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. $\Pi = \{\pi : \mathcal{X} \to \Delta_{\mathcal{A}}\}$ 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 $V^\pi_M(x)$ (resp. state-action value function $Q^\pi_M(x,a)$) 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 $M = \langle \mathcal{X}, \mathcal{A}, R, P, \gamma \rangle$:

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

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: $\rho(\pi, M) = V^\pi_M(x_0)$. 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:

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

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:

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

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:

$$|\rho(\pi^{\odot}_{spibb}, M^*) - \rho(\pi^{\odot}_{spibb}, \widehat{M})| = |\rho(\pi^{\odot}_{spibb}, \ddot{M}^*) - \rho(\pi^{\odot}_{spibb}, \widehat{\ddot{M}})| \tag{29}$$

$$= |V^{\pi^{\odot}_{spibb}}_{\ddot{M}^*}(x_0) - V^{\pi^{\odot}_{spibb}}_{\widehat{\ddot{M}}}(x_0)| \tag{30}$$

$$= |Q^{\pi^{\odot}_{spibb}}_{\ddot{M}^*}(x_0, o_\emptyset) - Q^{\pi^{\odot}_{spibb}}_{\widehat{\ddot{M}}}(x_0, o_\emptyset)| \tag{31}$$

$$\leq \frac{2\epsilon V_{max}}{1-\gamma} \tag{32}$$

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:

$$\zeta = \frac{4\epsilon V_{max}}{1-\gamma} - \Big(\rho(\pi^{\odot}_{spibb}, \widehat{M}) - \rho(\pi_b, \widehat{M})\Big) \tag{38}$$

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

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 $\dot{\pi}_t$:

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

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

As a consequence, we obtain:

$$Q^{\pi_t}_{\widehat{M}}(x,a) = \widehat{R}(x,a) + \gamma \sum_{x' \in \mathcal{X}} \sum_{a' \in \mathcal{A}} Q^{\pi_t}_{\widehat{M}}(x',a') \pi_t(a'|x') \widehat{P}(x'|x,a) \tag{43}$$

$$= \widehat{R}(x,a) + \gamma \sum_{x' \in \mathcal{X}} \sum_{a' \in \mathcal{A}} Q^{\pi_t}_{\widehat{M}}(x',a') \left(\tilde{\pi}_t(a'|x') + \dot{\pi}_t(a'|x')\right) \widehat{P}(x'|x,a) \tag{44}$$

$$= \widehat{R}(x,a) + \gamma \sum_{x' \in \mathcal{X}} \widehat{P}(x'|x,a) \left[\sum_{a'|(x',a') \in \mathfrak{B}} \pi_b(a'|x') Q^{\pi_t}_{\widehat{M}}(x',a') \right] \tag{45}$$

$$+ \gamma \sum_{x' \in \mathcal{X}} \widehat{P}(x'|x,a) \left[\left(\sum_{a'|(x',a') \notin \mathfrak{B}} \pi_b(a'|x') \right) \max_{a'|(x',a') \notin \mathfrak{B}} Q^{\pi_t}_{\widehat{M}}(x',a') \right]$$

$$= \frac{\sum_{\langle x_j = x, a_j = a, r_j, x'_j \rangle \in \mathcal{D}} r_j}{N_{\mathcal{D}}(x,a)} + \gamma \sum_{x' \in \mathcal{X}} \frac{\sum_{\langle x_j = x, a_j = a, r_j, x'_j = x' \rangle \in \mathcal{D}} 1}{N_{\mathcal{D}}(x,a)} \left[\sum_{a'|(x',a') \in \mathfrak{B}} \pi_b(a'|x') Q^{\pi_t}_{\widehat{M}}(x',a') \right] \tag{46}$$

$$+ \gamma \sum_{x' \in \mathcal{X}} \frac{\sum_{\langle x_j = x, a_j = a, r_j, x'_j = x' \rangle \in \mathcal{D}} 1}{N_{\mathcal{D}}(x,a)} \left[\left(\sum_{a'|(x',a') \notin \mathfrak{B}} \pi_b(a'|x') \right) \max_{a'|(x',a') \notin \mathfrak{B}} Q^{\pi_t}_{\widehat{M}}(x'_j,a') \right]$$

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:

$$Q^{\pi_t}_{\widehat{M}}(x,a) = \frac{\sum_{\langle x_j=x,a_j=a,r_j,x'_j\rangle\in\mathcal{D}} r_j}{N_{\mathcal{D}}(x,a)} + \frac{\gamma}{N_{\mathcal{D}}(x,a)} \sum_{\langle x_j=x,a_j=a,r_j,x'_j\rangle\in\mathcal{D}} \sum_{a'|(x'_j,a')\in\mathfrak{B}} \pi_b(a'|x'_j) Q^{\pi_t}_{\widehat{M}}(x'_j,a') \tag{47}$$

$$+ \frac{\gamma}{N_{\mathcal{D}}(x,a)} \sum_{\langle x_j=x,a_j=a,r_j,x'_j\rangle\in\mathcal{D}} \left(\sum_{a'|(x'_j,a')\notin\mathfrak{B}} \pi_b(a'|x'_j) \right) \max_{a'|(x'_j,a')\notin\mathfrak{B}} Q^{\pi_t}_{\widehat{M}}(x'_j,a')$$

$$= \frac{1}{N_{\mathcal{D}}(x,a)} \sum_{\langle x_j=x,a_j=a,r_j,x'_j\rangle\in\mathcal{D}} \Bigg[r_j + \gamma \sum_{a'|(x'_j,a')\in\mathfrak{B}} \pi_b(a'|x'_j) Q^{\pi_t}_{\widehat{M}}(x'_j,a') \tag{48}$$

$$+ \gamma \left(\sum_{a'|(x'_j,a')\notin\mathfrak{B}} \pi_b(a'|x'_j) \right) \max_{a'|(x'_j,a')\notin\mathfrak{B}} Q^{\pi_t}_{\widehat{M}}(x'_j,a') \Bigg]$$

$$= \frac{1}{N_{\mathcal{D}}(x,a)} \sum_{\langle x_j=x,a_j=a,r_j,x'_j\rangle\in\mathcal{D}} y_j^{\pi_t} \text{ when } N_{\mathcal{D}}(x,a) > 0 \text{ and is undefined otherwise.} \tag{49}$$

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 π_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 $a^* = \operatorname{argmax}_{a \in \mathcal{A}} Q^*(x, a)$ 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

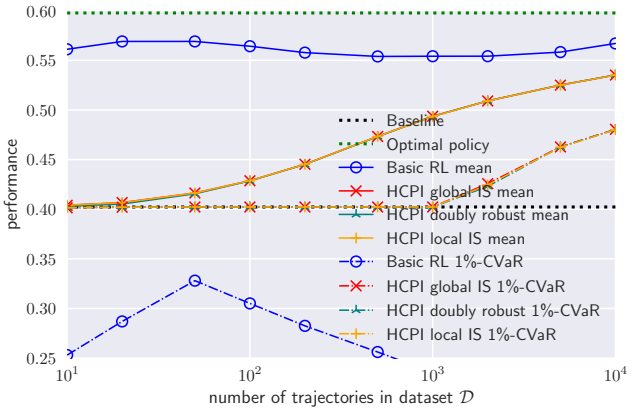
The mean performance is simply the average of performance over all the runs.

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.

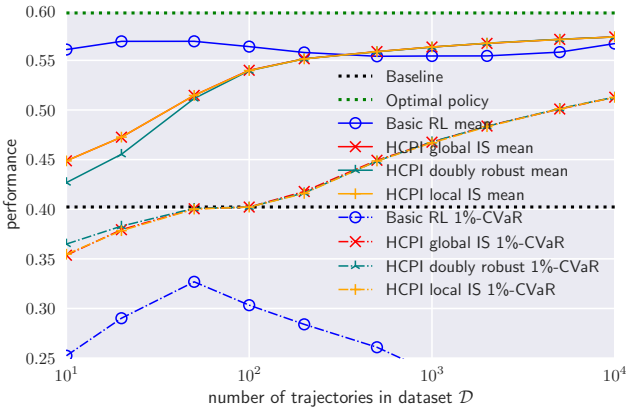
B.1.8. FIGURES

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

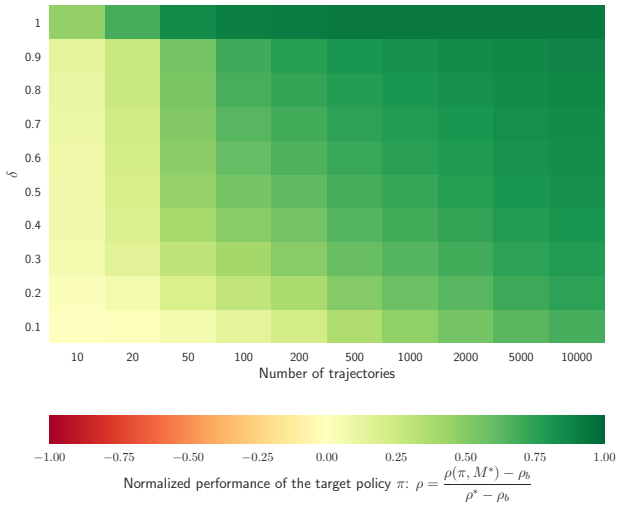
Performance vs. dataset size: These figures (for instance Figure 5(a)) show the (mean and/or CVaR) performance of the algorithms as a function of the dataset size.



(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):

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

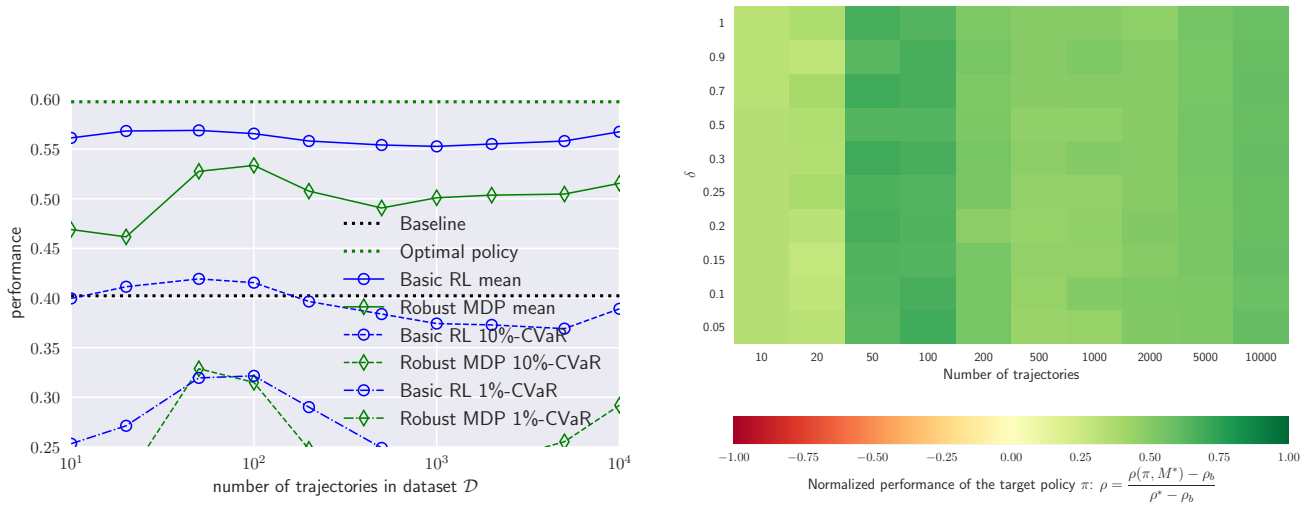
  with $\sigma = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} \left(X_i - \hat{X}_i)^2\right)}$ the sample standard deviation of $X_1, .., X_n$ with Bessel's correction.

- 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:

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

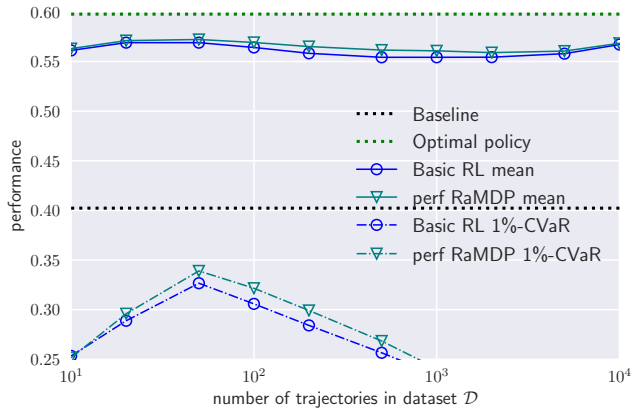
Also, we do not assume that $R^*(x,a)$ is known in our experiments and there is consequently a γ factor disappearing. We obtain:

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

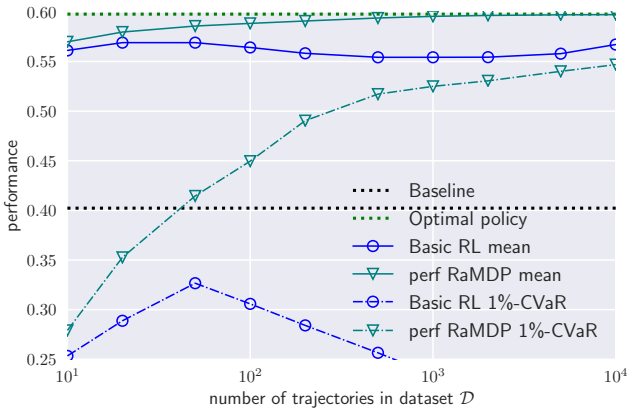
$$\leftarrow \widehat{R}(x,a) - \frac{100}{\sqrt{N_{\mathcal{D}}(x,a)}}, \tag{55}$$

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

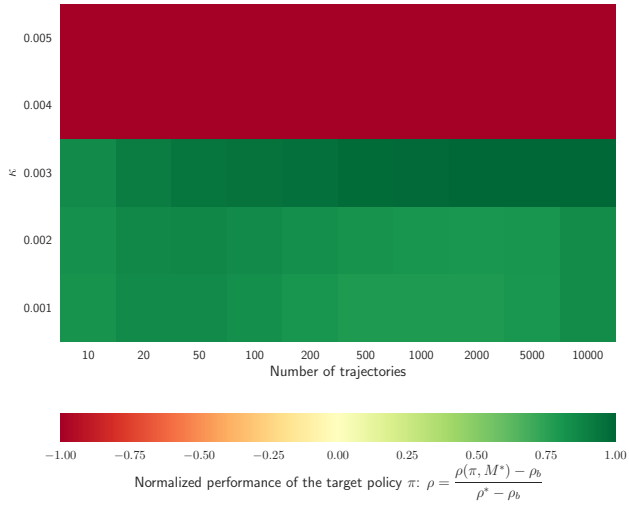
$$\widetilde{R}(x,a) \leftarrow \widehat{R}(x,a) - \frac{\kappa_{adj}}{\sqrt{N_{\mathcal{D}}(x,a)}}. \tag{56}$$



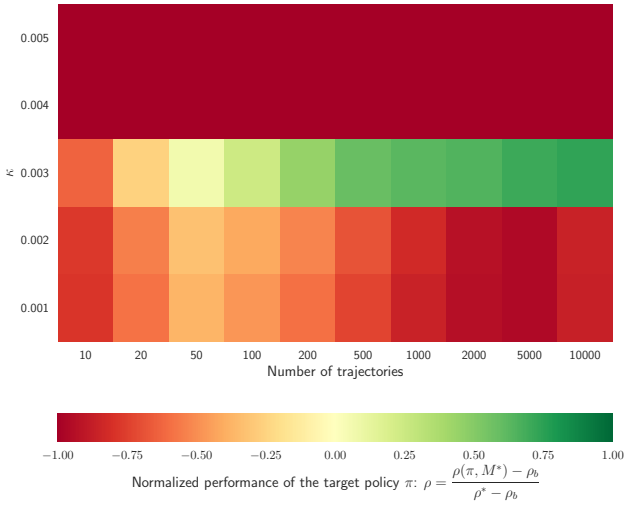
(a) RaMDP with $\kappa_{adj} = 0.002$ (Gridworld)



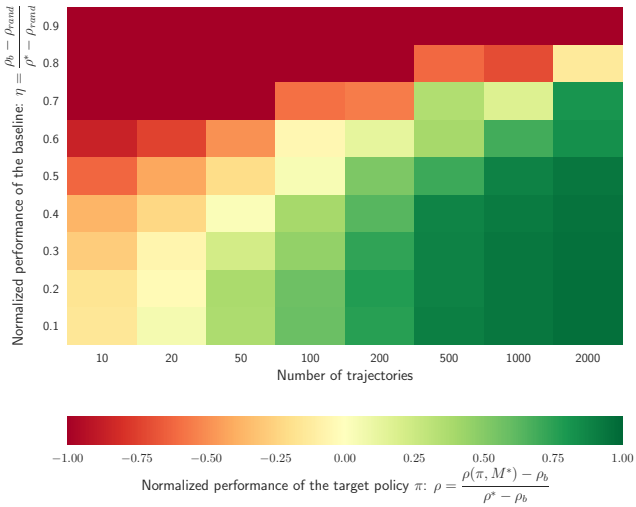
(b) RaMDP with $\kappa_{adj} = 0.003$ (Gridworld)



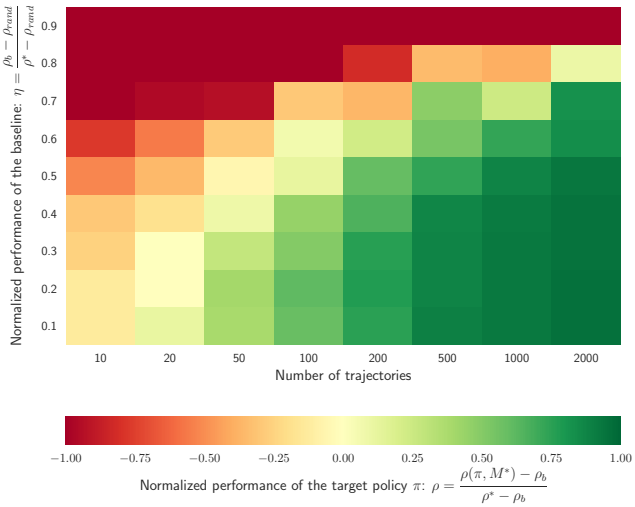
(c) Mean performance RaMDP heatmap (Gridworld)



(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.

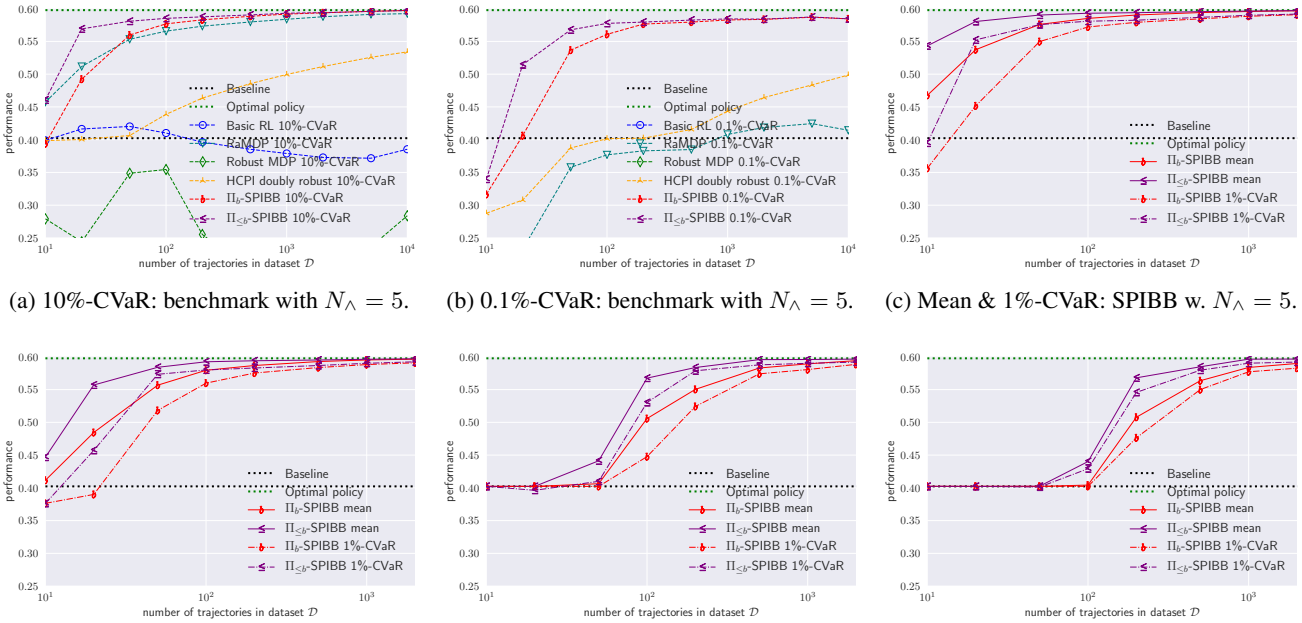
We perform a hyper-parameter seach for:

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

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

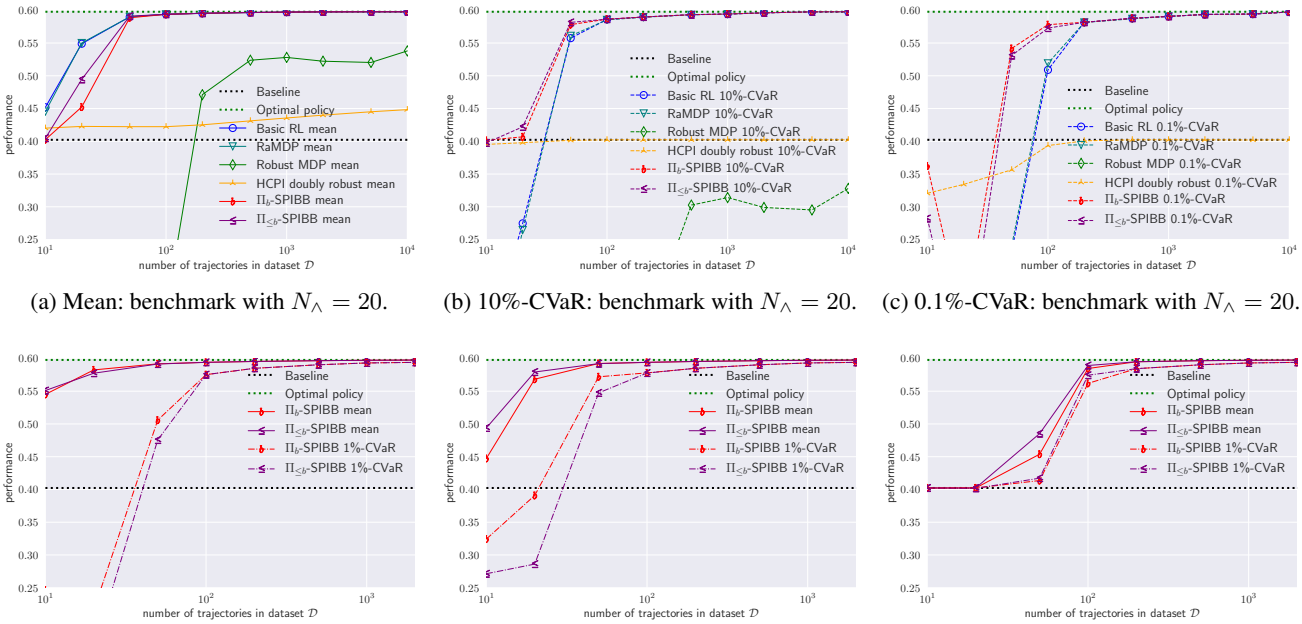


(a) 10%-CVaR: benchmark with $N_\wedge = 5$. (b) 0.1%-CVaR: benchmark with $N_\wedge = 5$. (c) Mean & 1%-CVaR: SPIBB w. $N_\wedge = 5$.

(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

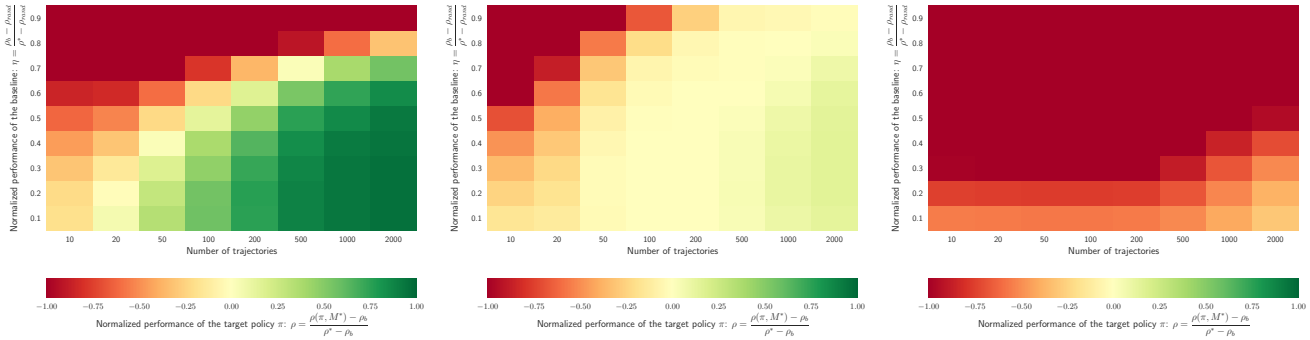


(a) Mean: benchmark with $N_\wedge = 20$. (b) 10%-CVaR: benchmark with $N_\wedge = 20$. (c) 0.1%-CVaR: benchmark with $N_\wedge = 20$.

(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.

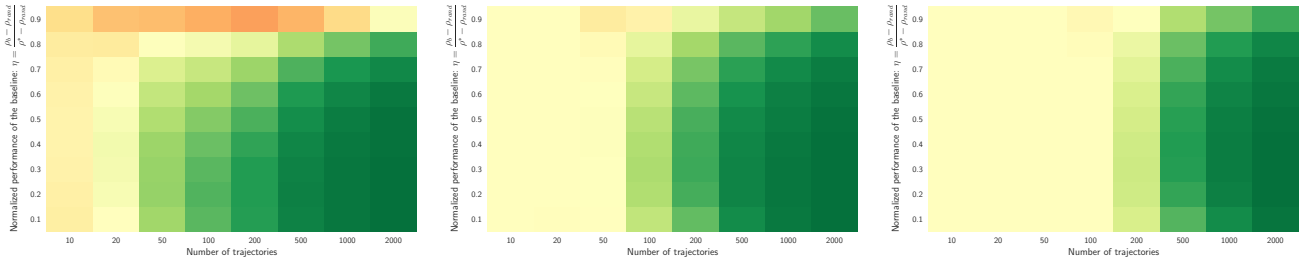
C.3. Full Random MDPs experiment results



(a) 1%-CVaR: Basic RL.

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

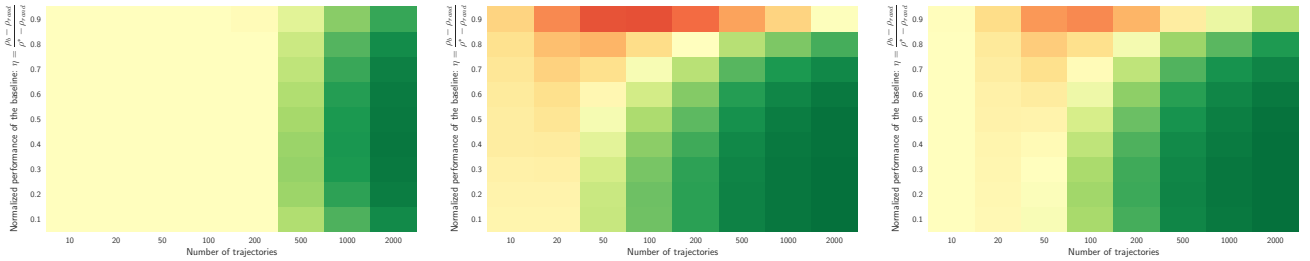
(c) 1%-CVaR: Robust MDP.



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

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

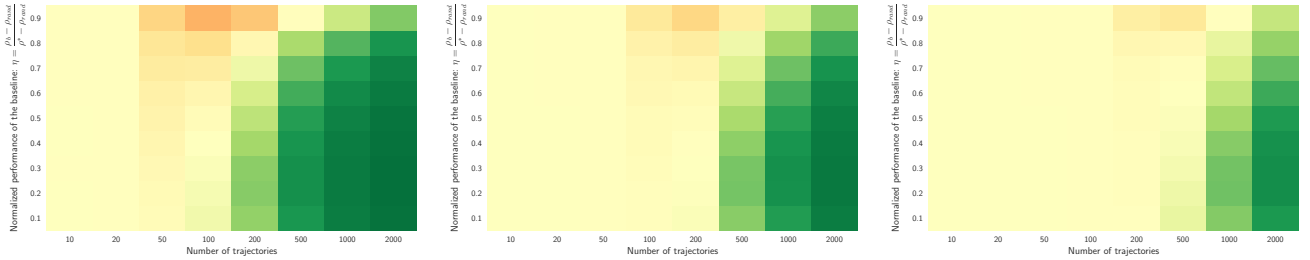
(f) 1%-CVaR: $\Pi_{\leq b}$-SPIBB, $N_\wedge = 50$.



(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, $N_\wedge = 10$.

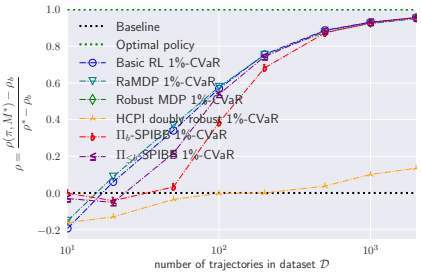


(j) 1%-CVaR: Π_b-SPIBB, $N_\wedge = 20$.

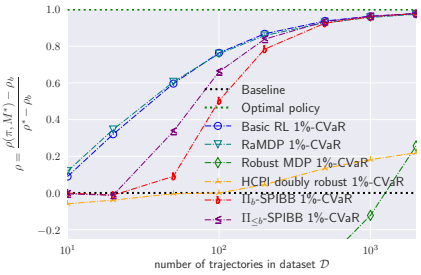
(k) 1%-CVaR: Π_b-SPIBB, $N_\wedge = 50$.

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

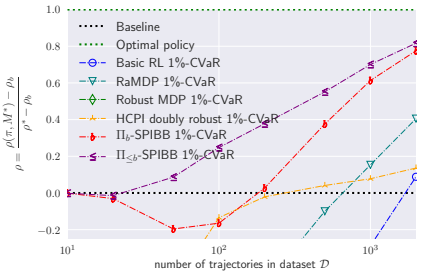
Figure 10. Random MDPs: 1%-CVaR performance heatmaps. The abscissae is the dataset size, the ordinate is the baseline hyperparameter η, and the color is the normalized performance: red, yellow, and green respectively mean below, equal to, and above baseline performance. Heatmaps for the mean normalized 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.



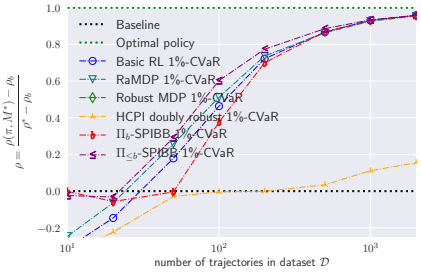
(a) 1%-CVaR: with $\eta = 0.1$ and $N_\wedge = 10$.



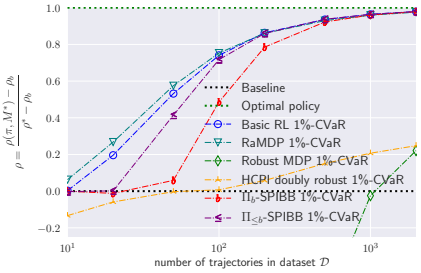
(b) 10%-CVaR: with $\eta = 0.1$ and $N_\wedge = 10$.



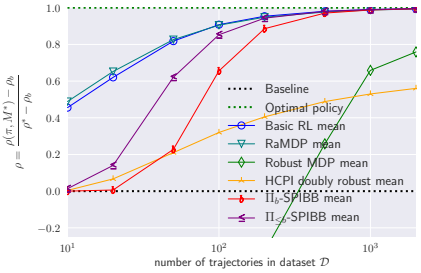
(c) 10%-CVaR: with $\eta = 0.9$ and $N_\wedge = 10$.



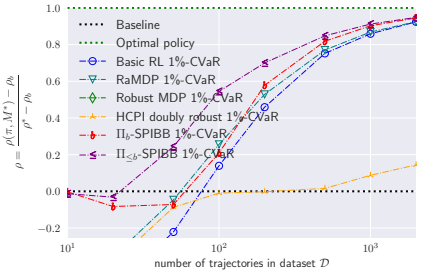
(d) 1%-CVaR: with $\eta = 0.3$ and $N_\wedge = 10$.



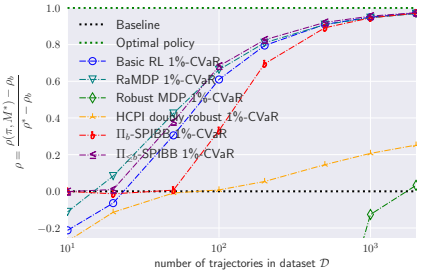
(e) 10%-CVaR: with $\eta = 0.3$ and $N_\wedge = 10$.



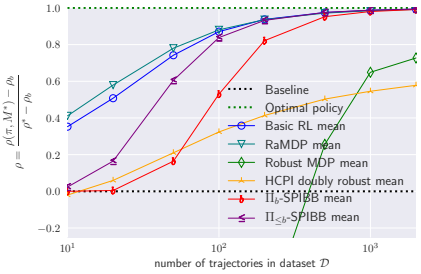
(f) Mean: with $\eta = 0.3$ and $N_\wedge = 10$.



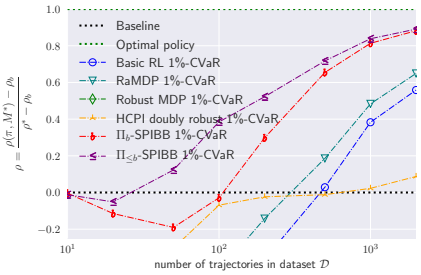
(g) 1%-CVaR: with $\eta = 0.5$ and $N_\wedge = 10$.



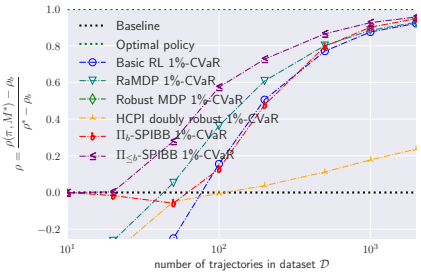
(h) 10%-CVaR: with $\eta = 0.5$ and $N_\wedge = 10$.



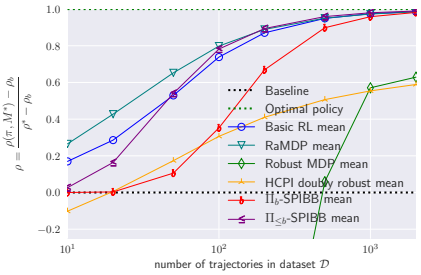
(i) Mean: with $\eta = 0.5$ and $N_\wedge = 10$.



(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

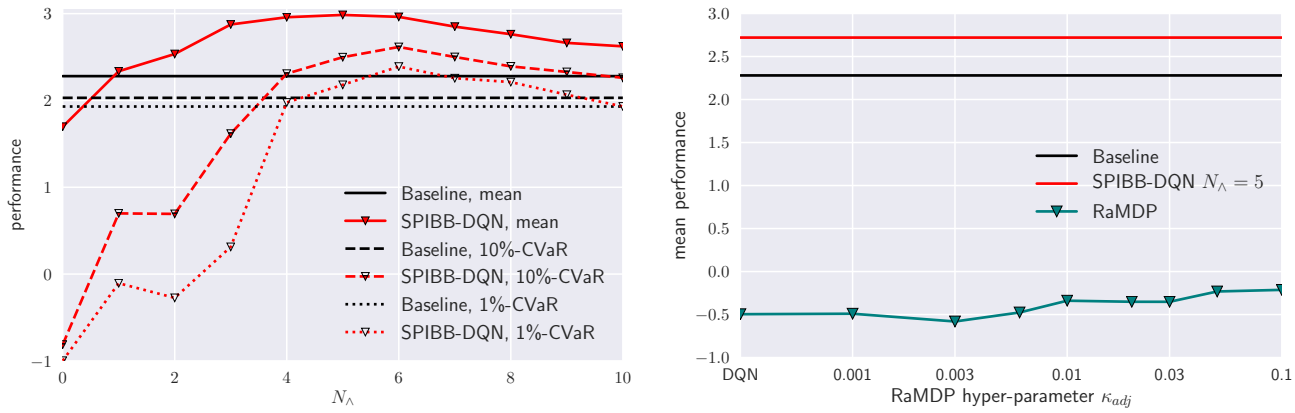
The batch version of DQN simply consists in replacing the experience replay buffer by the dataset we are training on. Effectively, we are not sampling from the environment anymore but from the transitions collected a priori following the baseline. The same methodology applies for SPIBB, except that the 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.
  - $\Rightarrow$ 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.
  - $\Rightarrow$ 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.
  - $\Rightarrow$ 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.
  - $\Rightarrow$ See Theorems 1, 2, and 3.
- A clear explanation of any assumptions.
  - $\Rightarrow$ See Sections 1 and 2.
- A complete proof of the claim.
  - $\Rightarrow$ 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.
  - $\Rightarrow$ 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.
  - $\Rightarrow$ See Section E.2.
- An explanation of how sample were allocated for training / validation / testing.
  - $\Rightarrow$ 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.
  - $\Rightarrow$ 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.

  $\Rightarrow$ See Sections 3, B.1.3, B.1.4, B.1.5, B.2, and D.3.

- The exact number of evaluation runs.

  $\Rightarrow$ 100,000+ for finite MDPs experiments and 300 for SPIBB-DQN experiments.

- A description of how experiments were run.

  $\Rightarrow$ See Sections 3, B, and D.

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

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

- Clearly defined error bars.

  $\Rightarrow$ 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).

  $\Rightarrow$ All our work is motivated and analyzed with respect to this matter.

- The computing infrastructure used.

  $\Rightarrow$ 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.