribution over the state-action pairs. Consequently, our policy matrix obtained through the following expansion through the diagonal:

$$\begin{bmatrix} \pi_{11} & \cdots & \pi_{1j} & \cdots & \pi_{1|\mathcal{X}|} \\ \vdots & & \vdots & & \vdots \\ \pi_{i1} & \cdots & \pi_{ij} & \cdots & \pi_{i|\mathcal{X}|} \\ \vdots & & \vdots & & \vdots \\ \pi_{|\mathcal{A}|1} & \cdots & \pi_{|\mathcal{A}|j} & \cdots & \pi_{|\mathcal{A}||\mathcal{X}|} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\pi}_{\cdot 1} & \cdots & \boldsymbol{\pi}_{\cdot j} & \cdots & \boldsymbol{\pi}_{\cdot |\mathcal{X}|} \end{bmatrix} \longrightarrow \begin{bmatrix} \boldsymbol{\pi}_{\cdot 1} & & 0 & & 0 \\ & \ddots & & & \\ 0 & & \boldsymbol{\pi}_{\cdot j} & & 0 \\ & & & \ddots & \\ 0 & & 0 & & \boldsymbol{\pi}_{\cdot |\mathcal{X}|}. \end{bmatrix}$$

The transition matrices P have $|\mathcal{X}|$ rows and $|\mathcal{X}||\mathcal{A}|$ columns.

The reward matrices R have 1 row and $|\mathcal{X}||\mathcal{A}|$ columns.

The discount rate matrices Γ have $|\mathcal{X}|$ rows and $|\mathcal{X}||\mathcal{A}|$ columns.

The expression AB is the matrix product between matrices A and B for which column and row dimensions coincide.

The expression $(A \circ B)$ is the element-wise product between matrices A and B of the same dimension.

$\mathbb{I}$ denotes the identity matrix (the diagonal matrix with only ones), which dimension is given by the context.

$\mathbb{1}(y)$ denotes the column unit vector with zeros everywhere except for the element of index y which equals 1. For instance $Q\mathbb{1}_{x,a}$ denotes the value of performing action a in state x.

The regular and option Bellman equations are therefore respectively written as follows:

$$Q = R + \gamma Q\pi P \tag{12}$$

$$Q = R + Q\pi(\Gamma \circ P) \tag{13}$$

A.3. Convergence and safe policy improvement of Π_b-SPIBB

Lemma 1 (Q-function error bounds with Π_b-SPIBB)**.** *Consider two semi-MDPs $M_1 = \langle \mathcal{X}, \Omega_\mathcal{A}, P_1, R_1, \Gamma_1 \rangle$ and $M_2 = \langle \mathcal{X}, \Omega_\mathcal{A}, P_2, R_2, \Gamma_2 \rangle$. Consider a policy π. Also, consider Q_1 and Q_2 be the state-action value function of the policy π in M_1 and M_2, respectively. If:*

$$\forall a \in \mathcal{A}, \forall x \in \mathcal{I}_a, \begin{cases} |R_1\mathbb{1}_{x,o_a} - R_2\mathbb{1}_{x,o_a}| \leq \epsilon R_{max} \\ ||(\Gamma_1 \circ P_1)\mathbb{1}_{x,o_a} - (\Gamma_2 \circ P_2)\mathbb{1}_{x,o_a}||_1 \leq \epsilon, \end{cases} \tag{14}$$

then, we have:

$$\forall a \in \mathcal{A}, \forall x \in \mathcal{I}_a, \quad |Q_1\mathbb{1}_{x,o_a} - Q_2\mathbb{1}_{x,o_a}| \leq \frac{2\epsilon V_{max}}{1-\gamma}, \tag{15}$$

where V_{max} is the known maximum of the value function.

Proof. We adopt the matrix notations. The difference between the two state-option value functions can be written:

$$\begin{aligned} Q_1 - Q_2 &= R_1 + Q_1\pi(\Gamma_1 \circ P_1) - R_2 - Q_2\pi(\Gamma_2 \circ P_2) & (16)\\ &= R_1 + Q_1\pi(\Gamma_1 \circ P_1) - R_2 - Q_2\pi(\Gamma_2 \circ P_2) + Q_2\pi(\Gamma_1 \circ P_1) - Q_2\pi(\Gamma_1 \circ P_1) & (17)\\ &= R_1 - R_2 + (Q_1 - Q_2)\pi(\Gamma_1 \circ P_1) + Q_2\pi((\Gamma_1 \circ P_1) - (\Gamma_2 \circ P_2)) & (18)\\ &= [R_1 - R_2 + Q_2\pi((\Gamma_1 \circ P_1) - (\Gamma_2 \circ P_2))] (\mathbb{I} - \pi(\Gamma_1 \circ P_1))^{-1}. & (19) \end{aligned}$$

Now using Holder's inequality and the second assumption, we have:

$$|Q_2\pi((\Gamma_1 \circ P_1) - (\Gamma_2 \circ P_2))\mathbb{1}_{x,o_a}| \leq \|Q_2\|_\infty \|\pi\|_\infty \|(\Gamma_1 \circ P_1)\mathbb{1}_{x,o_a} - (\Gamma_2 \circ P_2)\mathbb{1}_{x,o_a}\|_1 \leq \epsilon V_{max}. \tag{20}$$

Inserting (20) into Equation (19) and using the first assumption, we obtain:

$$\begin{aligned} |Q_1\mathbb{1}_{x,o_a} - Q_2\mathbb{1}_{x,o_a}| &\leq [\epsilon R_{max} + \epsilon V_{max}] \, ||(\mathbb{I} - \pi(\Gamma_1 \circ P_1))^{-1}\mathbb{1}_{x,o_a}||_1 & (21)\\ &\leq \frac{2\epsilon V_{max}}{1-\gamma}, & (22) \end{aligned}$$

which proves the lemma. There is a factor 2 that might require some discussion. It comes from the fact that we do not control that the maximum R_{max} might be as big as V_{max} in the semi-MDP setting and we do not control the γ factor in front of the second term. As a consequence, we surmise that a tighter bound down to $\frac{\epsilon V_{max}}{1-\gamma}$ holds, but this still has to be proven. □

Proposition 1. *Consider an environment modelled with a semi-MDP (Parr & Russell, 1998; Sutton et al., 1999) $\ddot{M} = \langle \mathcal{X}, \Omega_\mathcal{A}, \ddot{P}^*, \ddot{R}^*, \Gamma^* \rangle$, where Γ^* is the discount rate inferior or equal to γ that varies with the state action transitions and* *the empirical semi-MDP* $\widehat{\ddot{M}} = \langle \mathcal{X}, \Omega_{\mathcal{A}}, \widehat{\ddot{P}}, \widehat{\ddot{R}}, \widehat{\Gamma} \rangle$ *estimated from a dataset* $\mathcal{D}$*. If in every state* x *where option* o_a *may be initiated:* $x \in \mathcal{I}_a$*, we have:*

$$\sqrt{\frac{2}{N_{\mathcal{D}}(x,a)} \log \frac{2|\mathcal{X}||\mathcal{A}|2^{|\mathcal{X}|}}{\delta}} \leq \epsilon, \tag{23}$$

then, with probability at least $1-\delta$*:*

$$\forall a \in \mathcal{A}, \forall x \in \mathcal{I}_a, \begin{cases} \|\Gamma^* \ddot{P}^*(x, o_a) - \widehat{\Gamma}\widehat{\ddot{P}}(x, o_a)\|_1 \leq \epsilon \\ |\ddot{R}^*(x, o_a) - \widehat{\ddot{R}}(x, o_a)| \leq \epsilon \ddot{R}_{max} \end{cases} \tag{24}$$

Proof. The proof is similar to that of Proposition 9 in Petrik et al. (2016). □

Lemma 2 (Safe policy improvement of $\pi^{\odot}_{spibb}$ over any policy $\pi \in \Pi_b$)**.** *Let* Π_b *be the set of policies under the constraint of following* π_b *when* $(x,a) \in \mathfrak{B}$*. Let* $\pi^{\odot}_{spibb}$ *be a* Π_b*-optimal policy of the reward maximization problem of an estimated MDP* $\widehat{M}$*. Then, for any policy* $\pi \in \Pi_b$*, the difference of performance between* $\pi^{\odot}_{spibb}$ *and* π *is bounded as follows with high probability* $1-\delta$ *in the true MDP* M^**:*

$$\rho(\pi^{\odot}_{spibb}, M^*) - \rho(\pi, M^*) \geq \rho(\pi^{\odot}_{spibb}, \widehat{M}) - \rho(\pi, \widehat{M}) - \frac{4\epsilon V_{max}}{1-\gamma}. \tag{25}$$

Proof. We transform the true MDP M^* and the MDP estimate $\widehat{M}$, to their bootstrapped semi-MDP counterparts $\ddot{M}^*$ and the MDP estimate $\widehat{\ddot{M}}$. In these semi-MDPs, the actions $\mathcal{A}$ are replaced by options $\Omega_{\mathcal{A}} = \{o_a\}_{a \in \mathcal{A}}$ constructed as follows:

$$o_a = \langle \mathcal{I}_a, a{:}\pi_b, \beta \rangle = \begin{cases} \mathcal{I}_a = \{x \in \mathcal{X}, \text{ such that } (x,a) \notin \mathfrak{B}\} \\ a{:}\tilde{\pi}_b = \text{ perform } a \text{ at initialization, then follow } \tilde{\pi}_b \\ \beta(x) = \|\dot{\pi}_b(x, \cdot)\|_1 \end{cases} \tag{26}$$

where π_b has been decomposed as the aggregation of two partial policies: $\pi_b = \dot{\pi}_b + \tilde{\pi}_b$, with $\dot{\pi}_b(a|x)$ containing the non-bootstrapped actions probabilities in state x, and $\tilde{\pi}_b(a|x)$ the bootstrapped actions probabilities:

$$\forall a \in \mathcal{A}, \begin{cases} \dot{\pi}(a|x) = \pi(a|x) & \text{if } (x,a) \notin \mathfrak{B} \\ \dot{\pi}(a|x) = 0 & \text{if } (x,a) \in \mathfrak{B} \end{cases} \tag{27}$$

$$\forall a \in \mathcal{A}, \begin{cases} \tilde{\pi}(a|x) = \pi(a|x) & \text{if } (x,a) \in \mathfrak{B} \\ \tilde{\pi}(a|x) = 0 & \text{if } (x,a) \notin \mathfrak{B} \end{cases} \tag{28}$$

Let $\ddot{\Pi}$ denote the set of policies over the bootstrapped semi MDPs. $\mathcal{I}_a$ is the initialization function: it determines the set of states where the option is available. $a{:}\pi_b$ is the option policy being followed during the length of the option. Finally, $\beta(x)$ is the termination function defining the probability of the option to terminate in each state.

Notice that all options have the same termination function. Please, also notice that some states might have no available options, but this is okay since every option has a termination function equal to 0 in those states, meaning that they are unreachable. This to avoid being in this situation at the beginning of the trajectory, we use the notion of starting option: a trajectory starts with a void option $o_{\emptyset} = \langle \{x_0\}, \pi_b, \beta \rangle$.

By construction $x \in \mathcal{I}_a$ if and only if $(x,a) \notin \mathfrak{B}$, *i.e.* if and only if the condition on the state-action counts of Proposition 1 is fulfilled[1]. Also, any policy $\pi \in \Pi_b$ is implemented by a policy $\ddot{\pi} \in \ddot{\Pi}$ in a bootstrapped semi-MDP. Reciprocally, any policy $\ddot{\pi} \in \ddot{\Pi}$ admits a policy $\pi \in \Pi_b$ in the original MDP.

Note also, that by construction, the transition and reward functions are only defined for (x, o_a) pairs such that $x \in \mathcal{I}_a$. By convention, we set them to 0 for the other pairs. Their corresponding Q-functions are therefore set to 0 as well.

[1] Also, note that there is the requirement here that the trajectories are generated under policy π_b, so that the options are consistent with the dataset.

This means that Lemma 1 may be applied with $\pi = \pi^{\odot}_{spibb}$ and $M_1 = \ddot{M}^*$ and $M_2 = \widehat{\ddot{M}}$. We have:

$$
\begin{aligned}
|\rho(\pi^{\odot}_{spibb}, M^*) - \rho(\pi^{\odot}_{spibb}, \widehat{M})| &= |\rho(\pi^{\odot}_{spibb}, \ddot{M}^*) - \rho(\pi^{\odot}_{spibb}, \widehat{\ddot{M}})| && (29)\\
&= |V^{\pi^{\odot}_{spibb}}_{\ddot{M}^*}(x_0) - V^{\pi^{\odot}_{spibb}}_{\widehat{\ddot{M}}}(x_0)| && (30)\\
&= |Q^{\pi^{\odot}_{spibb}}_{\ddot{M}^*}(x_0, o_\emptyset) - Q^{\pi^{\odot}_{spibb}}_{\widehat{\ddot{M}}}(x_0, o_\emptyset)| && (31)\\
&\leq \frac{2\epsilon V_{max}}{1-\gamma} && (32)
\end{aligned}
$$

Analogously to 32, for any $\pi \in \Pi_b$, we also have:

$$
|\rho(\pi, M^*) - \rho(\pi, \widehat{M})| \leq \frac{2\epsilon V_{max}}{1-\gamma} \tag{33}
$$

Thus, we may write:

$$
\rho(\pi^{\odot}_{spibb}, M^*) - \rho(\pi, M^*) \geq \rho(\pi^{\odot}_{spibb}, \widehat{M}) - \rho(\pi, \widehat{M}) - \frac{4\epsilon V_{max}}{1-\gamma}, \tag{34}
$$

where the inequality is directly obtained from equations 32 and 33. □

Theorem 1 (Convergence of Π_b-SPIBB)**.** *Π_b-SPIBB converges to a policy $\pi^{\odot}_{spibb}$ that is Π_b-optimal in the MLE MDP $\widehat{M}$.*

Proof. We use the same transformation of $\widehat{M}$ as in Lemma 2. Then, the problem is cast without any constraint in a well defined semi-MDP, and Policy Iteration is known to converge in semi-MDPs to the policy optimizing the value function (Gosavi, 2004). □

Theorem 2 (Safe policy improvement of Π_b-SPIBB)**.** *Let Π_b be the set of policies under the constraint of following π_b when $(x, a) \in \mathfrak{B}$. Then, $\pi^{\odot}_{spibb}$ is a ζ-approximate safe policy improvement over the baseline π_b with high probability $1-\delta$, with:*

$$
\zeta = \frac{4V_{max}}{1-\gamma}\sqrt{\frac{2}{N_\wedge}\log\frac{2|\mathcal{X}||\mathcal{A}|2^{|\mathcal{X}|}}{\delta}} - \rho(\pi^{\odot}_{spibb}, \widehat{M}) + \rho(\pi_b, \widehat{M}) \tag{35}
$$

Proof. It is direct to observe that $\pi_b \in \Pi_b$, and therefore that Lemma 2 can be applied to π_b. We infer that, with high probability $1-\delta$:

$$
\rho(\pi^{\odot}_{spibb}, M^*) - \rho(\pi_b, M^*) \geq \rho(\pi^{\odot}_{spibb}, \widehat{M}) - \rho(\pi_b, \widehat{M}) - \frac{4\epsilon V_{max}}{1-\gamma}. \tag{36}
$$

with:

$$
\epsilon = \sqrt{\frac{2}{N_\wedge}\log\frac{2|\mathcal{X}||\mathcal{A}|2^{|\mathcal{X}|}}{\delta}} \tag{37}
$$

Therefore, we obtain:

$$
\begin{aligned}
\zeta &= \frac{4\epsilon V_{max}}{1-\gamma} - \Big(\rho(\pi^{\odot}_{spibb}, \widehat{M}) - \rho(\pi_b, \widehat{M})\Big) && (38)\\
&= \frac{4V_{max}}{1-\gamma}\sqrt{\frac{2}{N_\wedge}\log\frac{|\mathcal{X}||\mathcal{A}|2^{|\mathcal{X}|}}{\delta}} - \rho(\pi^{\odot}_{spibb}, \widehat{M}) + \rho(\pi_b, \widehat{M}) && (39)
\end{aligned}
$$

Quod erat demonstrandum. □

Theorem 3. *In finite MDPs, Equation 9 admits a unique fixed point that coincides with the Q-value of the policy trained with model-based Π_b-SPIBB.*

Proof. Unicity of the fixed point is a classical result in RL, obtained from the fact that the Bellman operator is a contraction.

Let π_t denote the policy trained with model-based Π_b-SPIBB. By construction, we know that π_t satisfies the optimal Bellman equation in the MLE MDP, under the Π_b-constraint:

$$Q^{\pi_t}_{\widehat{M}} = \widehat{R} + \gamma Q^{\pi_t}_{\widehat{M}} \pi_t \widehat{P} \tag{40}$$

Moreover, π_t may be decomposed by its component $\tilde{\pi}_t$ on $\mathfrak{B}$ and its complementary $\dot{\pi}_t$:

$$\pi_t(a|x) = \tilde{\pi}_t(a|x) + \dot{\pi}_t(a|x) \tag{41}$$

$$\text{with: } \left\{ \begin{array}{l} \tilde{\pi}_t(a|x) = \left\{ \begin{array}{l} \pi_b(a|x) \text{ if } (x,a) \in \mathfrak{B} \\ 0 \text{ otherwise} \end{array} \right. \\ \dot{\pi}_t(a|x) = \left\{ \begin{array}{l} \sum_{a'|(x,a') \notin \mathfrak{B}} \pi_b(a'|x) \text{ if } a = \text{argmax}_{a'|(x,a') \notin \mathfrak{B}} Q^{\pi_t}_{\widehat{M}}(x,a) \\ 0 \text{ otherwise} \end{array} \right. \end{array} \right. \tag{42}$$

As a consequence, we obtain:

$$\begin{aligned}
Q^{\pi_t}_{\widehat{M}}(x,a) &= \widehat{R}(x,a) + \gamma \sum_{x' \in \mathcal{X}} \sum_{a' \in \mathcal{A}} Q^{\pi_t}_{\widehat{M}}(x',a') \pi_t(a'|x') \widehat{P}(x'|x,a) && (43) \\
&= \widehat{R}(x,a) + \gamma \sum_{x' \in \mathcal{X}} \sum_{a' \in \mathcal{A}} Q^{\pi_t}_{\widehat{M}}(x',a') \left(\tilde{\pi}_t(a'|x') + \dot{\pi}_t(a'|x') \right) \widehat{P}(x'|x,a) && (44) \\
&= \widehat{R}(x,a) + \gamma \sum_{x' \in \mathcal{X}} \widehat{P}(x'|x,a) \left[\sum_{a'|(x',a') \in \mathfrak{B}} \pi_b(a'|x') Q^{\pi_t}_{\widehat{M}}(x',a') \right] && (45) \\
&\quad + \gamma \sum_{x' \in \mathcal{X}} \widehat{P}(x'|x,a) \left[\left(\sum_{a'|(x',a') \notin \mathfrak{B}} \pi_b(a'|x') \right) \max_{a'|(x',a') \notin \mathfrak{B}} Q^{\pi_t}_{\widehat{M}}(x',a') \right] \\
&= \frac{\sum_{\langle x_j = x, a_j = a, r_j, x'_j \rangle \in \mathcal{D}} r_j}{N_\mathcal{D}(x,a)} + \gamma \sum_{x' \in \mathcal{X}} \frac{\sum_{\langle x_j = x, a_j = a, r_j, x'_j = x' \rangle \in \mathcal{D}} 1}{N_\mathcal{D}(x,a)} \left[\sum_{a'|(x',a') \in \mathfrak{B}} \pi_b(a'|x') Q^{\pi_t}_{\widehat{M}}(x',a') \right] && (46) \\
&\quad + \gamma \sum_{x' \in \mathcal{X}} \frac{\sum_{\langle x_j = x, a_j = a, r_j, x'_j = x' \rangle \in \mathcal{D}} 1}{N_\mathcal{D}(x,a)} \left[\left(\sum_{a'|(x',a') \notin \mathfrak{B}} \pi_b(a'|x') \right) \max_{a'|(x',a') \notin \mathfrak{B}} Q^{\pi_t}_{\widehat{M}}(x'_j,a') \right]
\end{aligned}$$

where $\sum_{\langle x_j = x, a_j = a, r_j, x'_j \rangle \in \mathcal{D}}$ denotes the sum over all transitions in the dataset that start from the state-action pair (x,a) and $\sum_{\langle x_j = x, a_j = a, r_j, x'_j = x' \rangle \in \mathcal{D}}$ is the sum over all transitions that start from the state-action pair (x,a) and transition to x'.

We then see that:

$$\begin{aligned}
Q^{\pi_t}_{\widehat{M}}(x,a) &= \frac{\displaystyle\sum_{\langle x_j=x,a_j=a,r_j,x'_j\rangle\in\mathcal{D}} r_j}{N_\mathcal{D}(x,a)} + \frac{\gamma}{N_\mathcal{D}(x,a)} \sum_{\langle x_j=x,a_j=a,r_j,x'_j\rangle\in\mathcal{D}} \sum_{a'|(x'_j,a')\in\mathfrak{B}} \pi_b(a'|x'_j) Q^{\pi_t}_{\widehat{M}}(x'_j,a') \qquad (47)\\
&\quad + \frac{\gamma}{N_\mathcal{D}(x,a)} \sum_{\langle x_j=x,a_j=a,r_j,x'_j\rangle\in\mathcal{D}} \left(\sum_{a'|(x'_j,a')\notin\mathfrak{B}} \pi_b(a'|x'_j)\right) \max_{a'|(x'_j,a')\notin\mathfrak{B}} Q^{\pi_t}_{\widehat{M}}(x'_j,a')\\
&= \frac{1}{N_\mathcal{D}(x,a)} \sum_{\langle x_j=x,a_j=a,r_j,x'_j\rangle\in\mathcal{D}} \Bigg[r_j + \gamma \sum_{a'|(x'_j,a')\in\mathfrak{B}} \pi_b(a'|x'_j) Q^{\pi_t}_{\widehat{M}}(x'_j,a') \qquad (48)\\
&\qquad + \gamma \left(\sum_{a'|(x'_j,a')\notin\mathfrak{B}} \pi_b(a'|x'_j)\right) \max_{a'|(x'_j,a')\notin\mathfrak{B}} Q^{\pi_t}_{\widehat{M}}(x'_j,a') \Bigg]\\
&= \frac{1}{N_\mathcal{D}(x,a)} \sum_{\langle x_j=x,a_j=a,r_j,x'_j\rangle\in\mathcal{D}} y_j^{\pi_t} \text{ when } N_\mathcal{D}(x,a) > 0 \text{ and is undefined otherwise.} \qquad (49)
\end{aligned}$$

This concludes the proof that $Q^{\pi_t}_{\widehat{M}}$ is the fixed point of Equation 9. □

A.4. Algorithms for the greedy projection of $Q^{(i)}$ on Π_b and $\Pi_{\leq b}$

The policy-based SPIBB algorithms rely on a policy iteration process that requires a policy improvement step under the constraint that the generated policy belongs to Π_b or $\Pi_{\leq b}$. Those are respectively described in Algorithms 1 (main document) and 2 (see below).

Algorithm 2 Greedy projection of $Q^{(i)}$ on $\Pi_{\leq b}$

```
Input: Baseline policy $\pi_b$
Input: Last iteration value function $Q^{(i)}$
Input: Set of bootstrapped state-action pairs $\mathfrak{B}$
Input: Current state $x$ and action set $\mathcal{A}$
Sort $\mathcal{A}$ in decreasing order of the action values: $Q^{(i)}(x,a)$
Initialize $\pi^{(i)}_{spibb} = 0$
for $a \in \mathcal{A}$ do
    if $(x,a) \in \mathfrak{B}$ then
        if $\pi_b(a|x) \geq 1 - \sum_{a'\in\mathcal{A}} \pi^{(i)}_{spibb}(a'|x)$ then
            $\pi^{(i)}_{spibb}(a|x) = 1 - \sum_{a'\in\mathcal{A}} \pi^{(i)}_{spibb}(a'|x)$
            return $\pi^{(i)}_{spibb}$
        else
            $\pi^{(i)}_{spibb}(a|x) = \pi_b(a|x)$
        end
    else
        $\pi^{(i)}_{spibb}(a|x) = 1 - \sum_{a'\in\mathcal{A}} \pi^{(i)}_{spibb}(a'|x)$
        return $\pi^{(i)}_{spibb}$
    end
end
```

A.5. Comprehensive illustration of the difference between Π_b-SPIBB and $\Pi_{\leq b}$-SPIBB policy improvement steps

Table 1 illustrates the difference between Π_b-SPIBB and $\Pi_{\leq b}$-SPIBB in the policy improvement step of the policy iteration process. It shows how the baseline probability mass is locally redistributed among the different actions for the two policy-based SPIBB algorithms. We observe that for Π_b-SPIBB, the bootstrapped state-action pairs probabilities remain untouched whatever their Q-value estimates are. On the contrary, $\Pi_{\leq b}$-SPIBB removes all mass from the bootstrapped state-action pairs that are performing worse than the current Q-value estimates.

Table 1. Policy improvement step at iteration (i) for the two policy-based SPIBB algorithms.

Q-value estimate	Baseline policy	Boostrapping	Π_b-SPIBB	$\Pi_{\leq b}$-SPIBB
$Q^{(i)}_{\widehat{M}}(x, a_1) = 1$	$\pi_b(a_1\|x) = 0.1$	$(x, a_1) \in \mathfrak{B}$	$\pi^{(i+1)}(a_1\|x) = 0.1$	$\pi^{(i+1)}(a_1\|x) = 0$
$Q^{(i)}_{\widehat{M}}(x, a_2) = 2$	$\pi_b(a_2\|x) = 0.4$	$(x, a_2) \notin \mathfrak{B}$	$\pi^{(i+1)}(a_2\|x) = 0$	$\pi^{(i+1)}(a_2\|x) = 0$
$Q^{(i)}_{\widehat{M}}(x, a_3) = 3$	$\pi_b(a_3\|x) = 0.3$	$(x, a_3) \notin \mathfrak{B}$	$\pi^{(i+1)}(a_3\|x) = 0.7$	$\pi^{(i+1)}(a_3\|x) = 0.8$
$Q^{(i)}_{\widehat{M}}(x, a_4) = 4$	$\pi_b(a_4\|x) = 0.2$	$(x, a_4) \in \mathfrak{B}$	$\pi^{(i+1)}(a_4\|x) = 0.2$	$\pi^{(i+1)}(a_4\|x) = 0.2$

B. Finite MDP Benchmark Design

B.1. Experiments details

B.1.1. Pseudo code for the Gridworld benchmark

Algorithm 3 Gridworld benchmark

```
Input: List of dataset size
Input: List of algorithms in the benchmark
Input: List of hyper-parameter values for each algorithm
repeat $10^5$ times
    for each dataset size do
        Generate a dataset. (see Section B.1.5)
        for each algorithm do
            for each algorithm hyper-parameter value do
                Train a policy. (see Sections 2.3 and B.2)
                Evaluate the policy. (see Section B.1.6)
                Record the performance of the trained policy.
            end
        end
    end
end
```

B.1.2. Pseudo code for the Random MDPs benchmark

Algorithm 4 Random MDPs benchmark

```
Input: List of hyper-parameter values for the baseline
Input: List of dataset size
Input: List of algorithms in the benchmark
Input: List of hyper-parameter values for each algorithm
repeat $10^5$ times
    Generate an MDP. (see Section B.1.3)
    for each hyper parameter value for the baseline do
        Generate a baseline. (see Section B.1.4)
        for each dataset size do
            Generate a dataset. (see Section B.1.5)
            for each algorithm do
                for each algorithm hyper-parameter value do
                    Train a policy. (see Sections 2.3 and B.2)
                    Evaluate the policy. (see Section B.1.6)
                    Record the performance of the trained policy.
                end
            end
        end
    end
end
```

B.1.3. MDP generation

We use three parameters for our MDP generation: the number of states, the number of actions in each state, and the connectivity of the transition function stating how many states are reachable after performing a given action in a given state. We tried out various values for those parameters and found little sensitivity in those preliminary experimental results and decided to fix their respective values to 25/4/4 in the reported experiments. The discount factor γ is set to 0.95.

The initial state is arbitrarily set to be x_0, then we search with dynamic programming the performance of the optimal policy for all potential terminal state $x_f \in \mathcal{X}/x_0$. We select the terminal state for which the optimal policy yields the smaller value function and set it as terminal: $R(x, a, x_f) = 1$ and $P(x|x_f, a) = 0$ for all $x \in \mathcal{X}$ and all $a \in \mathcal{A}$. The reward function is set to 0 everywhere else. We found that the optimal value-function is on average 0.6 and with a surprising low variance, which amounts to an average horizon of 10. Later on, we write this environmental MDP $M^* = \langle \mathcal{X}, \mathcal{A}, P^*, R^*, \gamma \rangle$, its optimal action-value function Q^*, its optimal performance $\rho^* = \rho(\pi^*, M^*)$, and its random policy performance $\widetilde{\rho} = \rho(\widetilde{\pi}, M^*)$, where $\widetilde{\pi}$ denotes the uniform random policy: $\widetilde{\pi}(a|x) = \frac{1}{|\mathcal{A}|}$ for all $x \in \mathcal{X}$ and all $a \in \mathcal{A}$.

B.1.4. Baseline generation

We use a hyper-parameter for the baseline generation:

$$\rho_b = \rho(\pi_b, M^*) = \eta \rho^* + (1 - \eta)\widetilde{\rho}. \tag{50}$$

Therefore, $\eta \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$ determines the performance of the baseline, normalized with respect to the performances of the random and the optimal performance. There are an infinite number of policies that yield this performance. We designed several heuristics to generate the actual baseline and again notice a moderate sensitivity in our preliminary results. All the reported results use the following heuristics which consists in two steps: softening and randomization.

Softening: We apply a softmax on Q^* with temperature such that $\rho(\pi_s, M^*) = \frac{\rho_b + \rho^*}{2}$, where π_s denotes the policy obtained after the softening operation.

Randomization: Until reaching the desired performance for the baseline we repeatedly apply the following process: we randomly select a state x, and move a 0.1 probability mass from $a^* = \text{argmax}_{a \in \mathcal{A}} Q^*(x, a)$ to another random action. When this loop stops, the output is the baseline π_b.

B.1.5. Dataset generation

The dataset generation depends a single parameter $|\mathcal{D}| \in \{10, 20, 50, 100, 200, 500, 1000, 2000\}$ ($\cup \{5000, 10000\}$ for the Gridworld experiments): its size expressed in the number of trajectories. A trajectory generation simply consists in sampling the environment and the baseline policy until reaching the final state. The output is the dataset $\mathcal{D}$.

B.1.6. Trained policy evaluation

In the Random MDPs experiments, we use different MDPs and baselines for each run. We need a standardized method for evaluating the trained policy π. We use the performance normalized with respect to the baseline and optimal policies:

$$\rho = \frac{\rho(\pi, M^*) - \rho_b}{\rho^* - \rho_b} \leq 1. \tag{51}$$

Then, the results are analyzed with respect to ρ as everywhere else in the paper: according to the mean and CVaR performances.

B.1.7. Mean and CVaR performance

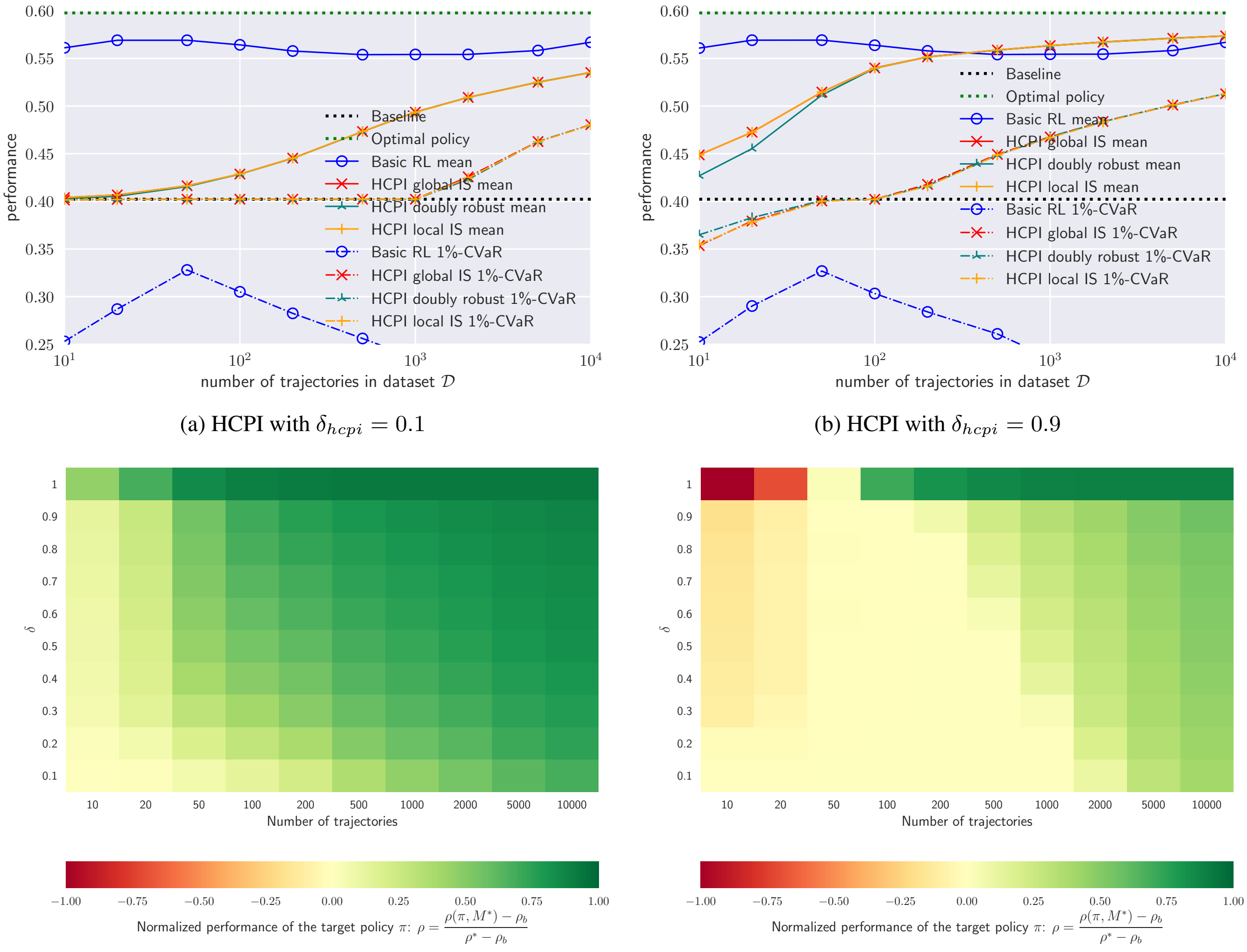
The mean performance is simply the average of performance over all the runs.

The X%-CVaR performance is the average performance of the X% worst runs. To compute this, we sort the performance of all the runs, and keep the lowest X% fraction and then take the average. The 100%-CVaR performance is obviously equivalent to the mean performance.

B.1.8. Figures

We present three types of figure in the paper (main document and appendix).

Performance vs. dataset size: These figures (for instance Figure 5(a)) show the (mean and/or CVaR) performance of the algorithms as a function of the dataset size.



(a) HCPI with $\delta_{hcpi} = 0.1$

(b) HCPI with $\delta_{hcpi} = 0.9$

(c) Mean performance HCPI doubly-robust heatmap

(d) 1%-CVaR performance HCPI doubly robust heatmap

Figure 5. HCPI hyper-parameter search results on the Gridworld domain.

Hyper-parameter search heatmaps: These figures (for instance Figure 5(c)) show the (mean or CVaR) normalized performance of the algorithms as a function of both the dataset size and the hyper-parameter value of the evaluated algorithm. The normalized performance is computed with Equation 51 and represented with colour. Red means that the performance is worse than that of the baseline, yellow means that it is equal and green means that it improves the baseline.

Random MDPs heatmaps: These figures (for instance Figure 7(f)) are very similar to the other heatmaps except that the normalized performance is shown as a function of both the dataset size and the hyper-parameter η used for the baseline generation (instead of the hyper-parameter of the evaluated algorithm).

B.2. Other benchmark algorithms: competitors

Since the *baseline* meaning is overridden in this paper, we refer to the non-SPIBB benchmark algorithms with the term *competitors*.

B.2.1. Basic RL

Basic RL is implemented by computing the MLE MDP and solving it with dynamic programming. In order to cover the state-action pairs absent from the dataset, two Q initializations were investigated in our experiments: optimistic (V_{max}), and pessimistic ($-V_{max}$). The former yields awful performances in our batch RL setting. This is not surprising because optimism makes it imprudently explore every unknown state-action pairs. All the presented results were therefore obtained with the pessimistic initialization as in Jiang & Li (2015).

B.2.2. HCPI

Safe policy improvement in a model-free setting is closely related to High Confidence Off-policy evaluation (Thomas et al., 2015b). Instead of relying on the model uncertainty, this class of methods relies on a high-confidence lower bound on the Importance Sampling (IS) estimate of the trained policy performance. Given a dataset $\mathcal{D}$, a part of it, $\mathcal{D}_{train}$, is used to derive a set of candidates policies. A policy π_t is first derived using an off policy reinforcement learning algorithm (Q-learning for instance) and is regularized using the baseline to obtain a set of candidate policies $\Pi_{candidates} = \{((1-\alpha)\pi_t + \alpha\pi_b), \alpha \in \{0, 0.1, 0.2, 0.3, ...1\}\}$. The remaining data $\mathcal{D}_{test}$ are used to evaluate the candidate policies. The policy with the highest lower bound on the estimated performance is returned. Thomas et al. (2015a) introduced three ways of obtaining the lower bound on the estimate.

- The first one is an extension of Maurer and Pontils empirical Bernstein inequality. Let $X_1, ...X_n$ be n independent real-valued random variables, such that for each $i \in \{1, ..., n\}$, we have $\mathbb{P}[0 \leq X_i] = 1$, $\mathbb{E}[X_i] \leq \nu$ and some threshold value $c_i \geq 0$. Let $\delta \geq 0$ and $Y_i = min\{X_i, x_i\}$. Then with probability at least $1-\delta$, we have:

$$\mu \geq \left(\sum_{i=1}^{n} \frac{1}{c_i}\right)^{-1} \sum_{i=1}^{n} \frac{Y_i}{c_i} - \left(\sum_{i=1}^{n} \frac{1}{c_i}\right)^{-1} \frac{7nln(\frac{2}{\delta})}{3(n-1)} - \left(\sum_{i=1}^{n} \frac{1}{c_i}\right)^{-1} \sqrt{\frac{ln(\frac{2}{\delta})}{n-1} \sum_{i,j=1}^{n} \left(\frac{Y_i}{c_i} - \frac{Y_j}{c_j}\right)^2}$$

  In the SPI setting, X_i is the unbiased estimate of the return related to each trajectory. The drawback of this method is the hyper-parameter c_i which needs to be tuned.

- The second method is based on the assumption that the mean return is normally distributed. Relying on this assumption, a less conservative lower bound can be obtained using Students t-test (with the same notations):

$$\mathbb{E}[X_i] \geq \frac{1}{n} \sum_{i=1}^{n} X_i - \frac{\sigma}{\sqrt{n}} t_{1-\delta, n-1}$$

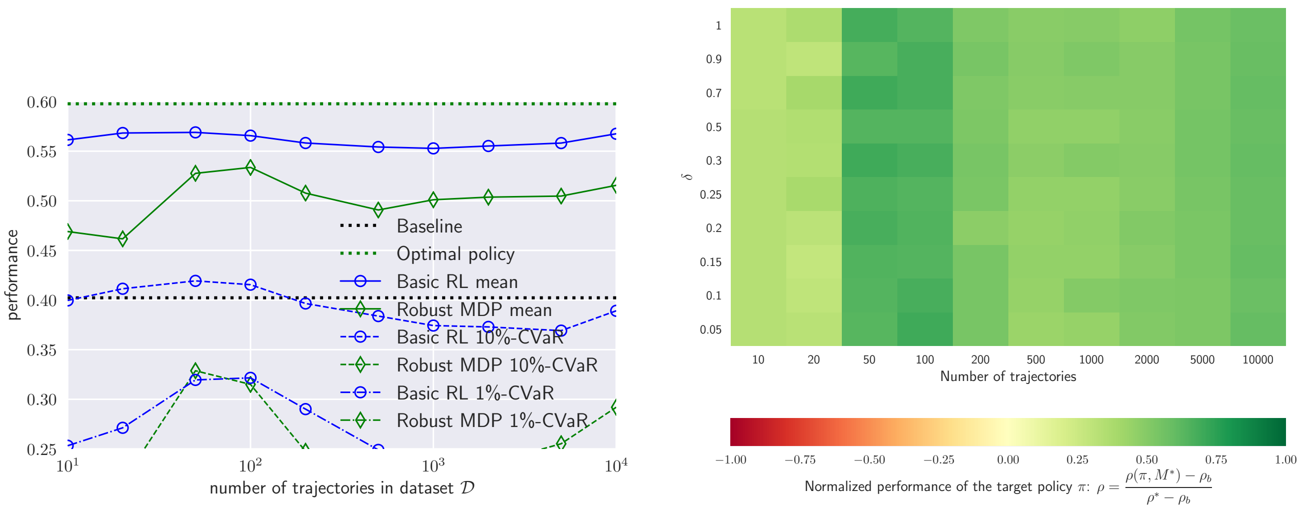
  with $\sigma = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} \left(X_i - \hat{X}_i)^2\right)}$ the sample standard deviation of $X_1, .., X_n$ with Bessel's correction.

- The last one is based on Efrons Bootstrap methods (Efron, 1987). It relies on bootstrapping to estimate the true distribution of the mean return instead of considering it as normally distributed.

In practice, the first method is too conservative and the third one is not computationally efficient. Therefore we limit our study to the second one, which relies on Student's t-test.
We implemented three versions of HCPI: with global importance sampling, with local importance sampling, and with the doubly robust method. As Figures 5(a) and 5(b) reveal, they all behave more or less the same on the Gridworld domain. We also searched for the best hyper-parameter $\delta_{hcpi} \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$ value. Figures 5(c) and 5(d) respectively display the mean and 1%-CVaR performances. One can observe that $\delta_{hcpi} = 1$ yields the best result in mean, but turns out to be strongly unsafe for small datasets. $\delta_{hcpi} = 0.9$ appears to offer the best compromise and this is the value we retain for the experiments reported in the main document. Note that those δ_{hcpi} values mean that the confidence is very small: 0.1 for $\delta_{hcpi} = 0.9$, and even null for $\delta_{hcpi} = 1$.

B.2.3. ROBUST MDP

Robust MDP also relies on a confidence hyperparameter δ_{rob}. We observe that the behaviour of the algorithm is not much dependent on $\delta_{rob} \in \{0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.5, 0.7, 0.9, 1\}$, and that it always fall back on the baseline when the dataset is under 50,000 trajectories. We observe also on Figure 6(a) that, for the smaller datasets we do our benchmark on, independently from the safety set, the policy trained with the Robust MDP algorithm, which is the best policy in the worst-case MDP, is worse that the policy trained with Basic RL on mean and also on CVaR. 1%-CVaR even falls down out of the figure. We interpret this as the fact that, in the Gridworld domain, there is a zone where all the states have been



(a) Robust MDP with $\delta_{rob} = 0.1$ (no safety test)

(b) Mean performance Robust MDP heatmap (no safety test)

Figure 6. Robust MDP hyper-parameter search results on the Gridworld domain.

experienced a reasonable amount of time, and where the algorithm infers that the outcome is well known: 0 reward. Near the goal, on the contrary, there are some states where there is a risk to go because of the stochastic transitions that are largely unknown. This behaviour seem to reproduce also frequently on the Random MDPs domain. Figure 6(b) displays the mean performance for a large set of δ_{rob} values without the safety test. The figures of Robust MDP with the safety test are omitted because it always fails and therefore the algorithm always outputs the baseline. On the main document figures, we report the Robust MDP performance without safety test for $\delta_{rob} = 0.1$.

B.2.4. Reward-adjusted MDP

The theory developed in Petrik et al. (2016) states that the reward should be adjusted as follows:

$$\widetilde{R}(x,a) \leftarrow R^*(x,a) - \frac{\gamma R_{max}}{1-\gamma} e(x,a), \tag{52}$$

where $R^*(x,a)$ is the true reward function, that they assume to be known, and $e(x,a)$ is the error function on the dynamics, with bounded with concentration bounds as in our Proposition 1:

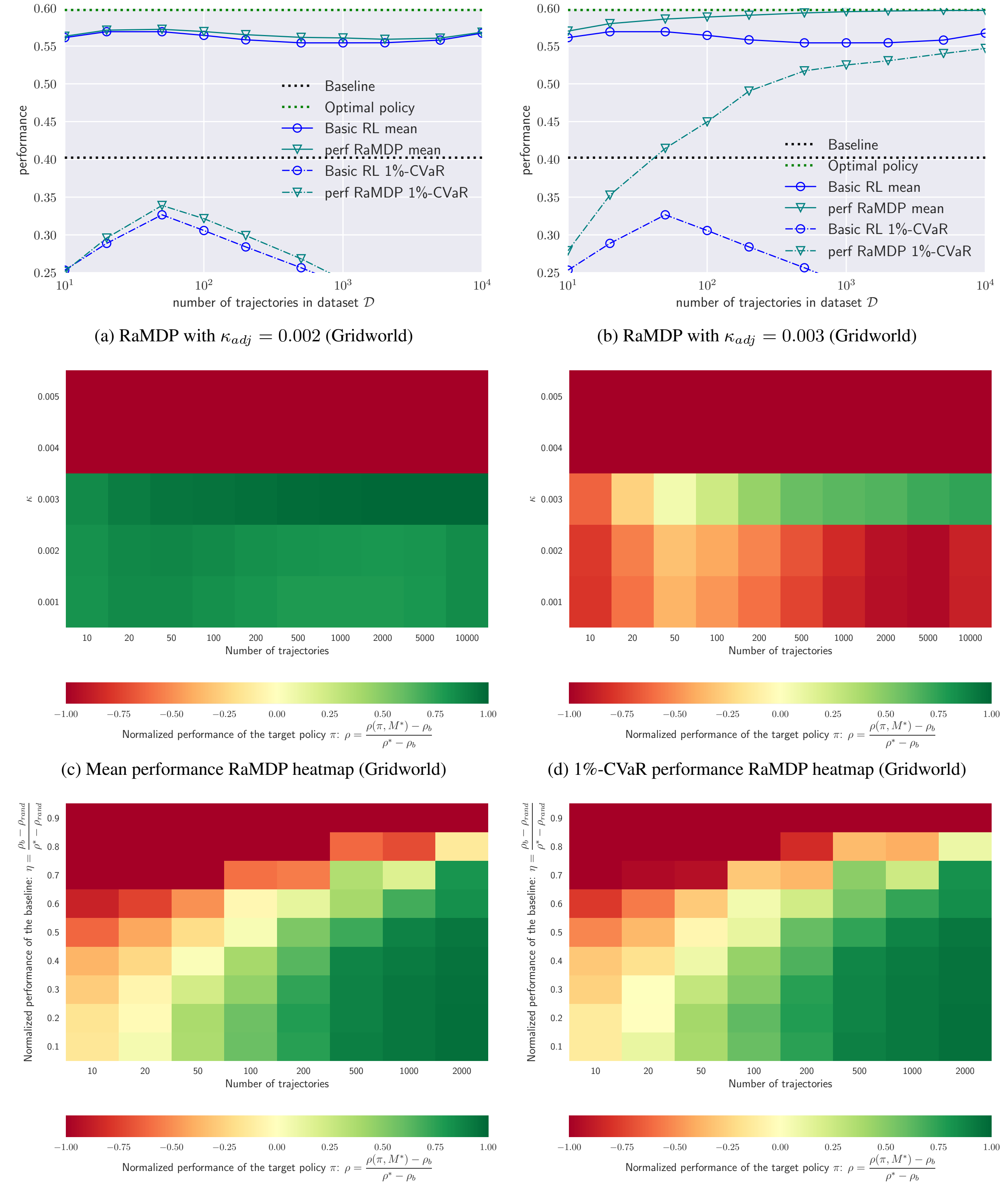
$$e(x,a) \leq \sqrt{\frac{2}{N_\mathcal{D}(x,a)} \log \frac{2|\mathcal{X}||\mathcal{A}|2^{|\mathcal{X}|}}{\delta_{adj}}} \tag{53}$$

Also, we do not assume that $R^*(x,a)$ is known in our experiments and there is consequently a γ factor disappearing. We obtain:

$$\begin{aligned} \widetilde{R}(x,a) &\leftarrow \widehat{R}(x,a) - \frac{R_{max}}{1-\gamma}\sqrt{\frac{2}{N_\mathcal{D}(x,a)} \log \frac{2|\mathcal{X}||\mathcal{A}|2^{|\mathcal{X}|}}{\delta_{adj}}} && (54) \\ &\leftarrow \widehat{R}(x,a) - \frac{100}{\sqrt{N_\mathcal{D}(x,a)}}, && (55) \end{aligned}$$

with our domain parameters and the choice of $\delta_{adj} = 0.1$. Instead, we consider the following hyper-parametrization:

$$\widetilde{R}(x,a) \leftarrow \widehat{R}(x,a) - \frac{\kappa_{adj}}{\sqrt{N_\mathcal{D}(x,a)}}. \tag{56}$$



(a) RaMDP with $\kappa_{adj} = 0.002$ (Gridworld)

(b) RaMDP with $\kappa_{adj} = 0.003$ (Gridworld)

(c) Mean performance RaMDP heatmap (Gridworld)

(d) 1%-CVaR performance RaMDP heatmap (Gridworld)

(e) 1%-CVaR RaMDP heatmap with $\kappa_{adj} = 0.002$ (Random MDPs) (f) 1%-CVaR RaMDP heatmap with $\kappa_{adj} = 0.003$ (Random MDPs)

Figure 7. RaMDP hyper-parameter search results on the Gridworld and Random MDPs domains.

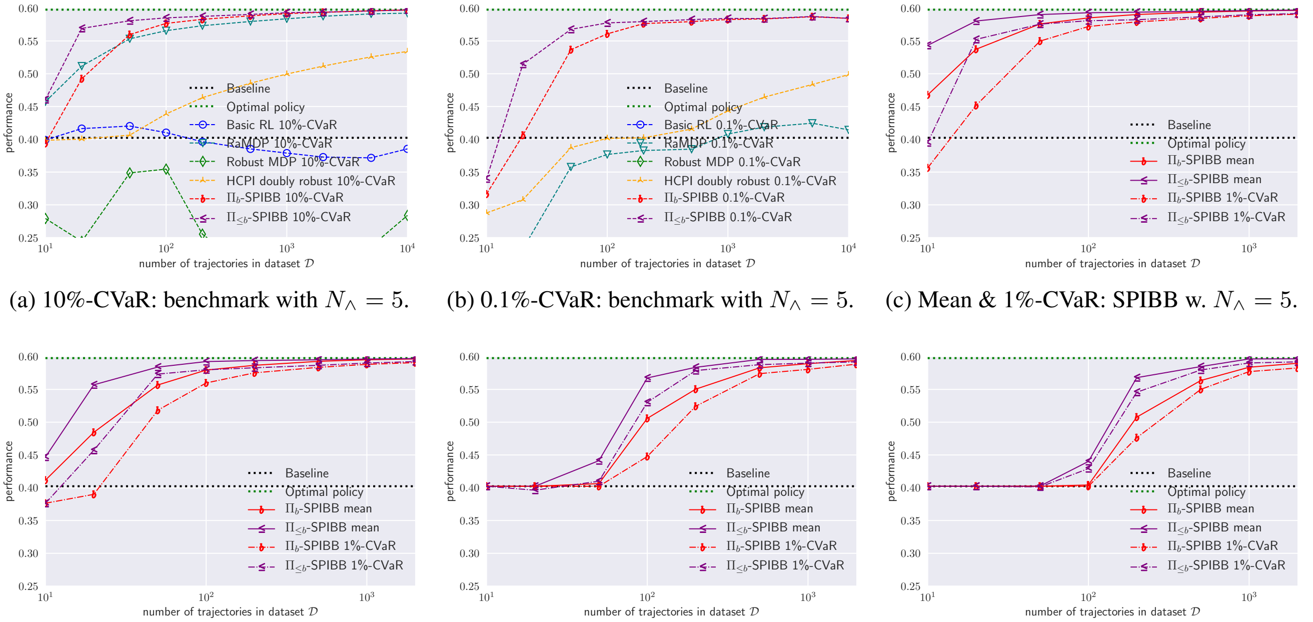
We perform a hyper-parameter seach for:

$$\kappa_{adj} \in \{0.0001, 0.0002, 0.0005, 0.001, 0.002, 0.003, 0.004, 0.005, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50, 100\}\,.$$

Figures 7(c) and 7(d) respectively show the mean and 1%-CVaR performance of RaMDP for $\kappa_{adj} \in \{0.001, 0.002, 0.003, 0.004, 0.005\}$. They reveal that for $\kappa_{adj} \geq 0.004$, RaMDP is overly frightened to go near the goal in the same way as with Robust MDP; and that for $\kappa_{adj} \leq 0.002$, RaMDP just ignores the penalty and yields results very close to the Basic RL's (see Figure 7(a)). In the middle, there is a tight spot ($\kappa_{adj} = 0.003$) where it works quite well on the Gridworld domain as may be seen on Figure 7(b), even though it is not safe for very small datasets. It has to be noted also that, in theory, RaMDP uses a safety test, which fails everytime similarly to that of Robust MDP. In addition to the sensitivity to the κ_{adj} parameter, on the Random MDPs benchmark, the unsafety of RaMDP is much more obvious (see Figures 7(e) and 7(f)), which tends us to think that the Gridworld domain is favorable to RaMDP. On the main document figures, we report the RaMDP performance without safety test for $\kappa_{adj} = 0.003$.

C. Extensive Empirical Results on Finite MDPs

C.1. Gridworld additional results

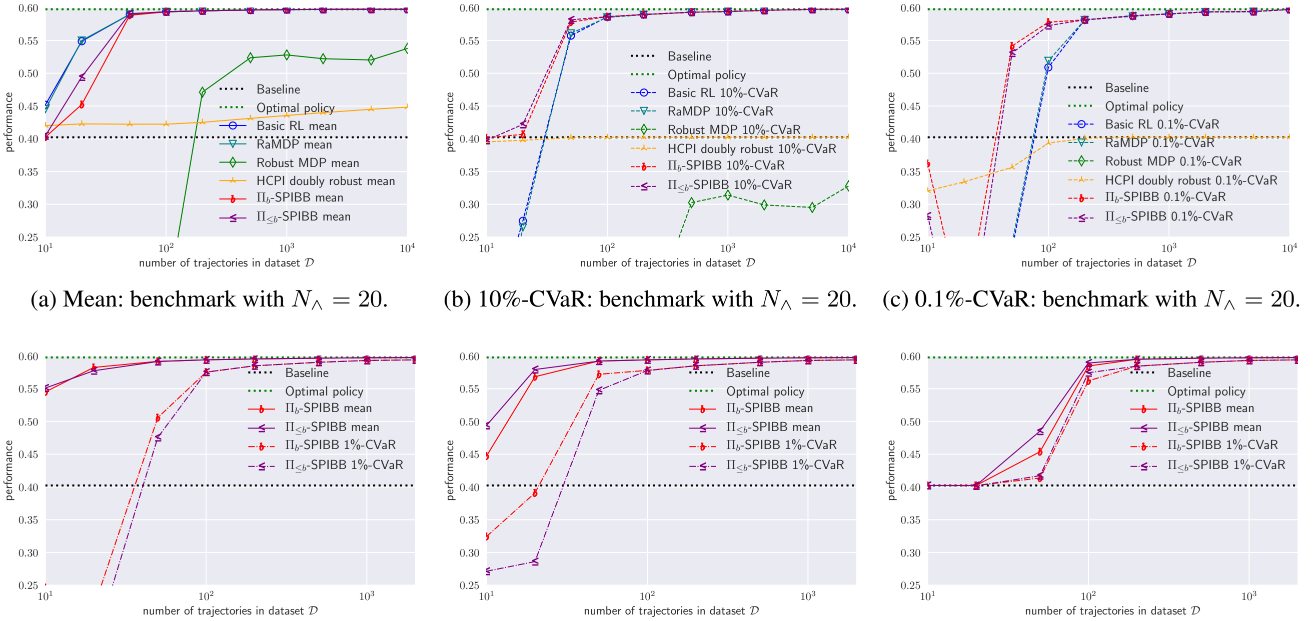


(a) 10%-CVaR: benchmark with $N_\wedge = 5$. (b) 0.1%-CVaR: benchmark with $N_\wedge = 5$. (c) Mean & 1%-CVaR: SPIBB w. $N_\wedge = 5$.

(d) Mean & 1%-CVaR: SPIBB w. $N_\wedge = 10$. (e) Mean & 1%-CVaR: SPIBB w. $N_\wedge = 50$. (f) Mean & 1%-CVaR: SPIBB w. $N_\wedge = 100$.

Figure 8. Gridworld experiment: Figures (a-b) respectively show the benchmark for the 10%-CVaR and 0.1%-CVaR performances. Figures (c-f) display additional curves for other $N_\wedge$ values: respectively 5, 10, 50, 100.

C.2. Gridworld full results with random behavioural policy

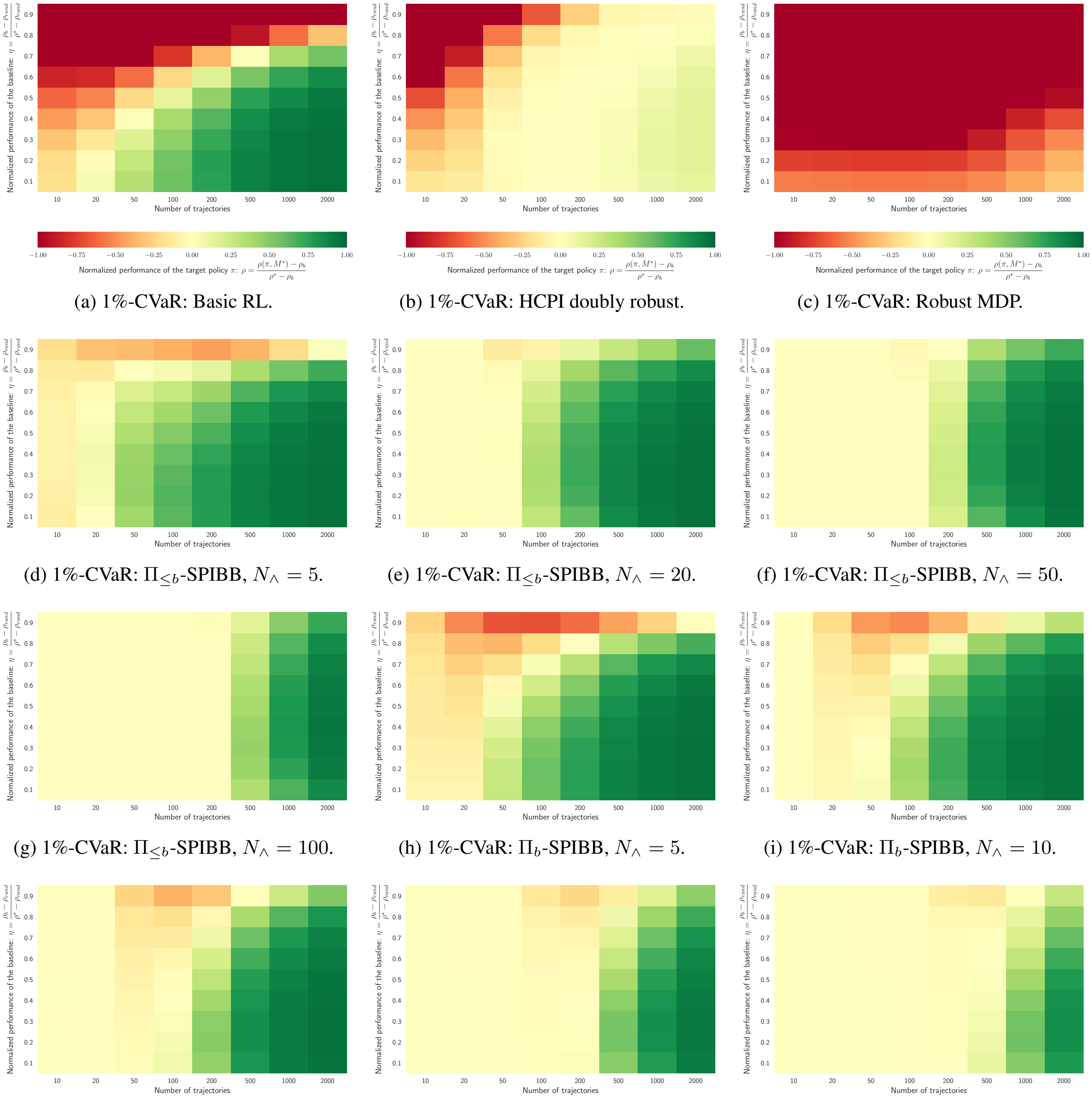


(a) Mean: benchmark with $N_\wedge = 20$. (b) 10%-CVaR: benchmark with $N_\wedge = 20$. (c) 0.1%-CVaR: benchmark with $N_\wedge = 20$.

(d) Mean & 1%-CVaR: SPIBB w. $N_\wedge = 5$. (e) Mean & 1%-CVaR: SPIBB w. $N_\wedge = 10$. (f) Mean & 1%-CVaR: SPIBB w. $N_\wedge = 50$.

Figure 9. Gridworld experiment with random behavioural policy: Figures (a-c) respectively show the benchmark for the mean, 10%-CVaR and 0.1%-CVaR performances. Figures (d-f) display additional curves for other $N_\wedge$ values: respectively 5, 10, 50.

C.3. Full Random MDPs experiment results



(a) 1%-CVaR: Basic RL.

(b) 1%-CVaR: HCPI doubly robust.

(c) 1%-CVaR: Robust MDP.

(d) 1%-CVaR: $\Pi_{\leq b}$-SPIBB, $N_\wedge = 5$.

(e) 1%-CVaR: $\Pi_{\leq b}$-SPIBB, $N_\wedge = 20$.

(f) 1%-CVaR: $\Pi_{\leq b}$-SPIBB, $N_\wedge = 50$.

(g) 1%-CVaR: $\Pi_{\leq b}$-SPIBB, $N_\wedge = 100$.

(h) 1%-CVaR: Π_b-SPIBB, $N_\wedge = 5$.

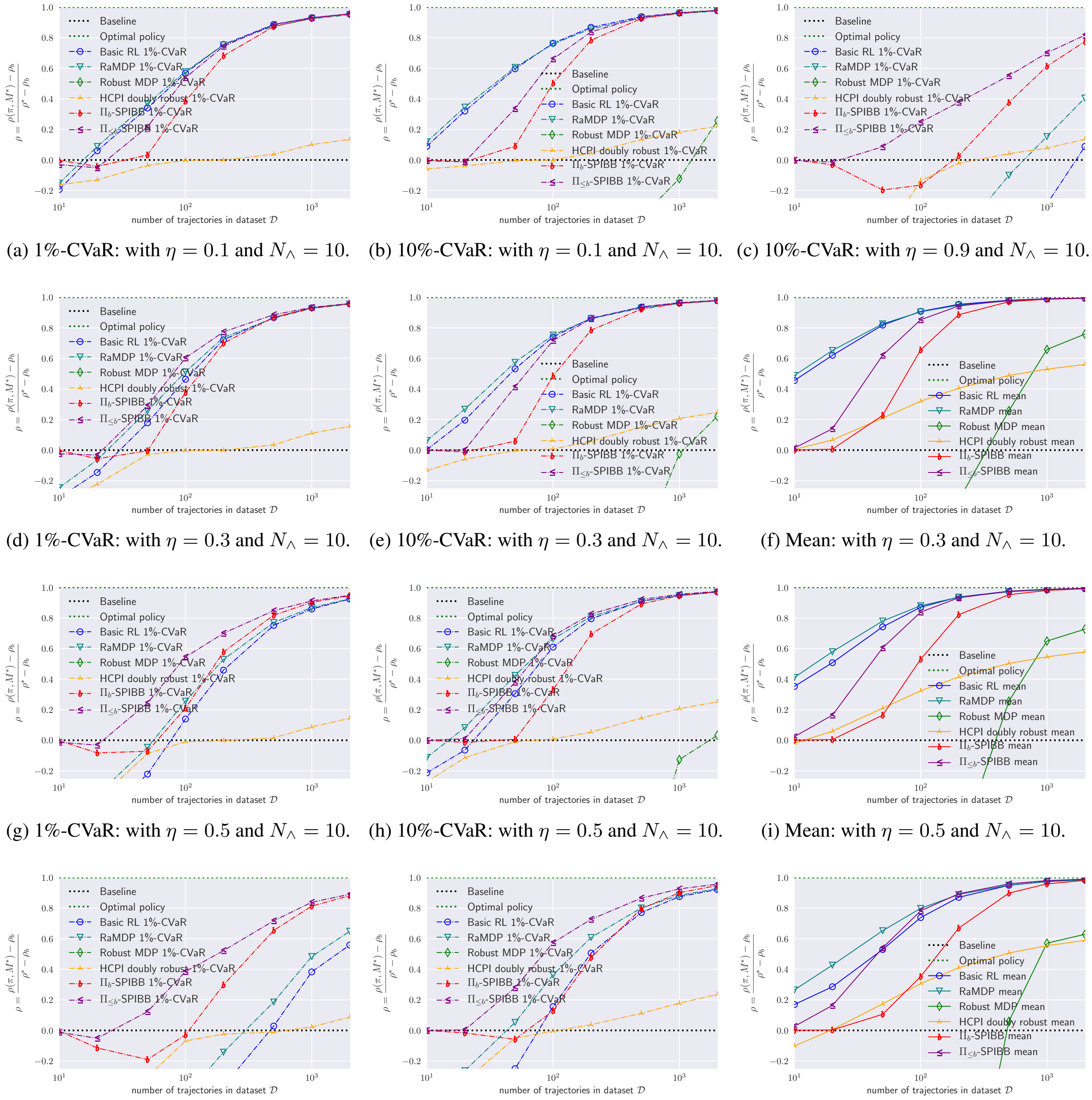
(i) 1%-CVaR: Π_b-SPIBB, $N_\wedge = 10$.

(j) 1%-CVaR: Π_b-SPIBB, $N_\wedge = 20$.

(k) 1%-CVaR: Π_b-SPIBB, $N_\wedge = 50$.

(l) 1%-CVaR: Π_b-SPIBB, $N_\wedge = 100$.

Figure 10. Random MDPs: 1%-CVaR performance heatmaps. The abscissae is the dataset size, the ordinate is the baseline hyperparameter η, and the color is the normalized performance: red, yellow, and green respectively mean below, equal to, and above baseline performance. Heatmaps for the mean normalized performance and for additional $N_\wedge$ values: 7, 15, 30, 70, may be found in the supplementary material package. The supplementary material package also contains more heatmaps on the sensitivity to $N_\wedge$, with fixed η values.



(a) 1%-CVaR: with $\eta = 0.1$ and $N_\wedge = 10$. (b) 10%-CVaR: with $\eta = 0.1$ and $N_\wedge = 10$. (c) 10%-CVaR: with $\eta = 0.9$ and $N_\wedge = 10$.

(d) 1%-CVaR: with $\eta = 0.3$ and $N_\wedge = 10$. (e) 10%-CVaR: with $\eta = 0.3$ and $N_\wedge = 10$. (f) Mean: with $\eta = 0.3$ and $N_\wedge = 10$.

(g) 1%-CVaR: with $\eta = 0.5$ and $N_\wedge = 10$. (h) 10%-CVaR: with $\eta = 0.5$ and $N_\wedge = 10$. (i) Mean: with $\eta = 0.5$ and $N_\wedge = 10$.

(j) 1%-CVaR: with $\eta = 0.7$ and $N_\wedge = 10$. (k) 10%-CVaR: with $\eta = 0.7$ and $N_\wedge = 10$. (l) Mean: with $\eta = 0.7$ and $N_\wedge = 10$.

Figure 11. Random MDPs: 1%-CVaR, 10%-CVaR, and mean performance benchmarks for various η values: respectively 0.1 (a-b), 0.9 (c), 0.3 (d-f), 0.5 (g-i), and 0.7 (j-l). The missing figures for $\eta = 0.1$ and $\eta = 0.9$ are in the main document. Figures for additional η values: 0.2, 0.4, 0.6, and 0.8 may be found in the supplementary material package. The abscissae is the dataset size, the ordinate is the normalized performance.

D. Helicopter Experiment Details

D.1. Details about the helicopter environment

We consider the following helicopter environment, where:

- The non terminal state space is the cross product of four features:
  - the abscissa position $s_x \in (0, 1)$,
  - the ordinate position $s_y \in (0, 1)$,
  - the abscissa velocity $v_x \in (-1, 1)$,
  - the ordinate velocity $v_y \in (-1, 1)$,
  - and the initial state is uniformly sampled in $(0, \frac{1}{3}) \times (0, \frac{1}{3}) \times (-1, 1) \times (-1, 1)$.
- The action set is a discrete thrust along each dimension:
  - the abscissa thrust $a_x \in \{-1, 0, 1\}$,
  - and the ordinate thrust $a_y \in \{-1, 0, 1\}$.
- The transition function is independently applied on each dimension:
  - $s_x(t+1) = s_x(t) + v_x(t)\tau + \frac{1}{2}a_x(t)\tau^2 + \Gamma(0, \sigma_s)$,
  - $s_y(t+1) = s_y(t) + v_y(t)\tau + \frac{1}{2}a_y(t)\tau^2 + \Gamma(0, \sigma_s)$,
  - $v_x(t+1) = v_x(t) + a_x(t)\tau + \Gamma(0, \sigma_v)$,
  - $v_y(t+1) = v_y(t) + a_y(t)\tau + \Gamma(0, \sigma_v)$,
  - where $\tau = 0.1$ is the time-step, $\Gamma(0, \sigma)$ is a centered Gaussian noise with standard deviation σ, $\sigma_s = 0.025$ is the position-wise noise standard deviation, $\sigma_v = 0.05$ is the velocity-wise noise standard deviation.
- The reward function is set to:
  - $r(t) = 0$ in every non-terminal state,
  - $r(t) = -1$ when one of the velocity features gets out of $(-1, 1)$: the motor melts and the episode terminates,
  - $r(t) = \min\left(10, \max\left(-1, \frac{1}{\sqrt{(s_x-1)^2+(s_y-1)^2}} - 4\right)\right)$ when one of the position features leaves $(0, 1)$: it landed and the episode terminates. It is good if it is close to the target coordinates $\{1, 1\}$, bad otherwise, see Figure 4(a) for a visual representation of this final reward.
- For the evaluation, similarly to what is commonly used in Atari or Go, the return is not discounted. Although, as next section specifies, the training of the SPIBB-DQN agents requires to set a discount factor lower than 1.

D.2. Details about the experimental design

See Algorithm 5.

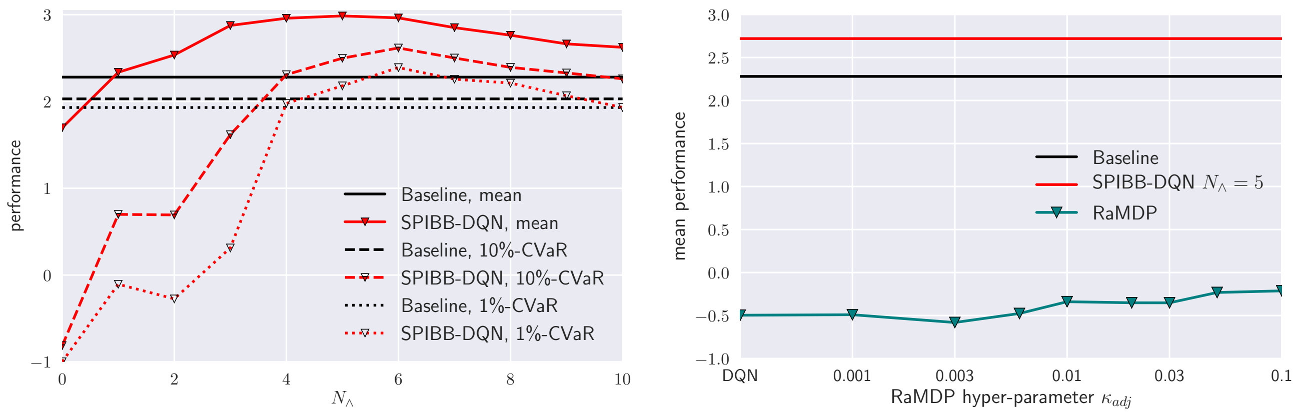
D.3. Details about the DQN and SPIBB-DQN implementations

The batch version of DQN simply consists in replacing the experience replay buffer by the dataset we are training on. Effectively, we are not sampling from the environment anymore but from the transitions collected a priori following the baseline. The same methodology applies for SPIBB, except that the targets we are using for our Q-values update verify the following modified Bellman equation:

$$\begin{aligned} y_j^{(i)} &= r_j + \gamma \max_{\pi \in \Pi_b} \sum_{a' \in \mathcal{A}} \pi(a'|x'_j) Q^{(i)}(x'_j, a') \\ &= r_j + \gamma \sum_{a' | (x'_j, a') \in \mathfrak{B}} \pi_b(a'|x'_j) Q^{(i)}(x'_j, a') + \gamma \left(\sum_{a' | (x'_j, a') \notin \mathfrak{B}} \pi_b(a'|x'_j) \right) \max_{a' | (x'_j, a') \notin \mathfrak{B}} Q^{(i)}(x'_j, a') \end{aligned}$$

Algorithm 5 Helicopter experimental process

```
Input: List of hyper-parameter values for $N_\wedge$
Input: List of dataset sizes
repeat 20 times
    for each dataset size do
        Generate a dataset.
        Compute the pseudo-counts.
        repeat 15 times
            for each $N_\wedge$ do
                Train a policy. ($N_\wedge = 0$ amounts to vanilla DQN, and $N_\wedge = \infty$ amounts to reproducing the baseline)
                Evaluate the trained policy.
                Record the performance of the trained policy.
            end
        end
    end
end
```



(a) SPIBB-DQN with a single dataset in function of $N_\wedge$.

(b) RaMDP hyper-parameter κ_{adj} search.

Figure 12. Robust MDP hyper-parameter search results on the Gridworld domain.

We notice in particular that when $\mathfrak{B} = \emptyset$ the targets fall back to the traditional Bellman ones. We used the now classic target network trick (Mnih et al., 2015), combined with Double-DQN (van Hasselt et al., 2015).

The network used for the baseline and for the algorithms in the benchmark is a fully connected network with 3 hidden layers of 32, 128 and 32 neurons, initialized using he_uniform (He et al., 2015). The network has 9 outputs corresponding to the Q-values of the 9 actions in the game. We train the Q-networks with RMSProp (Tieleman & Hinton, 2012) with a momentum of 0.95 and $\epsilon = 10^{-7}$ on mini-batches of size 32. The learning rate is initialized at 0.01 and is annealed every 20k transitions or every pass on the dataset, whichever is larger. The networks are trained for 2k passes on the dataset, and are fully converged by that time. We use the Keras framework (Chollet et al., 2015) with Tensorflow (Abadi et al., 2015) as backend. The policy is tested for 10k steps at the end of training, with the initial states of each trajectory sampled as described in Section D.1.

D.4. Preliminary SPIBB-DQN experiments

Before starting the experiments reported in the main document, Section 3.4, we led preliminary experiments with a single 10k-transition dataset. We found out, and report on Figure 12(a), that vanilla DQN trains very different Q-networks and therefore very different policies depending on the random seed, which influences the random initialization of the parameters of the network and the transitions sampled for the stochastic gradient. It is worth noticing a posteriori that this dataset was actually favorable to DQN on average (mean performance of 1.7 on this dataset vs. -0.5 reported in the main document), but that the reliability of DQN is still very low. In contrast, SPIBB-DQN shows stability for $N_\wedge \geq 4$.

We also performed a hyper-parameter search on RaMDP on 10k-transition datasets. Given that the reward / value function amplitude is larger than on our previous experiments (Gridworld and Random MDPs), we expect the optimal κ_{adj} value to be also larger than 0.003. We thus considered the following hyper-parameter values: $\kappa_{adj} \in \{0.001, 0.003, 0.006, 0.01, 0.02, 0.03, 0.05, 0.1\}$. To reduce the computational load, we only performed 75 runs per κ_{adj} value. We also added $\kappa_{adj} = 0$, which amounts to vanilla DQN. Figure 12(b) shows that, although it slightly improves the DQN abysmal performance, the RaMDP performance is very limited, far under the baseline.

E. Reproducible, Reusable, and Robust Reinforcement Learning

This paper's objective is to improve the robustness and the reliability of Reinforcement Learning algorithms. Inspired from Joelle Pineau's talk at NeurIPS 2018 about reproducible, reusable, and robust Reinforcement Learning[2], we intend to also make our work reusable and reproducible.

E.1. Pineau's checklist (slide 33)

For all algorithms presented, check if you include:

- A clear description of the algorithm.
  - ⇒ See Algorithm 1 for Π_b-SPIBB, Algorithm 2 for $\Pi_{\leq b}$-SPIBB, and Equation 9 for SPIBB-DQN.
- An analysis of the complexity (time, space, sample size) of the algorithm.
  - ⇒ We do not provide formal analysis for the complexity of the finite MDP SPIBB algorithms as it depends on the policy iteration implementation, but it can be said that the complexity increase in comparison with standard policy iteration is insignificant: it does not change neither the order of magnitude nor the multiplying constant. For SPIBB-DQN, the pseudo-count computation may increase significantly the complexity of the algorithm. It is once more impossible to formally analyze since it depends on the pseudo-count implementation.
- A link to downloadable source code, including all dependencies.
  - ⇒ We provide all the code on github at these addresses: `https://github.com/RomainLaroche/SPIBB` and `https://github.com/rems75/SPIBB-DQN`. See Section E.2.

For any theoretical claim, check if you include:

- A statement of the result.
  - ⇒ See Theorems 1, 2, and 3.
- A clear explanation of any assumptions.
  - ⇒ See Sections 1 and 2.
- A complete proof of the claim.
  - ⇒ See Section A.

For all figures and tables that present empirical results, check if you include:

- A complete description of the data collection process, including sample size.
  - ⇒ See Sections 3, B.1.3, B.1.4, B.1.5, and D.1.
- A link to downloadable version of the dataset or simulation environment.
  - ⇒ See Section E.2.
- An explanation of how sample were allocated for training / validation / testing.
  - ⇒ The complete dataset is used for training. There is no need for validation set. Testing is performed in the true environment.
- An explanation of any data that was excluded.
  - ⇒ Does not apply to our simulated environments.

[2] https://nips.cc/Conferences/2018/Schedule?showEvent=12486

- The range of hyper-parameters considered, method to select the best hyper-parameter configuration, and specification of all hyper-parameters used to generate results.

  ⇒ See Sections 3, B.1.3, B.1.4, B.1.5, B.2, and D.3.

- The exact number of evaluation runs.

  ⇒ 100,000+ for finite MDPs experiments and 300 for SPIBB-DQN experiments.

- A description of how experiments were run.

  ⇒ See Sections 3, B, and D.

- A clear definition of the specific measure or statistics used to report results.

  ⇒ Mean and X% conditional value at risk (CVaR), described in Sections 3 and B.1.6.

- Clearly defined error bars.

  ⇒ Given the high number of runs we considered, the error bar are too thin to be displayed. Any difference visible with the naked eye is significant. We use CVaR everywhere instead to account for the uncertainty.

- A description of results including central tendency (e.g. mean) and variation (e.g. stddev).

  ⇒ All our work is motivated and analyzed with respect to this matter.

- The computing infrastructure used.

  ⇒ For the finite-MDPs experiment, we used clusters of CPUs. The full results were obtained by running the benchmarks with 100 CPUs running independently in parallel during 24h. For the helicopter experiment, we used a GPU cluster. However, only one GPU is necessary for a single run. Using a cluster allowed to launch several runs in parallel and considerably sped up the experiment. On a single GPU (a GTX 1080 Ti), a dataset of $|\mathcal{D}| = 10$k transitions is generated in 5 seconds. The dataset generation scales linearly in $|\mathcal{D}|$. Computing the counts for that dataset takes approximately 20 minutes, it scales quadratically with the size of the dataset. As far as training is concerned, 2000 passes on a dataset of 10k transitions takes around 25 minutes, it scales linearly in N. Finally, evaluation of the trained policy on 10k trajectories takes 15 minutes. It scales linearly in $|\mathcal{D}|$ as it requires the computation of the pseudo-count for each state encountered during the evaluation and this pseudo-count computation is linear in $|\mathcal{D}|$. Overall, a single run for a dataset of 10k transitions takes around one hour.

E.2. Code attached to the submission

The attached code can be used to reproduce the experiments presented in the submitted paper. It is split into two projects: one for finite MDPs (Sections 3.1, 3.2, and 3.3), and one for SPIBB-DQN (Section 3.4).

E.2.1. Finite MDPs

Found at this address: `https://github.com/RomainLaroche/SPIBB`.

Prerequisites The finite MDP project is implemented in Python 3.5 and only requires *numpy* and *scipy*.

Content We include the following:

- Libraries of the following algorithms:
  - Basic RL,
  - SPIBB:
    - Π_b-SPIBB,
    - $\Pi_{\leq b}$-SPIBB,
  - HCPI:
    - doubly-robust,

        - importance sampling,
        - weighted importance sampling,
        - weighted per decision IS,
        - per decision IS,
    - Robust MDP,
    - and Reward-adjusted MDP.

- Environments:
    - Gridworld environment,
    - Random MDPs environment.

- Gridworld experiment of Section 3.1. Run:

  ```
  python gridworld_main.py #name_of_experiment# #random_seed#
  ```

- Gridworld experiment with random behavioural policy of Section 3.2. Run:

  ```
  python gridworld_random_behavioural_main.py #name_of_experiment# #random_seed#
  ```

- Random MDPs experiment of Section 3.3. Run:

  ```
  python randomMDPs_main.py #name_of_experiment# #random_seed#
  ```

Not included We DO NOT include the following:

- The hyper-parameter search (Appendix B.2): it should be easy to re-implement.
- The figure generator: it has too many specificities to be made understandable for a user at the moment. Also, it is not hard to re-implement with one's own visualization tools.

License This project is BSD-licensed.

E.2.2. SPIBB-DQN

Found at this address: `https://github.com/rems75/SPIBB-DQN`.

Prerequisites SPIBB-DQN is implemented in Python 3 and requires the following libraries: Keras, Tensorflow, pickle, glob, yaml, argparse, numpy, yaml, pathlib, csv, scipy and click.

Content The SPIBB-DQN project contains the helicopter environment, the baseline used for our experiments and the code required to generate datasets and train vanilla DQN and SPIBB-DQN.

Commands To generate a dataset, use the following command:

```
python baseline.py baseline --generate_dataset --dataset_size 10000 --dataset_dir
baseline/dataset --seed 1
```

It will generate a dataset with 10000 transitions using the baseline defined in the baseline folder and save the dataset in the `baseline/dataset/10000/1/1_0/dataset.pkl` folder. It will also compute the counts associated with each state-action pair in the dataset, and store those with the dataset in `baseline/dataset/10000/1/1_0/dataset.pkl`. With other parameters, it creates a subfolder of the `dataset_dir` you specify, the subfolder has the form: `dataset_size/seed/noise_factor` (`noise_factor` is 1.0 by default, denoted as a 1_0 folder).

To train a policy using SPIBB-DQN with a parameter n_wedge (denoted minimum_count in the command) of 10, on a dataset generated following the method above, run the following command:

```
python train_batch.py --seed 1 --dataset-path baseline/dataset/10000/1/1_0/counts_dataset.pkl
--baseline-path baseline --options minimum_count 10
```

This will create, in the folder containing the dataset (`baseline/dataset/10000/1/1_0` in that specific command), a csv file with the performance of the policy (one for each run on that dataset, 15 by default).

To repeat the experiment, simply define a different seed for the dataset generation and train on that new dataset. The default values set in the code are the ones that produced the results from the paper. To run vanilla DQN, simply set the `minimum_count` to 0.

To run Reward-adjusted MDP on a dataset, simply add the following flag `--options learning_type ramdp` and specify the value of kappa with e.g. `--options kappa 0.003`.

Not included We DO NOT include the following:

- The multi-CPU/multi-GPU implementation: its structure is too much dependent on the cluster tools. It would be useless for somebody from another lab.

License This project is BSD-licensed.