Technische Universität München

Department of Mathematics

Master's Thesis

Evaluation of Safe Policy Improvement with Soft Baseline Bootstrapping

Philipp Scholl

Supervisor: Prof. Dr. Hans-Joachim Bungartz

Advisor: Dr. Felix Dietrich, Dr. Clemens Otte, Dr. Steffen Udluft

Submission Date: 15.04.2021

I assure the single handed composition of this master's thesis only supported by declared resources.

Garching, 15.04.2021

P. Scholl

Abstract

Due to the high computing power of modern computers and the increasing availability of data, reinforcement learning is gaining importance in many areas of industry. In safety-critical areas, however, reinforcement learning poses dangers, which is why algorithms that promise a safe improvement of the behavior policy are of high interest. Therefore, this master's thesis examines the Soft-SPIBB algorithms introduced in "Safe Policy Improvement with Soft Baseline Bootstrapping" by Nadjahi et al. Some shortcomings of the mathematical theory underlying the algorithms are revealed, which have the consequence that the theoretical safety bounds can no longer be applied to the original algorithms from the paper. However, this can be remedied by further restricting the algorithms and this leads to new algorithms to which the theoretical safeties apply.

In addition to adapting further algorithms that incorporate the uncertainty of state-action pairs into their calculations, and implementing all of these into a unified Python framework, these algorithms are tested on two different benchmarks. Here, in particular, a heuristic adaptation of the original algorithms turns out to be both very safe and performant.

Zusammenfassung

Durch die hohe Rechenleistung moderner Computer und die steigende Verfügbarkeit von Daten gewinnt Reinforcement Learning in vielen Bereichen der Industrie an Bedeutung. In sicherheitskritischen Bereichen birgt Reinforcement Learning jedoch Gefahren, weswegen Algorithmen, die eine sichere Verbesserung der Behavior Policy versprechen, von hohem Interesse sind. Deshalb untersucht diese Masterarbeit die Soft-SPIBB Algorithmen, welche in "Safe Policy Improvement with Soft Baseline Bootstrapping" von Nadjahi et al. eingeführt wurden. Dabei werden einige Unzulänglichkeiten der den Algorithmen zugrundeliegenden mathematischen Theorie aufgedeckt, die zur Folge haben, dass die theoretischen Sicherheitsschranken nicht mehr auf die ursprünglichen Algorithmen aus dem Paper angewendet werden können. Dies kann jedoch durch eine weitere Einschränkung der Algorithmen behoben werden, was zu neuen Algorithmen führt, auf die die theoretischen Sicherheiten zutreffen.
Neben der Adaption weiterer Algorithmen, die die Unsicherheit von Zustand-Aktion-Paaren in ihre Berechnungen mit einbeziehen, und der Implementierung all dieser Algorithmen in ein einheitliches Python Framework werden diese Algorithmen auf zwei verschiedenen Benchmarks getestet. Dabei erweist sich insbesondere eine heuristische Anpassung der ursprünglichen Algorithmen als sehr sicher und performant.

Contents

1 Introduction

Due to the increase in computational power and availability of data over the last decades, machine learning has gained immense importance. Nowadays, machine learning is integrated in various different applications like image recognition [1], speech recognition [2], medical diagnosing [3], recommender systems [4], and many more.
The general idea behind machine learning is to not directly implement the solution to a problem, but instead to provide data which the algorithm can use to find the solution autonomously. This is especially useful, if enough data is available and the task which is supposed to be learned is too complicated to be implemented directly [5].
Two prominent kinds of machine learning are supervised and unsupervised learning which have multiple use cases in prediction, classification and pattern recognition [5]. This master's thesis focuses on a different kind of machine learning, namely, reinforcement learning. Reinforcement learning, also called feedback learning, follows the basic idea of learning by trying various actions and getting constant feedback on past decisions. This method is used, for example, for controlling robots, in scheduling in computer systems, in finance, in the energy sector, and transportation [6].

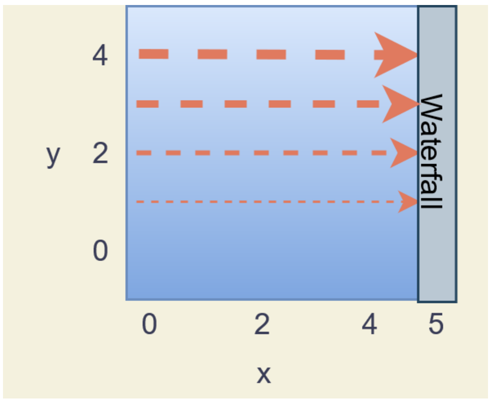


Fig. 1. *The setting of the Wet Chicken benchmark used for reinforcement learning. The boat starts at $(x, y) = (0, 0)$ and starts there again upon falling down the waterfall at $x = 5$. The arrows show the direction and strength of the stream towards the waterfall. Additionally, there are turbulences which are stronger for small y. The goal for the boat is to stay as close as possible to the waterfall without falling down.*

An example problem used in the experiments in Sections 3.4.2 and 3.4.4 is the Wet Chicken benchmark which is depicted in Figure 1. The general setting is that a person in a boat is trying to achieve the highest possible reward over a long time on this river. The person gets a higher reward for staying close to the waterfall without falling down, as they have to start at $(x, y) = (0, 0)$ again, which yields a low reward. The arrows show that the bigger y gets, the stronger the stream towards the waterfall becomes. The opposite effect holds for the turbulence which causes the boat to drift randomly either towards the waterfall or away from it [7].
The person in the boat has paddles and, thus, the possibility to influence their trajectory on the river through a variety of different actions. To learn which action to choose in which state, i.e., position on the river, is the goal of reinforcement learning. In online

reinforcement learning, this is usually done by choosing an exploration-exploitation trade-off, which means that the person in the boat tries to *exploit* prior knowledge through choosing the action they think is the best, while still *exploring* the different options by randomly choosing some which they think are sub-optimal. After sufficient training time, the person is expected to have improved their initial strategy—called *policy* in reinforcement learning—substantially [8].

The problem with these online approaches—particularly in safety-related applications—is that they naturally perform poorly at the beginning of their training as they will explore novel options. A possible counter to this is offline reinforcement learning [8]. In the Wet Chicken benchmark this would mean that there are two different persons at the river. One person sits in the boat and has most probably already some experience with this specific river, while the other is at the shore and simply observes the first person. The goal of offline reinforcement learning is that the observer can use the experience of the person in the boat directly to devise a good strategy and can, therefore, enter the river and follow this new strategy without any new exploration done.

This technique could be applied in many optimal-control settings in the industry, for example, in controlling the gas flow in a gas turbine [9] or controlling wind turbines [10] to increase the efficiency and durability of these. However, deploying a new policy for a safety-critical tool without observing it in action beforehand comes at a risk for a company. Thus, the newly learned policy should not only be expected to be significantly better than the policy it observed, there should also be some safety mechanism enabling bounds on the probability the new policy is performing significantly worse than the old one.

How to achieve safe offline reinforcement learning is the topic of this master's thesis. For this I build on "Safe Policy Improvement with Soft Baseline Bootstrapping" by Nadjahi et al. [11] from 2019, as the authors claim to have found provably safe algorithms to accomplish this in a discrete state and action space, which also empirically outperform many existing algorithms designed for the same task and, furthermore, show good performance in the continuous state space. Their interpretation of *safe* is formally defined in Definition 1.

The basic idea is to use "Baseline Bootstrapping", which means to follow the same policy, which was used to collect the data, at points with a high uncertainty and to concentrate the optimization process onto states in which the uncertainty is small. To examine the authors' claims, this thesis starts with an introduction to reinforcement learning in Sections 2.1 and 2.2, focusing only on the concepts and methods necessary for the rest of the thesis. Section 2.3 then presents the main ideas behind the algorithms of Nadjahi et al. [11] in detail.

Section 3 contains the new work done for this master's thesis. Here, I begin in Section 3.1 with showing that the original Soft-SPIBB algorithms are not provably safe. However, one of the safety theorems is corrected and proven as Theorem 3.2 in Section 3.2. Consequently, a new class of algorithms is introduced, which are natural extensions of the algorithms of Nadjahi et al. [11] for which the safety theorem truly holds. The limitations of the theory behind these algorithms and various ways of countering them are discussed in Section 3.2.4.

After this purely theoretical analysis of their paper, Section 3.3 introduces a framework containing several different uncertainty incorporating offline reinforcement learning al-

gorithms. These are either from existing literature or newly developed in this master's thesis. In Section 3.4 they are used as a comparison to assess the empirical strength of the algorithms from Nadjahi et al. [11] and the newly derived variations. As already shown by the authors, all their algorithms and these variations derived in this thesis perform well, especially regarding their safety. The majority of the experiments are conducted using a finite state space. However, I also start first experiments using the continuous version of the Wet Chicken benchmark to test the implementations using function approximation of some of the algorithms. Through the promising results of their algorithms in this benchmark, I can again support the authors' empirical investigation.
Section 4 concludes this master's thesis and points at open questions. I try to only include the most important plots in the main part of this thesis, but for the sake of completeness I deliver all relevant ones in the appendix.

2 State of the Art

In the first two subsections of this chapter the goal is to give an introduction to reinforcement learning to a reader with no prior knowledge about reinforcement learning to enable them to understand the most important concepts of the theory and to follow the rest of this thesis. Subsection 2.1 introduces the general problems which can be tackled by reinforcement learning. In Subsection 2.2 Dynamic Programming is introduced, which is a solution method for reinforcement learning and shows the basic structure and idea behind most reinforcement learning algorithms which are considered in this thesis. These subsections are based on the book of Sutton and Barto [8]. After this general introduction to the wide field of reinforcement learning, I present the work done about safe policy improvement in Laroche et al. [12] and Nadjahi et al. [11]. Especially the work done in the latter one is thoroughly analyzed in the rest of this master's thesis.

2.1 Finite Markov Decision Processes

This section is completely based on Sutton and Barto [8]. A *Markov Decision Process* (MDP) is a discrete-time process which models the interaction between an *agent* (decision maker) and the *environment* (everything else). The goal of reinforcement learning is to find a "good" *policy* (strategy) according to which the agent makes its decisions. The agent and the environment interact in the following way (as visualized in Figure 2):

- At time t, the agent observes the current *reward* $R_t \in \mathcal{R}$ and *state* $S_t \in \mathcal{S}$ and responds by choosing *action* $A_t \in \mathcal{A}$.
- Depending on this action, the environment then produces a new reward $R_{t+1} \in \mathcal{R}$ and state $S_{t+1} \in \mathcal{S}$.

$\mathcal{R}$, $\mathcal{S}$, and $\mathcal{A}$ are called the *reward*, *state*, and *action spaces*, respectively. Thus, the agent and the environment together produce the sequence $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, ...$ Note that while $\mathcal{S}$ and $\mathcal{A}$ can be just any sets, $\mathcal{R}$ has to be a subset of $\mathbb{R}$. However, as this thesis focuses on finite MDPs at first, assume that $\mathcal{A}$ and $\mathcal{S}$ are both finite.

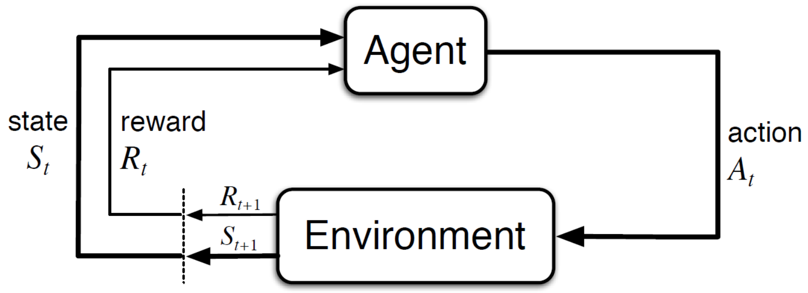


Fig. 2. *Basic model of an MDP, p. 48 in Sutton and Barto [8].*

Usually, the environment acts in a non-deterministic manner and the probability of the environment producing the new reward R_{t+1} and state S_{t+1}, after observing the sequence

$S_0 = s_0, A_0 = a_0, R_1 = r_1, S_1 = s_1, A_1 = a_1, R_2 = a_2, ..., S_t = s, R_t = r, A_t = a$ is described by

$$P(S_{t+1} = s', R_{t+1} = r' | A_t = a, S_t = s, R_t = r, A_{t-1} = a_{t-1}, ..., S_0 = s_0) \tag{1}$$

$$= P(S_t = s', R_t = r' | S_{t-1} = s, A_{t-1} = a) \tag{2}$$

$$= p(s', r' | s, a), \tag{3}$$

where $p : \mathcal{S} \times \mathcal{A} \times \mathcal{R} \times \mathcal{S} \to [0, 1]$, $(s, a, s', r') \mapsto p(s', r'|s, a)$ models the *transition probabilities* or *dynamics* of the MDP. The step from Equation 1 to Equation 2 is called the *Markov Property* of MDPs and ensures that the future only depends on the current state and the correspondingly chosen action. The step from Equation 2 to Equation 3 is called the *Stationarity* of the MDP and ensures that these probabilities are time independent.
There are two possible forms of MDPs, *episodic* ones and *non-episodic* ones. In an episodic MDP the agent-environment interaction can be split up into finite subsequences, which are called *episodes*. Each of these episodes ends when the environment enters a *terminal state*. A non-episodic MDP is the opposite, as its state-reward-action sequence never ends.
The goal of the agent is to maximize the collected rewards. Instead of just trying to get the maximal reward in the next step, one tries to maximize the *return* which includes also future rewards. Depending on the kind of MDP, the return has to be defined differently:

- in an episodic MDP, the return at time t is defined as the sum of rewards $G_t = \sum_{i=t+1}^{T} R_i$, where T is the *time of termination* and
- in a non-episodic MDP, the return at time t is defined as the discounted sum of rewards $G_t = \sum_{i=t+1}^{\infty} \gamma^{i-(t+1)} R_i$, where $0 < \gamma < 1$ is called the *discount factor*, which influences how high the return values future rewards in relation to the current one. The closer the discount factor is to 1, the more weight is given to future rewards.

These two definitions can be unified as

$$G_t = \sum_{i=t+1}^{T} \gamma^{i-t-1} R_i, \tag{4}$$

where T is again the time of termination, if the MDP is episodic and infinity otherwise. The discount factor $0 \leq \gamma \leq 1$ can be chosen as 1 in the episodic case to get the same definitions for the return as before.
The agent is modeled by a *policy* π, which is defined as

$$\pi : \mathcal{S} \times \mathcal{A} \to [0, 1], \quad \pi(a|s) = P(A_t = a | S_t = s). \tag{5}$$

Thus, for every state $s \in \mathcal{S}$, $\pi(\cdot|s)$ is a probability distribution over the action space $\mathcal{A}$. A special class of policies are the *deterministic* policies, i.e., policies π such that for each state $s \in \mathcal{S}$, there is an action $a \in \mathcal{A}$ with $\pi(a|s) = 1$ and $\pi(a'|s) = 0$ for every other action $a' \neq a$.
As mentioned in the beginning, the goal in reinforcement learning is to find a policy which maximizes the return. However, as S_t, R_t, and A_t are random variables, the return is a

random variable as well and, therefore, it is helpful to look at the expected value of the return.
Thus, one defines the *state-value function* $v_\pi(s)$ of some state s as the expected return when starting in s and following some policy π:

$$v_\pi : \mathcal{S} \to \mathbb{R}, \quad v_\pi(s) = E_\pi[G_t | S_t = s] \tag{6}$$

where t is arbitrary and the expected value is over the MDP with an environment governed by the dynamics p and an agent following the policy π.
Analogously, one defines for a state-action pair the *action-value function* $q(s,a)$ for some state s and action a as the expected return when starting in s by choosing a and from then on following some policy π:

$$q_\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}, \quad q_\pi(s,a) = E_\pi[G_t | S_t = s, A_t = a] \tag{7}$$

where t is again arbitrary and the expected value is over the same quantities as before. Note that both value functions are well-defined due to the Markov Property and Stationarity of the MDP.
Before completing all the definitions, I start with computing the relations between the state- and action-value functions:

$$\begin{aligned} v_\pi(s) = E_\pi[G_t | S_t = s] &= \sum_a \pi(a|s) E_\pi[G_t | S_t = s, A_t = a] \\ &= \sum_a \pi(a|s) q_\pi(s,a) \end{aligned} \tag{8}$$

and

$$\begin{aligned} q_\pi(s,a) &= E_\pi[G_t | S_t = s, A_t = a] = E_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\ &= E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a] = \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')]. \end{aligned} \tag{9}$$

The second step of Equation 9 is due to the formula

$$G_t = R_{t+1} + \gamma G_{t+1} \tag{10}$$

which follows directly from the definition of G_t in Equation 4.
Now, it can be defined what it means for a policy to be better than another one. One writes

$$\pi \geq \pi' \iff v_\pi(s) \geq v_{\pi'}(s) \text{ for all } s \in \mathcal{S}. \tag{11}$$

There is always a policy π_* s.t. $\pi_* \geq \pi$ for any policy π and this policy is called the *optimal policy* [8]. Furthermore, there is always an optimal policy which is deterministic [8]. One defines the *optimal value functions* as the value functions of any optimal policy:

$$\begin{aligned} v_*(s) &= \max_\pi v_\pi(s) \\ q_*(s,a) &= \max_\pi q_\pi(s,a) \end{aligned} \tag{12}$$

The next step is to compute some fundamental equations of the theory of Markov Decision Processes. Using at first Equation 8 and then Equation 9 yields

$$v_\pi(s) = \sum_a \pi(a|s) q_\pi(s,a) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')] \tag{13}$$

which is known as the *Bellman equation* for the state-value function. Similarly, one derives the *Bellman equation* for the action-value function by computing

$$q_\pi(s,a) = \sum_{s',r} p(s',r|s,a)[r+\gamma v_\pi(s')] = \sum_{s',r} p(s',r|s,a)(r+\gamma \sum_{a'} \pi(a'|s')q_\pi(s',a')). \quad (14)$$

The value functions not only satisfy the Bellman equations, but they are also the unique solutions to them [8]. The uniqueness is used in the following sections to compute the value functions. To conclude this subsection, I state the *Bellman optimality equation* for the state-value function, of which the unique solution is the optimal state-value function:

$$v_*(s) = \max_a q_*(s,a) = \max_a \sum_{s',r} p(s',r|s,a)[r+\gamma v_*(s')] \quad (15)$$

where the first step follows again from Equation 8 and the second step from Equation 9.

2.2 Dynamic Programming

In the previous subsection, the problem which is supposed to be solved with reinforcement learning and its most important quantities were introduced. This subsection shows how to solve this kind of problems with tabular methods, i.e., by trying to find the exact solution instead of relying on function approximation. This is only possible if the action and state spaces are finite and small enough to make exact solutions computationally feasible.
The method I present here is *Dynamic Programming*, which assumes that the dynamics of the environment are known. Even though this assumption is not used in the rest of this thesis, this subsection builds the basis for most of the algorithms considered in later sections, as most of them estimate the transition probabilities first and then use some variation of Dynamic Programming.
In this subsection the goal is to find the optimal policy π_* for a given MDP of which the transition probabilities are known. I only introduce *Policy Iteration*, as this is used in almost any algorithm considered in later chapters. This method can be divided into two parts:

- *Policy Evaluation* (PE)
- *Policy Improvement* (PI)

During the *Policy Evaluation* step, the goal is to compute the state- or action-value function for a given policy π and transition probabilities p. In Sutton and Barto [8], the authors only describe the computations for the state-value function in detail, whereas the computations for the action-value function are left as Exercise 4.5 on page 82. As the computations are almost exactly the same for the action-value function and I use this function later, the Policy Evaluation step for the action-value function is presented in the following.
Assuming the current policy π and the true dynamics p are known, Equation 14 can be seen as a system of $|\mathcal{S}||\mathcal{A}|$ linear equations and $|\mathcal{S}||\mathcal{A}|$ unknowns. Thus, one way to compute the action-value function would be to directly solve this system, which is done in the implementation of the algorithms of Nadjahi et al. [11].

Another approach is the *iterative policy evaluation*, which tries to approximate the solution of Equation 14 iteratively, by a sequence of action-value functions $q_0, q_1, q_2, \ldots$. It is possible to initialize q_0 arbitrarily and the update from k to $k+1$ is the following:

$$q_{k+1}(s,a) = \sum_{s',r} p(s',r|s,a)(r + \gamma \sum_{a'} \pi(a'|s')q_k(s',a')) \tag{16}$$

This sequence converges to its unique fixed point q_π [8].
Once the action-value function is known, the goal is to use this new knowledge about the expected return of different actions in different states to improve the current policy. The obvious idea might be to always choose the action with the highest value, i.e., to set

$$\pi'(a|s) = \begin{cases} 1, & \text{if } a = \arg\max_a q_\pi(s,a) \\ 0, & \text{otherwise} \end{cases} \tag{17}$$

for all states $s \in \mathcal{S}$. This is known as the *Policy Improvement* step. The fact that it yields an improvement, i.e., $\pi' \geq \pi$, is stated in the *Policy Improvement Theorem* on page 78 [8].

Theorem 2.1: Policy Improvement Theorem

Let π and π' be two arbitrary policies. Then

$$\forall s \in \mathcal{S} : \sum_{a \in \mathcal{A}} \pi'(a|s)q_\pi(s,a) \geq v_\pi(s) \implies \pi' \geq \pi. \tag{18}$$

If, additionally, there is at least one state s, such that

$$\sum_{a \in \mathcal{A}} \pi'(a|s)q_\pi(s,a) > v_\pi(s)$$

holds, it follows that there is at least one state s' with $v_{\pi'}(s') > v_\pi(s')$.

Thus, if the action-value function is known, it is easy to improve the current policy, by simply shifting probability mass of the policy from an action with a small value to an action with a high value, as it is done, for example, in the Policy Improvement step in Equation 17. Furthermore, it can be deduced from Theorem 2.1, that if the Policy Improvement step does not yield an improvement of π, i.e., $v_{\pi'}(s) = v_\pi(s)$ for every state s, it follows that π' and π are optimal policies [8].
In conclusion, the Policy Improvement step improves the current policy if the current policy is not already optimal and produces an optimal policy otherwise.
Policy Iteration starts with an arbitrary policy and from then on alternates until convergence between Policy Evaluation and Policy Improvement. This general concept works also for other ways of implementing the PE and PI step and is coined as *Generalized Policy Iteration (GPI)* by Sutton and Barto [8]. Figure 3 depicts the alternating structure of GPI and shows that it continues until convergence to the optimal policy value-function is reached, which is the unique fixed point of this process.

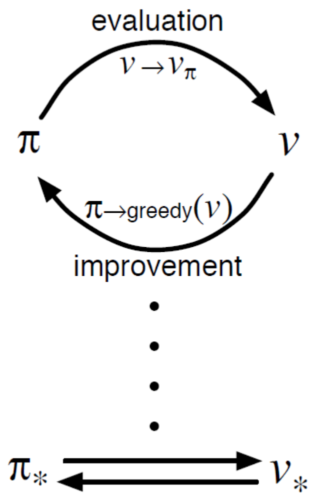


Fig. 3. *Generalized Policy Iteration: Policy Evaluation and Policy Improvement alternate until convergence to the optimal policy, see page 94 in Sutton and Barto [8].*

For finite MDPs, there are only finitely many deterministic policies and, thus, by the above mentioned properties of the Policy Improvement step, Policy Iteration finds the optimal policy in finitely many steps.

2.3 Safe Policy Improvement with Soft Baseline Bootstrapping

Having introduced the basic concepts of reinforcement learning, the precise problem being dealt with by this thesis is detailed in the following. In practice one usually does not know the true transition probabilities and, thus, cannot simply conduct the PE step as done for Dynamic Programming in Subsection 2.2. Therefore, one usually has to rely on the experience of an initial policy to gain knowledge about the transition probabilities or the value functions directly.
One possibility to use this information is to utilize it as soon as it is observed in order to improve the current policy to get more informative data. This is called *online* reinforcement learning. However, in safety critical applications it is not recommended to use such an online algorithm, as it will try to explore different actions to gain knowledge and, thus, possibly result in fatal consequences.
For this reason, I am interested in *offline* reinforcement learning, also called *batch* reinforcement learning [13], as a policy is observed for some time without changing it and then this batch of data is used to derive a new policy. However, Batch RL introduces new dangers, especially in scenarios in which wrong actions by the policy may result in terrible outcomes.
Therefore, the goal of this thesis is to determine how to find the best policy π in a safe way upon receiving data generated from a *behavior policy* π_b without access to the true transition probabilities and ever testing π on the real MDP.
The paper "Safe Policy Improvement with Soft Baseline Bootstrapping" by Nadjahi et al. [11] from 2019 introduces a method to solve exactly this problem. It is the successor paper of Laroche et al. [12].

There are many attempts on trying to find a *safe* solution to this problem and as many definitions of *safe* [14]. In this thesis I focus on safe policy improvement and, thus, use Definition 1 which is similar to the one given in Laroche et al. [12]:

Definition 1 *A policy π is said to be a ξ-approximate $1 - \delta$-safe policy improvement w.r.t. π_b if*

$$\mathbb{P}_{\mathcal{D}}(\forall s \in \mathcal{S} : v_\pi(s) - v_{\pi_b}(s) \geq -\xi) \geq 1 - \delta \tag{19}$$

where π_b is an arbitrary behavior policy used to generate a data sets $\mathcal{D}$ on an arbitrary MDP M, $\mathbb{P}_{\mathcal{D}}$ is the probability over all possible data sets $\mathcal{D}$ induced by π_b and M, $\xi > 0$ is the approximation parameter and $\delta > 0$ is the confidence parameter.

With respect to Definition 1, both Laroche et al. [12] and Nadjahi et al. [11] claim to have found mathematically provably safe Batch RL algorithms, which are practical at the same time. Since these are, as explained before, favorable properties, this master's thesis concentrates on these algorithms and their underlying theory. The focus lies on the successor paper "Safe Policy Improvement with Soft Baseline Bootstrapping" [11], as the algorithms introduced here have shown better results in experiments.
In Section 2.3.1 I introduce the mathematical framework from Nadjahi et al. [11] with all the definitions they use and the theorems stating the safety of their algorithms. Building upon this framework, I present the algorithms and their properties introduced in Nadjahi et al. [11] in Section 2.3.2. In Section 2.3.3 the Random MDPs benchmark is presented, which was used in Nadjahi et al. [11] to evaluate their algorithms. In the last section, Section 2.3.4, I quickly introduce the general idea behind using function approximation for reinforcement learning for continuous state spaces and describe how Nadjahi et al. [11] applied this to their algorithms.

2.3.1 Mathematical framework

The mathematical framework used in Nadjahi et al. [11] is in its basics very similar to the one introduced in Section 2.1. However, they denote an MDP as $M = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ are finite state and action spaces and $\gamma < 1$ is the discount factor. P is now the state-action-state transition probability, i.e., the margin of the transition probabilities

$$P(s'|s, a) = \sum_r p(s', r|s, a). \tag{20}$$

and R the stochastic *reward function* corresponding to the density

$$p(r|s, a) = \sum_{s'} p(s', r|s, a). \tag{21}$$

Thus, $R(s, a)$ can be seen as a random variable with density $p(\cdot|s, a)$ and is assumed to be bounded, i.e., $R \in [-R_{max}, R_{max}]$.
The dynamics modeled by p are assumed to be unknown and, thus, also P and R. To use methods similar to Dynamic Programming one has to estimate these unknown quantities. Given data $\mathcal{D} = (s_j, a_j, r_j, s'_j)_{j=1,\ldots,n}$ collected by the behavior policy, with $n \in \mathbb{N}$ the

number of transitions done by the behavior policy, Nadjahi et al. [11] use the Maximum Likelihood Estimator (MLE)

$$\hat{P}(s'|s,a) = \frac{\sum_{(s_j=s,a_j=a,r_j,s'_j=s')\in\mathcal{D}} 1}{N_{\mathcal{D}}(s,a)} \tag{22}$$

for P and

$$\hat{R}(s,a) = \frac{\sum_{(s_j=s,a_j=a,r_j,s'_j)\in\mathcal{D}} r_j}{N_{\mathcal{D}}(s,a)} \tag{23}$$

for R, where $N_{\mathcal{D}}(s,a)$ denotes the number of times the state-action pair (s,a) occurred in $\mathcal{D}$. Note that contrary to $\hat{P}$, $\hat{R}$ is not an estimator of the distribution of R but instead of its expected value, which is sufficient to estimate the action Bellman Equation 14.
Using these two quantities, it is possible to define the MLE MDP as $\hat{M} = (\mathcal{S}, \mathcal{A}, \hat{P}, \hat{R}, \gamma)$. This is again a valid MDP in which all the theory presented in the previous sections holds. The occurrence of a second MDP makes it necessary to clarify in the notation of value functions which MDP is used. So, I follow Nadjahi et al. [11] and notate from now on v_π^M and q_π^M when referring to the value functions in the MDP M.
Since the authors plan on using this estimated MDP $\hat{M}$ and want to prove the safety of their algorithms, they need to bound the error of $\hat{M}$ as in the following. With probability $1-\delta$,

$$||P(\cdot|s,a) - \hat{P}(\cdot|s,a)||_1 \leq e_P(s,a), \tag{24}$$

$$|R(s,a) - \hat{R}(s,a)| \leq e_P(s,a) R_{max} \tag{25}$$

and

$$|q_{\pi_b}^M(s,a) - q_{\pi_b}^{\hat{M}}(s,a)| \leq e_q(s,a) v_{max} \tag{26}$$

holds for all state-action pairs $(s,a) \in \mathcal{S} \times \mathcal{A}$. Here, v_{max} denotes the maximum of the value function and is bounded by $v_{max} \leq \frac{R_{max}}{1-\gamma}$ and the two error functions e_P and e_q are, according to the authors, directly derived by the Hoeffding's inequality [15] as

$$e_P(s,a) = \sqrt{\frac{2}{N_{\mathcal{D}}(s,a)} \log \frac{2|\mathcal{S}||\mathcal{A}|2^{|\mathcal{A}|}}{\delta}} \tag{27}$$

and

$$e_q(s,a) = \sqrt{\frac{2}{N_{\mathcal{D}}(s,a)} \log \frac{2|\mathcal{S}||\mathcal{A}|}{\delta}}. \tag{28}$$

Note that I changed the wording of the bounds a bit to the one on page 3 in Nadjahi et al. [11], as the original wording would imply that the bound only holds with probability $1-\delta$ for a single state-action pair. However, the bounds are supposed to be valid for all state-action pairs at the same time. I get to this in detail in Section 3.2.2, where I also examine the derivation of the bounds. It is still a bit misleading, as there is no guarantee that all three bounds in Equations 24, 25, and 26 hold simultaneously. In the later chapters I demonstrate that the bound on q is sufficient, so this poses no problem.
The plan of the authors is to use these bounds to derive a bound on the performance of the newly generated policy in the true MDP M. As one might have expected, the bounds are tighter for a state-action pair if it has been visited many times, which can be directly

inferred from Equations 27 and 28. In Laroche et al. [12] they used a strict cut-off value $N_\wedge$ and gained the safety of their algorithms by only allowing a change of the behavior policy for state-action pairs with at least $N_\wedge$ visits, so that the new policy ended up with the property

$$\forall (s,a) \in \mathcal{S} \times \mathcal{A} : N_\mathcal{D}(s,a) < N_\wedge \Rightarrow \pi(a|s) = \pi_b(a|s). \tag{29}$$

For the other state-action pairs they allow computation similar to the Dynamic Programming introduced in Section 2.2. For this algorithm they proved a safety bound depending on $N_\wedge$, which is explicitly stated in Section 3.3.3.
To improve their old algorithm, the authors use a "softer" version of this kind of bootstrapping the behavior policy. That is where the name of the algorithm comes from, as they call the behavior policy the *baseline* policy. The soft baseline bootstrapping is done by using the error functions to penalize changes of the behavior policy. To quantify this, they define the following:

Definition 2 *A policy π is (π_b, ϵ, e)-constrained w.r.t. a baseline policy π_b, an error function e and a hyper-parameter ϵ, if*

$$\sum_{a \in \mathcal{A}} e(s,a)|\pi(a|s) - \pi_b(a|s)| \leq \epsilon \tag{30}$$

holds for all states $s \in \mathcal{S}$.

The first theorem in Nadjahi et al. [11] uses the following two definitions, which are extended in Section 3.2.

Definition 3 *The advantage of taking action a in state s against following policy π in the MDP M is defined as*

$$A^M_\pi(s,a) = q^M_\pi(s,a) - v^M_\pi(s). \tag{31}$$

Definition 4 *A policy π is π_b-advantageous in an MDP M, if*

$$\sum_{a \in \mathcal{A}} A^M_{\pi_b}(s,a)\pi(a|s) \geq 0 \tag{32}$$

holds for all $s \in \mathcal{S}$.

By transforming Equation 32 to

$$0 \leq \sum_{a \in \mathcal{A}} A^M_{\pi_b}(s,a)\pi(a|s) = \sum_{a \in \mathcal{A}} \left(q^M_{\pi_b}(s,a) - v^M_{\pi_b}(s)\right)\pi(a|s) = \left(\sum_{a \in \mathcal{A}} q^M_{\pi_b}(s,a)\pi(a|s)\right) - v^M_{\pi_b}(s) \tag{33}$$

one directly obtains the inequality of the Policy Improvement Theorem (Theorem 2.1) and, thus, a policy π which is π_b-advantageous in M is an improvement to π_b in M, i.e., $\pi \geq \pi_b$ in M. Now, it is possible to state the first theorem proven in Nadjahi et al. [11]:
Theorem 1 (from Soft-SPIBB): Any (π_b, e_q, ϵ)-constrained policy π that is π_b-advantageous in $\hat{M}$ satisfies the following inequality in every state s with probability at least $1 - \delta$:

$$v^M_\pi(s) - v^M_{\pi_b}(s) \geq -\frac{\epsilon v_{max}}{1-\gamma}. \tag{34}$$

I use the exact same wording as the authors and do not write it in the formal theorem layout, as I state it again later, but in a slightly different version and provide the proof there.
This theorem yields that a (π_b, e_q, ϵ)-constrained policy π, that is π_b-advantageous in $\hat{M}$, is a $\frac{\epsilon v_{max}}{1-\gamma}$-approximate $1-\delta$-safe policy improvement w.r.t. π_b.
As ϵ and δ can be any positive real numbers, this statement can get arbitrary strong, with the only limitation, that the lower bound on the value function π given by Equation 34 can never surpass the value function of π_b. Also, using a smaller δ yields a bigger error function e_p, which follows directly by its definition in 28, which results in less space for change for the new policy while staying π_b-constrained. A smaller ϵ amplifies this effect, as can be seen in Definition 2. Therefore, one faces a trade-off between safety and freedom of the new policy.
The algorithms introduced in Nadjahi et al. [11] and presented in this thesis in Section 2.3.2, are not π_b-advantageous and, thus, the authors try to prove a similar statement in the absence of this property. For this they use an assumption on the true MDP:

Assumption 1: There exists a constant $\kappa < \frac{1}{\gamma}$ such that, for all state-action pairs $(s,a) \in \mathcal{S} \times \mathcal{A}$, the following holds:

$$\sum_{s',a'} e_P(s',a')\pi_b(a'|s')P(s'|s,a) \leq \kappa e_P(s,a) \tag{35}$$

The wording is again the same as on page 6 in Nadjahi et al. [11].
This assumption is to the best of my knowledge nowhere justified and it is proven in Section 3.1 that this cannot hold for all possible MDPs. Furthermore, I show there that even in the experiments of Nadjahi et al. [11] this assumption does not hold and elaborate on the consequences of these results.
By using Assumption 1 they proved the following theorem in Nadjahi et al. [11]:

Theorem 2 (from Soft-SPIBB): Under Assumption 1, any (π_b, e_P, ϵ)-constrained policy π satisfies the following inequality in every state s with probability at least $1-\delta$:

$$v_\pi^M(s) - v_{\pi_b}^M(s) \geq v_\pi^{\hat{M}}(s) - v_{\pi_b}^{\hat{M}}(s) + 2||d_\pi^M(\cdot|s) - d_{\pi_b}^M(\cdot|s)||_1 v_{max} - \frac{1+\gamma}{(1-\gamma)^2(1-\kappa\gamma)}\epsilon v_{max} \tag{36}$$

Here, $d_\pi^M(s'|s)$ denotes the expected discounted sum of visits to s' when starting in s and I get into this in more detail in Section 3.2.1.
This theorem is the justification why it is sufficient for a policy to be (π_b, e_P, ϵ)-constrained to be safe, however, it only holds if Assumption 1 holds. Also, one could go into detail discussing if the lower bound in Equation 36 is useful, but this is not necessary as I show in Section 3.1 that Assumption 1 is not true and, so, Theorem 2 is in general not applicable.

2.3.2 Algorithms

The authors present two different algorithms, *Exact-Soft-SPIBB* and *Approx-Soft-SPIBB*, which both rely on the Generalized Policy Iteration as discussed in Section 2.2. Thus, the algorithms can be divided into a Policy Evaluation step and a Policy Improvement step.

The PE step is done by solving the system of linear equations from the action Bellman Equation given in Equation 14, but using the iterative version would work as well. The only difference is that one does not know the true transition probabilities and reward and, therefore, uses the MLE estimates $\hat{P}$ and $\hat{R}$ from Equations 22 and 23, respectively.
The PI step is what makes these algorithms special. Both try to solve the same optimization problem for every state $s \in \mathcal{S}$ at step $(i+1)$:

$$\pi^{(i+1)} = \arg\max_{\pi} \sum_{a \in \mathcal{A}} q^{\hat{M}}_{\pi^{(i)}}(s,a)\pi(a|s) \tag{37}$$

subject to:
Constraint 1: $\pi^{(i+1)}(\cdot|s)$ being a probability over $\mathcal{A}$: $\sum_{a\in\mathcal{A}} \pi^{(i+1)}(a|s) = 1$ and $\forall a \in \mathcal{A}: \pi^{(i+1)}(a|s) \geq 0$.
Constraint 2: $\pi^{(i+1)}$ being (π_b, e, ϵ)-constrained.
Here, $e : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_{\geq 0}$ is a general error function and can be chosen as e_q or e_P, for example. Note that without Constraint 2, this optimization problem would be solved by the greedy policy π' as defined in Equation 17. The difference in the two Soft-SPIBB algorithms is how they solve this optimization problem.
Exact-Soft-SPIBB: For Exact-Soft-SPIBB the optimization problem is reformulated as a Linear Program. Since $(q^{\hat{M}}_{\pi^{(i)}}(s,a_1)), ..., q^{\hat{M}}_{\pi^{(i)}}(s,a_{|\mathcal{A}|}))$ in Equation 37 may be interpreted as the (linear) cost function, where all the elements of $\mathcal{A}$ are labeled as $a_1, a_2, ..., a_{|\mathcal{A}|}$, and Constraint 1 is clearly a linear constraint on π, it suffices to reformulate Constraint 2. This is done by using, for a fixed state $s \in \mathcal{S}$, the following $2|\mathcal{A}|+1$ constraints:

$$\forall a \in \mathcal{A}: \quad \pi^{(i+1)}(a|s) - \pi_b(a|s) \leq z(s,a) \tag{38}$$

$$\forall a \in \mathcal{A}: \quad -\pi^{(i+1)}(a|s) + \pi_b(a|s) \leq z(s,a) \tag{39}$$

$$\sum_a e(s,a) z(s,a) \leq \epsilon \tag{40}$$

On page 7 in Nadjahi et al. [11], it is stated that this Linear Program is solved by the simplex algorithm. Contrary to that, in their code accompanying the paper one can see in line 235 in the file spibb.py, which implements all the SPIBB and Soft-SPIBB algorithms, that the default solver of scipy.optimize.linprog is used to solve the Linear Program in the Policy Iteration step of Exact-Soft-SPIBB.[1] However, the default solver of scipy.optimize.linprog is the interior point method as specified in Andersen and Andersen [16].[2] This difference in description and implementation does not change the algorithm fundamentally, as both solvers try to solve the same problem and should in theory obtain the same results and Nadjahi et al. [11] even discuss the similarity between both in the previous section, in 2.3.
Approx-Soft-SPIBB: As solving a Linear Program exactly is computationally expensive, Nadjahi et al. [11] also provide an approximate version. The idea behind this algorithm is to move the probability mass of the policy for each state from actions with a low value to actions of high value, while ensuring that the policy stays (π_b, e_q, ϵ)-constrained.

[1]https://github.com/RomainLaroche/SPIBB/spibb.py, assessed on 20.03.2021
[2]https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.linprog.html, assessed on 20.03.2021

Pseudo code for this algorithm is shown in Algorithm 1, which is almost the same as in the Appendix A.8 in [11]. I only changed it so that it shows the whole PI step instead of the improvement for just one specific state and changed some of the notation to fit to the rest of this thesis. Also, I added the for-loop in line 7, to make it closer to their implementation.[3]

Algorithm 1: Approx-Soft-SPIBB: Policy Improvement

```
Input      : Behavior policy $\pi_b$, action space $\mathcal{A}$, state space $\mathcal{S}$, errors $e(s,a)$ for each
             $(s,a) \in \mathcal{S} \times \mathcal{A}$, hyper-parameter $\epsilon > 0$ and last iteration action-value
             function $q^{\hat{M}}_{\pi^{(i)}}$
Initialize: $\pi^{(i+1)} = \pi_b$
1  for $s \in \mathcal{S}$ do
2      $E = \epsilon$
3      Define $\mathcal{A}^-$ as $\mathcal{A}$ in increasing order of $q^{\hat{M}}_{\pi^{(i)}}(s,\cdot)$
4      for $a^- \in \mathcal{A}^-$ do
5          $m^- = \min\{\pi^{(i+1)}(a^-|s), \frac{E}{2e(s,a^-)}\}$
6          Define $\mathcal{A}^+$ as $\mathcal{A}$ in decreasing order of $\frac{q^{\hat{M}}_{\pi^{(i)}}(s,a) - q^{\hat{M}}_{\pi^{(i)}}(s,a^-)}{e(s,a)}$
7          for $a^+ \in \mathcal{A}^+$ do
8              if $a^+ == a^-$ then
9                  break
10             end
11             $m^+ = \min\{m^-, \frac{E}{2e(s,a^+)}\}$
12             if $m^+ > 0$ then
13                 $\pi^{(i+1)}(a^+|s) = \pi^{(i+1)}(a^+|s) + m^+$
14                 $\pi^{(i+1)}(a^-|s) = \pi^{(i+1)}(a^-|s) - m^+$
15                 $E = E - m^+(e(s,a^+) + e(s,a^-))$
16                 $m^- = m^- - m^+$
17             end
18         end
19     end
20 end
```

The algorithm always starts with improving the behavior policy instead of the policy from the previous iteration, which makes it easier to ensure that the newly computed policy is also (π_b, ϵ, e)-constrained. This is done by using the budget E, which is initialized as ϵ. To determine how much probability mass is moved from one action to another, one iterates through $\mathcal{A}$, starting with the action a^- which has the lowest value in the state s. m^- is then calculated as the most one can subtract from $\pi_b(a^-|s)$ and still have a (π_b, ϵ, e)-constrained policy. In Line 6, all the actions are sorted such that the action with the highest value and lowest error function gets chosen first. This is done by looking at

[3]https://github.com/RomainLaroche/SPIBB, assessed on 20.03.2021

the quotient

$$\frac{q^{\hat{M}}_{\pi^{(i)}}(s,a) - q^{\hat{M}}_{\pi^{(i)}}(s,a^-)}{e(s,a)}. \tag{41}$$

In combination with Lines 8-10, this ensures that mass is only moved from an action with a lower value to one with a higher value and never the other way around. Line 11 establishes that the policy in the end is (π_b, ϵ, e)-constrained. In Lines 13 and 14 the probability mass is moved from the low value action a^- to the high value action a^+. The budget is updated accordingly in Line 15. In the case that $e(s,a^+) > e(s,a^-)$, it can happen that $m^+ < m^-$ and, thus, there is still some mass which could be moved away from $\pi^{(i+1)}(a^-|s)$, but it is not possible to move any more probability mass to $\pi^{(i+1)}(a^+|s)$, while staying (π_b, ϵ, e)-constrained. Thus, the movable mass is updated in Line 16 and the algorithm continues to move mass to sub-optimal actions, i.e., actions with a positive value for the quotient from Equation 41, but not the biggest one. If there is still some budget E left at the end, Approx-Soft-SPIBB also tries to move mass from other actions, which have not the lowest value, to better ones.

By the definition of the algorithms Theorem 2.2 holds [11].

Theorem 2.2: Soft-SPIBB are constrained

The policy improvement step of Exact-Soft-SPIBB and Approx-Soft-SPIBB generate policies that are guaranteed to be (π_b, ϵ, e)-constrained.

Regarding the theoretical safety of both Soft-SPIBB algorithms, the authors state that Theorem 1 is applicable to the algorithms if only one iteration is performed and the error function e_q is used. Since both algorithms are then (π_b, ϵ, e_q)-constrained, this means that Nadjahi et al. [11] think the full versions of their algorithms are not π_b-advantageous, but the one-step versions are. Why this is the case is not addressed in the paper. However, the argument relies on the optimization problem described in Equation 37, which shows that $\pi^{(i+1)}$ is always $\pi^{(i)}$-advantageous. Therefore, $\pi^{(1)}$ is π_b-advantageous, as $\pi_b = \pi^{(0)}$. However, it cannot be ensured that $\pi^{(i)}$ is π_b-advantageous for $i > 1$.

2.3.3 Empirical evaluations

After the theoretical evaluation of their algorithms the authors present an empirical one on the Random MDPs benchmark against several other algorithms. An overview of this procedure is given in Algorithm 2 which is almost the same as on page 27 in Nadjahi et al. [11].

Algorithm 2: Random MDPs benchmark

```
Input: Number of iterations $N$, list of hyper-parameter values for the behavior, list
       of dataset sizes, list of algorithms in the benchmark, list of hyper-parameter
       values for each algorithm
for $i = 1, .., N$ do
    Generate an MDP
    for each hyper-parameter value for the behavior do
        Generate a behavior.
        for each dataset size do
            Generate a dataset.
            for each algorithm do
                for each algorithm hyper-parameter value do
                    Train a policy.
                    Evaluate the policy.
                    Record the performance of the trained policy on this run.
                end
            end
        end
    end
end
```

So, in every iteration a new MDP is generated. The generation follows these rules:

1. There are 50 states among which 1 state is terminal. Therefore, it is an episodic task, always starting in the state 0.
2. In each state, except the terminal one, there are 4 actions available.
3. From each state-action pair (s, a), where s is not terminal, there are 4 possible next states, i.e., the transition probability $P(s'|s, a)$ is nonzero for 4 different states s'.
4. For which states s' the probability $P(s'|s, a)$ is nonzero is randomly determined and also the value of $P(s'|s, a)$ is randomly determined.
5. The reward at time t only depends on the next state S_{t+1} and it is 0 for every non-terminal state, but 1 for the terminal state. Therefore, as the discount factor is less than 1, the whole task can be described as trying to find the fastest way to the terminal state.
6. The terminal state is chosen such that the optimal policy achieves the lowest performance if this state is chosen as the terminal state. This ensures that the randomly generated MDP does not turn out to be too simple.

Thus, the random quantities of these Random MDPs are the transition function and the terminal state. The goal of the benchmark is to use behavior policies with different performances to check how well the algorithms perform when confronted with either already very good policies or very bad ones. This is managed by a ratio parameter η from which the goal performance of the behavior can be calculated by $\rho_{\pi_b} = \eta\rho_{\pi_*} + (1-\eta)\rho_{\tilde{\pi}}$, where ρ_π is the performance $\rho_\pi = v_\pi(0)$ of some policy π, $\tilde{\pi}$ denotes the uniform policy, i.e., $\tilde{\pi}(a, s) = \frac{1}{|\mathcal{A}|}$ for every $a \in \mathcal{A}$ and $s \in \mathcal{S}$, and π_* is the optimal policy. The behavior policy is then generated using the following three steps [12]:

1. The optimal policy and its action-value function q_* for this specific MDP are calculated.

2. The softmax function $\pi(a|s) = \sigma_\lambda \left((q_*(s,0), q_*(s,1), q_*(s,2), q_*(s,3))\right)_a = \frac{e^{\lambda q_*(s,a)}}{\sum_{i=0}^{3} e^{\lambda q_*(s,i)}}$ is applied to generate the new policy π in state s. This is done for every state and for decreasing heat parameter λ until a performance of $\frac{\rho_{\pi_b} + \rho_{\pi_*}}{2}$ is reached. Note that for $\lambda \to \infty$ the optimal policy is approached and for $\lambda = 0$ the uniform policy is reached. Thus, the heat parameter is chosen higher if a higher performance is the goal.

3. In the last step, a random state s is chosen and $\pi(a|s)$ is decreased by some constant, where $a = \arg\max_a q_*(s,a)$. $\pi(\cdot|s)$ is then normalized to be a distribution again. This is repeated until the goal performance ρ_b is reached.

By observing a specified number of episodes of the behavior policy on the MDP, the batch dataset for the learning algorithms is generated. The data collected during one episode will in the following be referred to as one *trajetory*. Now, each algorithm is used for a list of hyperparameter-values to train a new policy, which will be evaluated on the MDP and its performance will be saved. The algorithms used are [11]:

- Basic RL: Dynamic Programming as explained in Section 2.2 using the estimated transition probabilities and reward function from Equations 22 and 23.

- Soft-SPIBB: Approx-Soft-SPIBB and Exact-Soft-SPIBB as explained in Section 2.3.2 and their 1-step versions [11].

- SPIBB: The predecessor algorithms of the Soft-SPIBB algorithms [12].

- HCPI: High Confidence Policy Improvement, which uses a standard off-policy [8] learning algorithm (e.g., the authors here use Q-learning [8]) to learn a policy from the batch data. This policy is regularized to become a mixture between the policy learned and the behavior policy. For each mixture policy, a lower bound of the performance is calculated, such that the probability that this mixture policy has a performance better than this lower bound is high. Among those mixtures, the one with the highest lower bound on the estimate is chosen and if its lower bound is higher than the performance of the behavior policy it is returned. Otherwise the behavior policy is returned [17].

- Reward-adjusted MDPs: This algorithm incorporates the uncertainty of a state-action pair by using $\tilde{R}(s,a) = \hat{R}(s,a) - \frac{\kappa}{\sqrt{N_\mathcal{D}(s,a)}}$ instead of $\hat{R}(s,a)$ and then applying Dynamic Programming from Section 2.2 [18].

- Robust MDP: An algorithm which is able to provide safety guarantees by considering the worst-case scenario [19].

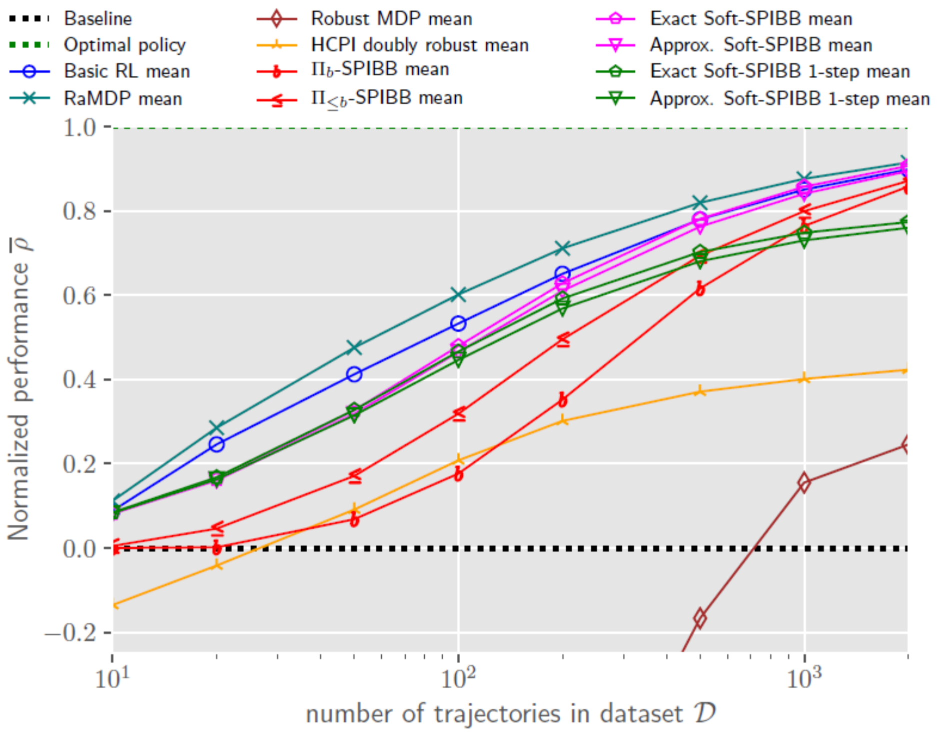
Figure 4 from page 11 in Nadjahi et al. [11] shows the result for a behavior policy with a ratio performance of 0.9, which was the highest value used in this experiment and all the algorithms are run with their best hyper-parameter. The number of iterations run in this experiment are 100, 000 and the performance is normalized to make them more comparable between different runs by calculating

$$\bar{\rho}_\pi = \frac{\rho_\pi - \rho_{\pi_b}}{\rho_{\pi_*} - \rho_{\pi_b}}. \tag{42}$$

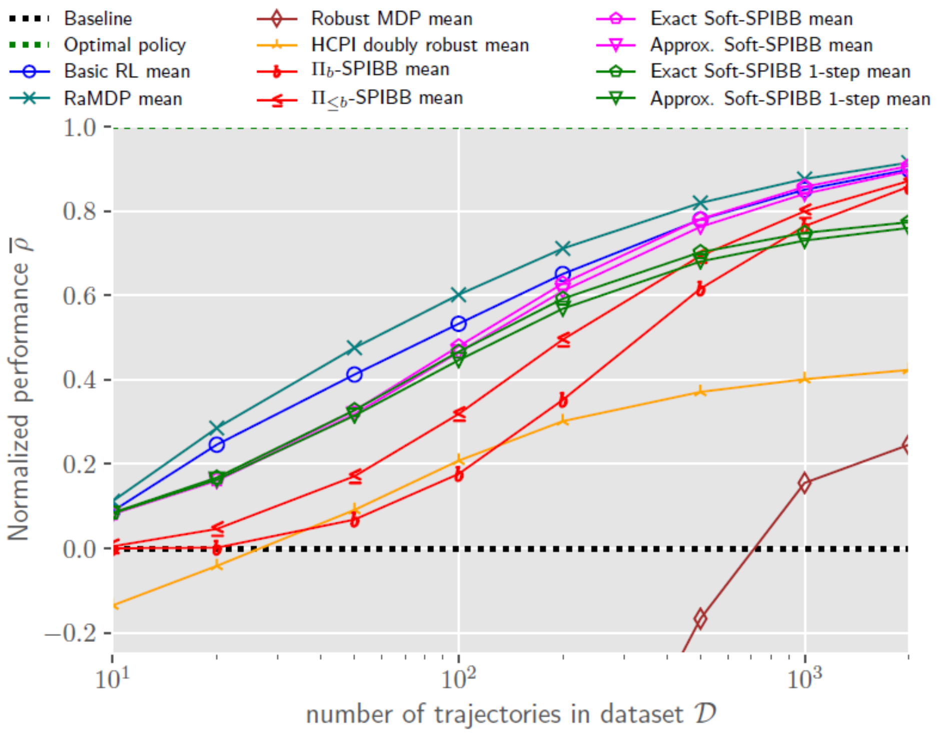
Thus, $\bar{\rho}_\pi < 0$ means a worse performance than the behavior policy, $\bar{\rho}_\pi > 0$ means an improvement w.r.t. the behavior policy and $\bar{\rho}_\pi = 1$ means the optimal performance was reached. In Figure 4a) the mean normalized performance over all the iterations are plotted with the number of trajectories used to learn from as the x-axis. RaMDP and Basic RL outperform all the others, followed by the full Soft-SPIBB algorithms. The worst performance is achieved by Robust MDP and HCPI.
As Nadjahi et al. [11] are concerned with safe policy improvement, their goal is not to simply maximize the achieved mean, but to develop algorithms with a low probability of performing extremely poorly. For this reason, they look at the 1%-CVaR (Critical Value at Risk) which is the mean performance over the 1% worst runs. This is depicted in Figure 4b) and the advantage of the Soft-SPIBB and SPIBB algorithms now becomes obvious.
In the course of this thesis I present similar experiments including the same algorithms except—due to their poor performance shown here—Robust MDP and HCPI. Instead I add further algorithms from the literature or developed in this thesis.

2.3.4 Function approximation

So far, this thesis focused on reinforcement learning with small and finite state and action spaces, however big or even continuous state spaces are becoming more important for the industry as many tasks involve continuous parameters, e.g., for the gas turbine already mentioned in the introduction [9].
One of the problems of a very large finite or continuous state space is that it is infeasible or even impossible to learn the value functions exactly. Nadjahi et al. [11] circumvent this by applying function approximation to the value function similar to how it is also done for other kinds of machine learning [8]. Naturally, there are various possibilities to approximate the value function and Laroche et al. [12] and Nadjahi et al. [11] apply a dense neural network [20] for this.
In this subsection, I start with explaining function approximation for reinforcement learning in general and then introduce all the concepts used by Nadjahi et al. [11] to be able to describe their method for implementing Approx-Soft-SPIBB, Π_b-SPIBB and RaMDP using function approximation.
As already mentioned the idea is to use a neural network Q_w with weights w to map a state s to a vector $Q_w(s) \in \mathbb{R}^{|\mathcal{A}|}$ which gives the approximate value for each action in this state s. A problem that arises is that training a neural network requires targets which one does not have in reinforcement learning, since the true action-value function is in general not known. Thus, similar to the update in Equation 16 for iterative policy iteration, one can utilize the Bellman equation (Equation 14) and reformulate it using the expected



(a) *Mean*



(b) *1%-CVaR*

Fig. 4. *Random MDPs benchmark: mean (a) and 1%-CVaR (b) performances for a strong behavior policy ($\eta = 0.9$) and Soft-SPIBB with $\epsilon = 2$.*

value for a continuous state-space [8]:

$$q_\pi(s,a) = E[R_{t+1} + \gamma \sum_{a'} \pi(a'|S_{t+1}) q_\pi(S_{t+1}, a')|S_t = s, A_t = a]. \tag{43}$$

Therefore, it is possible to use

$$y_a = E[R_{t+1} + \gamma \sum_{a'} \pi_n(a'|S_{t+1}) Q_{w_n}(S_{t+1}, a')|S_t = s, A_t = a] \tag{44}$$

as the target for a state s and action a to update the neural network using the weights w_n in the n-th step. A popular adaption of this, which is used in the discrete but also in the continuous setting, is *Q-learning* [8]. Q-learning incorporates the greedy choice of the policy in the Policy Improvement step into the Policy Evaluation step. This is done by using the maximal action-value in the next step instead of the current policy π_n to compute the target, which yields the target

$$y_a = E[R_{t+1} + \gamma \max_{a'} Q_{w_n}(S_{t+1}, a')|S_t = s, A_t = a]. \tag{45}$$

The expected value can be substituted by the unbiased estimator

$$y_a = r + \gamma \max_{a'} Q_{w_n}(s', a') \tag{46}$$

for each step consisting out of a state s, an action a, a reward r and a next state s'. As the loss function Nadjahi et al. [11] used the huber loss [21]

$$L_\delta(x,y) = \begin{cases} \frac{1}{2}(x-y)^2, & \text{if } |x-y| \leq \delta, \\ \delta|x-y| - \frac{1}{2}\delta^2, & \text{otherwise,} \end{cases} \tag{47}$$

which works similar to the L^2 loss for small errors, but grows linearly as soon as the error is greater than some $\delta > 0$ to counter the influence of outliers.
For online reinforcement learning Mnih et al. [22] present two additional modifications. Firstly, they propose to not use the data in the order it is generated, but to instead use *experience replay* which means that they collect a set of the last k transitions and use for the next update a randomly sampled one from this set. Secondly, they recommend not to use the latest weights in the target in Equation 45 as continuously changing targets make the algorithm unstable. Instead, they make use of two neural networks: the *training network* which one wants to train and the *target network* which is only used to generate the targets. After a fixed number of updates, the weights of the target network are set to the weights of the training network.
The problem of defining the target as in Equation 45 is that using the maximal action-value for the next state in the target can result in an overestimation of the value function. For this reason, Hasselt et al. [23] propose to use *Double Q-learning*, which decouples the selection of the best action and the evaluation of this action by training two different neural networks to approximate the action-value. So, let w_n and w'_n be the weights for

two different neural networks and assume one wants to update w_n in this step. Then, the target is defined as

$$y_a = E[R_{t+1} + \gamma Q_{w_n'}(S_{t+1}, \arg\max_{a'} Q_{w_n}(S_{t+1}, a'))|S_t = s, A_t = a]. \tag{48}$$

Overestimating action-values is not too problematic for online reinforcement learning, as it increases the exploration. The opposite holds for offline reinforcement learning, as the algorithm lacks the opportunity to correct its bias through new observations. For this reason Nadjahi et al. [11] use an adaption of Double Q-learning for the implementation of all their algorithms. However, they do not rely on training two neural networks in parallel. Instead they reuse the target neural network introduced before.
The whole family of SPIBB algorithms relies heavily on the number of times a state-action pair has been visited. For continuous state spaces, this is usually not possible, as most of the state-action pairs have never been visited. Furthermore, the idea behind function approximation is to generalize the knowledge of one state-action pair to other close state-action pairs. Therefore, it makes sense to use a *pseudo-count* which can be interpreted as a continuous version of state-action counts. There are many sophisticated possibilities to compute them, see Bellemare et al. [24] for example, and Nadjahi et al. [11] rely for computational reasons on the euclidean norm to compute the pseudo-count for one state-action pair by

$$N_{\mathcal{D}}(s, a) = \sum_{(s',a') \in \mathcal{D}, a'=a} \max\left(0, 1 - \frac{1}{\eta}||s - s'||\right), \tag{49}$$

where $\eta > 0$ should be chosen to fit the MDP.
Nadjahi et al. [11] implemented various different algorithms to use for the benchmark.
Vanilla DQN: They call the first one Vanilla DQN, due to the naming in Mnih et al. [22], which is exactly what is described above and, thus, it uses the targets

$$y_a = E[R_{t+1} + \gamma Q_{w_m}(S_{t+1}, \arg\max_{a'} Q_{w_n}(S_{t+1}, a'))|S_t = s, A_t = a] \tag{50}$$

in step n, where $m \leq n$ and w_m gets updated to w_n after a fixed number of steps.
RaMDP-DQN: The approximative version of RaMDP simply adapts the target in Equation 50 to

$$\begin{aligned} y_a = E[R_{t+1} + \gamma Q_{w_m}(S_{t+1}, \arg\max_{a'} Q_{w_n}(S_{t+1}, a')) \\ -\kappa N_{\mathcal{D}}(s, \arg\max_{a'} Q_{w_n}(S_{t+1}, a'))|S_t = s, A_t = a] \end{aligned} \tag{51}$$

and, thus, follows the same heuristic as the tabular version of RaMDP.
SPIBB-family-DQN: Nadjahi et al. [11] provide the implementation for SPIBB-DQN and Approx-Soft-SPIBB-DQN. The idea behind these algorithms is to keep the structure of Dynamic Programming. In each iteration the Policy Improvement step is conducted following the same rules as specified in Algorithm 1 for all the states which come up as a "next state" in the batch which is used to learn from in this iteration. To avoid running into an overestimation here, Nadjahi et al. [11] apply the idea from Double Q-learning and use the training network to compute the action-value function which is used to determine the policy in the PI step. The Policy Evaluation step is done by optimizing the parameters of the training network as shown in Equation 44 by utilizing the just computed policy.

Nadjahi et al. [11] conduct experiments for these 4 algorithms, in which their algorithms show a significant advantage w.r.t. to Vanilla-DQN and RaMDP-DQN. This serves as the motivation to try to reproduce these results on a new benchmark in Section 3.4.4.

3 Main Part

After this introduction to the essence of reinforcement learning and its most important concepts for this thesis and an overview of the paper “Safe Policy Improvement by Soft-Baseline Bootstrapping”[11], I start now towards the main results in the thesis: the correction of the theory presented in Nadjahi et al. [11] and the adaption of their algorithms such that the theory is applicable to them.
Section 3.1 completely focuses on Assumption 1 from the paper as well as its flaws and consequences for the theory.
Section 3.2 aims at saving the mathematical theory backing up the Soft-SPIBB algorithms. In order to do this, Theorem 1 is revisited and while some minor flaws in the proof are eliminated, a fundamental difference in the statement is discovered, which shows that the theorem is not only invalid for the Soft-SPIBB algorithms in their full version, but also in their 1-step version. Instead, I present an adaption of the Soft-SPIBB algorithms, for which the theorem holds.
In Section 3.3, I introduce algorithms that are used as a comparison against the original Soft-SPIBB algorithms and the new ones introduced in Section 3.2. Apart from using already existing algorithms, I adapt the exploratory algorithm R-MAX from Brafmann and Tennenholtz [28] to create the uncertainty incorporating algorithm R-MIN and add a heuristic to DUIPI [29] to make it more suitable for offline reinforcement learning. Furthermore, I develop the Lower-Approx-Soft-SPIBB which applies a heuristic already used in Laroche et al. [12] to the original Approx-Soft-SPIBB algorithm.
The comparison is done in Section 3.4, where I test on the one hand if the original Soft-SPIBB algorithms perform as well as in Nadjahi et al. [11] and on the other hand if their new versions, which are provably safe, can compete with them. Furthermore, I start some exploratory experiments to test the performance of the implementations of RaMDP, Π_b-SPIBB, Approx-Soft-SPIBB, and Lower-Approx-Soft-SPIBB using function approximation on a benchmark with a continuous state space.

3.1 Discussion of Assumption 1

This section focuses on Assumption 1 from Nadjahi et al. [11]. Section 3.1.1 starts with a thorough mathematical analysis including an interpretation of Assumption 1 and also the proof, why it cannot hold in general. In Section 3.1.2, I show that this is not only true for some artificially designed MDPs, but does also not hold for the MDPs from the Random MDPs benchmark. I conclude the discussion with the mathematical consequences of these results in Section 3.1.3.

3.1.1 Mathematical analysis

To start with the mathematical analysis of Assumption 1, I repeat its statement:
Assumption 1: There exists a constant $\kappa < \frac{1}{\gamma}$ such that, for all state-action pairs $(s, a) \in \mathcal{S} \times \mathcal{A}$, the following holds:

$$\sum_{s',a'} e_P(s', a')\pi_b(a'|s')P(s'|s, a) \leq \kappa e_P(s, a) \tag{52}$$

The wording is again the same as on page 6 in Nadjhai et al. [11].
In the paper I did not find any validation, nor justification nor interpretation of it. However, the interpretation is straightforward, as the product $\pi_b(a'|s')P(s'|s,a)$ can be seen as the probability of receiving the state-action pair (s', a') after observing (s, a). So, Equation 52 can be rewritten as

$$E_{P,\pi_b}[e_P(S_{t+1}, A_{t+1})|S_t = s, A_t = a] \leq \kappa e_P(s,a). \tag{53}$$

In words, Assumption 1 assumes that the expectation of the error function of the next state-action pair can be bounded with a constant $\kappa < \frac{1}{\gamma}$ times the error function of the current state-action pair. And this has to hold for any state-action pair, for any behavior policy, for any by data set collected by the behavior policy and for any MDP.
For a further analysis, it might be helpful to repeat the definition of e_p:

$$e_P(s,a) = \sqrt{\frac{2}{N_{\mathcal{D}}(s,a)} \log \frac{2|\mathcal{S}||\mathcal{A}|2^{|\mathcal{A}|}}{\delta}}. \tag{54}$$

Thus, for the same MDP and confidence parameter $\delta > 0$, the whole variation of e_P is due to the variation in $N_{\mathcal{D}}$, the number of visits of the state-action pairs in the data $\mathcal{D}$. Consequently, Assumption 1 might appear reasonable, as a relation between the number of visits of one state-action pair and the number of visits of state-action pairs which have a high probability of occurring after visiting the first one should be expected. However, it is shown in the following that Assumption 1 does not hold in general.
To show that Assumption 1 does not hold in every scenario, let (s, a) and $(\hat{s}, \hat{a})$ be two state-action pairs with

$$1 > \pi_b(\hat{a}|\hat{s})P(\hat{s}|s,a) > 0. \tag{55}$$

Let $\mathcal{D}$ be data collected by the behavior policy π_b, such that $N_{\mathcal{D}}(s,a) > 0$ but $N_{\mathcal{D}}(\hat{s},\hat{a}) = 0$. This is possible, as Equation 55 is assumed. Therefore, by the definition of e_P in Equation 54 one gets

$$e_P(s,a) < \infty \quad \text{and} \quad e_P(\hat{s},\hat{a}) = \infty. \tag{56}$$

Thus, the left hand side of Equation can be bounded 52 by using Equation 55 and 56

$$\sum_{s',a'} e_P(s',a')\pi_b(a'|s')P(s'|s,a) \geq e_P(\hat{s},\hat{a})\pi_b(\hat{a}|\hat{s})P(\hat{s}|s,a) = \infty. \tag{57}$$

Consequently, Assumption 1 cannot hold in this example as the right hand side of Equation 52 is finite due to Equation 56.
The setup of this counterexample is not very unrealistic, as it only assumes that there is a state-action pair $(\hat{s}, \hat{a})$ which has never been visited by the behavior policy, but there is another state-action pair (s, a), which has been visited and the probability of observing $(\hat{s}, \hat{a})$ after observing (s, a) is not zero, but can still be arbitrary small. As long as not enough data is used to visit all the state-action pairs in the experiments in Section 3.4, this is a scenario which happens almost every time. This can also be seen in Section 3.1.2.
The previous considerations is the easiest way to see that Assumption 1 cannot hold for every MDP, every behavior policy π_b, every data set observed when following π_b and every state-action pair. However, there are two possible arguments against it. Firstly, one

could counter it, by observing enough data that every possible state-action pair, i.e., every state-action pair with a non-zero probability, is encountered at least once. Obviously, this might take quite some time, but is theoretically possible. Secondly, one could change the definition of e_P as Equations 24 and 25 are sufficient for the theory in Nadjahi et al. [11] and the definition given in Equation 54 is just one possible way of achieving this. Therefore, I define

$$e'_P(s,a) = \begin{cases} e_P(s,a), & \text{if } N_{\mathcal{D}}(s,a) > 0 \\ 2, & \text{otherwise} \end{cases} \tag{58}$$

as the new error function. Since $P(\cdot|s,a)$ and $\hat{P}(\cdot|s,a)$ are probability distributions and $R(s,a) \in [-R_{max}, R_{max}]$ is bounded,

$$||P(\cdot|s,a) - \hat{P}(\cdot|s,a)||_1 \leq 2 \quad \text{and} \quad |R(s,a) - \hat{R}(s,a)| \leq 2R_{max} \tag{59}$$

hold. Therefore, e'_p fulfills Equations 24 and 25, but the argument for the invalidity of Assumption 1 cannot be applied here. However, I show in the following that there are MDPs for which Assumption 1 cannot hold for this new error function. For the proof, it is necessary to prove Lemma 1 first.

Lemma 1 *Let $A, B \in \mathbb{R}^{n\times n}$ be matrices and $\lambda > 0$. If A is positive definite and B is semi-positive definite, $A + \lambda B$ is also positive definite.*

Proof: Let $x \in \mathbb{R}^n$ be an arbitrary non-zero vector. Then

$$x^T(A + \lambda B)x = x^T Ax + \lambda x^T Bx > 0 \tag{60}$$

holds, since $x^T Ax > 0$ and $x^T Bx \geq 0$ by assumption. □

Theorem 3.1: Invalidity of Assumption 1

Let the discount factor $0 < \gamma < 1$ be arbitrary. Then there exists an MDP M with transition probabilities P such that for any behavior policy π_b and any data set $\mathcal{D}$, which contains every state-action pair at least once, it holds that, for all $0 < \delta < 1$,

$$\sum_{s',a'} e'_P(s',a')\pi_b(a'|s')P(s'|s,a) > \frac{1}{\gamma} e'_P(s,a). \tag{61}$$

This means that Assumption 1 can, independent of the discount factor, never be true for all MDPs.

Proof: Let $0 < \gamma < 1$ be arbitrary and $n \in \mathbb{N}$ be such that $\sqrt{n} > \frac{1}{\gamma}$.
Let M be the MDP displayed in Figure 5. It has $n+1$ states, from which n states are terminal states, labeled 1, 2, ..., n. In the only non-terminal state 0, there is only one action available and choosing it results in any of the terminal states with probability $\frac{1}{n}$. Also, 0 is the starting state of the MDP. As there is only one action, one can omit the

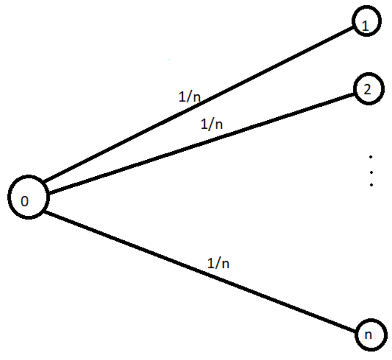


Fig. 5. *MDP with* $n+1$ *states,* n *of them are final states and in the non final state, there is only* 1 *action.*

action in the notation of e'_P and there is only one behavior policy. So, Equation 61 can be reduced to

$$\sum_{i=1}^{n} \frac{e'_P(i)}{n} > \frac{e'_P(0)}{\gamma}. \tag{62}$$

Now, I show that

$$\sum_{i=1}^{n} \frac{e'_P(i)}{n} \geq \sqrt{n} e'_P(0), \tag{63}$$

which implies Equation 62 as $\sqrt{n} > \frac{1}{\gamma}$. Let $\mathcal{D}$ denote the data collected on this MDP such that every state has been visited at least once. Thus, $N_\mathcal{D}(i) > 0$—the number of visits to state i—holds for every i. Equation 63 is equivalent to

$$\frac{1}{n} \sum_{i=1}^{n} \frac{1}{\sqrt{N_\mathcal{D}(i)}} \geq \frac{\sqrt{n}}{\sqrt{N}} \tag{64}$$

where $N = N_\mathcal{D}(0) = \sum_{i=1}^{n} N_\mathcal{D}(i)$.
Define the function $f : X \to \mathbb{R}$ by

$$f(x) = \sum_{i=1}^{n-1} \frac{1}{\sqrt{x_i}} + \frac{1}{\sqrt{N - \sum_{i=1}^{n-1} x_i}}, \tag{65}$$

with $X = \{x \in \mathbb{R}^{n-1} : x_1, ..., x_{n-1} > 0, \sum_{i=1}^{n-1} x_i < N\}$. Then for every possible data set $\mathcal{D}$, $(N_\mathcal{D}(1), ..., N_\mathcal{D}(n-1)) \in X$ and

$$f((N_\mathcal{D}(1), ..., N_\mathcal{D}(n-1))) = \sum_{i=1}^{n} \frac{1}{\sqrt{N_\mathcal{D}(i)}} \tag{66}$$

holds. Consequently, Equation 64 follows if

$$f(x) \geq \frac{n\sqrt{n}}{\sqrt{N}} \quad \forall x \in X \tag{67}$$

holds. The only thing left to show is Equation 67. At first calculate the gradient:

$$\nabla f(x) = \begin{pmatrix} -\frac{1}{2}x_1^{-\frac{3}{2}} + \frac{1}{2}(N - \sum_{i=1}^{n-1} x_i)^{-\frac{3}{2}} \\ \vdots \\ -\frac{1}{2}x_{n-1}^{-\frac{3}{2}} + \frac{1}{2}(N - \sum_{i=1}^{n-1} x_i)^{-\frac{3}{2}} \end{pmatrix} \tag{68}$$

As X is open, any minimum x' of f has $\nabla f(x') = 0$ and, thus, one arrives at the system

$$\frac{1}{2}(N - \sum_{i=1}^{n-1} x_i')^{-\frac{3}{2}} = \frac{1}{2}x_i'^{-\frac{3}{2}}, \quad i = 1, ..., n-1. \tag{69}$$

This is equivalent to

$$N - \sum_{i=1}^{n-1} x_i' = x_1' = x_2' = ... = x_{n-1}', \tag{70}$$

which is uniquely solved by $x' = (x_1', ..., x_{n-1}') = (\frac{N}{n}, ..., \frac{N}{n})$. To show that x' is a minimum of f on X, it is necessary to calculate the Hessian matrix

$$\nabla^2 f(x) = \text{diag}\left(\frac{3}{4}x_1^{-\frac{5}{2}}, ..., \frac{3}{4}x_{n-1}^{-\frac{5}{2}}\right) + \frac{3}{4}(N - \sum_{i=1}^{n-1} x_i)^{\frac{5}{2}} \begin{pmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{pmatrix}, \tag{71}$$

where diag(v) denotes a diagonal matrix with the vector v on its diagonal. As $x \in X$, it holds that $\frac{3}{4}x_i^{-\frac{5}{2}} > 0$ for all $i = 1, ..., n-1$ and $(N - \sum_{i=1}^{n-1} x_i)^{\frac{5}{2}} > 0$. Thus, all eigenvalues of $\text{diag}\left(\frac{3}{4}x_1^{-\frac{5}{2}}, ..., \frac{3}{4}x_{n-1}^{-\frac{5}{2}}\right)$ are positive and so the matrix is positive definite. Also, denoting A for the last matrix in Equation 71, one can easily check that A is positive semi-definite as the rank of A is 1, which is due to the fact that all rows are linearly dependent from each other. Therefore, there is only one non-zero eigenvalue and this is $n-1$ since

$$A \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{pmatrix} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} = \begin{pmatrix} n-1 \\ \vdots \\ n-1 \end{pmatrix} = n-1 \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}. \tag{72}$$

From Lemma 1 it follows that $\nabla^2 f(x)$ is positive definite for all $x \in X$. Thus, f is strictly convex and as in particular $\nabla^2 f(x')$ is positive definite, x' is a local minimum of the convex function f and, so, also the global minimum of f. Therefore, one gets that for all $x \in X$

$$f(x) \geq f(x') = \sum_{i=1}^{n-1} \frac{1}{\sqrt{\frac{N}{n}}} + \frac{1}{\sqrt{N - \sum_{i=1}^{n-1} \frac{N}{n}}} = (n-1)\frac{1}{\sqrt{\frac{N}{n}}} + \frac{1}{\sqrt{\frac{N}{n}}} = \frac{n\sqrt{n}}{\sqrt{N}} \tag{73}$$

holds and, thus, also Equation 67 holds, which concludes this proof. □

There are several things one should note about Theorem 3.1 and its proof. First of all, as already stated it shows that Assumption 1 cannot be assumed to be true in the generality it has been done in Nadjahi et al. [11].

Furthermore, Theorem 3.2 is stronger than necessary to disprove Assumption 1, as this assumption is a claim about all MDPs, discount factors, behavior policies on the MDP, data sets generated by the behavior policy on this MDP and safety parameters δ. So, it would have been enough to find one specific behavior policy and data set on one MDP for one specific γ and δ to show the flaw of Assumption 1. However, as γ and δ can—to some degree—be freely chosen by the developers, it is in my opinion important to realize that there is no a priori choice of the discount factor or safety parameter to counter this theorem. Additionally, as the only assumption on $\mathcal{D}$ was that every state had to be visited at least once, it shows that weakening the assumption by assuming that Equation 52 does only hold with some probability cannot work.
Even though an episodic MDP has been used in the proof, its statement also holds for non-episodic ones. This can be seen, since the class of MDPs used in the proof and depicted in Figure 5 could have been easily changed to be non-episodic, but still exhibit the same qualities for e'_P, by letting the states $1, \ldots, n$ be non-terminal and making instead the only action available in those states lead back to 0.
The class of MDPs used in the proof and depicted in Figure 5 gives a good impression what kind of constellations are critical for Assumption 1. An MDP does not have to exhibit exactly the same structure to have similar effects, it might already be enough if there is a state-action pair from which a lot of different states-action pairs are accessible, that are only accessible after this original state.
A reasonable question is whether although Assumption 1 is invalid in its generality shown at some specific class of MDPs it might hold on simple MDPs which are not built in order to disprove Assumption 1. One consideration here is that n does not need to be especially big as the proof only required $\sqrt{n} > \frac{1}{\gamma}$. So, for any $\gamma > \frac{1}{\sqrt{2}} \approx 0.707$ it suffices to choose $n = 2$. Another consideration in this direction is done by using the benchmark MDPs which were used in Nadhaji et al. [11].

3.1.2 Experiments

In this section I show that Assumption 1 also fails in the Random MDPs benchmark from Nadjahi et al. [11] which has already been introduced in Section 2.3.3.
The experiments clearly show that if e'_P is used, Equation 52 is invalid for every state-action pair for every possible κ, unless a low behavior performance ratio of $\eta = 0.1$ is used and at least 200 trajectories are observed. Then the number of invalid state-action pairs slowly goes down to about 50, which is around 25% of all state-action pairs.
If e'_P is considered, the number of state-action pairs invalidating Equation 52 for all possible κ is significantly lower and decreases further when using a higher performance ratio for the behavior policy and more trajectories, but is still not getting lower than 30 for up to $10,000$ trajectories, which is already 5 times the maximum number of trajectories used in the experiments in Nadjahi et al. [11].
The previous paragraphs focus on a discount factor of $\gamma = 0.95$ as it is used in the experiments of the paper. But even when trying a lower discount factor, e.g., $\gamma = 0.7$, it cannot be ensured that Equation 52 holds for all $\kappa < \frac{1}{\gamma}$ and for all state-action pairs as can be seen in Figure 6, where it is shown that independent of the number of trajectories and the performance of the behavior policy, the number of state-action pairs failing Equation 52 for all possible κ is never less than 10.

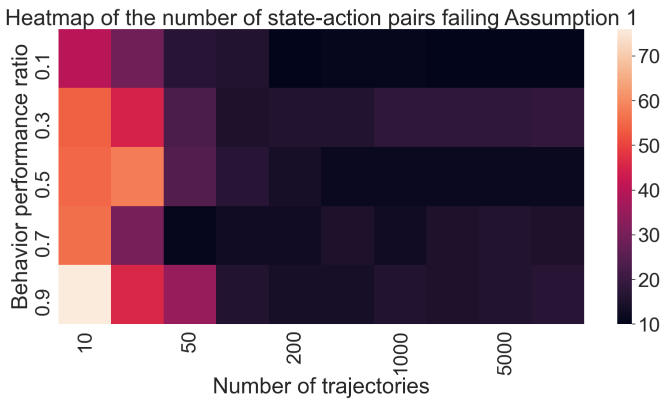


Fig. 6. *Heatmap of the number of state-action pairs for which Assumption 1 fails for* $\gamma = 0.7$ *using the error function* e'_p *on the Random MDPs benchmark [11] with the seed 1234. The x-Axis shows the number of trajectories in* $\mathcal{D}$ *and the y-axis the performance ratio* η *of the behavior policy.*

One should not pay too much attention to the exact number of state-action pairs failing Equation 52, but rather notice that Assumption 1 requires all state-action pairs to follow this inequality and, therefore, Assumption 1 does not hold for this Benchmark.

3.1.3 Mathematical consequences

The consequence of Assumption 1 not holding is that Theorem 2 cannot be proven and, therefore, the full versions, i.e., the not 1-step versions, of the Soft-SPIBB algorithms are not theoretically safe, in the sense of Definition 1.
As it was shown in Section 3.1.1, there cannot be an easy fix to Assumption 1 and, thus, there cannot be an easy workaround for Theorem 2. Therefore, the only theoretical safety provided in Nadjahi et al. [11] stems from Theorem 1, which only covers the 1-step versions, so far.

3.2 Discussion of Theorem 1

In the last Section it was shown that Assumption 1 and, therefore, also Theorem 2 do not hold. Thus, I concentrate on Theorem 1 now, as it is the only theoretical safety guarantee. I start with an introduction to the notation used in Nadjahi et al. [11] in Section 3.2.1 and prove one preliminary result there. Subsequently, in Section 3.2.2 I prove the bound on the action-value function from Equation 26, which has to be slightly modified. Here, it is also discussed in detail which action-value function can be bounded, which becomes important for the results in the end. In Section 3.2.3 I state and prove the corrected version of Theorem 1. Section 3.2.4 digs deeper into the theorem to interpret it in more detail, discusses its strengths and weaknesses, and shows some possible improvements.
Afterwards I analyze its usability for the Soft-SPIBB algorithms in Section 3.2.5. In this section, it is covered why the full versions of the Soft-SPIBB algorithms are not π_b-advantageous and, thus, Theorem 1 is not applicable to them. Furthermore, I show that the corrected version of the theorem is not applicable to the 1-step versions of the

algorithms, but it turns out that there is an easy solution for this. In the last part of this chapter, I introduce the advantageous versions of the Soft-SPIBB algorithms, which are similar to the Soft-SPIBB algorithms. However, Theorem 3.2, the corrected version of Theorem 1, is applicable to them and, thus, they can be considered as safe with respect to Definition 1.

3.2.1 Introduction of matrix notation and a first use case

To be able to prove Theorem 1, it is necessary to introduce some notation from the predecessor paper [12]. To enable the use of matrix operations all the quantities introduced so far are denoted as vectors and matrices in the following. The value functions become

$$q_\pi = (q_\pi(1,1), ..., q_\pi(1,|\mathcal{A}|), q_\pi(2,1), ..., q_\pi(2,|\mathcal{A}|), ..., q_\pi(|\mathcal{S}|,1), ..., q_\pi(|\mathcal{S}|,|\mathcal{A}|)) \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|} \tag{74}$$

and

$$v_\pi = (v_\pi(1), ..., v_\pi(|\mathcal{S}|)) \in \mathbb{R}^{|\mathcal{S}|}. \tag{75}$$

Similarly, the reward vector is

$$R = (R(1,1), ..., R(1,|\mathcal{A}|), R(2,1), ..., R(2,|\mathcal{A}|), ..., R(|\mathcal{S}|,1), ..., R(|\mathcal{S}|,|\mathcal{A}|)) \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|}. \tag{76}$$

The policy is denoted by

$$\pi = \begin{pmatrix} \pi_{\cdot 1} & 0 & 0 \\ & \ddots & \\ 0 & \pi_{\cdot j} & 0 \\ & & \ddots \\ 0 & 0 & \pi_{\cdot |\mathcal{S}|} \end{pmatrix} \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}| \times |\mathcal{S}|}, \tag{77}$$

where

$$\pi_{\cdot j} = (\pi(1|j), ..., \pi(|\mathcal{A}||j))^T \in \mathbb{R}^{|\mathcal{A}|}, \tag{78}$$

and the transition probabilities are

$$P = \begin{pmatrix} P(1|1,1) & \cdots & P(1|1,|\mathcal{A}|) & \cdots & P(1||\mathcal{S}|,|\mathcal{A}|) \\ \vdots & & \vdots & & \vdots \\ P(|\mathcal{S}||1,1) & \cdots & P(|\mathcal{S}||1,|\mathcal{A}|) & \cdots & P(|\mathcal{S}||\mathcal{S}|,|\mathcal{A}|) \end{pmatrix} \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{S}||\mathcal{A}|}. \tag{79}$$

This notation might seem unintuitive, especially for π, but with this Equation 9 becomes

$$q_\pi = R + \gamma v_\pi P \tag{80}$$

and the Bellman Equation for the state-value function from Equation 13 becomes

$$v_\pi = (R + \gamma v_\pi P)\pi. \tag{81}$$

Note that I omit the current MDP M in all these notations, as I always do, if it is clear or not relevant which MDP is currently used.
A useful quantity already mentioned in Section 2.3.1 is the following from Nadjahi et al. [11].

Definition 5 *One denotes the expectation of the discounted sum of visit to state s', when starting in state s and following policy π in MDP M, with transition probabilities P by*

$$d_\pi^M(s'|s) = \sum_{t=0}^{\infty} \gamma^t \mathbb{P}(S_t = s'|X_t \sim P\pi S_t, S_0 = s). \tag{82}$$

Using the matrix notation for π and P, it can be seen that

$$d_\pi^M(s'|s) = \sum_{t=0}^{\infty} \gamma^t (P\pi)^t_{s,s'} = (\mathbb{I} - \gamma P\pi)^{-1}_{s,s'} \tag{83}$$

where $\mathbb{I} \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$ is the identity matrix and it is known that the inverse on the right exists, since $P\pi$ is a stochastic matrix and $\gamma < 1$ [11].
Using this, I can present Proposition 1 and its proof from Nadjahi et al. [11].

Proposition 1 *Let π_1 and π_2 be two policies on an MDP M. Then*

$$v_{\pi_1}^M - v_{\pi_2}^M = q_{\pi_2}^M(\pi_1 - \pi_2)d_{\pi_1}^M \tag{84}$$

holds.

Proof: I start with applying Equation 81 on the left hand side of Equation 84 to get

$$\begin{aligned} v_{\pi_1}^M - v_{\pi_2}^M &= (R + \gamma v_{\pi_1}^M P)\pi_1 - (R + \gamma v_{\pi_2}^M P)\pi_2 \\ &= (R + \gamma v_{\pi_2}^M P)(\pi_1 - \pi_2) + \gamma(v_{\pi_1}^M - v_{\pi_2}^M)P\pi_1 \\ &= q_{\pi_2}^M(\pi_1 - \pi_2) + \gamma(v_{\pi_1}^M - v_{\pi_2}^M)P\pi_1. \end{aligned} \tag{85}$$

The second step is achieved by adding zero and reordering the summands. The last step uses Equation 80. From Equation 85 one gets that

$$(v_{\pi_1}^M - v_{\pi_2}^M)(\mathbb{I} - \gamma P\pi_1) = q_{\pi_2}^M(\pi_1 - \pi_2) \tag{86}$$

and, thus, Equation 83 yields

$$v_{\pi_1}^M - v_{\pi_2}^M = q_{\pi_2}^M(\pi_1 - \pi_2)d_{\pi_1}^M \tag{87}$$

which concludes the proof. □
Proposition 1 yields a decomposition of the difference of the state value function for two different policies on the same MDP which is utilized to prove the corrected version of Theorem 1 from Nadjahi et al. [11].

3.2.2 The correct value function

Before proving Theorem 1 it is necessary to go back to the definition of e_q in Equation 28 and the statement by Nadjahi et al. [11] that

$$\mathbb{P}\left(\forall(s,a) \in \mathcal{S} \times \mathcal{A} : |q_{\pi_b}^M(s,a) - q_{\pi_b}^{\hat{M}}(s,a)| \le e_q(s,a)v_{max}\right) \ge 1 - \delta. \tag{88}$$

Here, $q_{\pi_b}^{\hat{M}}$ refers to the true action-value function of π_b on the estimated MDP $\hat{M}$ using $\hat{P}$ and $\hat{R}$ from Equations 22 and 23, respectively. But considering the proof, however, this seems wrong. It is necessary to use the Monte-Carlo estimate for $q_{\pi_b}^{M}(s,a)$ from data $\mathcal{D}$

$$\hat{q}_{\pi_b}^{\mathcal{D}}(s,a) = \frac{1}{n}\sum_{i=1}^{n} G_{t_i} \tag{89}$$

where G_t is the return as defined in Equation 4 and $t_1, ..., t_n$ are times such that $(S_{t_i}, A_{t_i}) = (s,a)$, for all $i = 1, ..., n$. As $E[G_{t_i}] = q_{\pi_b}^{M}(s,a)$ it is possible to apply the Hoeffding's inequality for $\hat{q}_{\pi_b}^{\mathcal{D}}$, as it is done in the paper. But to the best of my knowledge I do not know how this should be done for $q_{\pi_b}^{\hat{M}}(s,a)$.

Lemma 2 ***(Hoeffding)*** *Let $X_1, ..., X_n$ be independent variables with values in $[0,1]$. Let $\bar{X}$ denote the empirical mean $\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$. Then,*

$$\mathbb{P}(\bar{X} - E[\bar{X}] \geq t) \leq \exp(-2nt^2) \tag{90}$$

for any $t \geq 0$.

This concentration inequality has been proven in Hoeffding [15] and is known as Hoeffding's inequality.

Lemma 3 *For e_q as defined in Equation 28, it holds for an arbitrary MDP M with G_{max} as an upper bound on the absolute value of the return in any state-action pair of any policy and behavior policy π_b, that*

$$\mathbb{P}_{\mathcal{D}}\left(\forall (s,a) \in \mathcal{S} \times \mathcal{A} : |q_{\pi_b}^{M}(s,a) - \hat{q}_{\pi_b}^{\mathcal{D}}(s,a)| \leq e_q(s,a) G_{max}\right) \geq 1 - \delta, \tag{91}$$

if the G_{t_i} used to estimate $\hat{q}_{\pi_b}^{\mathcal{D}}(s,a)$ are independent. Here, the subscript $\mathcal{D}$ of $\mathbb{P}$ emphasizes that the dataset generated by π_b is the random quantity.

Proof: First, note that Hoeffding's inequality implies that

$$\mathbb{P}(|\bar{X} - E[\bar{X}]| \geq t) \leq 2\exp(-2nt^2), \tag{92}$$

which follows by applying it for X and for $1 - X$.
Fix the state-action pair $(s,a) \in \mathcal{S} \times \mathcal{A}$. Then,

$$\begin{aligned} &\mathbb{P}_{\mathcal{D}}\left(\left|q_{\pi_b}^{M}(s,a) - \hat{q}_{\pi_b}^{\mathcal{D}}(s,a)\right| > e_q(s,a) G_{max}\right) \\ = \mathbb{P}_{\mathcal{D}}&\left(\left|\frac{(q_{\pi_b}^{M}(s,a) + G_{max}) - (\hat{q}_{\pi_b}^{\mathcal{D}}(s,a) + G_{max})}{2G_{max}}\right| > \sqrt{\frac{1}{2N_{\mathcal{D}}(s,a)} \log\frac{2|\mathcal{S}||\mathcal{A}|}{\delta}}\right) \\ &\leq 2\exp\left(-2N_{\mathcal{D}}(s,a)\frac{1}{2N_{\mathcal{D}}(s,a)}\log\frac{2|\mathcal{S}||\mathcal{A}|}{\delta}\right) = \frac{\delta}{|\mathcal{S}||\mathcal{A}|}, \end{aligned} \tag{93}$$

where it is possible to apply Hoeffding's inequality in the second step since

$$\frac{\hat{q}^{\mathcal{D}}_{\pi_b}(s,a) + G_{max}}{2G_{max}} = \frac{1}{n}\sum_{i=1}^{n} \frac{G_{t_i} + G_{max}}{2G_{max}} \tag{94}$$

is the empirical mean of independent variables which are bounded in $[0,1]$ as G_{t_i} are independent and bounded in $[-G_{max}, G_{max}]$ and

$$E[\hat{q}^{\mathcal{D}}_{\pi_b}(s,a)] = E[G_{t_i}] = q^M_{\pi_b}(s,a). \tag{95}$$

□

In Nadjahi et al. [11], it is not explained why the Hoeffding's inequality works for $q^{\hat{M}}_{\pi_b}$. To the best of my knowledge, the only way Hoeffding's inequality is applicable, is when $\hat{q}^{\mathcal{D}}_{\pi_b}$ is used. For this reason, I substitute $q^{\hat{M}}_{\pi_b}$ by $\hat{q}^{\mathcal{D}}_{\pi_b}$. That both are two different quantities and not just computed differently, can be tested in the file *difference_in_action_value_functions.py*.[4]
Furthermore, Nadjahi et al. [11] use v_{max} instead of G_{max} as stated in Equation 88. For this kind of proof, however, v_{max} does not work, since G_t does not necessarily need to lie in the interval $[-v_{max}, v_{max}]$. This causes another change in their theorem.
The independence of G_{t_i} is essential, as otherwise Hoeffding's inequality cannot be applied. In an episodic setting this is easily achieved by ensuring that G_{t_i} and G_{t_j} are never from the same episode for $i \neq j$. In a continuous setting this is harder and might be approximated by choosing the distance between two adjacent time steps t_i and t_{i+1}, such that $\gamma^{t_{i+1}-t_i}$ is so small that the dependence between G_{t_i} and $G_{t_{i+1}}$ becomes negligible. This issue is addressed in more detail in Section 3.2.4.
To enable the use of $\hat{q}^{\mathcal{D}}_{\pi_b}$ for the advantageous property for Theorem 1, it is necessary to extend the definition given in the paper to arbitrary functions and omit the dependence on an MDP.

Definition 6 *A policy π is π_b-advantageous w.r.t. the function $q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, if*

$$\sum_a q(s,a)(\pi(a|s) - \pi_b(a|s)) \geq 0 \tag{96}$$

holds for all states $s \in \mathcal{S}$.

A policy π is π_b-advantageous in M if and only if π is π_b-advantageous w.r.t. $q^M_{\pi_b}$ since

$$\begin{aligned}\sum_a q^M_{\pi_b}(s,a)(\pi(a|s) - \pi_b(a|s)) &= \sum_a q^M_{\pi_b}(s,a)\pi(a|s) - \sum_{a'} q^M_{\pi_b}(s,a')\pi_b(a'|s) \\ &= \sum_a \left(q^M_{\pi_b}(s,a) - v^M_{\pi_b}(s)\right)\pi(a|s) = \sum_a A^M_{\pi_b}\pi(a|s).\end{aligned} \tag{97}$$

Thus, Definition 6 is an extension of Definition 4 to also allow functions which are independent of the underlying MDP, i.e., it is now possible to state this property without access to an MDP, estimated or true.

[4]https://github.com/Philipp238/Evaluation-of-Safe-Policy-Improvement-with-Baseline-Bootstrapping/tree/master/auxiliary_tests, assessed on 20.03.2021

3.2.3 The corrected Theorem 1 and its proof

I am now ready to state and prove the corrected version of Theorem 1 as Theorem 3.2.

Theorem 3.2: Safety of constrained and advantageous policies

For any (π_b, e_q, ϵ)-constrained policy that is π_b-advantageous w.r.t. $\hat{q}^{\mathcal{D}}_{\pi_b}$, which is estimated with independent returns for each state-action pairs, the following inequality holds:

$$\mathbb{P}_{\mathcal{D}}\left(\forall s \in \mathcal{S} : v^M_\pi(s) - v^M_{\pi_b}(s) \geq -\frac{\epsilon G_{max}}{1-\gamma}\right) \geq 1 - \delta, \tag{98}$$

where M is the true MDP on which the data $\mathcal{D}$ gets sampled by the behavior policy π_b, $0 \leq \gamma < 1$ is the discount factor and $\delta > 0$ is the safety parameter for e_q.

Proof: I start by applying Proposition 1:

$$\begin{aligned} v^M_\pi - v^M_{\pi_b} = q^M_{\pi_b}(\pi - \pi_b)d^M_\pi &= (q^M_{\pi_b} - \hat{q}^{\mathcal{D}}_{\pi_b} + \hat{q}^{\mathcal{D}}_{\pi_b})(\pi - \pi_b)d^M_\pi \\ &= (q^M_{\pi_b} - \hat{q}^{\mathcal{D}}_{\pi_b})(\pi - \pi_b)d^M_\pi + \hat{q}^{\mathcal{D}}_{\pi_b}(\pi - \pi_b)d^M_\pi . \end{aligned} \tag{99}$$

The goal of this proof is to lower bound these two summands. For the first summand, notice that it is a row vector and, so, a bound on the 1-norm gives a simultaneous bound for all states. Thus, applying a vector-matrix version of the Hoelder's inequality, which is proven in Lemma 5 later on, yields

$$||(q^M_{\pi_b} - \hat{q}^{\mathcal{D}}_{\pi_b})(\pi - \pi_b)d^M_\pi||_1 \leq ||((q^M_{\pi_b} - \hat{q}^{\mathcal{D}}_{\pi_b})(\pi - \pi_b))^T||_\infty ||d^M_\pi||_1, \tag{100}$$

where all the norms refer to matrix norms. Using Lemma 3 and that π is (π_b, e_q, ϵ)-constrained yields that

$$\begin{aligned} || \left((q^M_{\pi_b} - \hat{q}^{\mathcal{D}}_{\pi_b})(\pi - \pi_b)\right)^T ||_\infty &\leq \max_s \sum_a |q^M_{\pi_b}(s,a) - \hat{q}^{\mathcal{D}}_{\pi_b}(s,a)||\pi(a|s) - \pi_b(a|s)| \\ &\leq \max_s \sum_a e_q(s,a) G_{max} |\pi(a|s) - \pi_b(a|s)| \leq \epsilon G_{max} \end{aligned} \tag{101}$$

holds with probability $1 - \delta$. The next step is to compute

$$\begin{aligned} ||d^M_\pi||_1 &= \max_s \sum_{s'} \sum_{t=0}^{\infty} \gamma^t \mathbb{P}(S_t = s' | X_t \sim P\pi S_t, S_0 = s) \\ &= \max_s \sum_{t=0}^{\infty} \gamma^t \sum_{s'} \mathbb{P}(S_t = s' | X_t \sim P\pi S_t, S_0 = s) = \max_s \sum_{t=0}^{\infty} \gamma^t = \frac{1}{1-\gamma}. \end{aligned} \tag{102}$$

Thus, one can lower bound the first summand with $-\frac{\epsilon G_{max}}{1-\gamma}$ with probability $1-\delta$. To lower bound the second summand, note that all entries of d^M_π are non-negative, so Theorem 3.2 follows if $\hat{q}^{\mathcal{D}}_{\pi_b}(\pi - \pi_b)(s) \geq 0$ for every state $s \in \mathcal{S}$. So, let $s \in \mathcal{S}$ be arbitrary. Then,

$$\hat{q}^{\mathcal{D}}_{\pi_b}(\pi - \pi_b)(s) = \sum_a \hat{q}^{\mathcal{D}}_{\pi_b}(s,a)(\pi(a|s) - \pi_b(a|s)) \geq 0 \tag{103}$$

holds, since π is π_b-advantageous w.r.t. $\hat{q}^{\mathcal{D}}_{\pi_b}$. □
As mentioned in the proof, it remains to show that the inequality in Equation 100 holds. The inequality is very similar to Hoelder's inequality for vectors [25].

Lemma 4 ***(Hoelder)*** *Let $n \in \mathbb{N}$ and $v, w \in \mathbb{R}^n$ be arbitrary. Let $p, q \in [1, \infty]$ such that $\frac{1}{p} + \frac{1}{q} = 1$, where one defines "$\frac{1}{\infty} = 0$". Then,*

$$|v^T w| \leq ||v||_p ||w||_q \tag{104}$$

holds.

However, Equation 100 has a vector-matrix product on the left hand side, instead of a vector-vector product. In order to show that extending Hoeffding's inequality to matrices in general has to be done carefully, consider

$$A = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}. \tag{105}$$

Then $||A||_2 = 1$, as its eigenvectors are $(1,0)^T$ and $(1,1)^T$ with the eigenvalues 0 and 1, respectively, and $||B||_2 = 1$, as its eigenvectors are $(1,0)^T$ and $(0,1)^T$ with the eigenvalues 0 and 1, respectively. However,

$$AB = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \tag{106}$$

and, thus,

$$||AB||_1 = 2 > 1 = ||A||_2 ||B||_2. \tag{107}$$

Even though this shows that a general matrix version of Hoelder's inequality does not hold, the extension required for Equation 100 is true and is proven in Lemma 5.

Lemma 5 *Let $n \in \mathbb{N}$, $v \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n \times n}$ be arbitrary. Then,*

$$||v^T A||_1 \leq ||v||_\infty ||A||_1 \tag{108}$$

holds, where all norms refer to matrix norms.

Proof: Denote the columns of A by $a^1, ..., a^n$. Then,

$$\begin{aligned} ||v^T A||_1 = ||(v^T a^1, ..., v^T a^n)||_1 = \max_{i=1,...,n} \{|v^T a^i|\} \\ \leq \max_{i=1,...,n} \{||v||_\infty ||a^i||_1\} = ||v||_\infty \max_{i=1,...,n} \{||a^i||_1\} = ||v||_\infty ||A||_1 \end{aligned} \tag{109}$$

where I used the definition of the 1-norm for matrices as the maximum absolute column sum in the second and the last step and Hoeffding's inequality for vectors in the third step. □
I focus now on the difference of Theorem 3.2 and Theorem 1 from Nadjahi et al. [11]. One important difference discussed in the previous section is that the policy π has to be π_b-advantageous w.r.t. $\hat{q}^{\mathcal{D}}_{\pi_b}$ instead of π_b-advantageous in the MDP $\hat{M}$.

Another change is that it was necessary to substitute v_{max} by G_{max} for reasons explained in Section 3.2.2.
The last difference is that in the original formulation of their theorem, Nadjahi et al. [11] state that for one state-action pair the lower bound on the difference of the state-value functions holds with probability $1 - \delta$. However, this lower bound holds simultaneously for all state-action pairs with probability $1 - \delta$, as it should be clear from Theorem 3.2. The difference in the proof is the consideration of the extension of Hoelder's inequality in Equation 100. In the original paper, the authors used on the left hand side the supremum-norm, possibly interpreting it as a vector norm for a row vector instead of a matrix norm. Using Hoelder's inequality for this norm is, however, questionable, as usually the 1-norm is required.

3.2.4 Analysis of the theorem

In this subsection Theorem 3.2.1 is analyzed in more detail. I start by interpreting its statement and also applying it to the Random MDPs benchmark the authors tested the algorithms on. As the given bound turns out to be pretty weak, I discuss various attempts to strengthen it. The first one is done by revisiting the proof of Theorem 3.2 and the second one tries to improve the error function. In the end, I try to tackle the problem that the returns for each state-action pair need to be independent.
Theorem 3.2 yields that any (π_b, e_q, ϵ)-constrained and π_b-advantageous w.r.t. $\hat{q}^{\mathcal{D}}_{\pi_b}$ policy is a $\frac{\epsilon G_{max}}{1-\gamma}$-approximate $(1 - \delta)$-safe policy improvement w.r.t. π_b. It is only natural that the lower bound only holds with some probability close to 1 instead of 1, since there is always the possibility of observing untypical data and, thus, the action-value function is estimated poorly. However, by choosing a smaller δ, the error function is increased, and, thus, more action-value functions are encompassed at the same time, which increases the probability that the theoretical bound applies.
Furthermore, it should not be of surprise that the lower bound on the difference in the value functions is negative. One reason for this is that Theorem 3.2 holds for every behavior policy π_b and, thus, also for the optimal policy on the MDP. A positive lower bound would not be possible here and a lower bound of zero would mean a high probability that the behavior policy remains unchanged or finds another optimal policy. An additional reason for this, besides this extreme case, stems directly from the proof, where it can be seen how the lower bound is derived. Especially the bound on the first summand from Equation 99, shows that the safety provided by this theorem is rooted in the idea that it cannot be assumed that the action-value function is correctly estimated, but there are state-action pairs for which the range of possible values can be bounded more narrowly with a high probability than for others. By expressing this through the error function, the (π_b, e_q, ϵ)-constrained property is utilized to derive a maximal risk, assuming the error function actually bounded the action-value function and otherwise the δ probable case applies, meaning that the performance cannot be bounded from below. And as one never knows if the action-value function for a state-action pair is overestimated or underestimated, one cannot get rid of the possibility of a slight loss of the state-value function for the new policy. The strength of the lower bound is that it involves, besides two constants, the algorithm hyper-parameter ϵ. So, theoretically one can decrease $\epsilon > 0$ as much as desired and, thus, obtain a lower bound which is practically zero.

In the last two paragraphs I mentioned the limits of Theorem 3.2, namely, that it is only approximately and probably safe, but countered these restrictions by explaining that δ and ϵ are hyper-parameters and can, thus, be chosen as small as desired. This deserves some more considerations as the choice of these parameters have consequences for the freedom of π. Assume the extremes $\epsilon = 0$ and $\epsilon = \infty$. In the first case, any $(\pi_b, e_q, 0)$-constrained policy π has to be exactly the same as π_b, since e_q is always positive. This shows that the only way to get a lower bound of 0 in Theorem 3.2 is by setting $\pi = \pi_b$. For the other extreme case $\epsilon = \infty$, every policy is (π_b, e_q, ∞)-constrained and the theorem yields consequently a lower bound of $-\infty$. This shows the trade-off between a high ϵ which leads to high freedom of the policy and low safety and a small ϵ which results in low freedom and high safety. The remaining question is which role δ plays in this. As can be seen in the definition of the error function e_q in Equation 28, a lower value of δ increases the error function and, therefore, it becomes harder for a policy to be (π_b, e_q, ϵ)-constrained, but at the same time the safety of the theorem is reinforced. Therefore, one faces for both, ϵ and δ, a safety-freedom trade-off.
How this trade-off exactly looks like is shown experimentally in Section 3.4. For now, I examine what parameters the authors use in their benchmarks. They perform a hyper-parameter optimization for all the algorithms and for the Soft-SPIBB algorithms $\epsilon = 2$ turned out to be the best one. In the paper [11] itself, they do not talk about which δ is used, but in the code it seems like they set $\delta = 1$.[5] Obviously, this choice of δ makes no sense if one plans on using Theorem 3.2, so, they used both hyper-parameters as such, without considering what it means for the theoretical safety bound. This also becomes clear considering the choice of $\epsilon = 2$, as this means a lower bound on the difference of the value functions of -40, for their choice of $\gamma = 0.95$, as the absolute value of the return is bounded by 1. This can be lowered to 20, since in the Random MDPs benchmark $G_t \in [0, 1]$, which can be translated to $G_t \in [-\frac{1}{2}, \frac{1}{2}]$ without changing the problem of the MDP by simply changing the reward function (by setting the reward of the initial state to -0.5). As a bound on the return is always a bound on the state-value function, since it is the expected value of the returns, the difference between the value function at one state for two different policies in such a random MDP is at most 1. Thus, a choice of $\epsilon = 2$ yields a trivial lower bound. Another issue which complicates the application of this theorem is that G_{max} is a parameter not necessarily known about an MDP without having access to the transition probabilities or at least the reward function.
The previous paragraphs showed on the one hand the theoretical strength of Theorem 3.2 and on the other hand its practical limitations. As the proof used several inequalities without discussing their sharpness, there might still be some potential hidden in it. Especially the second summand in Equation 99 might be more exploitable as, so far, it is only used that all the entries involved are non-negative. Equation 99 and the inequalities from Equations 101 and 102 yield that with probability $1 - \delta$

$$v_{\pi}^{M}(s) - v_{\pi_b}^{M}(s) \geq -\frac{\epsilon G_{max}}{1-\gamma} + \sum_{s'} \left(\sum_{a} \hat{q}_{\pi_b}^{\mathcal{D}}(s', a)(\pi(a|s') - \pi_b(a|s')) \right) d_{\pi}^{M}(s'|s) \tag{110}$$

holds for all states $s \in \mathcal{S}$. The issue, one faces when trying to infer the exact value

[5]https://github.com/RomainLaroche/SPIBB/blob/master/soft_randomMDPs_main.py, line 25, assessed on 20.03.2021

of $d_\pi^M(s'|s)$, is that it involves knowledge about the transition probabilities. Instead, a possible approach might be to utilize an estimation of the transition probabilities and bounds on this estimation, similar to the MLE estimate $\hat{P}$ in Equation 22 and its error function e_P in Equation 27. A detailed discussion of this idea is beyond the scope of this thesis and might be part of later research. In this master's thesis, I show a simple fact about $d_\pi^M(s'|s)$, which enables a slightly improved version of Equation 34. For this observe that

$$d_\pi^M(s|s) = \sum_{t=0}^{\infty} \gamma^t \mathbb{P}(S_t = s | X_t \sim P\pi S_t, S_0 = s) \geq \gamma^0 \mathbb{P}(S_0 = s | X_t \sim P\pi S_t, S_0 = s) = 1, \tag{111}$$

i.e., the expected discounted sum of visits to the state in which the MDP started is at least 1. Inserting this into Equation 110 yields

$$v_\pi^M(s) - v_{\pi_b}^M(s) \geq -\frac{\epsilon G_{max}}{1-\gamma} + \sum_a \hat{q}_{\pi_b}^{\mathcal{D}}(s,a)(\pi(a|s) - \pi_b(a|s)). \tag{112}$$

Here, the second summand is easily calculated when the policy π has already been computed. This additional summand is non-negative, so, it cannot decrease the lower bound. How big is depends on the context and is investigated in the experiments in Section 3.4. Another possibility to improve Theorem 3.2 is to increase the sharpness of the error function. For this, note that it is not important how the error function is defined, for the proof of Theorem 3.2 it is sufficient that it fulfills the inequality from Lemma 3. Thus, defining e_q as the authors in Equation 28 by using Hoeffding's inequality is just one way of achieving it. One disadvantage of using Hoeffding's inequality is that it only incorporates the number of observations to estimate the uncertainty. Even though this is a valid approach, it neglects additional knowledge of the data, e.g., its variance. It should be clear that less observations of a quantity are necessary to estimate it with high confidence, if the quantity has a low variance. This is incorporated in Bennett's inequality [15]:

Lemma 6 ***(Bennett)*** *Let $X_1, ..., X_n$ be i.i.d. (independent and identical distributed) variables with values in $[0,1]$. Then,*

$$\mathbb{P}\left(\bar{X} - E[\bar{X}] \leq \sqrt{\frac{2\,\mathrm{Var}(X_1)\log\frac{1}{\delta}}{n}} + \frac{\log\frac{1}{\delta}}{3n} \right) \geq 1 - \delta \tag{113}$$

for any $\delta > 0$.

Observe that Hoeffding's inequality from Lemma 2 can be rewritten to

$$\mathbb{P}\left(\bar{X} - E[\bar{X}] \leq \sqrt{\frac{\log\frac{1}{\delta}}{2n}} \right) \geq 1 - \delta \tag{114}$$

to make it easier comparable to Equation 113. Now, one can identify several differences. Bennett's inequality states the uncertainty through two separate summands, one which

only depends on the number of observations and one which depends on both, the number of observations and of the variance. However, while the bound of the Hoeffding's inequality depends on $\sqrt{\frac{1}{n}}$, the second summand of the bound in Bennett's inequality depends directly on $\frac{1}{n}$, which converges substantially faster to zero. The first summand, opposingly, depends again on $\sqrt{\frac{1}{n}}$ but also incorporates the variance of the variables. Thus, assuming that the random variables are deterministic, i.e., there is a constant $c \in [0,1]$ such that $X_1 = c$ almost surely, one gets $\mathrm{Var}(X_1) = 0$ and, therefore, only the second summand remains, which is a great improvement in comparison to Hoeffding's bound.
Assuming the other worst case, namely, that $\mathrm{Var}(X) = \frac{1}{4}$ it can be seen that Hoeffding's inequality is sharper in this scenario as the first summand of Bennett's bound becomes exactly the bound from Hoeffding's inequality. However, for any random variable X which is bounded in $[0,1]$, Popoviciu's inequality [26] states that the variance $\mathrm{Var}(X)$ fulfills $\mathrm{Var}(X) \leq \frac{1}{4}$ and the equality is only achieved by a random variable with $\mathbb{P}(X = 0) = \frac{1}{2} = \mathbb{P}(X = 1)$. Thus, for any other sequence of i.i.d. random variables which are bounded by 0 and 1, the asymptotic bound of Bernett's inequality is lower than the asymptotic bound of Hoeffding's inequality.
However, to be able to use Bernett's inequality to compute the error function one needs to know the variance of the action-value function and this cannot be assumed to be known in general. Consequently, it is necessary to use an empirical bound like the one derived by Maurer and Pontil [27].

Lemma 7 ***(Maurer and Pontil)*** *Let $X_1, ..., X_n$ be i.i.d. variables with values in $[0,1]$ and denote with X the random vector $X = (X_1, ..., X_n)$. Then,*

$$\mathbb{P}\left(\bar{X} - E[\bar{X}] \leq \sqrt{\frac{2\widehat{\mathrm{Var}}(X)\log\frac{2}{\delta}}{n}} + \frac{7\log\frac{2}{\delta}}{3(n-1)}\right) \geq 1 - \delta \tag{115}$$

for any $\delta > 0$. Here, $\widehat{\mathrm{Var}}(X)$ denotes the sample variance

$$\widehat{\mathrm{Var}}(X) = \frac{1}{n(n-1)} \sum_{1 \leq i < j \leq n} (Z_i - Z_j)^2 \tag{116}$$

This lemma gives a similar bound as Bennett's inequality, however, adds some more constants and is, thus, weaker, as should have been expected. Following Lemma 7 I define

$$e_q^B(s,a) = 2\left(\sqrt{\frac{2\widehat{\mathrm{Var}}(\frac{G}{2G_{max}})\log(\frac{4|\mathcal{S}||\mathcal{A}|}{\delta})}{N_{\mathcal{D}}(s,a)}} + \frac{7\log(\frac{4|\mathcal{S}||\mathcal{A}|}{\delta})}{3(N_{\mathcal{D}}(s,a)-1)}\right) \tag{117}$$

with $G = (G_{t_1}, ..., G_{t_n})$. With exactly the same proof as for Lemma 3 for e_q, it is possible to show the same inequality for e_q^B, if one applies the empirical version of Bennett's inequality instead of Hoeffding's inequality:

Lemma 8 *For e_q^B as defined in Equation 117, it holds for an arbitrary MDP M with G_{max} as an upper bound on the absolute value of the return in any state-action pair of any policy and behavior policy π_b, that*

$$\mathbb{P}_{\mathcal{D}}\left(\forall(s,a) \in \mathcal{S} \times \mathcal{A} : |q_{\pi_b}^M(s,a) - \hat{q}_{\pi_b}^{\mathcal{D}}(s,a)| \leq e_q^B(s,a)G_{max}\right) \geq 1-\delta, \tag{118}$$

if the G_{t_i} used to estimate $\hat{q}_{\pi_b}^{\mathcal{D}}(s,a)$ are independent.

The advantage of this new version of the error function has already been discussed before: the improved asymptotic bound, especially for low variance returns. There are also two disadvantages. The first is that if only a few observations are available, the bound is worse than the bound achieved by the Hoeffding's bound. Even more severe, the value G_{max} is used in the definition of e_q^B and one needs to know it a priori to be able to apply this error function. However, the maximal return is not necessarily known. There are various possible approaches to solve this issue.

The first is to derive an estimation for G_{max} from the observations G_t. The disadvantage of this method is that Theorem 3.2 does not hold anymore for an approximated version of the error function. How strongly it fails depends on the MDP and how good the approximation was.

As it is sufficient to find an upper bound on e_q^B to ensure that Theorem 3.2 is still applicable, the second solution tries to find a lower bound on G_{max}. For this, one might simply choose $\hat{G}_{max} = \frac{1}{2}(\max_t G_t - \min_t G_t)$. This has the disadvantage that it increases the error function and, so, the belief of the uncertainty of the action-value function is higher than necessary and, consequently, some power of Theorem 3.2 is lost.

The last solution is to observe that $G_{max} \leq \frac{R_{max}}{1-\gamma}$. How tight this upper bound is, depends on the MDP, so, choosing $\frac{R_{max}}{1-\gamma}$ as an approximation for G_{max} is not necessarily a good idea, but might be helpful and even a provably good approximation in some MDPs. However, it is also possible to change some of the theory to use $\frac{R_{max}}{1-\gamma}$ instead of G_{max}. This would mean substituting it in Equations 118 and 117 and, thus, in Theorem 3.2 as well. This would not only enable the use of e_q^B without any approximation, it would even make Theorem 3.2 more understandable as right now the user also cannot know the value of G_{max} and, therefore, cannot interpret its statement correctly. The drawback of this would be due to the fact that the role of G_{max} was to bound the absolute return. By using a non-sharp upper bound, some of the power of Hoeffding's or Bennett's inequality gets wasted. The size of this waste, again, depends strongly on the MDP.

In Section 3.4 I empirically investigate how the error function relying on the Maurer and Pontil bound compare to the one relying on Hoeffding's bound.

In Section 3.2.2, I explained why the independence of the returns G_t is important for Theorem 3.2 to hold. I also mentioned an easy solution for this in the episodic setting. For non-episodic MDPs this turns out to be more complicated and I illustrate here the idea already mentioned in Section 3.2.2.

The idea is to reach for approximate independence. For this assume that $S_t = s = S_{t+h}$ and $A_t = a = A_{t+h}$ and, thus, the two returns G_t and G_{t+h} would be used to estimate $\hat{q}_{\pi_b}^{\mathcal{D}}(s,a)$. To see why these two returns are not independent, remember Equation 4 which yields that

$$G_t = \sum_{i=t+1}^{\infty} \gamma^{i-t-1} R_i = \sum_{i=t+1}^{t+h} \gamma^{i-t-1} R_i + \gamma^h G_{t+h}. \tag{119}$$

By the Markov property, $R_{t+1}, ..., R_{t+h}$ are independent of G_{t+h}, conditioned on the fact that $S_t = s = S_{t+h}$ and $A_t = a = A_{t+h}$ holds. Thus, $G_t - \gamma^h G_{t+h}$ and G_{t+h} are independent. Since one wants to estimate $\hat{q}^{\mathcal{D}}_{\pi_b}(s, a)$ by unbiased samples, one might not want to simply use $G_t - \gamma^h G_{t+h}$ to estimate it, but instead enforce that if two returns G_t and G_{t+h} are used, that h is so big that $\gamma^h G_{t+h}$ is negligible or in other words $G_t - \gamma^h G_{t+h}$ is approximately G_t. This can be achieved by using a return for the estimation of $\hat{q}^{\mathcal{D}}_{\pi_b}(s, a)$ only if the last return, which was used to estimate $\hat{q}^{\mathcal{D}}_{\pi_b}(s, a)$, was enough time steps ago and, otherwise, omit this return. This procedure is formalized in Algorithm 3. Naturally,

Algorithm 3: Ensuring approximate independence of the returns

```
Input  : $\mathcal{G}_1$: list of returns achieved after being in the state-action pair $(s, a)$
         $\mathcal{T}$: list of the time steps of the returns in $\mathcal{G}_1$
         $\kappa$: the maximal correlation between two returns one would accept
         $\gamma$: discount factor
Output: sublist of $\mathcal{G}_1$ such that the maximal correlation between two returns in it is
        bounded by $\kappa$
1 Initialize $\mathcal{G}_2$ as an empty list
2 Set $t_0 = -\infty$
3 for $G \in \mathcal{G}_1$ do
4     Let $t$ be the corresponding time step to $G$ in $\mathcal{T}$.
5     if $\gamma^{t-t_0} \leq \kappa$ then
6         append $G$ to $\mathcal{G}_2$
7         $t_0 = t$
8     end
9 end
```

reducing the number of returns used to estimate the value function of the behavior policy increases the error function, which makes the algorithms less flexible. However, if one wants to apply Theorem 3.2 it is important that the uncertainty estimation in the form of the error functions is correct, which is only given if independent samples are used. The problem of highly correlated data becomes clear when looking at the extreme case where the agent visited the same state-action pair n times in a row. If Algorithm 3 is not used, the error function gives the confidence for the value function at this state-action pair considering only that there were n samples it could use. However, as the correlation between each sample and its successor is exactly the discount factor, see Equation 119, the information gained by these n samples is far less than by n samples which are distributed equally over the whole trajectory.

I do not advice to use Algorithm 3 in general as it comes with a loss of information which is usually, especially in offline reinforcement learning, not a good idea. However, if one wants to be able to estimate the uncertainty as good as possible and to utilize Theorem 3.2 one should definitely consider applying it. In the experiments in Section 3.4.3 I examine the influence of this procedure.

3.2.5 Usability for the Soft-SPIBB algorithms

After discussing Theorem 3.2, the question arises if there is an algorithm one can apply it to. To be able to apply it, the algorithm has to produce a policy which is

- (π_b, e_q, ϵ)-constrained and
- π_b-advantageous w.r.t. $\hat{q}^{\mathcal{D}}_{\pi_b}$.

As stated in Theorem 2.2, the Soft-SPIBB algorithms all fulfill the first property, but struggle with the second one. Remember that the easiest way to understand what the Soft-SPIBB algorithms aim for is the optimization problem described in Equation 37. As mentioned before, both algorithms produce a sequence of policies $\pi_b = \pi^{(0)}, \pi^{(1)}, ..., \pi^{(n)}$ which fulfill that $\pi^{(i+1)}$ is $\pi^{(i)}$-advantageous in the estimated MDP $\hat{M}$. This was the reason, Nadjahi et al. [11] stated that Theorem 1 is applicable to the 1-step versions of the Soft-SPIBB algorithms. However, in Section 3.2 I have shown that the policy has to be π_b-advantageous w.r.t. $\hat{q}^{\mathcal{D}}_{\pi_b}$ instead of π_b-advantageous in the estimated MDP $\hat{M}$.
The easy solution to make Theorem 3.2 applicable to the 1-step version of Soft-SPIBB would be to focus on the optimization problem

$$\pi^{(1)} = \arg\max_{\pi} \sum_{a \in \mathcal{A}} q^{\mathcal{D}}_{\pi_b}(s,a)\pi(a|s) \tag{120}$$

subject to:
Constraint 1: $\pi^{(1)}(\cdot|s)$ being a probability over $\mathcal{A}$: $\sum_{a\in\mathcal{A}} \pi^{(1)}(a|s) = 1$ and $\forall a \in \mathcal{A} : \pi^{(1)}(a|s) \geq 0$.
Constraint 2: $\pi^{(1)}$ being (π_b, e_q, ϵ)-constrained.
The difference to the optimization problem from Equation 37 is that I used here $\hat{q}^{\mathcal{D}}_{\pi_b}$ as the action-value function I want to optimize the new policy for and, thus, $\pi^{(1)}$ is π_b-advantageous w.r.t. $\hat{q}^{\mathcal{D}}_{\pi_b}$.
It is usually desired to perform more than one step to improve the policy, as the action-value function changes for the new policy. However, it is not clear how to do it without losing the advantageous property starting from Equation 120.
A modified version of this optimization problem is shown in the next section, which enables the derivation of two algorithms which produce policies to which Theorem 3.2 is applicable.

3.2.6 Advantageous versions of the Soft-SPIBB algorithms

In this section I introduce the advantageous version of the optimization problem from Equation 37 and deduce the two algorithms Adv-Exact-Soft-SPIBB and Adv-Approx-Soft-SPIBB. The tag Adv- relates to the fact that they always compute policies which are π_b-advantageous w.r.t. $\hat{q}^{\mathcal{D}}_{\pi_b}$.
For a given error function $e : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_{\geq 0}$, the new $(i+1)^{th}$ PI step is the following:

$$\pi^{(i+1)} = \arg\max_{\pi} \sum_{a \in \mathcal{A}} q^{\hat{M}}_{\pi^{(i)}}(s,a)\pi(a|s) \tag{121}$$

subject to:
Constraint 1: $\pi^{(i+1)}(\cdot|s)$ being a probability over $\mathcal{A}$: $\sum_{a\in\mathcal{A}} \pi^{(i+1)}(a|s) = 1$ and $\forall a \in \mathcal{A}$: $\pi^{(i+1)}(a|s) \geq 0$.
Constraint 2: $\pi^{(i+1)}$ being (π_b, e, ϵ)-constrained.
Constraint 3: $\pi^{(i+1)}$ being π_b-advantageous w.r.t. $\hat{q}^{\mathcal{D}}_{\pi_b}$.
Thus, it is the same problem already solved by the original Soft-SPIBB algorithms with the additional constraint that the new policy has to be π_b-advantageous w.r.t. $\hat{q}^{\mathcal{D}}_{\pi_b}$.
Adv-Exact-Soft-SPIBB: Since by definition Constraint 3 is a linear constraint, see Definition 6, this optimization problem can be solved exactly by adding Equation 96 as one of the constraints for the linear program already described in Section 2.3.2 and then solve it for any solver, just the same as its predecessor algorithm Exact-Soft-SPIBB does it.
Adv-Approx-Soft-SPIBB: Again I build on the original Soft-SPIBB algorithm here, by using the basic structure from Approx-Soft-SPIBB and adding a budget representing the right-hand side of Equation 96 to ensure that it is non-negative all the time. This is done by simply prohibiting the movement of probability mass which would result in a negative budget. The whole pseudo code is displayed in Algorithm 4.
Since the left-hand side of Equation 96 is 0 for $\pi(\cdot|s) = \pi_b(\cdot|s)$, the budget B can be initialized as 0 in Line 3. To check whether the left-hand side increases or decreases when increasing $\pi(a^+|s)$ and decreasing $\pi(a^-|s)$, one computes the difference in $\hat{q}^{\mathcal{D}}_{\pi_b}$ for the two different state-action pairs in Line 12. If this is non-negative, one does not have to pay attention to the budget when asserting how much probability mass to move and one can just use the same m^+ as one used in Approx-Soft-SPIBB in Algorithm 1. Otherwise, one diminishes the left-hand side of Equation 96 by m^+b, where

$$b = \hat{q}^{\mathcal{D}}_{\pi_b}(s, a^+) - \hat{q}^{\mathcal{D}}_{\pi_b}(s, a^-), \tag{122}$$

when moving probability mass m^+ from a^- to a^+. Thus, it has to be ensured that

$$m^+ \leq -\frac{B}{b} \tag{123}$$

holds, since the left-hand side of Equation 96 after moving m^+ from $\pi(a^-|s)$ to $\pi(a^+|s)$ is, see the proof of Theorem 3.3,

$$B + bm^+ \geq B - b\frac{B}{b} = 0. \tag{124}$$

Line 19 simply updates B to its new value and the rest is already known from Approx-Soft-SPIBB.

Theorem 3.3: Adv-Soft-SPIBB properties

The policy improvement steps of Adv-Exact-Soft-SPIBB and Adv-Approx-Soft-SPIBB generate policies that are guaranteed to be (π_b, ϵ, e)-constrained and π_b-advantageous w.r.t. $\hat{q}^{\mathcal{D}}_{\pi_b}$.

Algorithm 4: Adv-Approx-Soft-SPIBB: Policy Improvement

```
Input      : Behavior policy $\pi_b$, action space $\mathcal{A}$, state space $\mathcal{S}$, errors $e(s,a)$ for each
             $(s,a) \in \mathcal{S} \times \mathcal{A}$, hyper-parameter $\epsilon > 0$, function $\hat{q}^{\mathcal{D}}_{\pi_b}$ and last iteration
             action-value function $q^{\hat{M}}_{\pi^{(i)}}$
Initialize: $\pi^{(i+1)} = \pi_b$
for $s \in \mathcal{S}$ do
    $E = \epsilon$
    $B = 0$
    Define $\mathcal{A}^-$ as $\mathcal{A}$ in increasing order of $q^{\hat{M}}_{\pi^{(i)}}(s,\cdot)$
    for $a^- \in \mathcal{A}^-$ do
        $m^- = \min\{\pi^{(i+1)}(a^-|s), \frac{E}{2e(s,a^-)}\}$
        Define $\mathcal{A}^+$ as $\mathcal{A}$ in decreasing order of $\frac{q^{\hat{M}}_{\pi^{(i)}}(s,a) - q^{\hat{M}}_{\pi^{(i)}}(s,a^-)}{e(s,a)}$
        for $a^+ \in \mathcal{A}^+$ do
            if $a^+ == a^-$ then
                break
            end
            $b = \hat{q}^{\mathcal{D}}_{\pi_b}(s,a^+) - \hat{q}^{\mathcal{D}}_{\pi_b}(s,a^-)$
            if $b \geq 0$ then
                $m^+ = \min\{m^-, \frac{E}{2e(s,a^+)}\}$
            else
                $m^+ = \min\{m^-, \frac{E}{2e(s,a^+)}, -\frac{B}{b}\}$
            end
            if $m^+ > 0$ then
                $B = B + bm^+$
                $\pi^{(i+1)}(a^+|s) = \pi^{(i+1)}(a^+|s) + m^+$
                $\pi^{(i+1)}(a^-|s) = \pi^{(i+1)}(a^-|s) - m^+$
                $E = E - m^+(e(s,a^+) + e(s,a^-))$
                $m^- = m^- - m^+$
            end
        end
    end
end
```

Proof: The optimization problem described in Equation 121, which is a feasible problem since π_b is a feasible point, can be solved by Linear programming and by the constraints of this problem, the produced policy is always (π_b, ϵ, e)-constrained and π_b-advantageous w.r.t. $\hat{q}^{\mathcal{D}}_{\pi_b}$. Therefore, Theorem 3.3 is proven for Adv-Exact-Soft-SPIBB.

Some of the properties of Adv-Approx-Soft-SPIBB have already been shown. First of all, since the same mechanisms, which were used for Approx-Soft-SPIBB to ensure that the produced policy is (π_b, ϵ, e)-constrained, are still applied, Adv-Approx-Soft-SPIBB is also (π_b, ϵ, e)-constrained. Furthermore, the intuition behind the budget calculation for the advantageous property has already been explained. This will be rigorously proven now by showing at first inductively that, in Line 16, B is always the correct budget, i.e.,

$$B = \sum_a \hat{q}^{\mathcal{D}}_{\pi_b}(s,a)(\pi(a|s) - \pi_b(a|s)), \tag{125}$$

where π is the current policy.
The induction start is done by observing that $\pi(\cdot|s) = \pi_b(\cdot|s)$ which means that the right-hand side of Equation 125 is 0 and that B is initialized as 0 in Line 3.
For the induction step assume that B fulfills equation 125 and, afterwards, π is updated to π' and B to B' according to Lines 19 to 21. The goal is to show that

$$B' = \sum_a \hat{q}^{\mathcal{D}}_{\pi_b}(s,a)(\pi'(a|s) - \pi_b(a|s)) \tag{126}$$

holds. For this calculate

$$\begin{aligned}
&\sum_a \hat{q}^{\mathcal{D}}_{\pi_b}(s,a)(\pi'(a|s) - \pi_b(a|s)) \\
&= \sum_a \hat{q}^{\mathcal{D}}_{\pi_b}(s,a)(\pi'(a|s) - \pi(a|s)) + \sum_a \hat{q}^{\mathcal{D}}_{\pi_b}(s,a)(\pi(a|s) - \pi_b(a|s)) \\
&= \hat{q}^{\mathcal{D}}_{\pi_b}(s,a^+)(\pi'(a^+|s) - \pi(a^+|s)) + \hat{q}^{\mathcal{D}}_{\pi_b}(s,a^-)(\pi'(a^-|s) - \pi(a^-|s)) + B \\
&= \hat{q}^{\mathcal{D}}_{\pi_b}(s,a^+)m^+ - \hat{q}^{\mathcal{D}}_{\pi_b}(s,a^-)m^+ + B = bm^+ + B = B'.
\end{aligned} \tag{127}$$

Therefore, it is proven that the budget B fulfills Equation 125 in Line 16. Enforcing the inequality from Equation 123 in Line 16 yields that B is never negative as shown in Equation 124. Therefore, the current policy is always π_b-advantageous w.r.t. $\hat{q}^{\mathcal{D}}_{\pi_b}$. Consequently, the end policy of the PI step is also π_b-advantageous w.r.t. $\hat{q}^{\mathcal{D}}_{\pi_b}$. □
Therefore, if one chooses e_q from Equation 28 as the error function for the advantageous versions of the Soft-SPIBB algorithms, the generated policies fulfill the properties required in Theorem 3.2 and are consequently $\frac{\epsilon G_{max}}{1-\gamma}$ approximate $1-\delta$-safe policy improvements w.r.t. π_b.
Thus, the whole analysis from Section 3.2.3 is applicable to policies generated from Adv-Exact-Soft-SPIBB and Adv-Approx-Soft-SPIBB, too. In Section 3.4 I empirically test how these new algorithms compare to the original Soft-SPIBB and SPIBB algorithms and to some other uncertainty incorporating offline reinforcement learning algorithms. These algorithms are introduced in Section 3.3.

3.3 Algorithms for comparison

In the previous sections I discussed the theory behind the Soft-SPIBB algorithms and concluded that the theoretical safety bounds proven in Nadjahi et al. [11] apply only to a modified version of the 1-step Soft-SPIBB algorithms. However, the Soft-SPIBB algorithms were extended to Adv-Soft-SPIBB algorithms to make Theorem 3.2, the corrected version of Theorem 1 from Nadjahi et al. [11], applicable. To assess their practical relevance I use them against several other algorithms on different benchmarks in Section 3.4. The competitor algorithms are

- Basic RL: classical Dynamic Programming as explained in Section 2.2,

- Reward adjusted MDP: an algorithm adjusting the reward to incorporate the uncertainty as already mentioned in Section 2.3.3,
- Π_b-SPIBB and $\Pi_{\leq b}$-SPIBB: the predecessor algorithms of Soft-SPIBB from Laroche et al. [12],
- Lower-Approx-Soft-SPIBB: a newly derived, heuristic adaption of Approx-Soft-SPIBB,
- R-MIN: pessimistic adaption of the R-MAX algorithm from Brafman, Tennenholtz [28],
- DUIPI: an algorithm incorporating the covariance matrix of the action-value function when choosing the policy from Schneegass et al. [29].

All of the above algorithms, except Basic RL, incorporate some kind of theoretical or heuristical safety mechanism, which is explained in the following subsections in more detail and I present the framework of my implementation in Subsection 3.3.7.

3.3.1 Basic RL

Similar to Nadjahi et al. [11], I am using Basic RL as an competitor in the benchmarks. This algorithm simply consists out of estimating the transition probabilities and reward matrix by Equations 22 and 23, respectively, and using these instead of the real but unknown transition probabilities and rewards for the Policy Iteration as described in Section 2.2.
Therefore, after the estimation process in the beginning, all the computations can be divided into Policy Evaluation (PE) and Policy Improvement (PI) steps. This is a recurring theme for the batch reinforcement learning algorithms I am considering here, as it has already been stated in Section 2.2 for general RL and it is also exploited for the implementation in Section 3.3.7.

3.3.2 RaMDP

As already explained in Section 2.3.3, RaMDP (Reward adjusted MDP) from Petrik et al. [18] modifies the reward matrix to incorporate the uncertainty in the following way:

$$\tilde{R}(s,a) = \hat{R}(s,a) - \frac{\kappa}{\sqrt{N_{\mathcal{D}}(s,a)}}, \tag{128}$$

where $\kappa > 0$ is some hyper-parameter. This algorithm has also been used in the benchmarks of Nadjahi et al. [11]. Besides this change, RaMDP works exactly as Basic RL.

3.3.3 SPIBB

Laroche et al. [12] propose the Π_b- and $\Pi_{\leq b}$-SPIBB algorithms which follow a similar idea as the Soft-SPIBB algorithms. They both have the same preparation, i.e., they estimate P and R, and conduct the same PE step as Basic RL (and Soft-SPIBB). However, they use a different PI step.

This depends on the hyperparameter $N_\wedge$, which divides the state-action pairs (s, a) in two different sets, depending on their frequency $N_\mathcal{D}(s, a)$ in the collected data $\mathcal{D}$:

$$\mathcal{B} = \{(s, a) \in \mathcal{S} \times \mathcal{A} : N_\mathcal{D}(s, a) \leq N_\wedge\} \quad \text{and} \quad \mathcal{B}^C = (\mathcal{S} \times \mathcal{A}) \setminus \mathcal{B}. \tag{129}$$

$\mathcal{B}$ is called the bootstrapped set, as Π_b-SPIBB always sets in the PI step

$$\pi'(a|s) = \pi_b(a|s), \quad \forall (s, a) \in \mathcal{B} \tag{130}$$

for the new policy π' but optimizes in its complement, i.e., sets

$$\pi'(a, s) = \begin{cases} \sum\limits_{a'|(s,a') \in \mathcal{B}^C} \pi_b(a'|s), & \text{if } a = \underset{a'|(s,a') \in \mathcal{B}^C}{\arg\max}\, q_\pi^{\hat{M}}(s, a') \\ 0, & \text{otherwise} \end{cases} \tag{131}$$

where the action-value function of the previous policy π on the estimated MDP $\hat{M}$ is used. Thus, Laroche et al. [12] try to solve the optimization problem

$$\pi' = \arg\max_\pi \sum_{a \in \mathcal{A}} q_\pi^{\hat{M}}(s, a)\pi(a|s) \tag{132}$$

subject to:
Constraint 1: $\pi'(\cdot|s)$ being a probability over $\mathcal{A}$: $\sum_{a \in \mathcal{A}} \pi'(a|s) = 1$ and $\forall a \in \mathcal{A}$: $\pi'(a|s) \geq 0$.
Constraint 2: $\pi'(a|s) = \pi_b(a|s)$ for all $(s, a) \in \mathcal{B}$.
The fact that a policy computed by Π_b-SPIBB equals the behavior policy for any state-action pair in the bootstrapped set, enables the authors to prove that Π_b-SPIBB produces policies which are ξ-approximate $1 - \delta$-safe policy improvements w.r.t. to the behavior policy π_b, where

$$\xi = \frac{4v_{max}}{1-\gamma}\sqrt{\frac{2}{N_\wedge}\log\frac{2|\mathcal{S}||\mathcal{A}|2^{|\mathcal{S}|}}{\delta}}. \tag{133}$$

Its heuristic variation $\Pi_{\leq b}$-SPIBB does not exhibit any theoretical safety. However, it performed even better than Π_b-SPIBB in their benchmarks. Its most concise description is probably via its optimization problem:

$$\pi' = \arg\max_\pi \sum_{a \in \mathcal{A}} q_\pi^{\hat{M}}(s, a)\pi(a|s) \tag{134}$$

subject to:
Constraint 1: $\pi'(\cdot|s)$ being a probability over $\mathcal{A}$: $\sum_{a \in \mathcal{A}} \pi'(a|s) = 1$ and $\forall a \in \mathcal{A}$: $\pi'(a|s) \geq 0$.
Constraint 2: $\pi'(a|s) \leq \pi_b(a|s)$ for all $(s, a) \in \mathcal{B}$.
This only differs in Constraint 2 to the optimization problem corresponding to Π_b-SPIBB. While Π_b-SPIBB prohibits the change of the behavior policy for uncertain state-action pairs, $\Pi_{\leq b}$-SPIBB just does not allow to add more weight to them but allows to reduce it.

3.3.4 Lower-Approx-Soft-SPIBB

Lower-Approx-Soft-SPIBB is a new variant of the Soft-SPIBB algorithms which incorporates the idea from the heuristic adaption of $\Pi_{\leq b}$-SPIBB from Π_b-SPIBB. As seen in Section 3.3.3 this adaption follows the idea to only consider the uncertainty of a state-action pair if the new policy is supposed to give more weight to this pair, but it is always allowed to decrease the probability of a state-action pair, independent of its uncertainty. Even though this heuristic version lost its theoretical safety, Laroche et al. [12] noticed a significant improvement with this new algorithm.
In the following, I apply the same heuristic to Approx-Soft-SPIBB. this requires an adaption of the constrainedness-property.

Definition 7 *A policy π is (π_b, ϵ, e)-lower-constrained w.r.t. a baseline policy π_b, an error function e and a hyper-parameter ϵ, if*

$$\sum_{a\in\mathcal{A}} e(s,a)\max\{0, \pi(a|s) - \pi_b(a|s)\} \leq \epsilon \tag{135}$$

holds for all states $s \in \mathcal{S}$.

The optimization problem which is supposed to be solved by Lower-Approx-Soft-SPIBB in the PI step of the $(i+1)^{th}$-iteration is:

$$\pi^{(i+1)} = \arg\max_{\pi} \sum_{a\in\mathcal{A}} q^{\hat{M}}_{\pi^{(i)}}(s,a)\pi(a|s) \tag{136}$$

subject to:
Constraint 1: $\pi^{(i+1)}(\cdot|s)$ being a probability over $\mathcal{A}$: $\sum_{a\in\mathcal{A}} \pi^{(i+1)}(a|s) = 1$ and $\forall a \in \mathcal{A}$: $\pi^{(i+1)}(a|s) \geq 0$.
Constraint 2: $\pi^{(i+1)}$ being (π_b, e, ϵ)-lower-constrained.
The pseudo code of Lower-Approx-Soft-SPIBB is depicted in Algorithm 5 and has due to the proximity to Approx-Soft-SPIBB only slight changes to its pseudo code in Algorithm 1, where only Lines 5, 11, and 15 changed. In Line 5, the algorithm can now choose the whole probability mass $\pi^{(i+1)}(a^-|s)$, as being lower-constrained is independent of the uncertainty of states I remove probability mass from. Line 15 depicts the budget update for a lower-constrained policy and Line 11 has been changed to fit to this update to ensure that the policy stays (π_b, e, ϵ)-lower-constrained.
The property to be (π_b, e, ϵ)-lower-constrained does not yield any provable theoretical properties, however, it implements an interesting safety heuristic and, thus, I test it out in the benchmarks in Section 3.4 where it can be seen that it performs really well. Also, no theoretical safeties are lost by this, as it was shown in the preceding sections that there are none applying to Approx-Soft-SPIBB.

3.3.5 R-MIN

Brafman and Tannenholtz [28] introduce the R-MAX algorithm as a simple online algorithm for exploration. The basic idea is to use the hyper-parameter $N_\wedge$ and as long as a state-action pair has not been visited more than $N_\wedge$ times yet, its value is set to the

Algorithm 5: Lower-Approx-Soft-SPIBB: Policy Improvement

```
Input      : Behavior policy $\pi_b$, action space $\mathcal{A}$, state space $\mathcal{S}$, errors $e(s,a)$ for each
             $(s,a) \in \mathcal{S} \times \mathcal{A}$, hyper-parameter $\epsilon > 0$ and last iteration action-value
             function $q^{\hat{M}}_{\pi^{(i)}}$
Initialize: $\pi^{(i+1)} = \pi_b$
1  for $s \in \mathcal{S}$ do
2      $E = \epsilon$
3      Define $\mathcal{A}^-$ as $\mathcal{A}$ in increasing order of $q^{\hat{M}}_{\pi^{(i)}}(s, \cdot)$
4      for $a^- \in \mathcal{A}^-$ do
5          $m^- = \pi^{(i+1)}(a^-|s)$
6          Define $\mathcal{A}^+$ as $\mathcal{A}$ in decreasing order of $\frac{q^{\hat{M}}_{\pi^{(i)}}(s,a) - q^{\hat{M}}_{\pi^{(i)}}(s,a^-)}{e(s,a)}$
7          for $a^+ \in \mathcal{A}^+$ do
8              if $a^+ == a^-$ then
9                  break
10             end
11             $m^+ = \min\{m^-, \frac{E}{e(s,a^+)}\}$
12             if $m^+ > 0$ then
13                 $\pi^{(i+1)}(a^+|s) = \pi^{(i+1)}(a^+|s) + m^+$
14                 $\pi^{(i+1)}(a^-|s) = \pi^{(i+1)}(a^-|s) - m^+$
15                 $E = E - m^+ e(s,a^+)$
16                 $m^- = m^- - m^+$
17             end
18         end
19     end
20 end
```

highest possible value to encourage the algorithm to take these state-action pairs for an efficient exploration.
The problem considered in this master's thesis is quite contrary to that. First of all, I am confronted with a batch offline learning problem. Furthermore, I am not interested in exploration but in safety. Consequently, I invert their technique and instead of setting the value of uncertain state-action pairs to the best possible value, I set it to the worst possible value, which is why I renamed the algorithm to R-MIN.
The estimation of the MDP and the PI step are the same as for Basic RL, but the PE step changes from Equation 16 to

$$q_{k+1}(s,a) = \sum_{s',r} p(s',r|s,a)(r + \gamma \sum_{a'} \pi(a'|s') q_k(s',a')), \quad \text{if } N_{\mathcal{D}}(s,a) > N_\wedge \tag{137}$$

and

$$q_{k+1}(s,a) = \frac{R_{min}}{1-\gamma}, \quad \text{if } N_{\mathcal{D}}(s,a) \leq N_\wedge \tag{138}$$

as $\frac{R_{min}}{1-\gamma}$ is the sharpest lower bound which works for any MDP, without any additional knowledge about its structure. Note that it is important to apply Equation 138 in every iteration of the PE step and not only at the end, as the uncertainty of one state-action pair should also influence the value of state-action pairs leading to the uncertain one with a high probability.

3.3.6 DUIPI

The approach of Schneegass et al. [29] for developing a safe RL algorithm is to calculate not only the action-value function, but also its covariance function.
Contrary to most other papers which reduce the reward function to $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, they consider

$$\hat{R}_3 : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}, \quad \hat{R}_3(s,a,s') = \frac{\sum_{(s_j=s,a_j=a,r_j,s'_j=s')\in\mathcal{D}} r_j}{N_{\mathcal{D}}(s,a,s')}. \tag{139}$$

As described for Dynamic Programming in Section 2.2, the calculation of the action-value function for some policy π in the MDP with the estimates $\hat{P}$ and $\hat{R}_3$ is done iteratively following Equation 16. The variance of q is also calculated iteratively by using uncertainty propagation, i.e., the fact that

$$\operatorname{Var}(f(X)) = D \operatorname{Var}(X) D^T \tag{140}$$

holds for a function $f : \mathbb{R}^m \to \mathbb{R}^n$ with jacobian $D_{i,j} = \frac{\partial f_i}{\partial x_j}$.
They present two different approaches in their paper, the first makes use of the complete covariance matrix, while the second neglects the correlations between state-action pairs. Schneegass et al. [29] show that the second approach is significantly faster and that the correlation is less important for MDPs with bigger state and action spaces. In the benchmark where it made a difference there were 6 states with 2 actions and in the one where it did not matter anymore there were 18 states and 4 actions. As all benchmarks considered in this thesis use MDPs which are greater than those two, I concentrate on the second approach which is called DUIPI (Diagonal Approximation of Uncertainty Incorporating Policy Iteration). By neglecting the covariances Equation 140 becomes

$$\operatorname{Var}(f(X)) = \sum_i \left(\frac{\partial f}{\partial X_i}\right)^2 \operatorname{Var}(X_i), \tag{141}$$

which should be read component-wise. Interpreting the action-value update in Equation 16 as a function, Equation 141 yields

$$\begin{aligned}\operatorname{Var}(q^m(s,a)) = \sum_{s'} (D^m_{q,q})^2 \left(\sum_{a'} \pi(a'|s')^2 \operatorname{Var}(q^{m-1}(s',a'))\right) \\ + \sum_{s'} D^m_{q,\hat{P}} \operatorname{Var}(\hat{P}(s'|s,a)) \\ + \sum_{s'} D^m_{q,\hat{R}_3} \operatorname{Var}(\hat{R}_3(s,a,s')))\end{aligned} \tag{142}$$

with the derivatives

$$\begin{aligned} D^m_{q,q} = \gamma \hat{P}(s'|s,a),\ D^m_{q,\hat{P}} = \hat{R}_3(s,a,s') + \gamma \sum_{a'} \pi(a'|s')^2 q^{m-1}(s',a') \\ \text{and } D^m_{q,\hat{R}_3} = \hat{P}(s'|s,a). \end{aligned} \tag{143}$$

Equation 142 shows that the variance of the action-value function at the current step depends on the variance of the action-value function at the previous step and on the variance of the estimators $\hat{P}$ and $\hat{R}_3$.
To make use of the variance of the action-value function, which is simultaneously computed in the PE step, the authors propose to define

$$q_u(s,a) = q_\pi^{\hat{M}}(s,a) - \xi\sqrt{\mathrm{Var}(q_\pi^{\hat{M}}(s,a))} \tag{144}$$

and use the uncertainty incorporating action-value function q_u instead of $q_\pi^{\hat{M}}(s,a)$ in the PI step. By choosing $\xi > 0$, the uncertainty of one state-action pair has a negative influence on q_u and the new policy prefers state-action pairs with a low variance.
Assuming that neglecting covariances between the state-action pairs is a good enough approximation and additionally assuming some distribution on the action-value of the state-action pairs, $q_u(s,a)$ can work as a probable lower bound on the actual action-value.
One question that remains is how to determine the covariances of the estimators $\hat{P}$ and $\hat{R}_3$. The authors present a bayesian approach for this, which is used in this thesis. As the prior on $\hat{P}$ they propose the Dirichlet distribution with density

$$\mathbb{P}(P(s_1|s_i,a_j),...,P(s_{|\mathcal{S}|}|s_i,a_j)) = \frac{1}{B(\alpha_{i,j})}\prod_{k=1}^{|\mathcal{S}|} P(s_k|s_i,a_j)^{\alpha_{k,i,j}-1} \tag{145}$$

with $\alpha_{i,j} = \sum_{k=1}^{\mathcal{S}} \alpha_{k,i,j}$ and normalizing constant

$$B(\alpha_{i,j}) = \frac{\prod_{k=1}^{|\mathcal{S}|}\Gamma(\alpha_{k,i,j})}{\Gamma(\alpha_{i,j})} \tag{146}$$

since this is a conjugate prior for $\hat{P}$. The posterior parameters are

$$\alpha_{k,i,j}^d = \alpha_{k,i,j} + N_{\mathcal{D}}(s_i,a_j,s_k) \tag{147}$$

where $N_{\mathcal{D}}(s_i,a_j,s_k)$ denotes the number of times s_k has been visited after choosing action a_j in state s_i in the data set $\mathcal{D}$ [29]. Therefore, the posterior estimator $\hat{P}$ is

$$\hat{P}(s_k|s_i,a_j) = \frac{\alpha_{k,i,j}^d}{\alpha_{i,j}^d} \tag{148}$$

with variance

$$\mathrm{Var}(\hat{P}(s_k|s_i,a_j)) = \frac{\alpha_{k,i,j}^d(\alpha_{i,j}^d - \alpha_{k,i,j}^d)}{(\alpha_{k,i,j}^d)^2(\alpha_{k,i,j}^d+1)}. \tag{149}$$

I assume no prior knowledge on the MDPs and, thus, choose $\alpha = \alpha_{k,i,j}$, for all i, j, and k, to be constant as there is no state which should be more probable a priori. If one chooses $\alpha = 1$, one assumes a uniform distribution over all possible distributions over $(P(s_1|s_i,a_j),...,P(s_{|\mathcal{S}|}|s_i,a_j))$. If one chooses $\alpha < 1$, one assumes a sparse distribution which means that only a few of those are probably non-zero. This is reasonable for an MDP, since usually only some states are reachable from a fixed state-action pair. I follow the choice of Schneegass et al. [29] on the Trap benchmark and use $\alpha = 0.1$ in my experiments.

In both benchmarks the reward matrix is deterministic, if considered as a function on the state-action-next-state triplet. So, I set the reward matrix accordingly without a need to estimate it and, thus, its variance is set to 0. This can be easily changed to non-deterministic rewards and Schneegass et al. [29] also provide a bayesian estimator for this.

The actual implementation of DUIPI is still a bit different from what I have shown here, i.e., the policy is not chosen greedy in the PI-step and for more detail see Schneegass et al. [29] and the pseudo code on page 71. At first I tried to just follow along this pseudo code but I observed that this estimation of the transition probabilities results in the possibility that none observed state-action pairs are chosen. For this reason I implemented an additional step which ensures that this does not happen. Note that this is no change in the general theory backing up DUIPI, but yields a high performance improvement, see Figure 24 in Appendix B. For the experiments in Section 3.4 I only use this new implementation of DUIPI.

3.3.7 Implementation

All of these algorithms are implemented in the directory *batch_rl_algorithms* in the github repository.[6] Nadjahi et al. [11] published code for BasicRL, RaMDP, and the SPIBB and Soft-SPIBB algorithms. My framework aims at extending this framework to all the above algorithms mentioned so far and unifying all the computations necessary for the algorithms. The Python implementation is object oriented and tries to benefit from inheritance to avoid code duplicates.

Figure 7 shows the UML class diagram of the framework. Not all of the attributes are shown in this diagram, the focus lies on the input parameters and, also, only the important methods are included. The abstract class every algorithm is inheriting from is *BatchRLAlgorithm*. The most of the computation is implemented here. As can be seen, it gets as input some basic information about the MDP, the behavior policy π_b, and the raw data collected by it. These are used to compute the estimates $\hat{P}$ and $\hat{R}$, in the *_initial_calculations()* method which calls *_build_model()* and *_compute_R_state_action()*. This is all done in the initialization of the learner. To compute a new policy, the method *fit()* has to be called, which is the only method which should usually be called outside of the class except the constructor. The method *fit()* starts the actual training process, which consists out of repeating the PE and PI step, called *_policy_evaluation()* and *_policy_improvement()*, respectively, until the action-value function converges. The here implemented PE and PI steps are the basic ones for Dynamic Programming as explained in Section 2.2.

The estimation of $\hat{P}$ is done via Equation 22 and $\hat{R}$ is computed using $\hat{P}$, as I assume in my benchmarks—similar to Nadjahi et al. [11]—that the reward is a deterministic function of the pair of the current and the next state. To extend this to a state-action-next-state triplet is straightforward and assuming that R is unknown or even on the whole triplet

[6]https://github.com/Philipp238/Evaluation-of-Safe-Policy-Improvement-with-Baseline-Bootstrapping, assessed on 20.03.2021

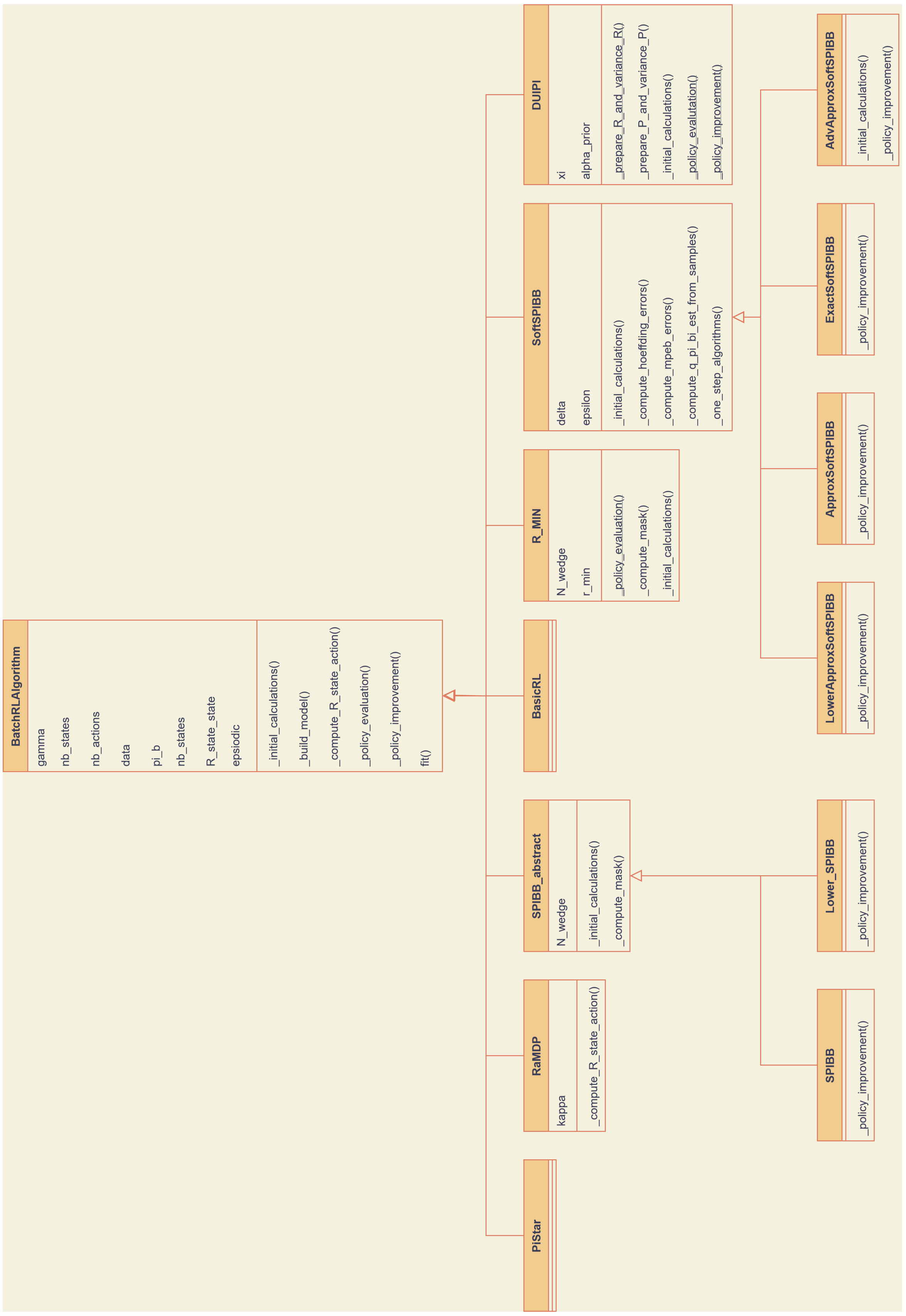


Fig. 7. *UML class diagram for the batch reinforcement learning algorithms used in this thesis.*

stochastic, would simply require substituting the current calculation with Equation 23. The rest of the implementation is independent on the calculation of $\hat{R}$.
The idea of this structure is that all the different learning algorithms inherit directly or indirectly from the base class *BatchRLAlgorithm* and only substitute the methods in which the algorithm differs. As I have already demonstrated in this section, these differences are mostly either in the

- initialization, i.e., the computation of $\hat{P}$ and $\hat{R}$,
- the policy evaluation step, or
- the policy improvement step.

This is one of the reasons for the structure of these classes. The other reason is that it gives a nice abstraction when looking at the differences between the approaches of the different algorithms. Most of the methods of this class including the ones for the PE and PI step are copied from the code from Nadjahi et al. [11] or are at least tested against their code to ensure that my implementation produces the same results. This includes the classical Dynamic Programming versions of the PI and PE step as presented in Section 2.2.
The easiest algorithm to implement is Basic RL, since it is the same as the base class and can, therefore, be left empty.
RaMDP has the same PE and PI step as Basic RL and only differs through the adjustment of its reward matrix in the beginning. This step is done exactly as it has been done in Nadjahi et al. [11].
R-MIN depends on the hyper-parameter $N_\wedge$ and it starts with a calculation of a mask marking all state-action pairs which have been visited more than $N_\wedge$ times. This is used later in its own variation of the PE step, as explained in Section 3.3.5. As this algorithm is a new adaption of R-MAX, there is no implementation I can build upon. Therefore, I implement it myself and use plausibility checks to ensure that there is no mistake in the code. For this, I especially check the influence of the parameter $N_\wedge$, which shows that R-MIN works the same as Basic RL if $N_\wedge = 0$ is used and gets more conservative for higher $N_\wedge$. Furthermore, for an extremely large $N_\wedge$ it sets the value for all state-action pairs simply to the worst possible value. These properties are exactly as R-MIN is expected to work.
DUIPI depends on the choice of the hyper-parameter ξ and the parameter α for the Dirichlet prior. During the initialization, $\hat{P}$, $\hat{R}_3$ and their variances are calculated. As I assume that the reward is deterministic, I set $\text{Var}(\hat{R}_3) = 0$. If one wants to remove this assumption from this framework, this should be corrected, too. DUIPI has a substantially different PI step from Basic RL and also the PE step is implemented individually for DUIPI, as the variance of the action-value function is computed there. Originally, DUIPI is planned to be used as an online algorithm, therefore, I cannot directly check if I can reproduce any of the results from Schneegass et al. [29]. However, I check if the influence of ξ works as expected, meaning that for a bigger ξ the choice of the new policy is driven more by the number of times a state is being visited, which turns out to be true. Furthermore, I conduct a thorough debugging process to validate every step of the algorithm and I observe a very good performance of DUIPI in the experiments in Section 3.4, which supports the believe that it is implemented correctly.

For the Soft-SPIBB and SPIBB algorithms, I add for each kind an additional abstract class between the base class and the actual algorithms, as there are a lot of computations the Soft-SPIBB algorithms and the SPIBB algorithms share. While this structure is completely new, the actual implementation of the algorithms is very similar to the one from Nadjahi et al. [11] and I validate that their code and my code produce the same results.
The abstract base class for the SPIBB algorithms contains the hyper-parameter $N_\wedge$ and implements a method to compute a mask marking all state-action pairs which have been visited more than $N_\wedge$ times, similar to the one for R-MIN. Thus, from an implementation point of view it would be possible to let R-MIN inherit as well from *SPIBB_abstract*, but this is not recommendable considering the difference in the nature of the algorithms. Also, the additional code this causes is negligible. The SPIBB algorithms, Π_b-SPIBB and $\Pi_{\leq b}$-SPIBB, renamed to SPIBB and Lower-SPIBB due to technical reasons in Python, then only need to implement their specific PI steps.
Similarly, the Soft-SPIBB and Adv-Soft-SPIBB algorithms all make use of the same hyper-parameters δ and ϵ and of the calculation of an error function. The user can choose between the error function relying on Hoeffding's inequality [15] or the bound derived by Maurer and Pontil [27] even though the latter should only be used with great caution due to the problems discussed in Section 3.2.4. Furthermore, I implement a method to compute $\hat{q}^{\mathcal{D}}_{\pi_b}$, as it is needed for the advantageous algorithms and for the 1-step algorithms, if they are supposed to fulfill the properties of Theorem 3.2. As there is usually no point in using the 1-step algorithms unless they hold an additional property, I add the method *_one_step_algorithms()*, which checks if one of the original Soft-SPIBB algorithms is used with only 1 iteration and, if it is, changes them so that they solve the optimization problem described in Equation 120. Again, all the explicit algorithms only need their specific implementation of the PI step.
Looking at the general picture of this framework shows that the big difference between the SPIBB algorithms including all their successors and all the other algorithms is that the SPIBB family concentrates on the PI step to incorporate some kind of safety mechanism, while all the others concentrate on the PE step by penalizing state-action pairs with a high uncertainty and then assume that the usual Dynamic Programming PI step finds a good trade-off between safety and performance on its own. DUIPI seems to contradict this classification, however, the deviation in its PI step is mostly technically and the idea of DUIPI is rooted in the PE step by taking the variance of the action-value function into account.

3.4 Benchmarks

The previous sections introduced a new class of algorithms which are variants of the Soft-SPIBB algorithms from Nadjahi et al. [11] and proved that they are supported by theoretical safety guarantees. In this section I test how well these algorithms perform empirically compared to their predecessor algorithms, classic Dynamic Programming and other algorithms which try to incorporate some safety mechanism, introduced in Section 3.3.
For this I conduct 3 experiments on MDPs with a finite state space. The first one on the Random MDPs benchmark from Nadjahi et al. [11] to see how these new algorithms

compare on this benchmark with their predecessors. As this is a very artificial benchmark, I use later on the discrete version of the Wet Chicken benchmark [7] which tries to depict a more realistic problem. Similar to Nadjahi et al. [11], during these two benchmarks I use the hyper-parameters just as such, i.e., I perform a hyper-parameter search for all of these algorithms and choose the "optimal" one without considering that some of them have an explicit meaning as stated in Theorem 3.2, for example. There are several reasons for this approach. Firstly, this makes the experiments more comparable to the ones done in Nadjahi et al. [11] which I am trying to extend. Secondly, for many of the algorithms there are no theoretical bounds, so, it would be difficult to choose the hyper-parameters for them to make a fair comparison. Thirdly, I think that the optimal hyper-parameter is a fair choice as it depends the least on my subjective view, e.g., in comparison to "guessing" hyper-parameters a priori. How the optimal hyper-parameter will be chosen exactly is described in Subsection 3.4.1. Additionally, I include all the hyper-parameter searches in Appendix B to make it possible to examine the influence of these parameters. Contrary, in the third experiment I reuse the Wet Chicken benchmark, but this time concentrate on the algorithms which are supported by some theoretical safety bounds and choose the parameters such that these become meaningful. The goal of this experiment is to see how well the algorithms perform under these circumstances to find out if the theoretical bounds can be used in practice.

In these three experiments I test all the algorithms from Sections 2.3.2, 3.2.6 and 3.3 except the Exact-Soft-SPIBB and Adv-Exact-Soft-SPIBB, as, on the one hand, they show in most cases numerical problems when applying the solver of the linear program due to an ill-conditioned matrix in the procedure. This happens independent of which solver was used. On the other hand they take on average ten times the amount of time from other algorithms, while performing very similar to their approximate versions—if they do not run in numerical issues—, which is also stated in Nadjahi et al. [11].

In Section 3.4.4 I conduct an exploratory experiment on the continuous version of the Wet Chicken benchmark to assess the possibility to use the function approximation versions of the Soft-SPIBB and SPIBB algorithms.

3.4.1 Random MDPs

This benchmark has already been described in Section 2.3.3. The only change in the MDPs I use is that this time, after the behavior policy is computed, one of the non-final and non-starting states is chosen as an easter egg. This easter egg is an additional final state and also yields a reward of 1. This makes the state special, as the behavior policy has only been optimized for the MDP without this state and, therefore, it is totally unaware of its existence. This is beneficial as the training of the behavior policy incorporates the optimal policy on the original MDP, so, if the MDP remains unchanged after the training, the behavior policy depends on knowledge about the true MDP. This can have strange effects, as in this scenario any algorithm, which simply generates the policy that greedily chooses in each state the most often observed action, might compute the optimal policy with a high probability. This easter egg should serve to counter effects like this. This method has already been used by Nadjahi et al. [11], too.

I also tried to substitute this positive easter egg with a negative one, i.e., the exact same procedure but instead of a reward of 1, this state should yield a reward of -1. The idea

was that the presence of a danger should make a more interesting scenario for safe RL. However, the results from these experiments did not yield any new findings as they were extremely similar to the ones using a good easter egg. Thus, I only present the results of the Random MDPs benchmark with a good easter egg.
I start the analysis with the hyper-parameter search. Figure 8 depicts the results for Adv-Approx-Soft-SPIBB for different values of ϵ. δ was set to 1, which does not make much sense in regards to Theorem 3.2, but remember that I am not interested in this benchmark in any of the theoretical bounds, but just in the general performance of these algorithms for optimized hyper-parameters. Also, the effect of δ is very small, which can be seen in Section 3.4.3, and very similar to the one by ϵ, so, fixing δ only reduces the number of hyper-parameters one has to optimize for. Here, I follow again Nadjahi et al. [11].

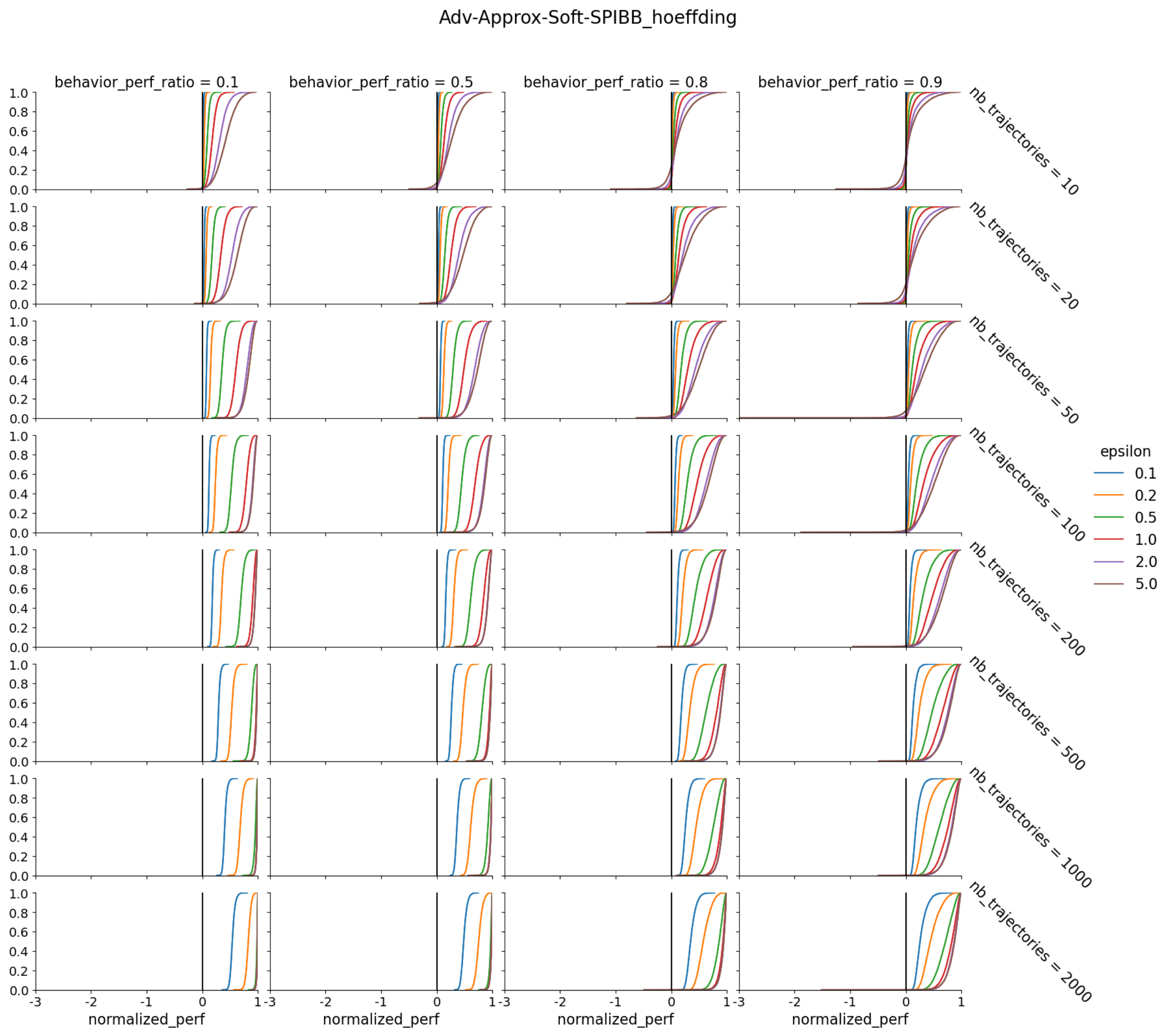


Fig. 8. *ECDF for the different hyper-parameter values for Adv-Approx-Soft-SPIBB on the Random MDPs benchmark.*

As shown in Algorithm 2, I consider different target performances for the behavior policy and different numbers of trajectories the offline reinforcement learning algorithms can

learn from. Figure 8 describes the hyper-parameter search exemplary for Adv-Approx-Soft-SPIBB using the error function relying on the Hoeffding's inequality. In each column the target performance of the behavior policy is fixed and in each row the number of trajectories is fixed, according to the value above and right to the plots, respectively. Each subplot shows the empirical cumulative distribution function (ECDF) of the normalized performance from Equation 42 for the six different values for ϵ. The black line shows the normalized performance of each behavior policy which is 0 by definition.
I use a ECDF plot since every constellation of number of trajectories, target performance of the behavior policy and ϵ is computed on more than 10,000 MDPs and the ECDF plot displays information of the whole distribution of all these runs. This is especially interesting as I am interested in safe policy improvement and not only a single value like the mean performance.
This multitude of information about the performance of each algorithm for different hyper-parameter values makes it harder to choose the "optimal" one. Consequently, I focus on three things. Firstly, I generally assume that the underlying behavior policy already performs quite well, which is hopefully a reasonable assumption when trying to improve a policy which is currently used. This means I can focus on the experiments using a target performance ratio of 0.9, i.e., I only have to consider the 4^{th} column. Secondly, the focus of this thesis lies on safe policy improvement, so, I also concentrate on this here and choose the hyper-parameters optimizing for this. Thus, I pay special attention to the amount of probability left to the black line for each hyper-parameter, which corresponds to the probability that the algorithm with this parameter performs worse than the behavior policy. For this reason, I do not want to choose an ϵ of 2 or 5, considering their performances for 10 and 20 trajectories. Thirdly, even though I am interested in a safe improvement, I am also interested in a strong improvement at the same time, if the safety is given. Therefore, I choose a parameter which does not contradict the safety like 2 or 5, but also performs overall very well. This seems to be achieved very well by $\epsilon = 1$, which I choose for a comparison against the other algorithms. Furthermore, one can see the effect of the hyper-parameter ϵ as already described in Section 3.2.4 as an trade-off between a high uncertainty for a big ϵ and a low mean performance for a small ϵ.
The same is done for all the other algorithm. Their ECDF plots can be found in Appendix B where I reduced the plots such that it only shows the right column. A table with my choices is shown in Appendix A.
To compare all the algorithms with optimized hyper-parameters I consider, similarly to Nadjahi et al. [11], two statistics:

- the mean performance and
- the 1%-CVaR performance.

As explained before, the focus lies on the behavior policy with the highest target performance and, so, the mean and the 1%-CVaR performance can be depicted as a line plot with the number of trajectories as the x-axis and the normalized performance as the y-axis. This is done in Figure 9 and 10 for the mean performance and the 1%-CVaR performance, respectively.
In the mean performance, RaMDP and DUIPI outperform every other algorithm as soon as at least 50 trajectories are observed. They are followed by Lower-Approx-Soft-SPIBB

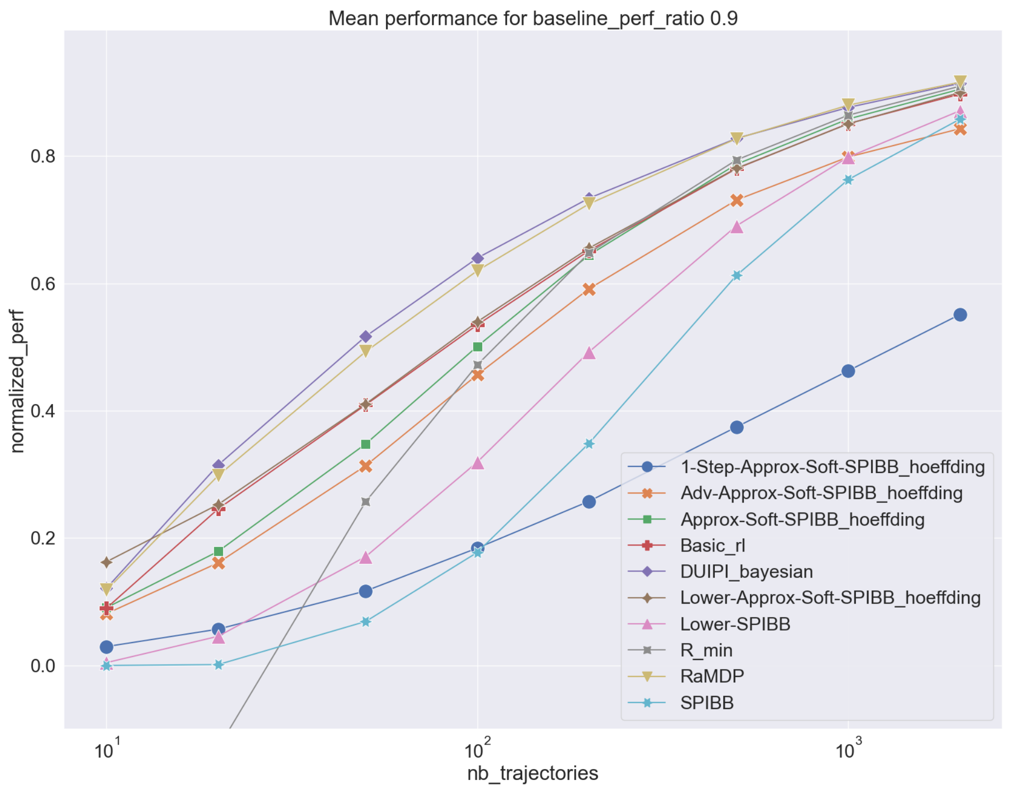


Fig. 9. *Mean normalized performance over 10,000 trials on the Random MDPs benchmark for the target performance ratio of 0.9 of the behavior policy.*

and Basic RL, which come just before the other Soft-SPIBB algorithms. Approx-Soft-SPIBB shows a slightly better performance than its successor Adv-Approx-Soft-SPIBB. 1-Step-Approx-Soft-SPIBB, Π_b-SPIBB (SPIBB) and $\Pi_{\leq b}$-SPIBB (Lower-SPIBB) seem to perform the worst. While R-MIN exhibits issues when the data set is very small, it catches up with the others for bigger data sets. Interestingly, Basic RL performs generally quite well, but is still outperformed by some algorithms, which might be surprising as the others are intended for safe RL instead of an optimization of their mean performance. The reason for this might be that considering the uncertainty of the action-value function is even for the mean performance beneficial.

The performance in the worst percentile looks very different. Here, it can be seen how well the safety mechanisms of some of the algorithms work, when they are compared to Basic RL, which shows the worst overall 1%-CVaR performance. Also, R-MIN and RaMDP perform very poorly especially for a low number of trajectories. For less than 100 trajectories, DUIPI performs also very poorly but outperforms every other algorithm for bigger data sets. An interesting observation is that all the SPIBB and Soft-SPIBB algorithms, except 1-step Approx-Soft-SPIBB, perform in the beginning extremely well, which is explainable by the fact that they fall back to the behavior policy if not much data is available. The ranking in the SPIBB family stays the same for the 1%-CVaR

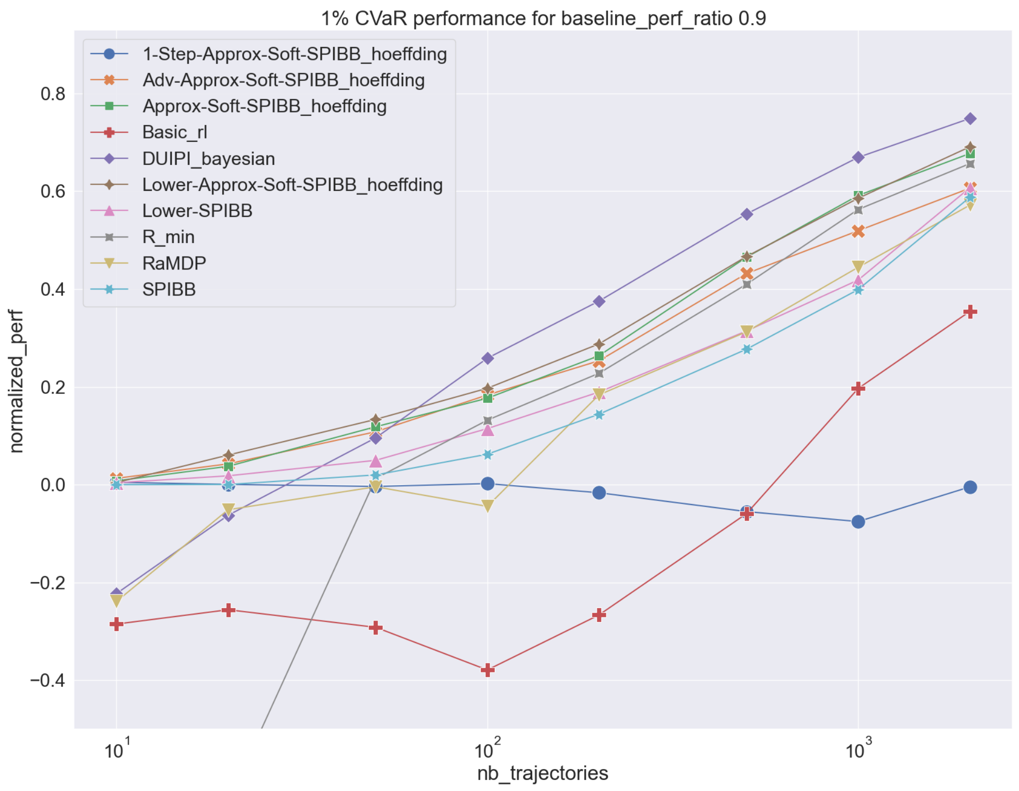


Fig. 10. *1%-CVaR normalized performance over 10,000 trials on the Random MDPs benchmark for the target performance ratio of 0.9 of the behavior policy.*

as it has been for the mean performance: Lower-Approx-Soft-SPIBB performs the best, closely followed first by Approx-Soft-SPIBB and Adv-Approx-Soft-SPIBB. $\Pi_{\leq b}$-SPIBB falls a bit behind but still manages to perform better than the original Π_b-SPIBB. The worst performance is achieved by 1-step Approx-Soft-SPIBB, which is the only algorithm to never exceed the behavior policy.

Figure 9 and Figure 10 show that the best overall performance of any of the algorithms from the SPIBB family is achieved by Lower-Approx-Soft-SPIBB, independent of which statistic is applied to compare the algorithms, the mean or the 1%-CVaR.

As indicated in the legend of the plots the error function relying on the Hoeffding's bound is used for the Soft-SPIBB algorithms. In the following, I want to check if it would be advantageous to use the error function relying on the Maurer and Pontil bound. As shown before in Section 3.2.4 it is crucial for this error function to know or to estimate an upper bound on the absolute return, assuming the return is centered. So, it is necessary to know the lowest and the highest possible returns. The lowest possible return is 0, as this happens if no terminal is ever reached, which is highly unlikely but always possible due to the high stochasticity of the MDP. The highest possible return is harder to estimate and needs specific knowledge about the MDP. If there is a possibility to get to a terminal state in only 1 step, it would be 1, otherwise it is reduced by the discount factor in each step. For the sake of simplicity, I estimate the upper bound with 1, so, G_{max} could be chosen as $\frac{1}{2}$. Fortunately, if one optimizes the hyper-parameter ϵ, this approximation has no influence on the algorithm itself, as this simply changes which ϵ delivers the best performance.

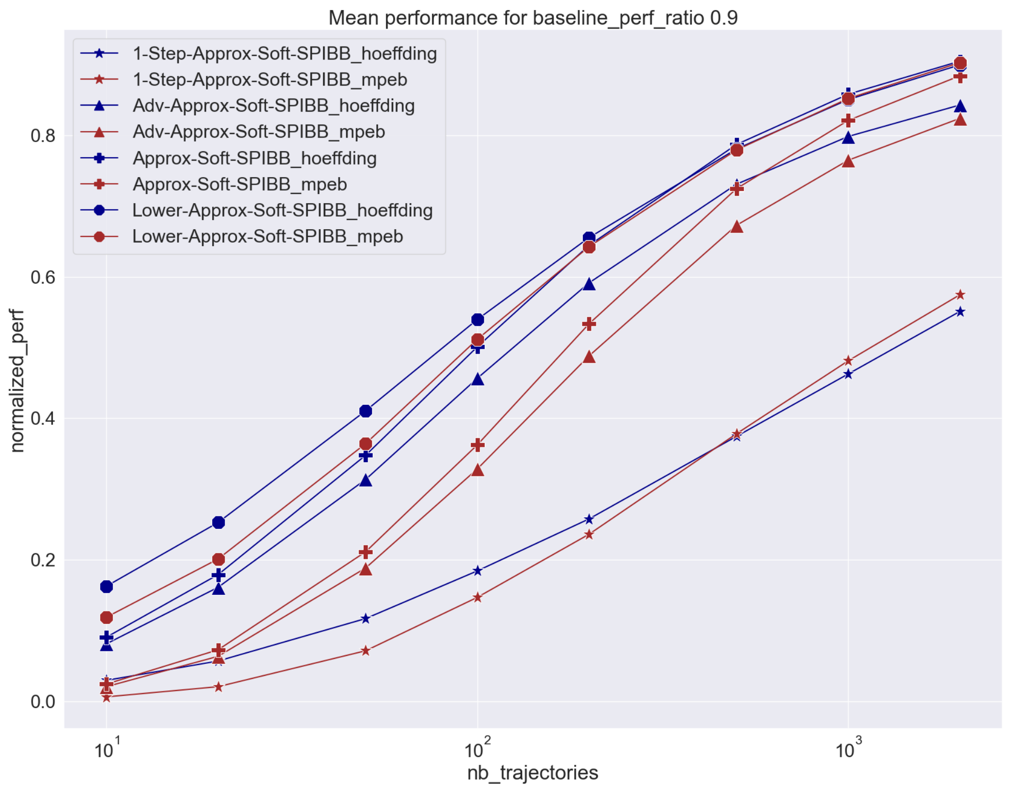


Fig. 11. *Mean normalized performance over 10,000 trials on the Random MDPs benchmark for the target performance ratio of 0.9 of the behavior policy. Comparison between the Soft-SPIBB algorithms using an error function relying on Hoeffding's bound (blue) vs. Soft-SPIBB algorithms using an error function relying on the Maurer and Pontil bound (brown).*

Figure 11 shows the comparison between the Soft-SPIBB algorithms using these two different error functions for the mean normalized performance in the Random MDPs benchmark. One clearly sees that the new error function relying on the Maurer and Pontil bound yields no improvement in this scenario.
Figure 12 depicts this comparison for the 1%-CVaR normalized performance. Here, the superiority of the error function relying on the Hoeffding's bound is not as clear as in the mean normalized performance plot before and is even reversed for the 1-step-Approx-Soft-SPIBB. However, considering especially Approx-Soft-SPIBB and Adv-Approx-Soft-SPIBB yields no substantial reason to switch to the more complicated error function relying on the Maurer and Pontil bound.
To investigate whether the reported differences in the performance between different al-

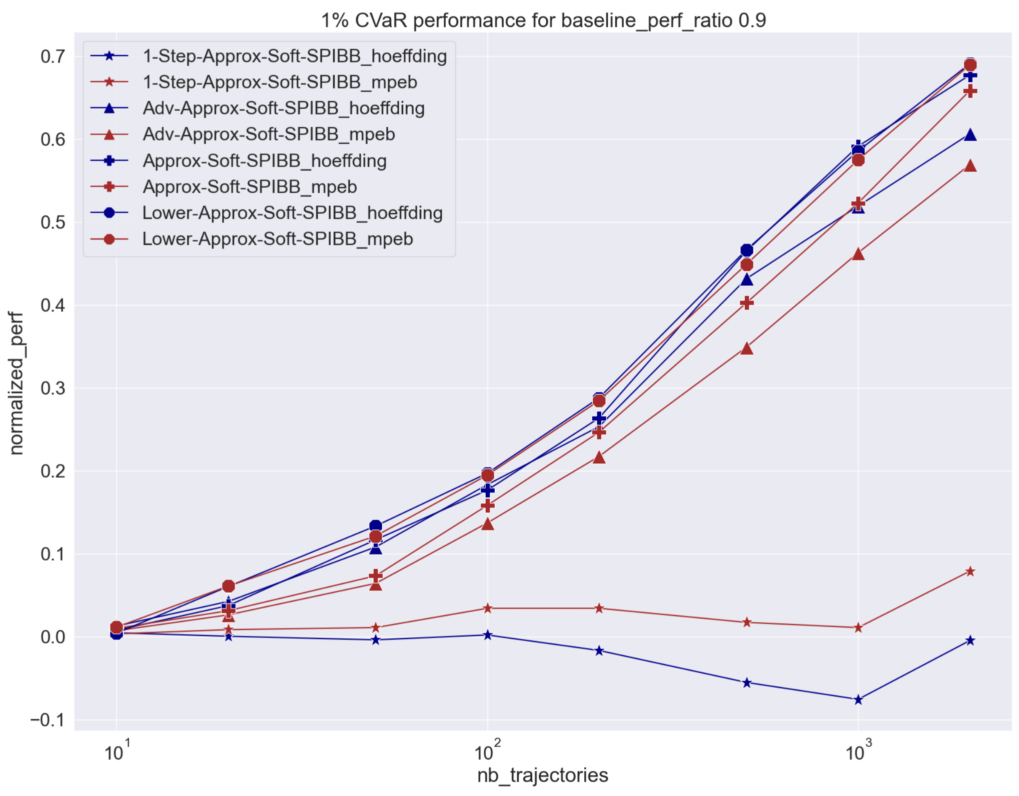


Fig. 12. *1%-CVaR normalized performance over 10,000 trials on the Random MDPs benchmark for the target performance ratio of 0.9 of the behavior policy. Comparison between the Soft-SPIBB algorithms using an error function relying on Hoeffding's bound (blue) vs. Soft-SPIBB algorithms using an error function relying on the Maurer and Pontil bound (brown).*

gorithms considered in this section are significant, I also compute the standard deviation. The standard deviation of the mean $\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$, with i.i.d. samples x_1,...,x_N, is calculated [30] as

$$\sigma_{\bar{x}} = \frac{1}{\sqrt{N}}\sigma_x, \tag{150}$$

where σ_x is the standard deviation of the samples, i.e.,

$$\sigma_x^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2. \tag{151}$$

This formula yields that the standard deviation of the mean performance is less than 0.001 for every algorithm and, so, every visible difference in the figures in this subsection is highly significant. Calculating the standard deviation of the 1%-CVaR performance is more complicated and, thus, I make use of the Bootstrap method introduced by Efron [31]. This method reveals that the standard deviation of the 1%-CVaR performance is always between 0.003 and 0.006. This shows that one has to be cautious with over interpreting every difference between algorithms. However, every trend I have mentioned in this section remains significant.

3.4.2 Discrete Wet Chicken benchmark

Besides reproducing the results of Nadjahi et al. [11] for additional algorithms, I want to extend their experiments to a more realistic scenario for which I have chosen the discrete version of the Wet Chicken benchmark [7] because of its heterogeneous stochasticity. Figure 13, already displayed in the introduction, visualizes the setting of the Wet Chicken benchmark. The basic idea behind it is that a person floats in a small boat on a river. The river has a waterfall at one end and the goal of the person is to stay as close to the waterfall as possible without falling down. Thus, the closer this person is to the waterfall the higher the reward gets, but upon falling down they start again at the starting place, which is as far away from the waterfall as possible. Therefore, this is modeled as a non-episodic MDP.

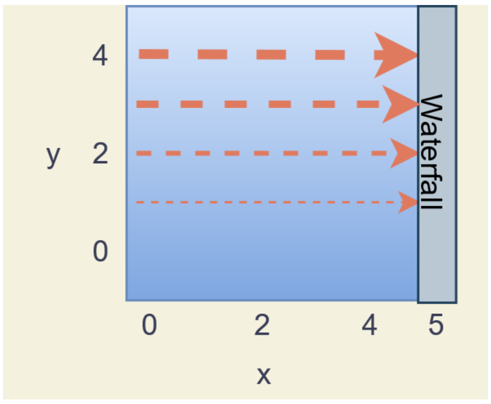


Fig. 13. *The setting of the Wet Chicken benchmark used for reinforcement learning. The boat starts at $(x, y) = (0, 0)$ and starts there again upon falling down the waterfall at $x = 5$. The arrows show the direction and strength of the stream towards the waterfall. Additionally, there are turbulences which are stronger for small y. The goal for the boat is to stay as close as possible to the waterfall without falling down.*

The whole river has a length and width of 5, so, there are 25 states. The starting point is $(x, y) = (0, 0)$ and the waterfall is at $x = 5$. The position of the person at time t is denoted by the pair (x_t, y_t). The river itself has a turbulence which is stronger near the shore the person starts close to $(y = 0)$ and a stream towards the waterfall which is stronger near the other shore $(y = 4)$. The velocity of the stream is defined as $v_t = y_t \frac{3}{5}$ and the turbulence as $b_t = 3.5 - v_t$. The effect of the turbulence is stochastic; so, let $\tau_t \sim U(-1, 1)$ be the parameter describing this stochasticity at time t.
The person itself has 5 actions, which are (a_x and a_y describe the influence of an action on x_t and y_t, respectively):

- Drift: The person does nothing, in formula $(a_x, a_y) = (0, 0)$.
- Hold: The person paddles back with half its power, in formula $(a_x, a_y) = (-1, 0)$.

- Paddle back: The person wholeheartedly paddles back, in formula $(a_x, a_y) = (-2, 0)$.
- Right: The person tries to go to the right parallel to the waterfall, in formula $(a_x, a_y) = (0, 1)$.
- Left: The person tries to go to the left parallel to the waterfall, in formula $(a_x, a_y) = (0, -1)$.

The new position of the person assuming no river constraints is then calculated by

$$\hat{x} = round(x_t + a_x + v_t + \tau_t s_t) \quad \text{and} \quad \hat{y} = round(x_t + a_y), \tag{152}$$

where the *round* function is the usual one, i.e., a number is getting rounded down if the first decimal is 4 or less and rounded up otherwise. Incorporating the boundaries of the river yields the new position as

$$x_{t+1} = \begin{cases} \hat{x}, & \text{if } 0 \leq \hat{x} \leq 4 \\ 0, & \text{otherwise} \end{cases} \quad \text{and} \quad y_{t+1} = \begin{cases} 0, & \text{if } \hat{x} > 4 \\ 4, & \text{if } \hat{y} > 4 \\ 0, & \text{if } \hat{y} > 0 \\ \hat{y}, & \text{otherwise} \end{cases}. \tag{153}$$

To evaluate the performance of a policy on an MDP exactly, i.e., without approximating it by simply testing it out several times on it, one has to compute the transition probabilities. This is done in the function *get_transition_function()*, which is also used to debug the implementation of the MDP.[7]
As the aim of this experiment is to have a realistic setting for Batch RL, I want to use a realistic behavior policy. Thus, I do not make use of the optimal policy or any knowledge about the transition probabilities as it has been done for the Random MDPs benchmark. Instead I devise heuristically a policy, considering the overall structure of the MDP.
Therefore, the behavior policy I apply follows the idea that the most beneficial state might lie in the middle of the river at $(x, y) = (2, 2)$. This idea stems from two trade-offs. The first trade-off is between low rewards for a small x and a high risk to fall down for a big x and the second trade-off is between a high turbulence and low velocity for a low y and the opposite for big y. To be able to ensure the boat stays at the same place turbulence and velocity should both be limited.
This idea is enforced through the following procedure. If the boat is not in the state $(2, 2)$, the person tries to get there and if they are already there the person uses the action *paddle back*.[8] I denote this policy with π_b'. The problem with this policy is that it is deterministic, i.e., in every state there is only one action which is chosen with probability 1. This means that for each state there is at most 1 action for which data is available when observing this policy. This is countered by making π_b' ϵ-greedy, i.e., define the behavior policy π_b as the mixture

$$\pi_b = (1 - \epsilon)\pi_b' + \epsilon\pi_u \tag{154}$$

[7]https://github.com/Philipp238/Evaluation-of-Safe-Policy-Improvement-with-Baseline-Bootstrapping/blob/master/wet_chicken_discrete/dynamics.py, assessed on 20.03.2021

[8]For details: https://github.com/Philipp238/Evaluation-of-Safe-Policy-Improvement-with-Baseline-Bootstrapping/blob/master/wet_chicken_discrete/baseline_policy.py, assessed on 20.03.2021

where π_u is the uniform policy which chooses every action in every state with the same probability.
Figure 14 gives a feeling about the distribution of performances of deterministic policies. As the number of possible deterministic policies is too high to compute all of them ($5^{25} \approx 2.98 \cdot 10^{17}$), 1,000,000 deterministic policies were randomly sampled and used to estimate this plot. The performance of π_b' is about 30 and, thus, π_b' is already better than a great percentage of the deterministic policies. The stochastic behavior policy π_b has a similar performance between 29 and 30 for an $0 < \epsilon \leq 0.2$. For $\epsilon = 0.5$, which is the least value I use, its performance is about 27.

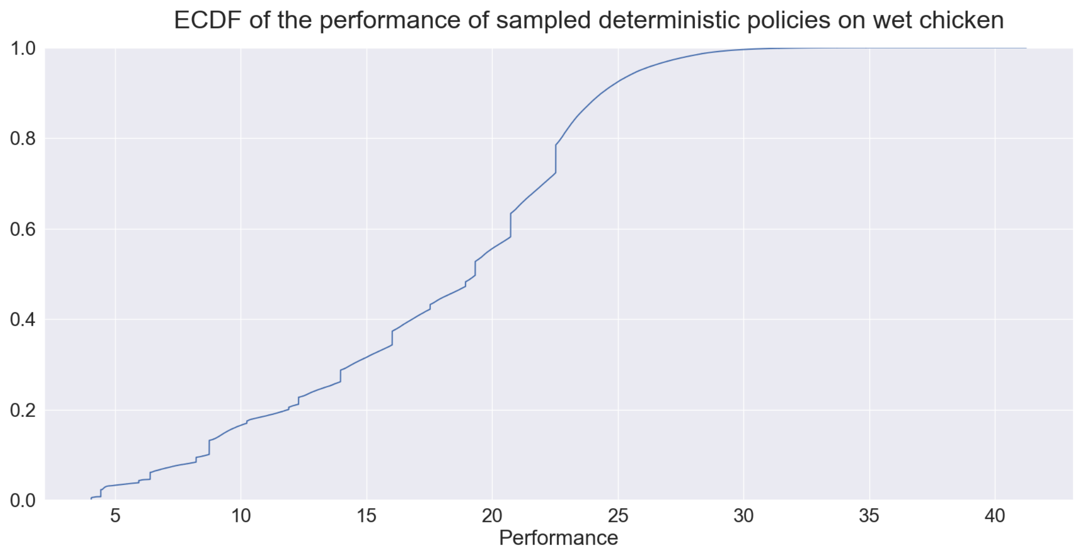


Fig. 14. *ECDF of the performance of randomly sampled deterministic policies on the Wet Chicken benchmark (sample size = 1,000,000).*

Figure 14 shows some additional very interesting properties of the Wet Chicken benchmark. First of all, the range of the performance of all deterministic policies can be seen, which is equivalent to the range of all policies, since there is always an optimal deterministic policy, as mentioned in Section 2.1. This range goes from about 3 up to almost 42. This is useful to interpret the results of the algorithms. The optimal policy is similar to π_b' in that it tries to reach an optimal position and then uses the *paddle back* action all the time, however, the optimal position turns out to be $(x, y) = (3, 3)$.
Furthermore, the distribution of the performances of the deterministic policies is heavily concentrated on lower values, which makes this benchmark quite complicated and different to the Random MDPs benchmark, as can be seen in Figure 15, where the same is done for the Random MDPs benchmark with a good easter egg. I only display the results for one MDP here, but I tried different ones and got always very similar results. Here, one can see an ECDF resembling the CDF of a normal distribution on a bounded interval and especially see a strong symmetry, i.e., high performances are exactly as likely as low performances. Another observation is that there are jumps in the ECDF for the Wet Chicken benchmark, which shows that there are many policies with the same performance which is at least partly due to the many possible actions which have the same effect, since,

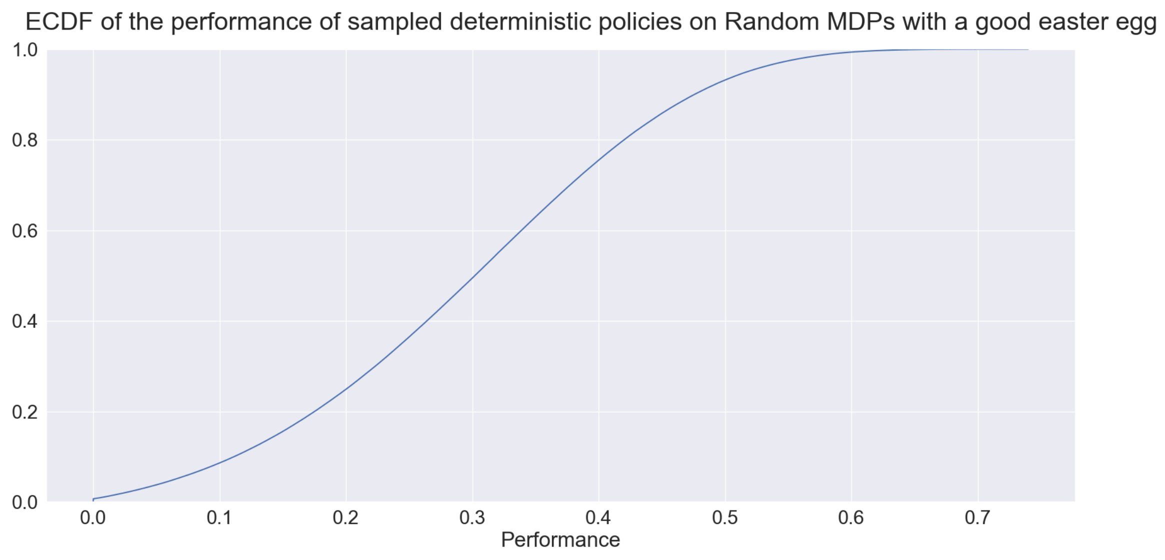


Fig. 15. *ECDF of the performance of randomly sampled deterministic policies on the Random MDPs with a good easter egg benchmark (sample size = 1,000,000).*

for example, a policy which chooses the action *left* when $y = 0$ gets the exact same result as one which chooses *drift* there. This should be kept in mind when analyzing the ECDFs of the performance of algorithms which produce deterministic policies, as they should be expected to have the same jumps.

The overall concept of the experiments is similar to the one described in Algorithm 2 which was used for the Random MDPs benchmark before. However, the MDP itself is fixed now and the behavior policy is chosen as described above. As it is a non-episodic task, only a single trajectory is sampled in each iteration but the length of this trajectory varies. The hyper-parameter selection is following the same ideas and using the same plots as for the Random MDPs benchmark, so, I refer to the Appendix B for the hyper-parameter search and to Appendix A for the hyper-parameter choices.

Again, I focus on the error function relying on the Hoeffding's inequality for the Soft-SPIBB algorithms, since they show the better overall performances. Figure 25 in Appendix B compares the Soft-SPIBB algorithms using the two error functions on the Wet Chicken benchmark.

In Figure 16 the mean performance over all 10,000 runs of the different algorithms can be seen. Apart from DUIPI, the results are similar to those on the Random MDPs benchmark. Again R-MIN performs extremely bad for few data but then improves strongly. Basic RL, 1-Step-Approx-Soft-SPIBB and DUIPI exhibit the worst performance. All the other algorithms from the SPIBB family perform very well, especially Lower-Approx-Soft-SPIBB. The best overall mean performance seems to be achieved by RaMDP again.

Again the 1%-CVaR performance is of high interest and can be seen in Figure 17. This confirms many of the observations from the Random MDPs benchmark as well. One can see again that especially Basic RL, but also RaMDP and R-MIN have problems competing with the SPIBB and Soft-SPIBB algorithms and DUIPI. Among the SPIBB and Soft-SPIBB algorithms Lower-Approx-Soft-SPIBB seems to perform again the best,

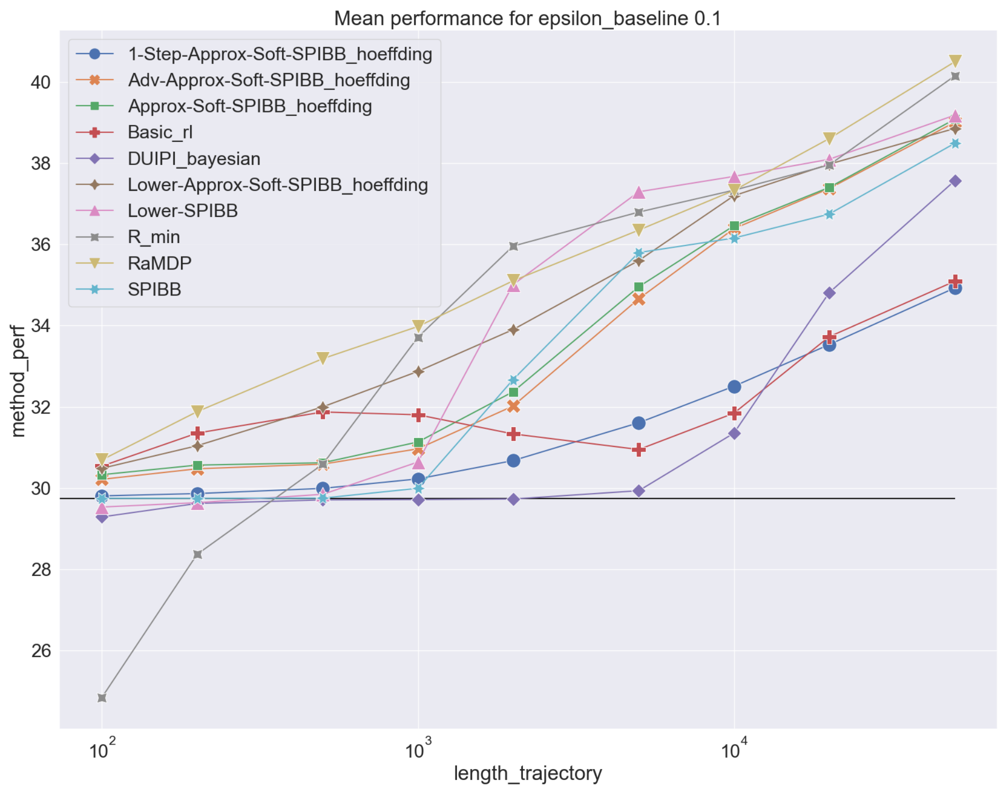


Fig. 16. *Mean performance over 10,000 trials on the Wet Chicken benchmark with a heuristic epsilon-greedy baseline with* $\epsilon = 0.1$*.*

followed by Adv-Approx-Soft-SPIBB and Approx-Soft-SPIBB.

Similarly to the experiments on the Random MDPs benchmark, I compute the standard deviation for the reported measures on the Wet Chicken benchmark. The computation is done exactly as in Section 3.4.1. The standard deviation of the mean performance is always below 0.1 and, thus, every clearly visible difference in Figure 16 is a significant one. The standard deviation of the 1%-CVaR performance amounts up to 0.7 and, so, Figure 17 has to be interpreted cautiously. However, firstly it is important to note the increased y-scale in Figure 17 in comparison to Figure 16. Secondly, the standard deviation of the 1%-CVaR performance varies strongly among the different algorithm classes. While it is between 0.2 and 0.7 for Basic RL, RaMDP, R-MIN and sometimes DUIPI, it is almost exclusively below 0.2 for all algorithms of the SPIBB and Soft-SPIBB family. This is interesting as those are mostly the ones in close range in this performance measure and, additionally, it shows that the performance of these algorithms is more stable compared to the others.

In conclusion, these two benchmarks have shown that especially in the 1%-CVaR performance, the whole SPIBB family performs extremely well and among them Lower-Approx-Soft-SPIBB excels in almost every scenario, including the mean and 1%-CVaR performance. Furthermore, it has been shown that the theoretically supported Adv-

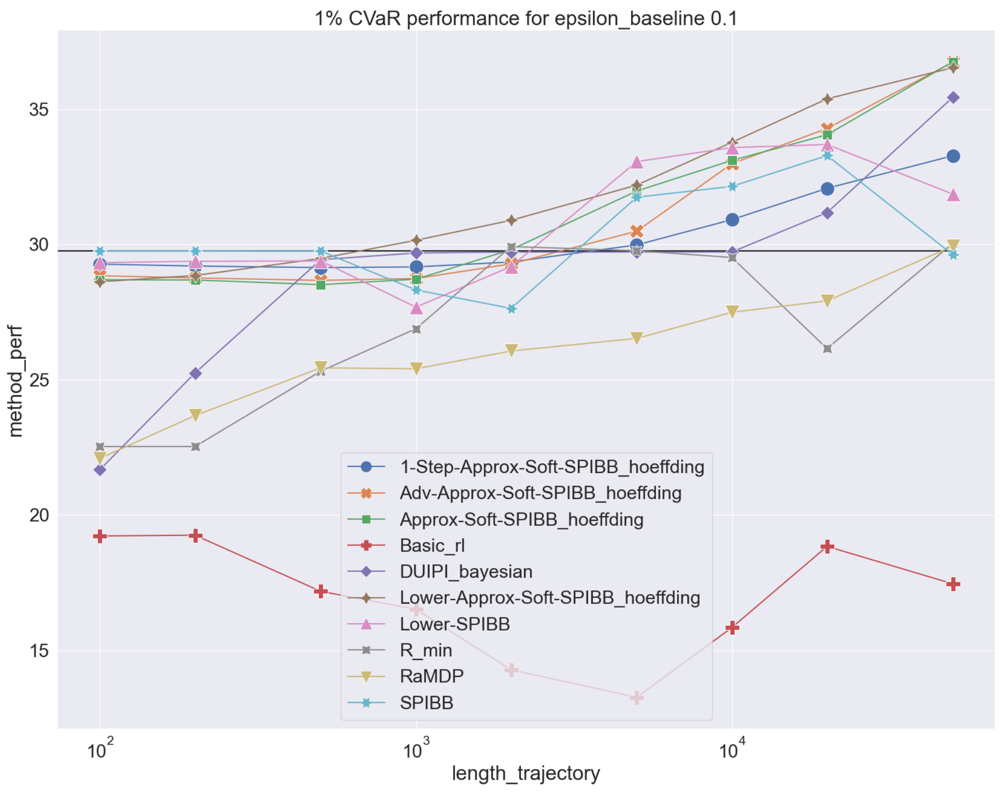


Fig. 17. *1%-CVaR performance over 10,000 trials on the Wet Chicken benchmark with a heuristic epsilon-greedy baseline with* $\epsilon = 0.1$.

Approx-Soft-SPIBB performs almost as well as its predecessor Approx-Soft-SPIBB, only falling slightly behind in the mean performance.

3.4.3 Theoretical safety benchmark

As already mentioned, I want to test the theoretical safety properties some of the algorithms exhibit. These algorithms are

- Adv-Approx-Soft-SPIBB,
- 1-Step-Approx-Soft-SPIBB,
- Π_b-SPIBB, and
- DUIPI.

The first three algorithms completely fulfill the properties needed for their safety bound, while DUIPI only uses an approximation of the variance and additionally comes with a strong assumption on the distribution of the action-value function, namely, that it is normal distributed. This has to be kept in mind when comparing the different bounds.

To test these bounds I repeat the experiments on the Wet Chicken benchmark and choose this time the hyper-parameters such that the bound is a useful one. The goal is not to directly compare the algorithms against each other. Instead I want to see if the algorithms can still compute good policies when they are secured by theoretical bounds. As the range of the returns or the value-functions is important for many of the theoretical safety bounds, I want to mention here that by adding or subtracting the same constant of the reward in every state, these two ranges can be arbitrarily translated in non-episodic settings. This can be seen directly by the definition of the returns as the series of discounted rewards and the linearity of the expected value implies the same for the value functions. Thus, it is always possible to find an MDP equivalent to the Wet Chicken MDP such that either the ranges of the returns or the value functions are centered.

For Adv-Approx-Soft-SPIBB and 1-Step-Approx-Soft-SPIBB the safety guarantee is Theorem 3.2, thus, I set $\delta > 0$ close to 0 and also $\epsilon > 0$ small enough to make Equation 34 meaningful. The good thing here is that all the parameters are known a priori so one can choose ϵ and directly know the value of the bound. As I use $\gamma = 0.95$, all the returns are between 0 and 80 ($80 = \frac{4}{1-\gamma}$) and, therefore, $G_{max} = 40$, as there is an MDP equivalent to the usual Wet Chicken benchmark but with centered returns and G_{max} only needs to be an upper bound on these centered ones, see Section 3.2.4. Remembering Figure 14, I choose ϵ to be at most 0.01, as this yields a lower bound of -8 and everything below that bound is not really useful. I also test smaller ϵ to see how the performance of the algorithm changes when I tighten the lower bound.

An ECDF plot similar to the one used for the hyper-parameter search in Section 3.4.1 is used again in Figure 18 to visualize the performance of Adv-Approx-Soft-SPIBB with $\epsilon = 0.01$ and $\delta = 0.01$ and its corresponding lower bound, which holds, thus, with a probability of at least 0.99. One can see here, that this lower bound actually holds all the time, which shows that Theorem 3.2 is not sharp. In general, the given lower bound is meaningful, however, it takes a lot of data to get even the slightest improvement over the behavior policy. If one decreases ϵ, the bound improves but the performance deteriorates further. The effect of δ is weaker, so, choosing a lower bound which holds with a higher probability does not influence the performance strongly. All these additional plots can be found in Appendix B. The plots for 1-step Approx-Soft-SPIBB are not shown here as they are essentially the same as for Adv-Approx-Soft-SPIBB. The reason for this is probably that such a small ϵ makes the algorithms inflexible and, therefore, one iteration is already enough to find the optimal policy which is π_b-advantageous w.r.t. $\hat{q}^{\mathcal{D}}_{\pi_b}$.

In Section 3.2.4 I proved that the bound of Theorem 3.2 can be improved to Equation 112. I hoped that this yields a substantial improvement of the bound and might make it possible to use a bigger ϵ and still have a meaningful lower bound. Unfortunately, this extra term in Equation 112 is not big enough for this. The biggest one in this experiment is about 0.27, which occurred for an ϵ of 0.01, so, it sharpened the bound from 8 below the performance of the behavior policy to 7.73 below the performance of the behavior policy. For smaller ϵ these improvements were even smaller, always shortening the distance to the performance of the behavior policy of the original bound by about 3.6%.

As mentioned before, G_{max} is known exactly in this setting, which enables the use of the error function relying on the Maurer and Pontil bound for Adv-Approx-Soft-SPIBB and 1-step Approx-Soft-SPIBB. In Section 3.2.4 I propose this new error function especially because of its improved asymptotic tightness, which turns out to be of great advantage

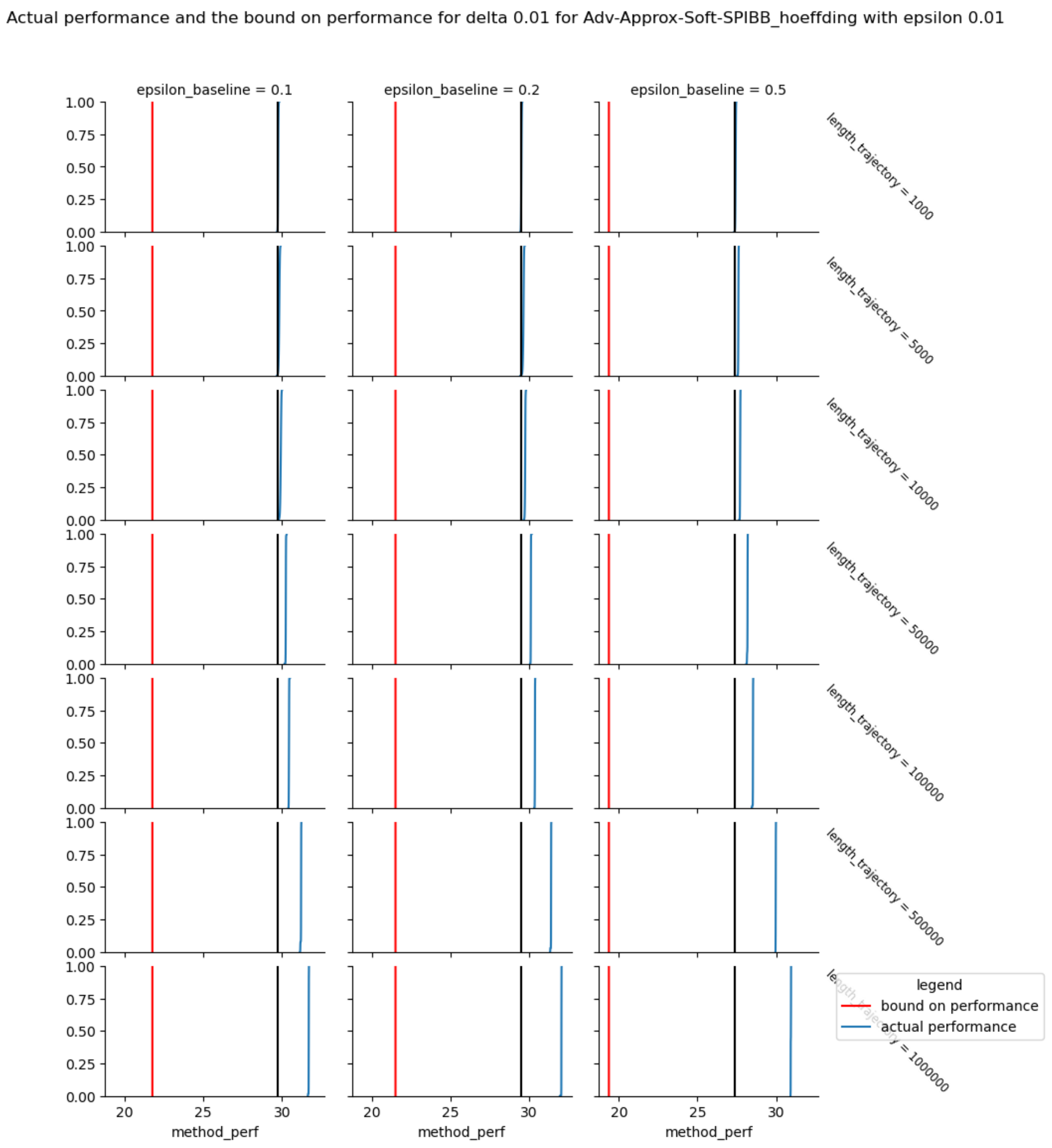


Fig. 18. *ECDF plot of the performance of Adv-Approx-Soft-SPIBB using the error function relying on Hoeffding's inequality and the lower bound on its performance on the Wet Chicken benchmark with $\epsilon = 0.01$ and $\delta = 0.01$ such that the lower bound holds with probability 0.99.*

in this setting as can be seen in Figure 19. While still achieving the same security for the same parameters as before, the performance of the computed policies is significantly higher by using the error function relying on the Maurer and Pontil bound.
For Theorem 3.2 to hold, it is necessary that the returns used to estimate the value function of the behavior policy are independent. In Section 3.2.4 it is discussed how this can be achieved. Therefore, I apply Algorithm 3 with $\kappa = 0.01$ to Adv-Approx-Soft-

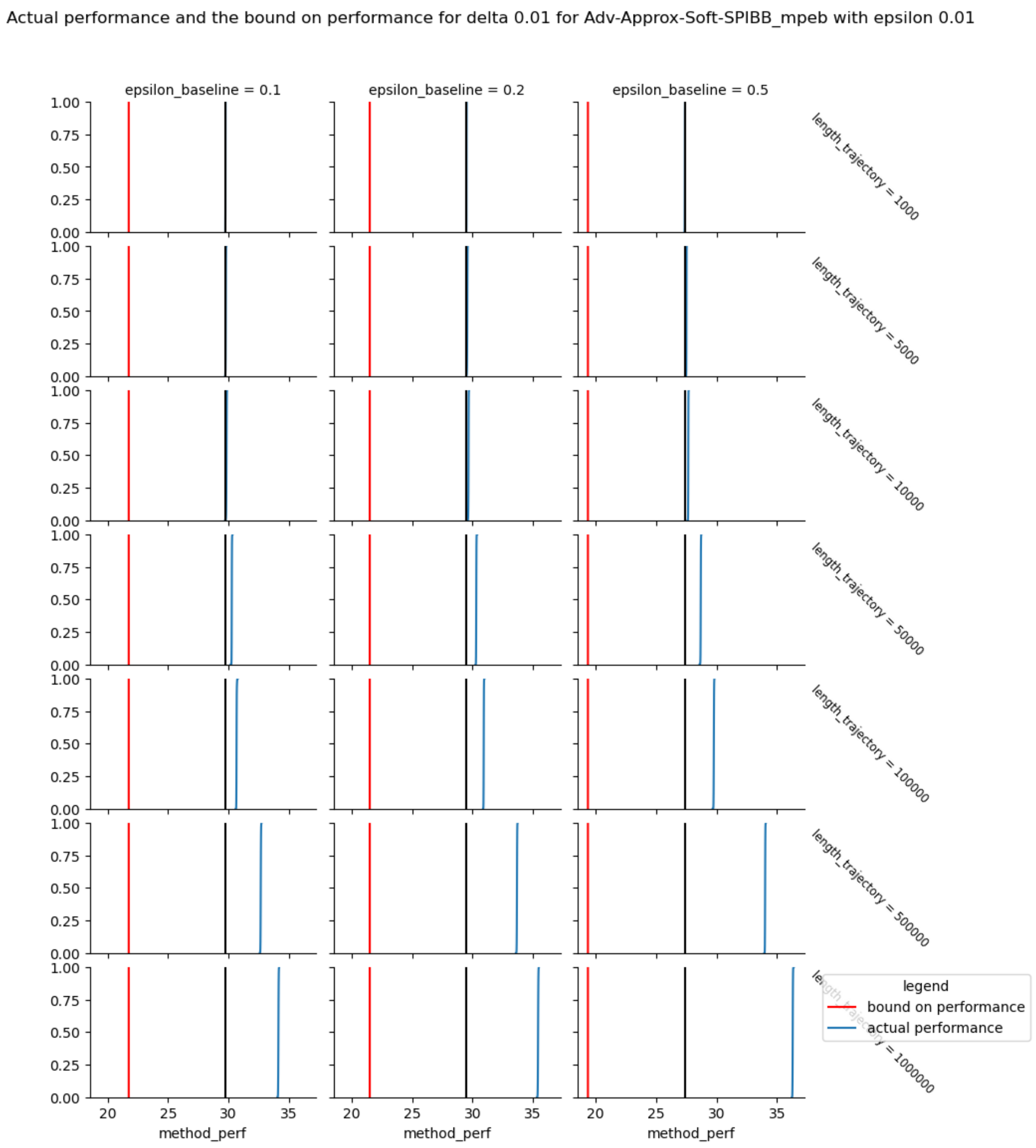


Fig. 19. *ECDF plot of the performance of Adv-Approx-Soft-SPIBB using the error function relying on the Maurer and Pontil bound and the lower bound on its performance on the Wet Chicken benchmark with* $\epsilon = 0.01$ *and* $\delta = 0.01$ *such that the lower bound holds with probability 0.99.*

SPIBB using the error function relying on the Maurer and Pontil bound. The results for this are shown in Figure 20. Omitting the returns yields more conservative algorithms and, thus, policies closer to the behavior policy are produced this time. However, even though I applied a very strict bound on the correlation of 0.01, the performance has not diminished substantially and, especially, the algorithms still performed better here, than when using the error function relying on the Hoeffding's bound, but not ensuring approximate independence.

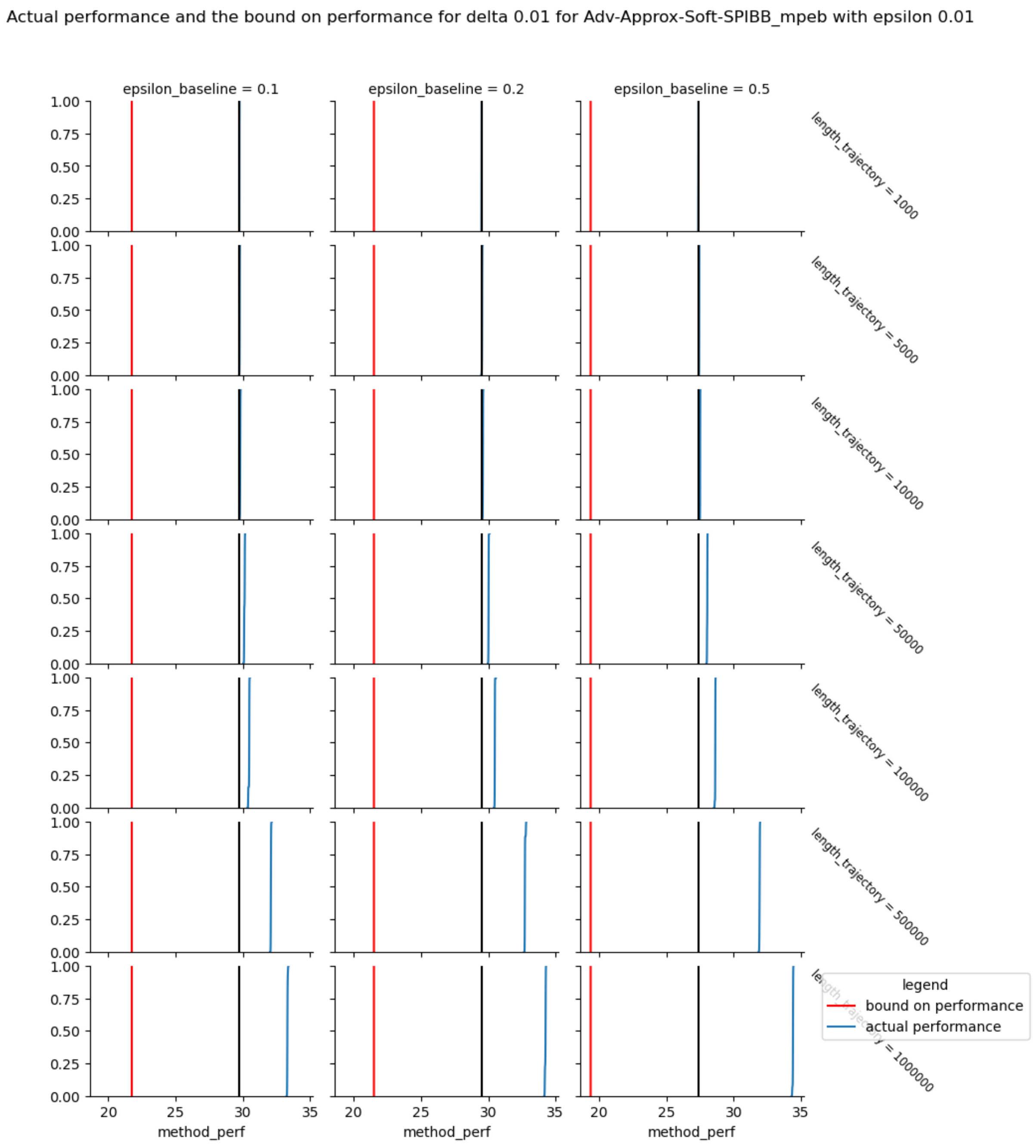


Fig. 20. *ECDF plot of the performance of Adv-Approx-Soft-SPIBB and the lower bound on its performance on the Wet Chicken benchmark with $\epsilon = 0.01$ and $\delta = 0.01$ such that the lower bound holds with probability 0.99, this time the approximate independence of the returns was ensured by applying Algorithm 3 with $\kappa = 0.01$.*

In Section 3.3.3 I mentioned that Π_b-SPIBB produces policies which are ξ-approximate

$1-\delta$-safe policy improvements w.r.t. to the behavior policy π_b, where

$$\xi = \frac{4v_{max}}{1-\gamma}\sqrt{\frac{2}{N_\wedge}\log\frac{2|\mathcal{S}||\mathcal{A}|2^{|\mathcal{S}|}}{\delta}}. \tag{155}$$

So, one has to choose the hyper-parameter $N_\wedge$, which is the number of visits a state-action pair needs for the behavior policy to be changed at this state-action pair, such that this lower bound becomes meaningful. In the Wet Chicken benchmark, $|\mathcal{S}| = 25$, $|\mathcal{A}| = 5$ and $v_{max} \approx 20$ (referring to an MDP with centered performances). I use $\gamma = 0.95$ and want $\delta \leq 0.05$. To choose $N_\wedge$, Equation 155 is transformed to

$$N_\wedge = \frac{32v_{max}\log\left(\frac{2\mathcal{S}||\mathcal{A}|2^{|\mathcal{S}|}}{\delta}\right)}{\xi^2(1-\gamma)^2}. \tag{156}$$

To gain again a lower bound of at least -8, one has to set $\xi = 8$ and inserting all the values into Equation 156 yields $N_\wedge = 1,832,114$ for $\delta = 0.05$. For a smaller δ this is even bigger. It is obvious that Π_b-SPIBB does not work for a $N_\wedge$ that high, as the output policy would always be the behavior policy unless there are far more than $10,000,000$ observations available, what I do not want to assume for a small MDP like this. For this reason, I exclude Π_b-SPIBB from the following experiments.
DUIPI's safety stems from the calculation of the variance of the action-value function. By assuming a normal distribution on the action-value function, this yields a confidence interval on the real action-value function q, as

$$\mathbb{P}(q(s,a) > q_u(s,a)) = 1 - F(\xi) \tag{157}$$

holds, with q_u as defined in Equation 144 and F as the CDF of a standard normal distribution [29]. For example, for a lower bound which applies to 99% one has to choose $\xi > 2.33$. The problem here is that one does not know in advance how well the lower bound is, as this depends on the variance which depends on the policy itself. The results are shown in Figure 21. One sees very well how much data is necessary to get a sharp bound and produce a good policy at the same time. However, it can be seen that this works very well with enough data and a highly explorative policy, especially when compared to the SPIBB based algorithms.
Nevertheless, I have to mention again that this lower bound is subject to some assumptions and approximations. First of all, DUIPI neglects the covariance between the action-value function on different state-action pairs. Also, the bound assumes a normal distribution on the action-value function. This might seem unreasonable as value functions are bounded, however, the ECDFs of the bounds in Figure 21 looks quite similar to the CDF of the normal distribution. Besides these theoretical issues with the lower bound, these experiments show that the DUIPI's lower bound is at least in this benchmark a useful one, as it is never violated. This cannot be inferred from Figure 21 as the plot of the lower bound is also an ECDF plot and, thus, they are not necessarily sorted according to how the performance is sorted. However, I checked this additionally and at least for $\delta = 0.01$ this holds. Also, it might be reassuring that the ECDF of the lower bound of the performance and of the performance itself often look similar to the CDF of a normal distribution.

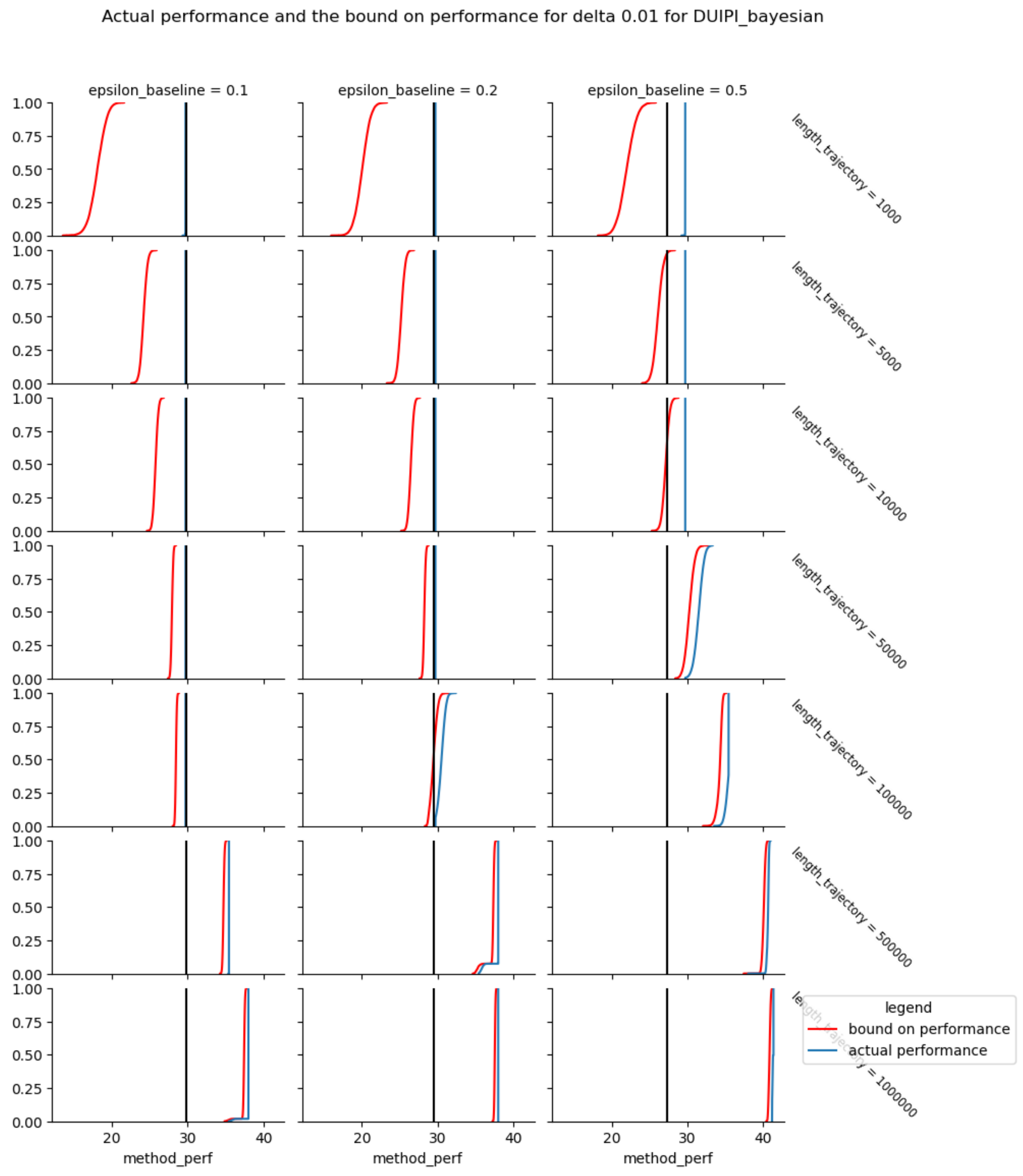


Fig. 21. *ECDF plot of the performance of DUIPI and the lower bound on its performance on the Wet Chicken benchmark with ξ such that the lower bound holds with probability 0.99.*

I have shown several things in this chapter. First of all, the bound of SPIBB turned out to be—at least on the Wet Chicken benchmark—not useful. The bound of Adv-Approx-Soft-SPIBB and 1-step Approx-Soft-SPIBB turned out to be a bit better. However, the algorithms are still strongly constrained if a meaningful bound should be achieved. The improvement of Theorem 3.2 in Equation 112 helps here, but is by far not strong enough. A stronger improvement is achieved by applying the error function relying on the Maurer and Pontil bound and its improvement even remains significant after omitting data to ensure independence. DUIPI's performance and bound improves substantially with respect to the one of the Soft-SPIBB algorithms as soon as at least 500,000 steps of the behavior policy have been observed. This bound of DUIPI depends on an assumption of the distribution of the true action-value function and it might be interesting to investigate if a similar assumption for Soft-SPIBB helps to improve the bound significantly.

3.4.4 Continuous Wet Chicken benchmark

In this section I use the continuous version of the Wet Chicken benchmark to find out if I can reproduce the promising results from Nadjahi et al. [11] in this setting. Naturally, I would like to include many of the algorithms introduced in this thesis, however, most of these algorithms do not come with a straightforward adaption to function approximation. Therefore, I only include the four algorithms used by Nadjahi et al. [11]—Vanilla-DQN, RaMDP-DQN, Π_b-SPIBB-DQN and Approx-Soft-SPIBB-DQN, see Section 2.3.4—and the function approximation version of Lower-Approx-Soft-SPIBB. Applying function approximation to Lower-Approx-Soft-SPIBB works exactly as for Π_b-SPIBB-DQN and Approx-Soft-SPIBB-DQN. So, Equation 44 is used as the target for the training neural network and in each step the policy is computed by using Algorithm 5, where the training neural network is used to estimate the action-value function.
The continuous Wet Chicken benchmark only omits the rounding in Equation 152 and is otherwise the same as the discrete version. Note that continuous actions are still not allowed and, thus, the boat always stays on one of the 5 lanes in the y-direction, i.e., the state space can be exactly modeled as $[0, 5) \times \{0, 1, 2, 3, 4\}$.
Nadjahi et al. [11] provide the framework for these experiments in their github repository[9], which enables the use of different environments and additional algorithms. For their computations, they apply a fully connected neural network with 3 hidden layers with 32, 128, and 32 neurons, with the rectified linear unit (ReLU) [32]

$$f(x) = \max\{0, x\} \tag{158}$$

as the activation function. The input layer has 1 neuron per state space dimension and the output layer has as many neurons as there are actions. So, in the Wet Chicken benchmark the networks have 2 input neurons and 5 output neurons. Due to computational reasons I omit one hidden layer and reduce the number of neurons to 32 and 16, as this still suffices to learn an appropriate action-value approximation. This was observed by trying to learn a policy online using Vanilla DQN and this smaller network achieved the same results as the big one while even smaller networks struggled to find a good policy. Thus, this network size should be already expressive enough to approximate the true value function, while it is small enough to conduct many experiments.
Following Nadjahi et al. [11], I also use the network initialization He et al. [33] proposed. This means I initialize the weights of the neural network by sampling for each weight w connecting a layer with n_{in} neurons to one with n_{out} neurons from a uniform distribution over the interval $[-a, a]$, where $a = \sqrt{6/n_{in}}$. This supports the convergence of neural

[9]https://github.com/rems75/SPIBB-DQN, assessed on 20.03.2021

networks using ReLU for the activation and a detailed theoretical analysis can be found in He et al. [33].
As an optimizer Nadjahi et al. [11] rely on the popular optimization algorithm *RMSProp* introduced by Tieleman and Hinton [34]. RMSProp uses a parameter update similar to stochastic gradient descent:

$$w_n^i = w_{n-1}^i - \frac{\alpha}{\sqrt{M_n + \zeta}} \frac{\partial L(w_{n-1})}{\partial w^i}. \tag{159}$$

Here, $w_m^i \in \mathbb{R}$ is one parameter of the weights at step n, L is the loss function, $\alpha > 0$ is the learning rate, $\zeta > 0$ is a constant improving the stability of the update and M_n is the rolling average of the mean of recent squared partial derivatives at this weight. M_n is computed inductively by

$$M_n = \beta M_{n-1} + (1-\beta)\left(\frac{\partial L(w_{n-1})}{\partial w^i}\right)^2, \tag{160}$$

where $\beta > 0$ is the forgetting factor. Following Nadjahi et al. [11], I set $\beta = 0.95$ and $\zeta = 10^{-7}$ and initialize the learning rate to $\alpha = 0.01$ and decay it by using $\frac{\alpha}{m}$ in the m-th pass over the data. The batch size I use is 32.
The behavior policy for this benchmark is trained using the online variant of Vanilla DQN as described before, which is basically a DQN [22] with double Q-learning [23]. Following Nadjahi et al. [11], the softmax function is applied with temperature 0.1 to the behavior policy, which means that

$$\pi_b(a|s) = \frac{\exp(0.1Q(s,a))}{\sum_{a'} \exp(0.1Q(s,a'))} \tag{161}$$

is the behavior policy used to create the data sets and Q is the action-value function learned using Vanilla DQN. This softmax step is necessary to make the behavior policy stochastic. Using this policy, data sets with 3,000 and 10,000 transitions are generated.
I try to use many different hyper-parameters for each algorithm as I did for the previous experiments. Due to computational and time limitations however, it is not possible to conduct a thorough hyper-parameter search. For the parameter η for the pseudo-count as defined in Equation 49 I try 0.5 and 1, following the logic that a state which is more than 2 apart from another state does not contain much direct information about the other one.
To start with, I want to state that it is inherently different to train a neural network to approximate an action-value function in reinforcement learning than to approximate a function in supervised learning. The big difference is that in supervised learning the labels are known and, thus, the calculation of the loss is always meaningful and with enough training steps, a good optimizer with well chosen parameters, and an expressive network the training error can be reduced as much as desirable. Contrary to that, one does not have access to the true action-value function in reinforcement learning and, thus, has to estimate the labels. This is done by using targets like these in Equation 44. Besides the convergence problem mentioned in Section 2.3.4 caused by changing targets, another issue that comes up is, that if the current approximation of the action-value function is bad, the loss computed by using the target will also be bad [35].

In the online setting, many algorithms like DQN solve these issues by correcting wrong estimates through collecting more data at desired places [22]. For offline reinforcement learning, this is not possible. This can be seen by the poor performance of Vanilla DQN in the table.

Algorithms	Mean performance (its standard deviation) for 3000 data steps	Mean performance (its standard deviation) for 10000 data steps
Vanilla DQN	1.068 (0.065)	1.119 (0.062)
RaMDP-DQN ($N_\wedge = 15$)	1.903 (0.054)	1.767 (0.065)
SPIBB-DQN ($N_\wedge = 3$)	2.474 (0.003)	2.483 (0.003)
Approx-Soft-SPIBB-DQN ($\epsilon = 2$)	2.450 (0.004)	2.464 (0.005)
Lower-Approx-Soft-SPIBB-DQN ($\epsilon = 0.2$)	2.450 (0.003)	2.473 (0.003)

This table records the performances over 100 trials for the different algorithms on the continuous Wet Chicken benchmark with the parameter for the pseudo-count set to $\eta = 0.5$, as the performance of the algorithms seems to be slightly better than for $\eta = 1$. The first column states the algorithms together with the hyper-parameter for which they performed the best. The second column states the mean performance and the standard deviation of the mean performance over 100 runs for a data set with 3,000 steps. The last column shows the same for 10,000 steps.

The gap between Vanilla DQN, RaMDP-DQN and the SPIBB and Soft-SPIBB algorithms becomes very clear, regardless if the smaller or the bigger data set is considered. As the standard derivation is small compared to the difference in mean performance it can be concluded that this difference is not by chance. The behavior policy has a performance of 2.411, so only the SPIBB and Soft-SPIBB algorithms manage to improve the behavior in the mean. Among the SPIBB and Soft-SPIBB algorithms, while there is no particularly clear preference, SPIBB-DQN appears—contrary to the experiments of Nadjahi et al. [11]—the foremost in this benchmark.

In the benchmarks before, one could see that the SPIBB and especially the Soft-SPIBB algorithms exhibit an extremely good 1%-CVaR performance. However, due to computational limitations it is not feasible for me to conduct more than 100 trials on the continuous Wet Chicken benchmark and, thus, it is not recommendable to consider the 1%-CVaR performance. Instead I want to point at the standard derivation again, as this is a measure of how much the samples differ from the mean. Here, one sees that the standard derivation is significantly smaller for the SPIBB and Soft-SPIBB algorithms, which means that incorporating the uncertainty of state-action pairs and bootstrapping the behavior policy helps to compute policies with a narrow range of performances.

These experiments are not aimed at providing insight into the ranking in performance among the algorithms used. The foremost goal is to complement the experiments conducted by Nadjahi et al. [11] and show that this class of algorithms gives rise to intuitive algorithms which can be used for a continuous state space and still manage to consistently improve the behavior policy, even though other algorithms like Vanilla DQN and RaMDP-DQN fail to do so.

4 Conclusion

Motivated by industrial applications, this master's thesis aimed at finding theoretical and empirical safe offline reinforcement learning algorithms following along the paper by Nadjahi et al. [11]. I quickly summarize the results of this master's thesis in Section 4.1 and provide possible topics which might be addressed by future research in Section 4.2.

4.1 Summary

In Section 2.3 the Soft-SPIBB algorithms introduced in Nadjahi et al. [11] and the theory backing them up were presented. To start the analysis of this thesis, the validity of this theory was assessed in Sections 3.1 and 3.2. Thereby, it was shown that there are several issues around the theory, which prohibits its applicability to the Soft-SPIBB algorithms. However, a variant of their safety theorem is proven as Theorem 3.2. As it cannot be applied to the original Soft-SPIBB algorithms, they are extended to the class of Adv-Soft-SPIBB algorithms for which this safety bound holds. Complementary, Section 3.2.4 includes a detailed discussion about the strengths and weaknesses of Theorem 3.2.
In Section 3.3 several more offline reinforcement learning algorithms from the literature, which try to incorporate the uncertainty of state-action pairs, were introduced. Some of these algorithms had to be adapted to the safe offline reinforcement learning setting and, additionally, the completely new Lower-Approx-Soft-SPIBB algorithm was introduced, which builds upon the original Approx-Soft-SPIBB by incorporating a reasonable heuristic. Section 3.3.7 described a new Python framework, which implements all of the algorithms used in this thesis and can be used for further experiments in the future.
In the following, experiments on the Random MDPs [11] and Wet Chicken [7] benchmarks were conducted to assess the empirical performance of all algorithms. Here, all variants of the Soft-SPIBB algorithms turned out to be empirically safe, meaning that their 1%-CVaR performance was in general better than the one of the remaining algorithms. Especially Lower-Approx-Soft-SPIBB excelled for this measure and showed also the best mean performance among the Soft-SPIBB family. Section 3.4.3 examined the strength of different theoretical bounds in the sense of Theorem 3.2, which showed that these bounds hold in practice, but are very conservative. However, methods like sharpening the bounds through better uncertainty estimation as explained in Section 3.2.4 turned out to be highly effective.
In Section 3.4.4 the performances of the approximate versions of the Soft-SPIBB algorithms were explored in an MDP with a continuous state space. They were the only algorithms which were able to improve the behavior policy in the mean and also exhibited a very small standard deviation compared to the other two algorithms, which hints at a consistent improvement and few outliers. Therefore, these experiments allow me to support the empirical findings of Nadjahi et al. [11], that the Soft-SPIBB family can be used to derive efficient deep offline reinforcement learning algorithms.
To conclude, this master's thesis corrected and extended the theory and algorithms developed in Nadjahi et al. [11] and showed that this class of algorithms indeed achieves remarkable performances in the discrete and continuous setting. Thus, I hope that it motivates future safe reinforcement learning research to include these algorithms. Possible theoretical and empirical research options are mentioned in the following section.

4.2 Outlook

Regarding the theory supporting the safety of the algorithms, there are three interesting points that should be subject of future research. The first is to try tightening the bound in Theorem 3.2, which might be achieved either directly by bounding the second summand of Equation 110 closer from below or indirectly by sharpening the uncertainty estimation of state-action pairs. The latter approach could prove rewarding by considering more concentration inequalities like the empirical version of Bennett's inequality [27] or by assuming reasonable distributions on the value function as it is done by Schneegass et al. [29].

In Section 3.2.4 it is discussed how to fulfill the assumption that all returns for one state-action pair have to be independent. The solution for non-episodic MDPs introduced there is only heuristic and works by bounding the correlation by some arbitrarily small number. There are two different ways two solve this problem in a theoretically exact way apparent to me. One could utilize an adaption of the classical concentration inequalities which deal with dependent random variables, see, for example, the work done by Pelekis and Ramon [36]. Another possibility is to use a similar approach to the one described in Section 3.2.4. However, instead of using slightly correlated but unbiased samples, one could change them to be slightly biased but independent. Adopting the bounds such that this bias is taken into account might be a feasible solution.

The last and most interesting open question about the theory is, whether there is a way of extending the theoretical safety in the discrete setting to the continuous setting. However, this most probably requires completely new concepts for the proof and even a different definition of safety.

Empirically, I tried to conduct the experiments very thoroughly in the discrete setting, thus, future research should be focused on function approximation for continuous state spaces. I only started some experiments in this setting in order to discern their viability, which yielded positive results. Therefore, it might be of high interest to conduct more thorough experiments iterating through a long range of hyper-parameters and using different behavior policies. Furthermore, one might want to try different pseudo-counts, network architectures, initializations, and optimizers as I mostly just followed along Nadjahi et al. [11]. Furthermore, the function approximation versions of SPIBB, Approx-Soft-SPIBB and Lower-Approx-Soft-SPIBB should be tested on various benchmarks against state of the art uncertainty incorporating offline reinforcement learning algorithms, similar to the experiments of Fujimoto et al. [35].

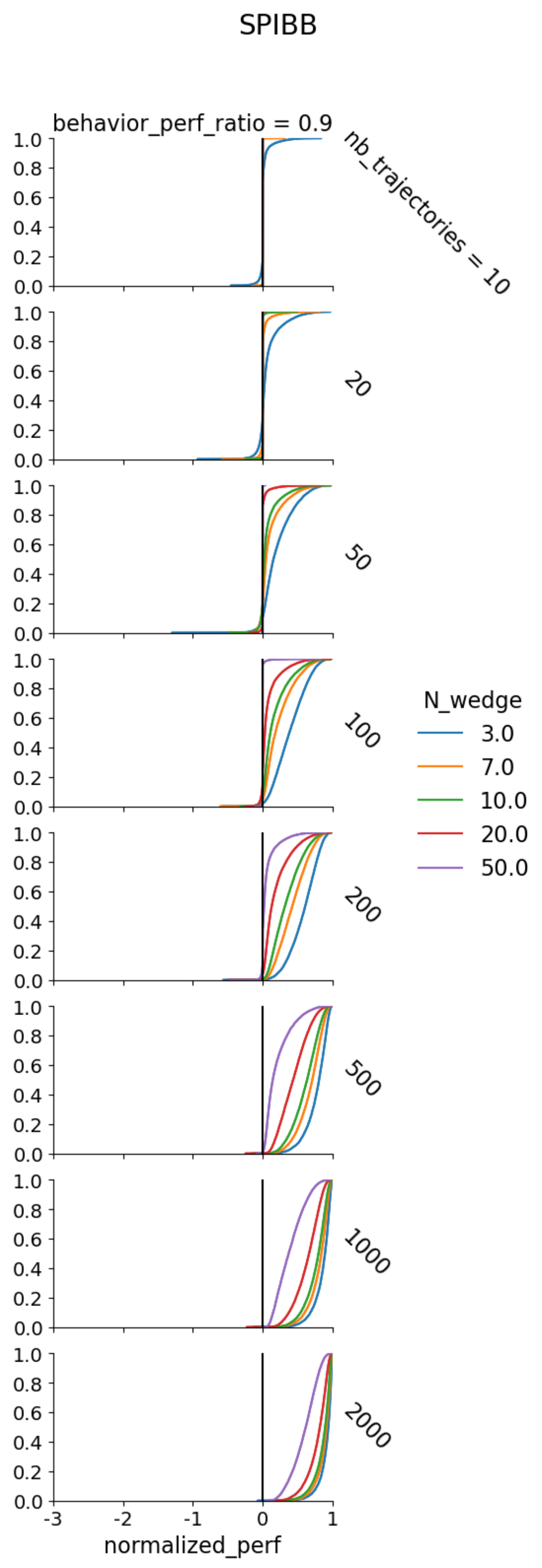
Appendices

A Hyper-parameter Choices

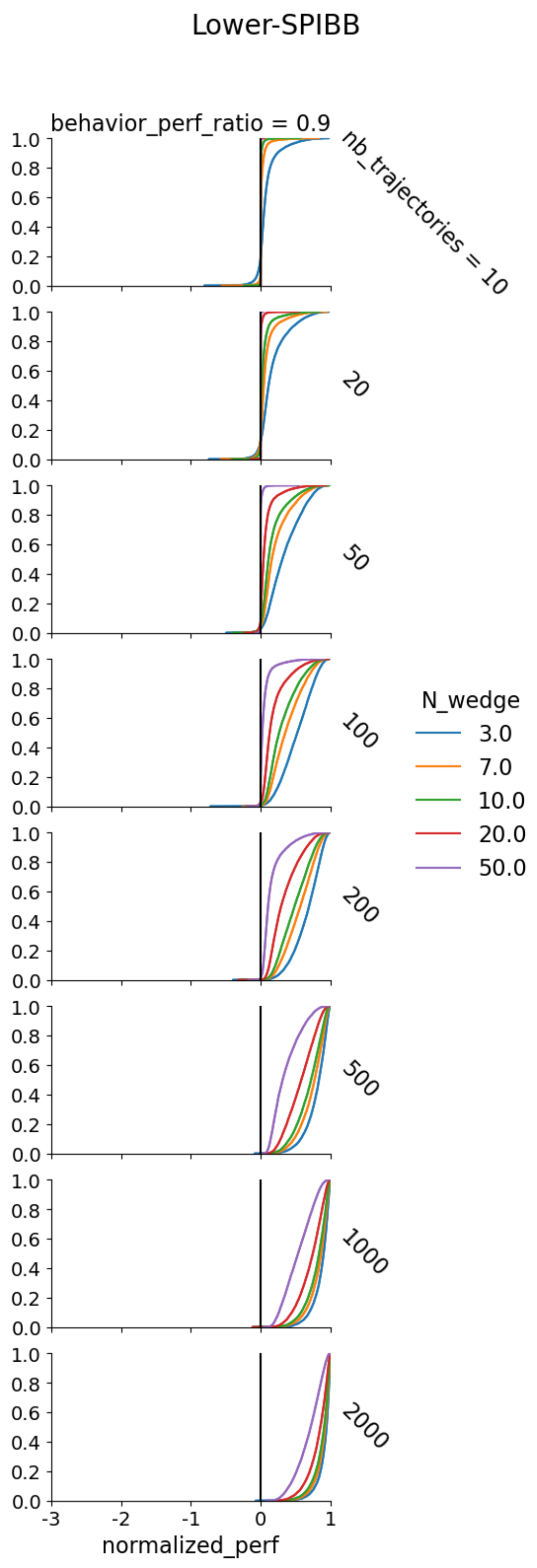
In the following the choices for the hyper-parameters for all algorithms used in Section 3.4 or Appendix B on the Random MDPs and discrete Wet Chicken benchmark are reported. DUIPI (native) refers to the original implementation of DUIPI, without the mechanism to avoid unvisited state-action pairs.

Algorithm	Hyper-parameter choice for the Random MDPs	Hyper-parameter choice for Wet Chicken
Π_b-SPIBB	$N_\wedge = 10$	$N_\wedge = 7$
$\Pi_{\leq b}$-SPIBB	$N_\wedge = 10$	$N_\wedge = 7$
Approx-Soft-SPIBB-Hoeffding	$\epsilon = 2$	$\epsilon = 1$
Approx-Soft-SPIBB-MPeB	$\epsilon = 5$	$\epsilon = 5$
1-Step-Approx-Soft-SPIBB-Hoeffding	$\epsilon = 1$	$\epsilon = 0.2$
1-Step-Approx-Soft-SPIBB-MPeB	$\epsilon = 2$	$\epsilon = 0.5$
Adv-Approx-Soft-SPIBB-Hoeffding	$\epsilon = 2$	$\epsilon = 1$
Adv-Approx-Soft-SPIBB-MPeB	$\epsilon = 5$	$\epsilon = 5$
Lower-Approx-Soft-SPIBB-Hoeffding	$\epsilon = 1$	$\epsilon = 0.5$
Lower-Approx-Soft-SPIBB-MPeB	$\epsilon = 5$	$\epsilon = 5$
DUIPI	$\xi = 0.1$	$\xi = 0.5$
DUIPI (native)	$\xi = 0.5$	-
R-MIN	$N_\wedge = 3$	$N_\wedge = 3$
RaMDP	$\kappa = 0.05$	$\kappa = 2$

B Additional Plots

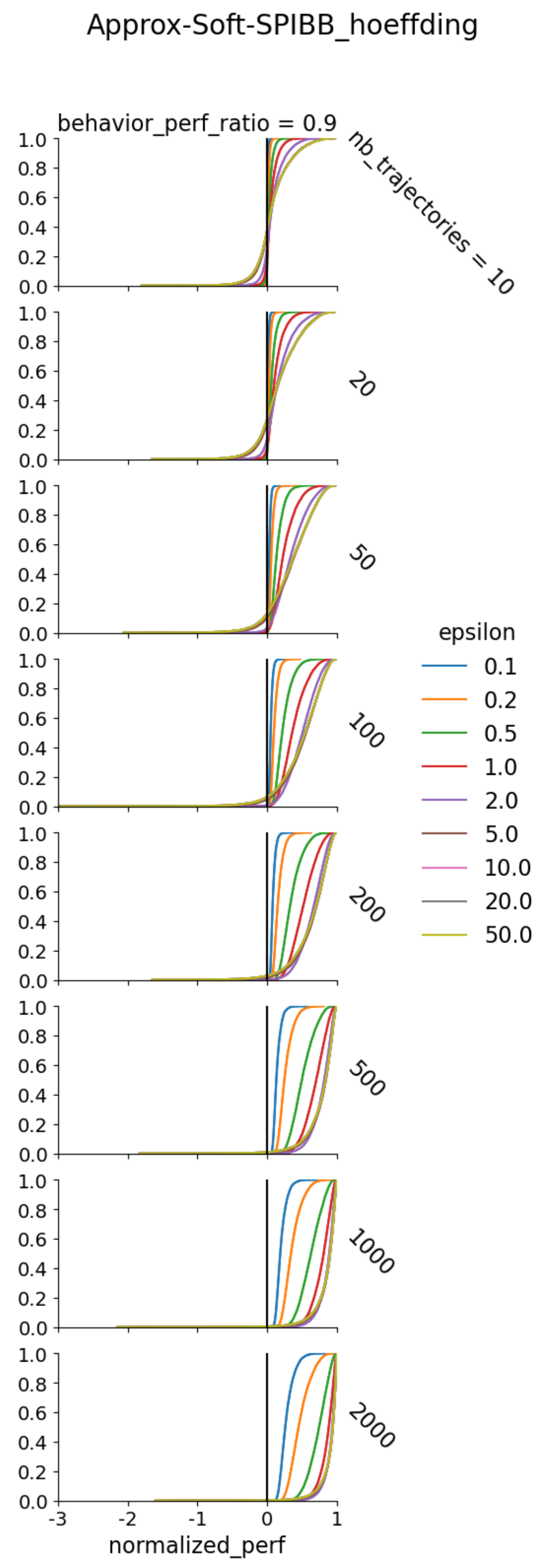


(a) Π_b-*SPIBB*

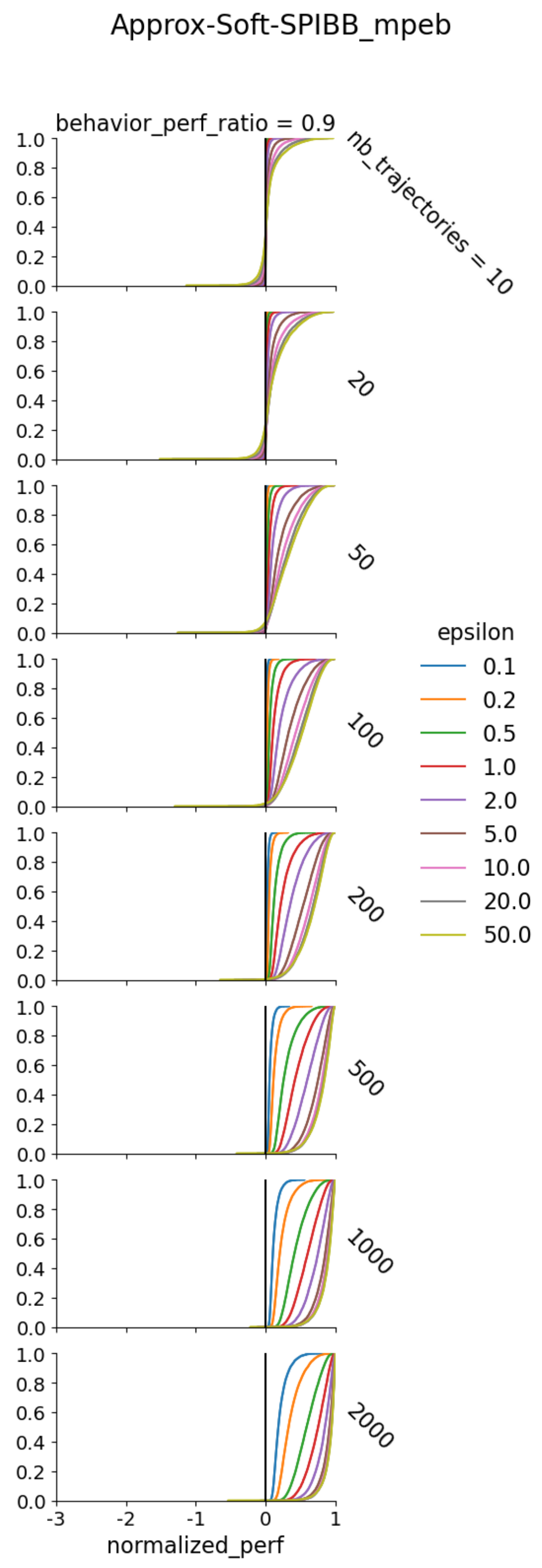


(b) $\Pi_{\leq b}$-*SPIBB*

Fig. 22. *ECDF for different hyper-parameter values for various algorithms on the Random MDPs benchmark.*

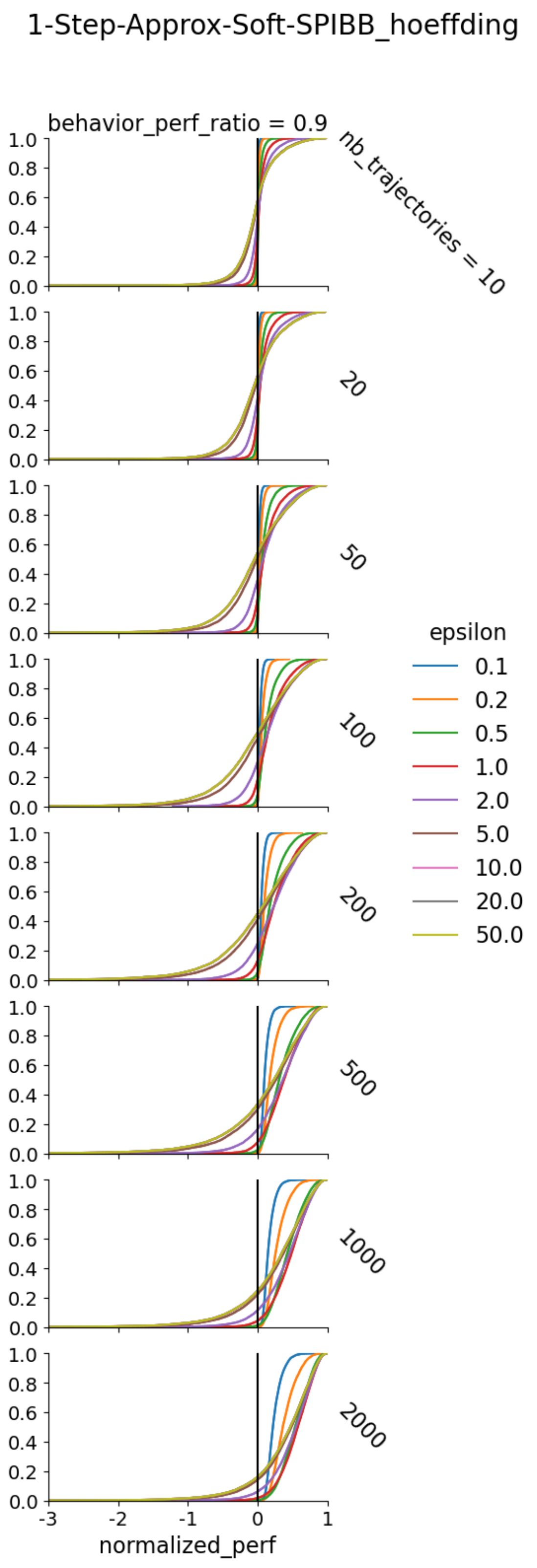


(c) *Approx-Soft-SPIBB-Hoeffding*

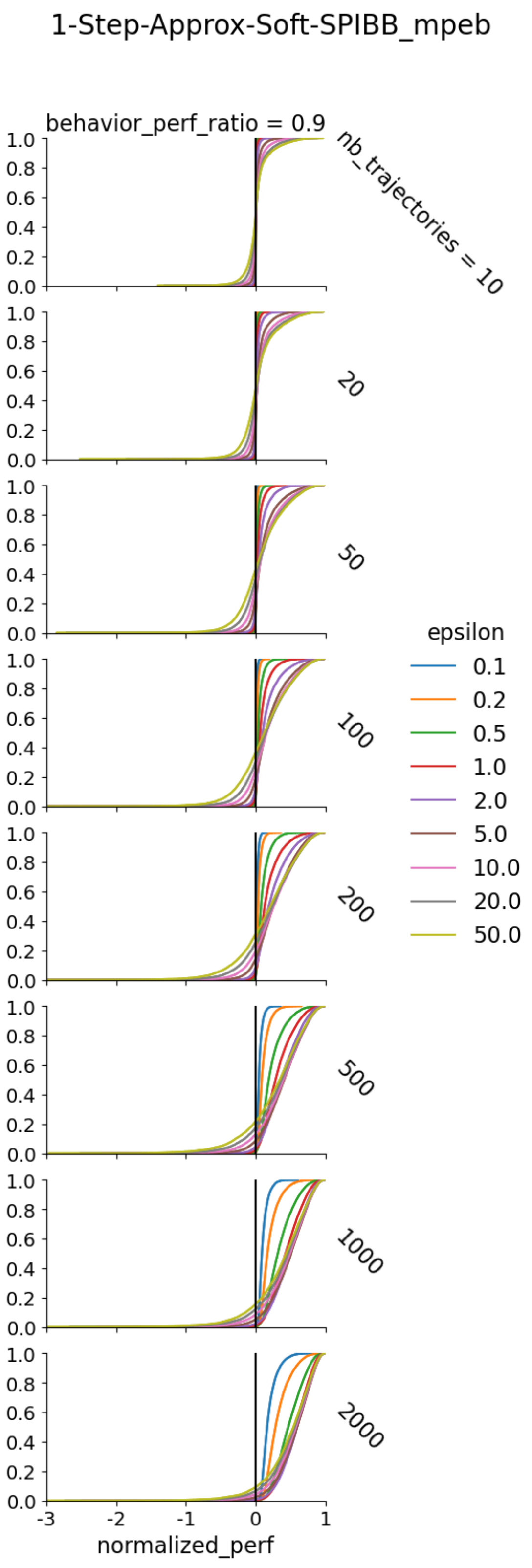


(d) *Approx-Soft-SPIBB-MPeB*

Fig. 22. *ECDF for different hyper-parameter values for various algorithms on the Random MDPs benchmark.*

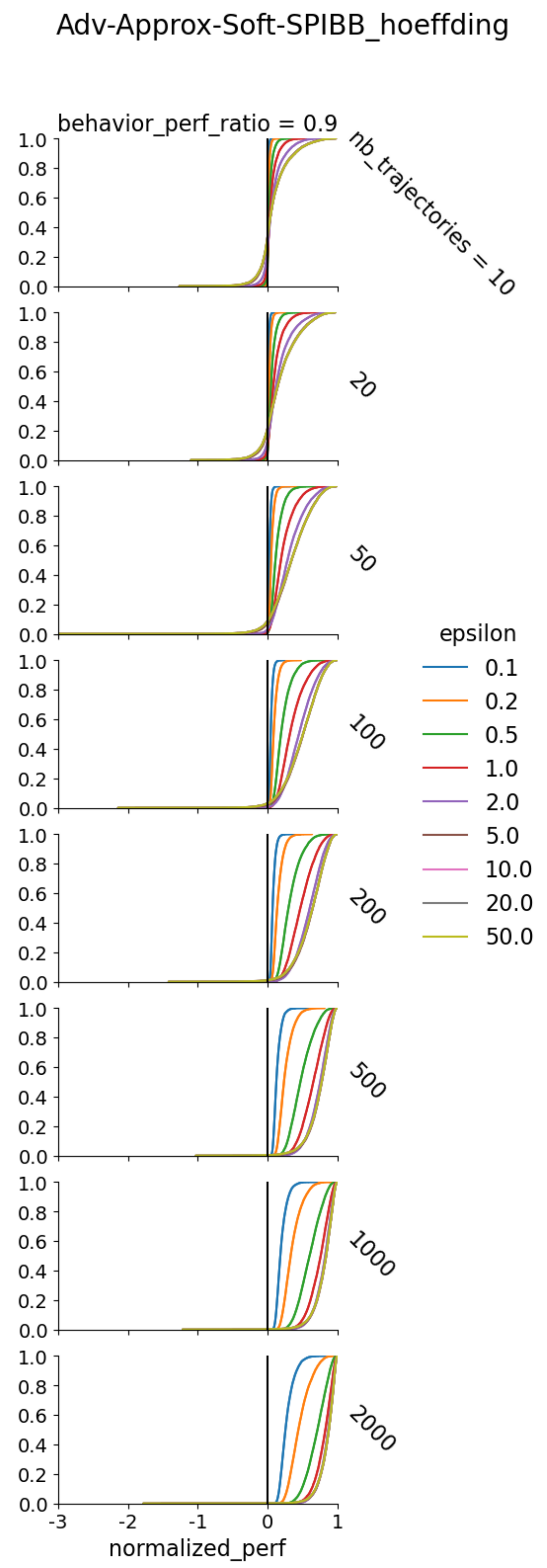


(e) *1-Step-Approx-Soft-SPIBB-Hoeffding*

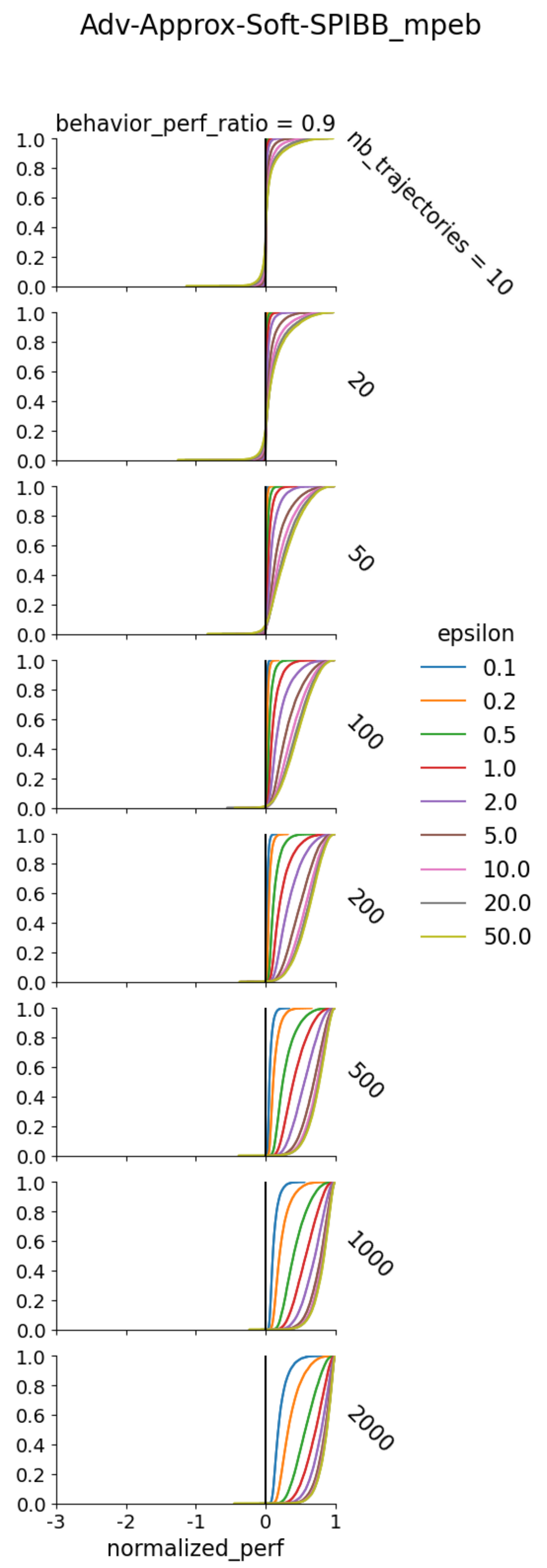


(f) *1-Step-Approx-Soft-SPIBB-MPeB*

Fig. 22. *ECDF for different hyper-parameter values for various algorithms on the Random MDPs benchmark.*

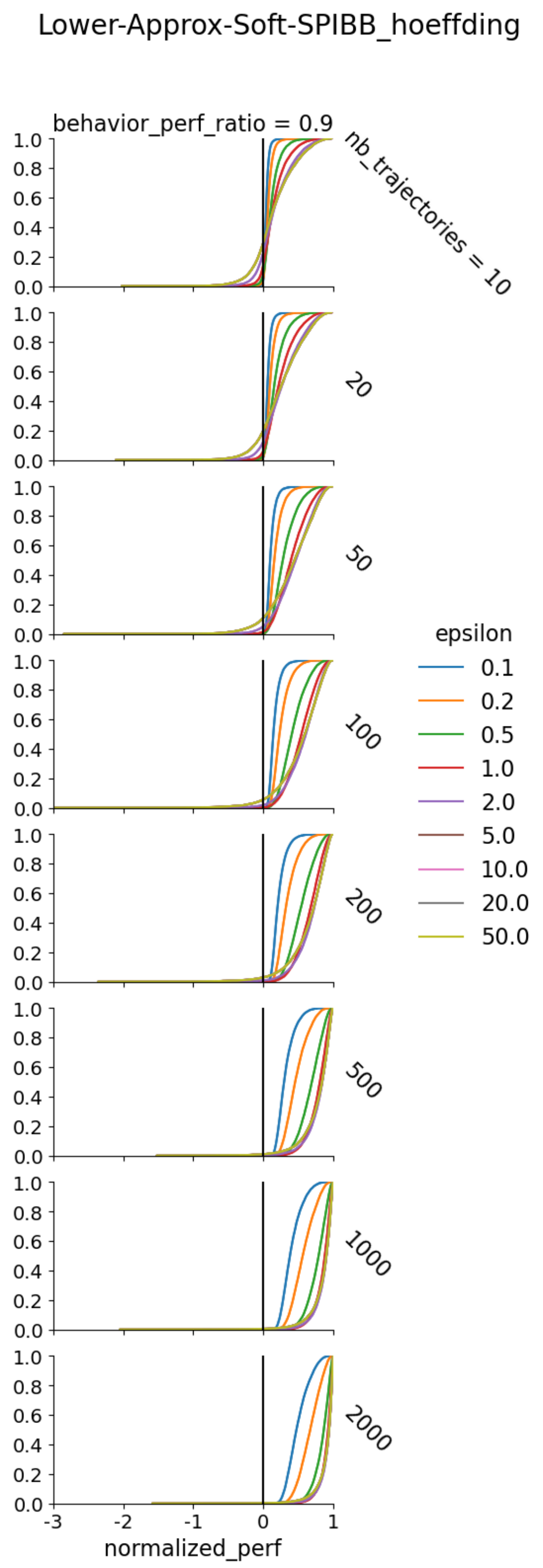


(g) *Adv-Approx-Soft-SPIBB-Hoeffding*

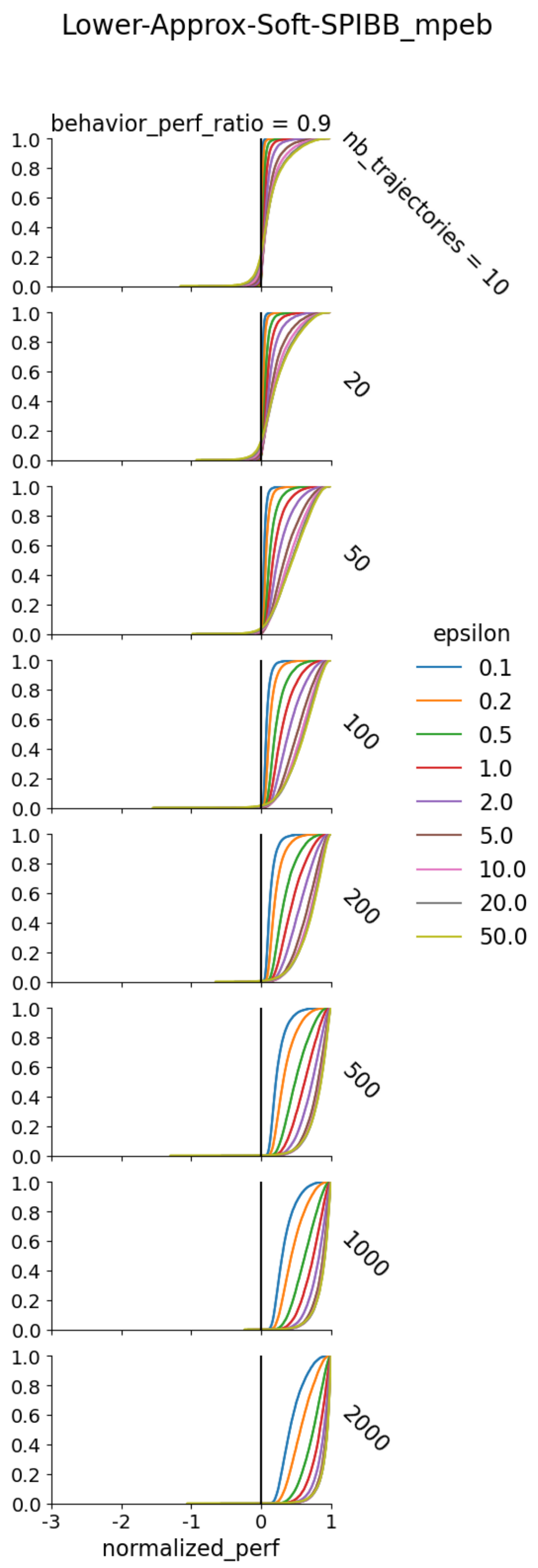


(h) *Adv-Approx-Soft-SPIBB-MPeB*

Fig. 22. *ECDF for different hyper-parameter values for various algorithms on the Random MDPs benchmark.*

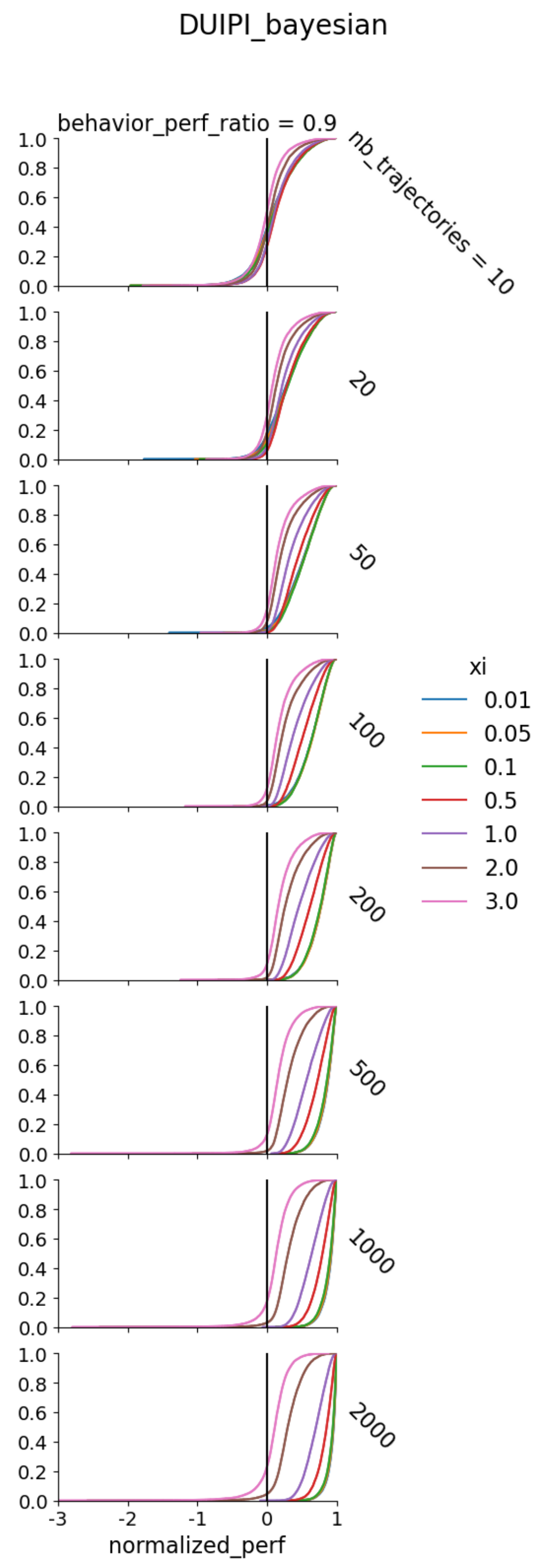


(i) *Lower-Approx-Soft-SPIBB-Hoeffding*

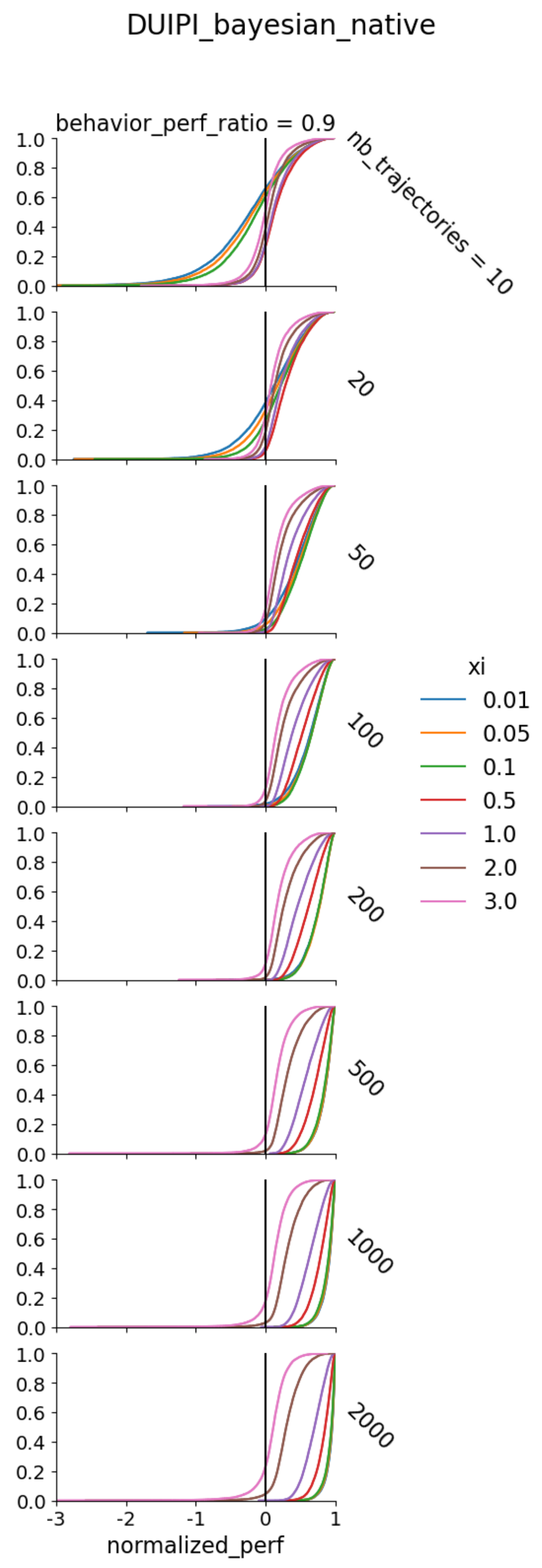


(j) *Lower-Approx-Soft-SPIBB-MPeB*

Fig. 22. *ECDF for different hyper-parameter values for various algorithms on the Random MDPs benchmark.*

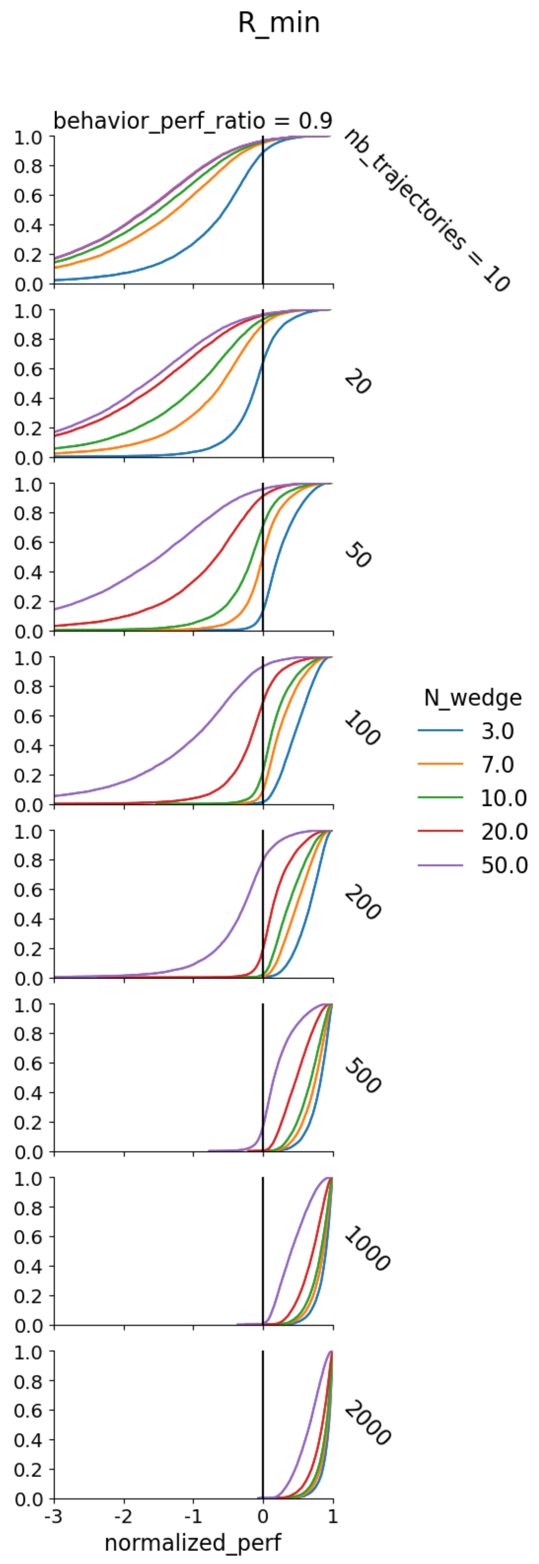


(k) *DUIPI*

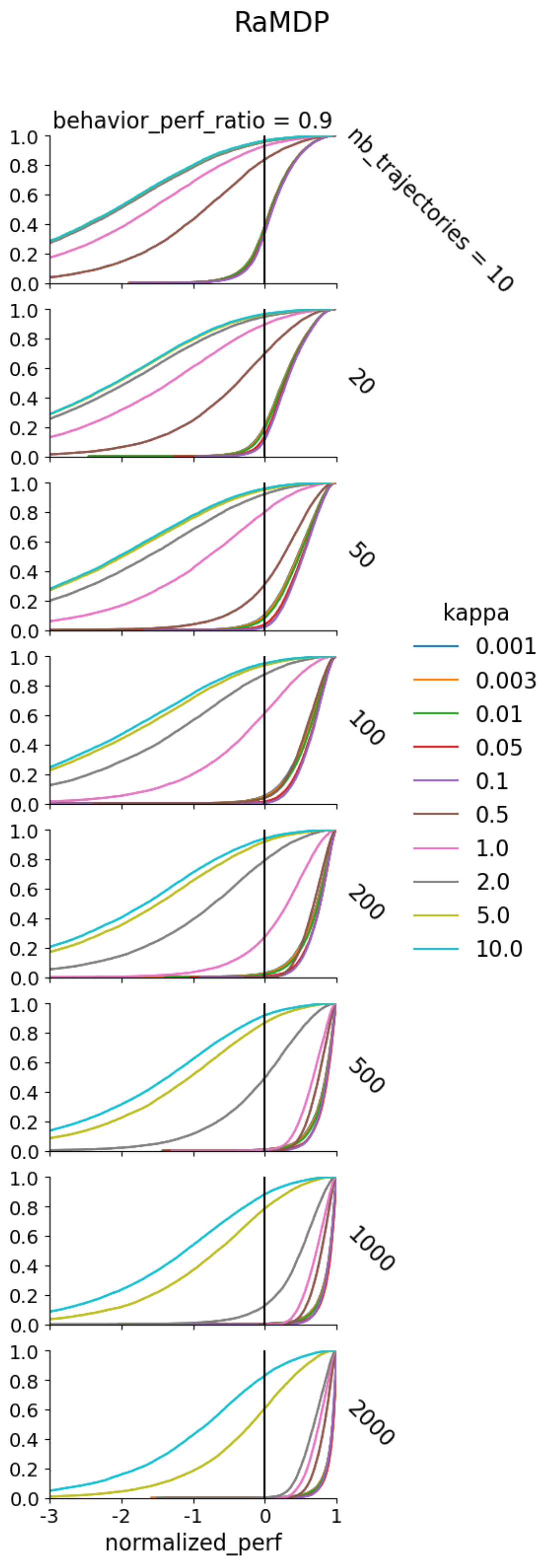


(l) *DUIPI (native)*

Fig. 22. *ECDF for different hyper-parameter values for various algorithms on the Random MDPs benchmark.*

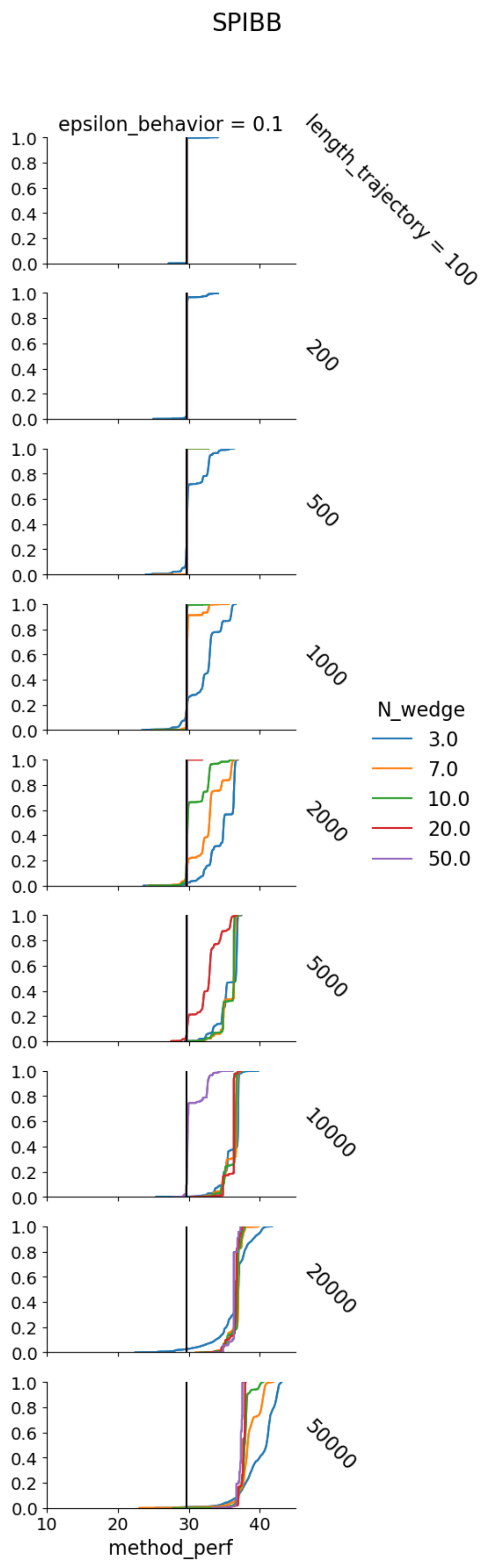


(m) *R-MIN*

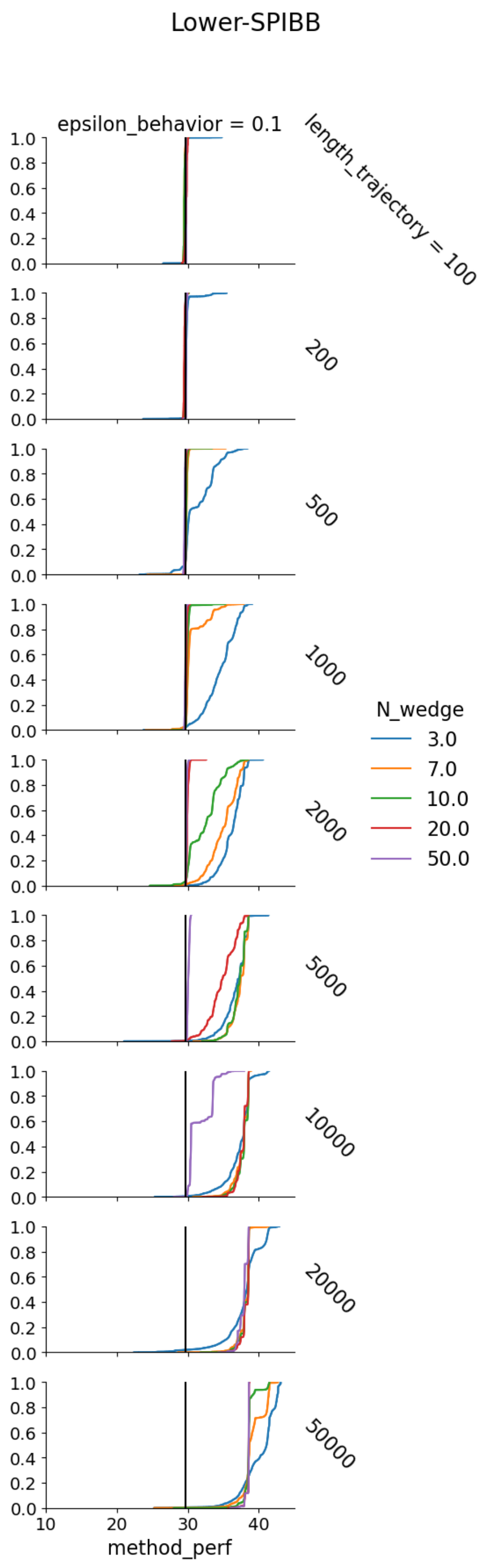


(n) *RaMDP*

Fig. 22. *ECDF for different hyper-parameter values for various algorithms on the Random MDPs benchmark.*

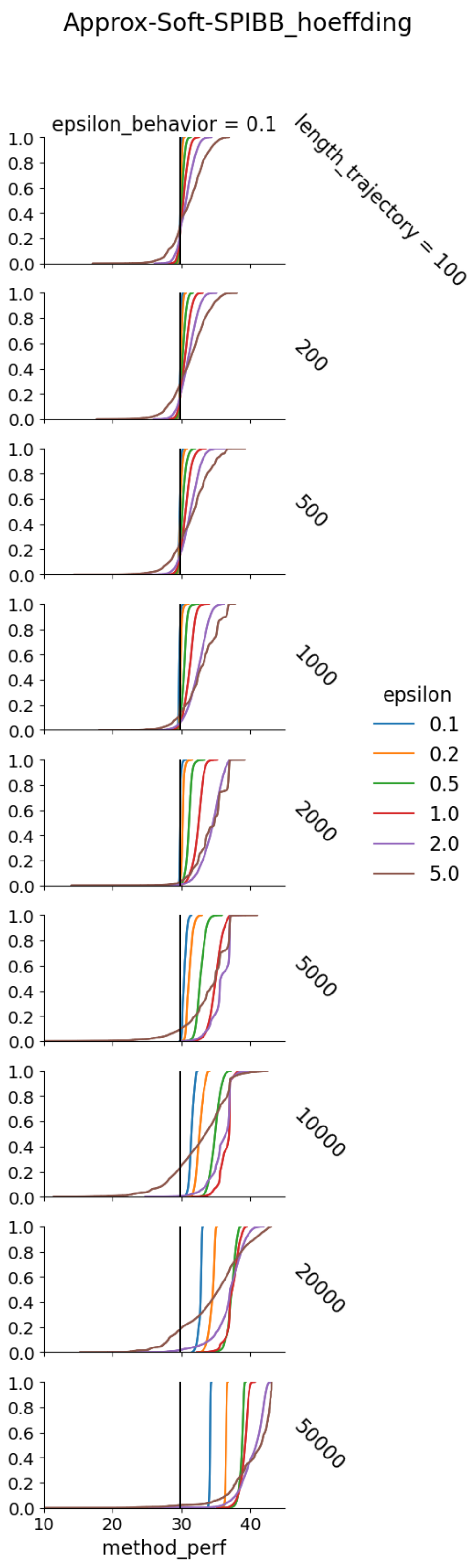


(a) Π_b-*SPIBB*

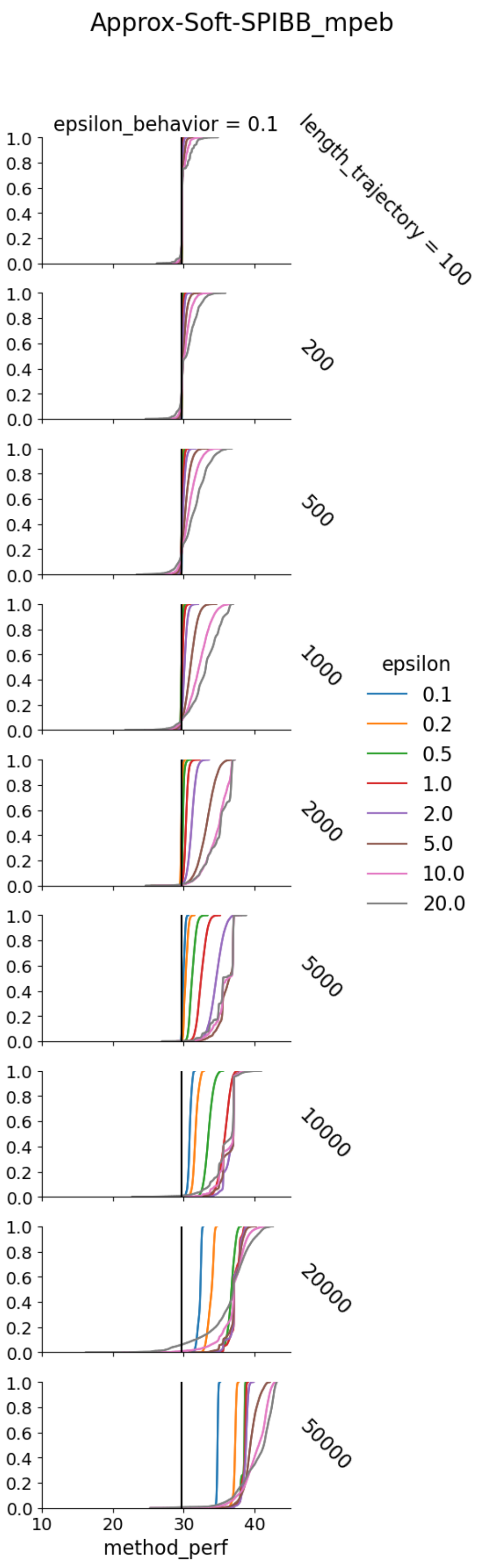


(b) $\Pi_{\leq b}$-*SPIBB*

Fig. 23. *ECDF for different hyper-parameter values for various algorithms on the Wet Chicken benchmark.*

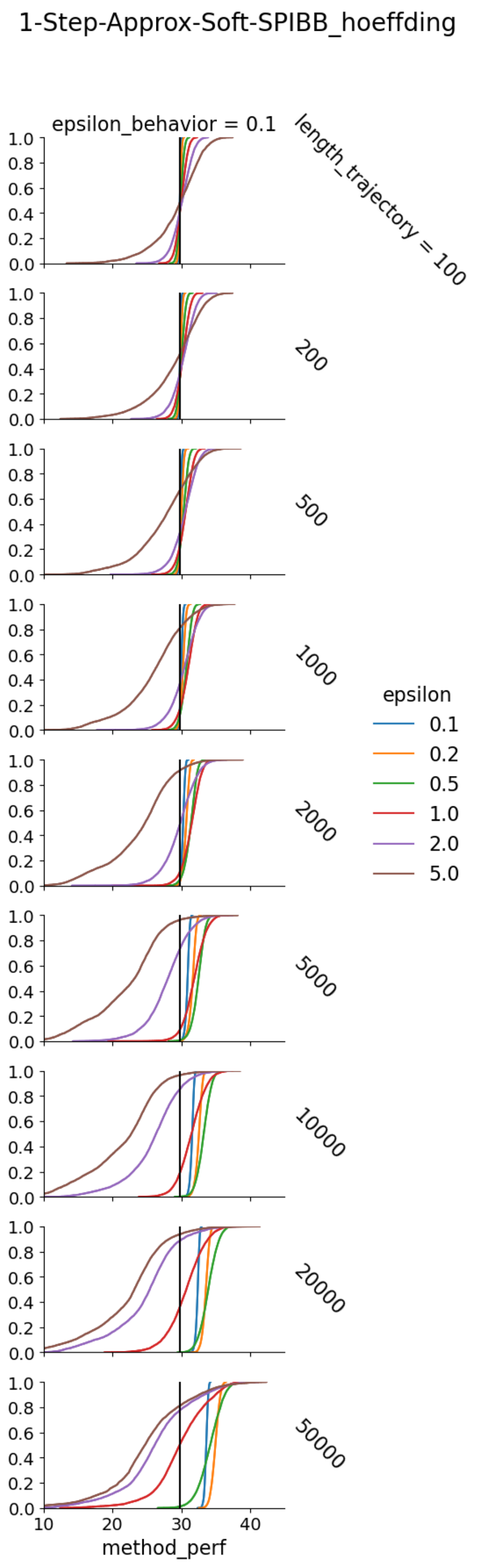


(c) *Approx-Soft-SPIBB-Hoeffding*

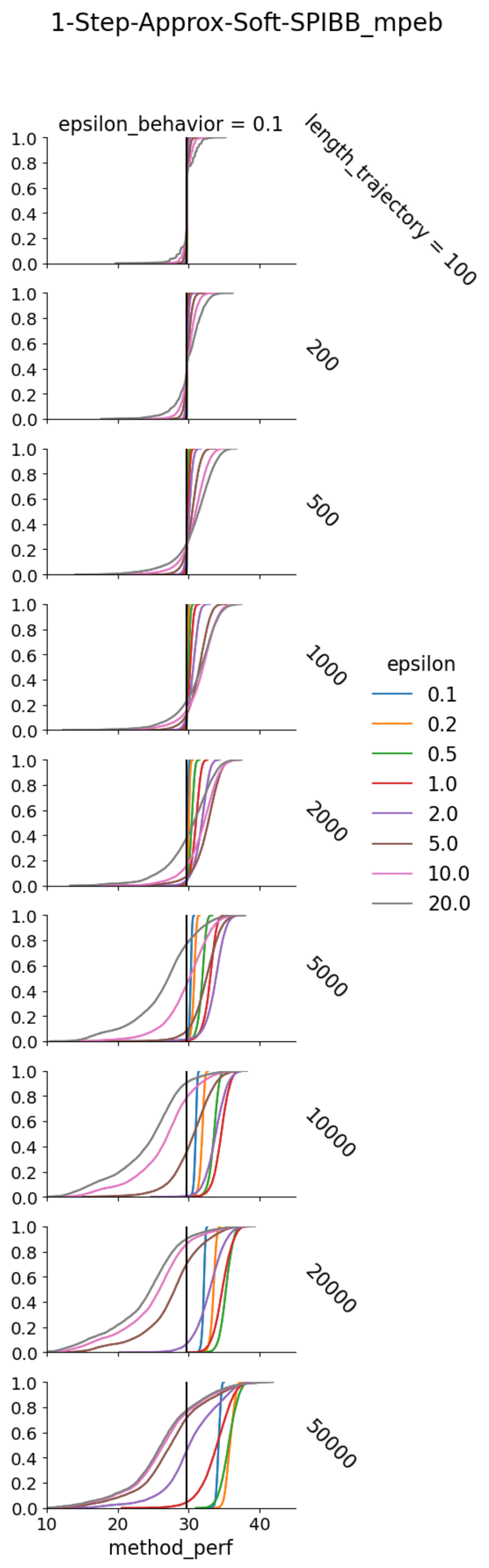


(d) *Approx-Soft-SPIBB-MPeB*

Fig. 23. *ECDF for different hyper-parameter values for various algorithms on the Wet Chicken benchmark.*

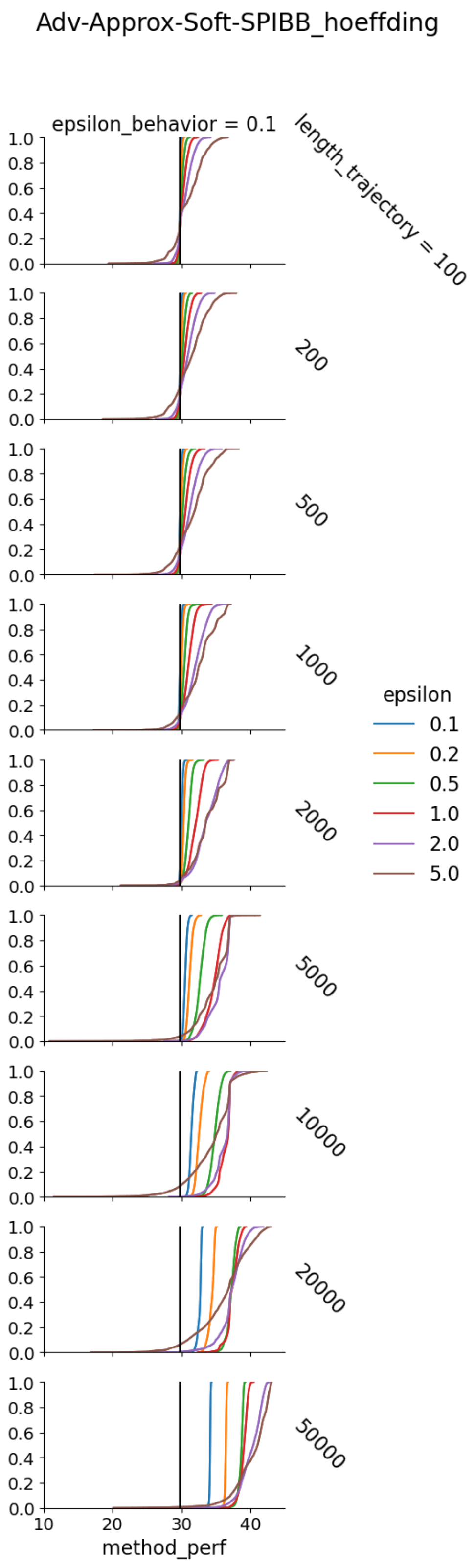


(e) *1-Step-Approx-Soft-SPIBB-Hoeffding*

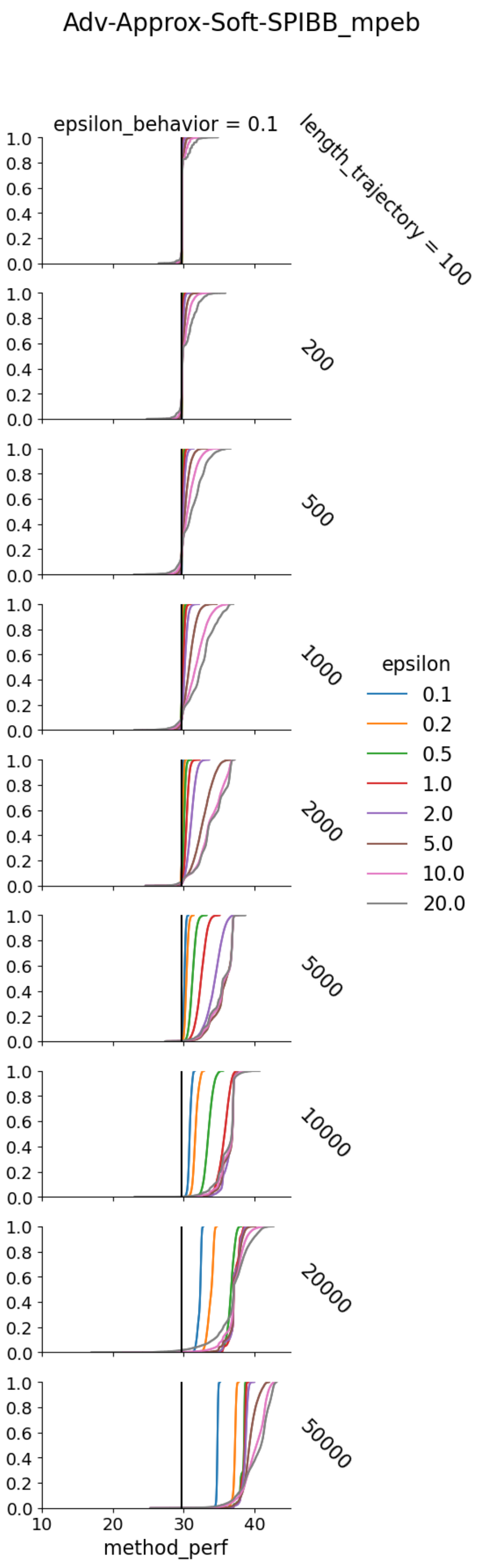


(f) *1-Step-Approx-Soft-SPIBB-MPeB*

Fig. 23. *ECDF for different hyper-parameter values for various algorithms on the Wet Chicken benchmark.*

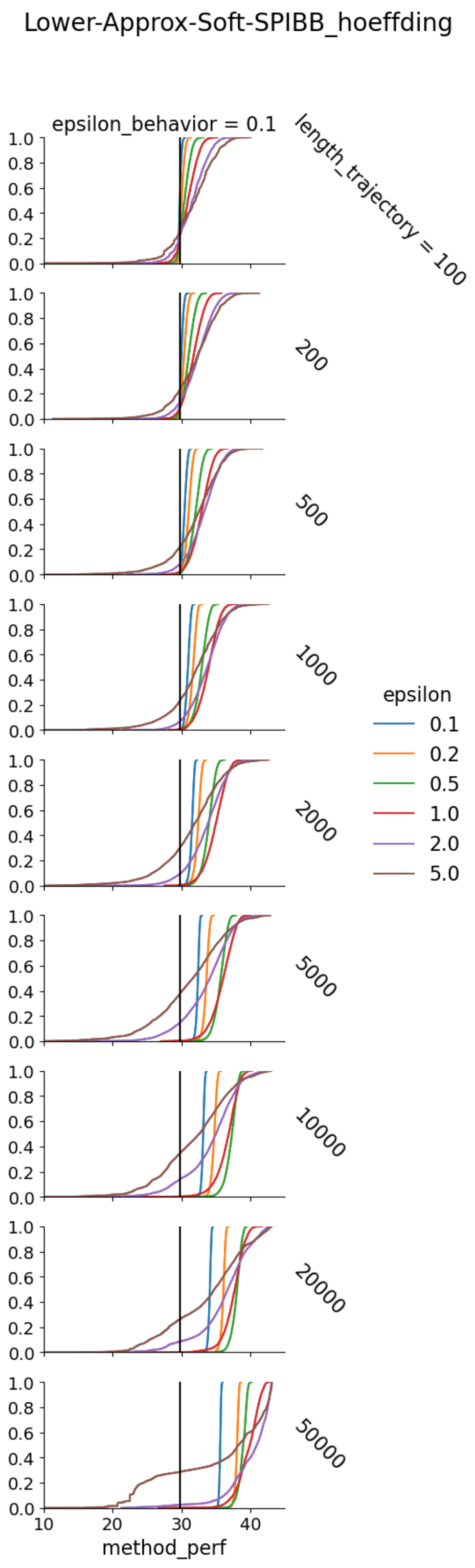


(g) *Adv-Approx-Soft-SPIBB-Hoeffding*

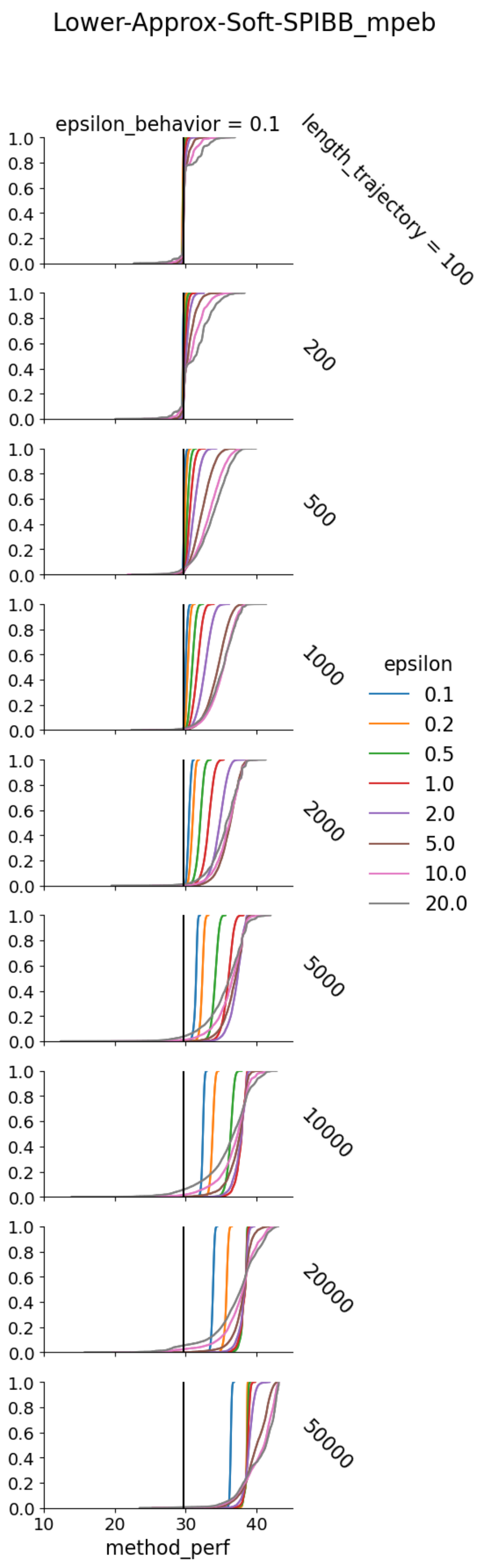


(h) *Adv-Approx-Soft-SPIBB-MPeB*

Fig. 23. *ECDF for different hyper-parameter values for various algorithms on the Wet Chicken benchmark.*

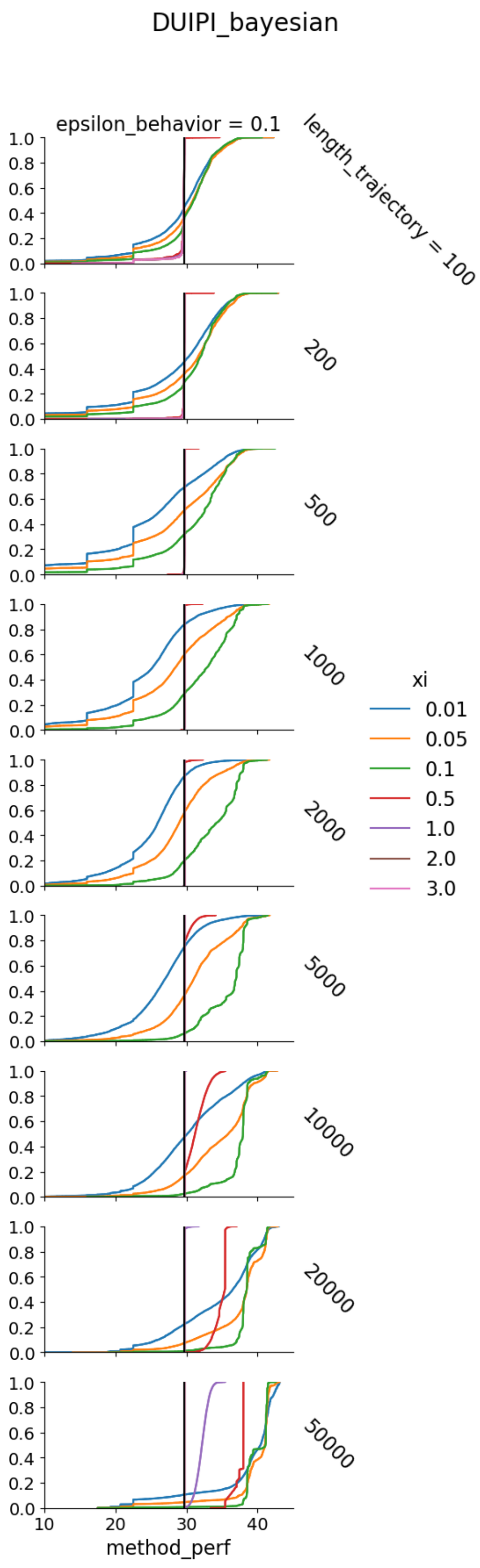


(i) *Lower-Approx-Soft-SPIBB-Hoeffding*

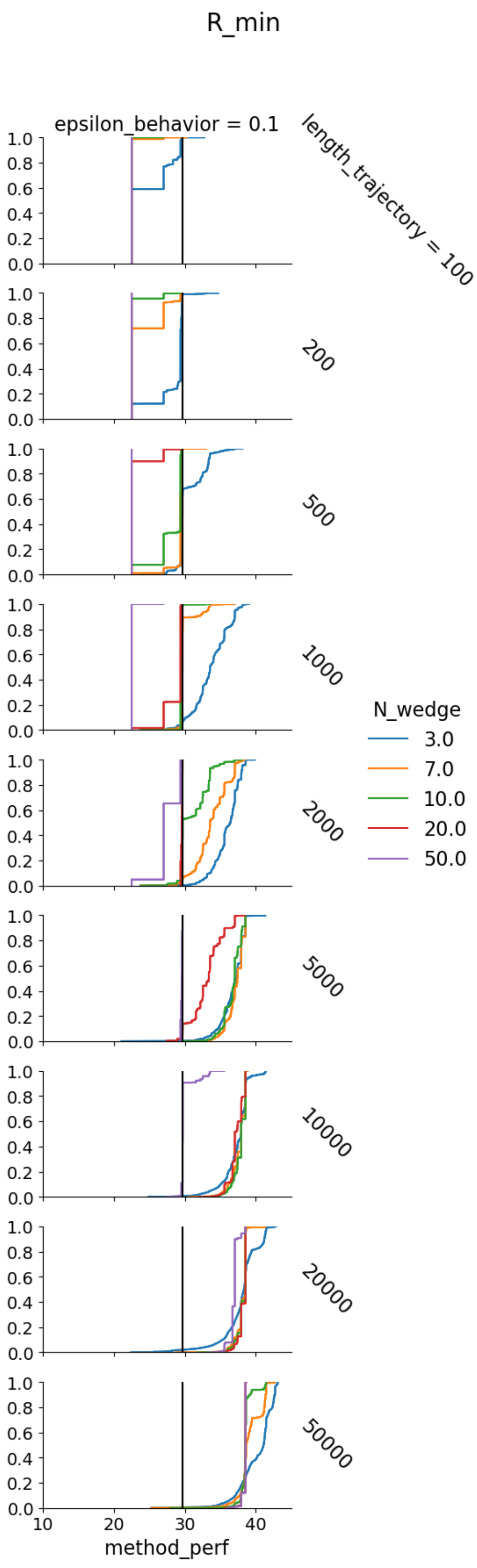


(j) *Lower-Approx-Soft-SPIBB-MPeB*

Fig. 23. *ECDF for different hyper-parameter values for various algorithms on the Wet Chicken benchmark.*

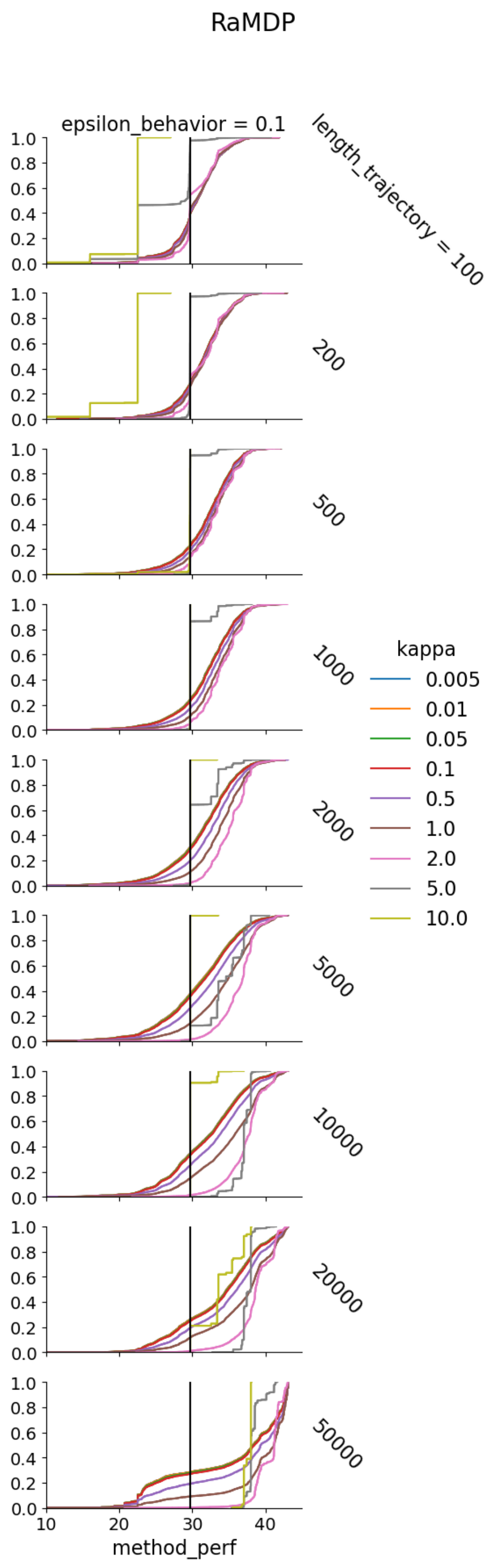


(k) *DUIPI*



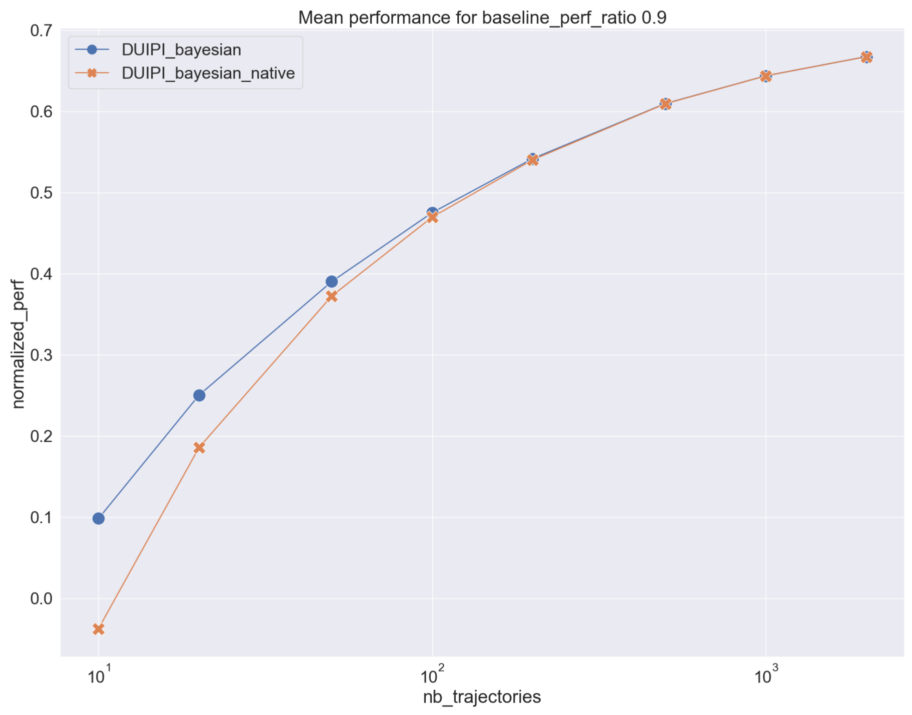
(l) *R-MIN*

Fig. 23. *ECDF for different hyper-parameter values for various algorithms on the Wet Chicken benchmark.*

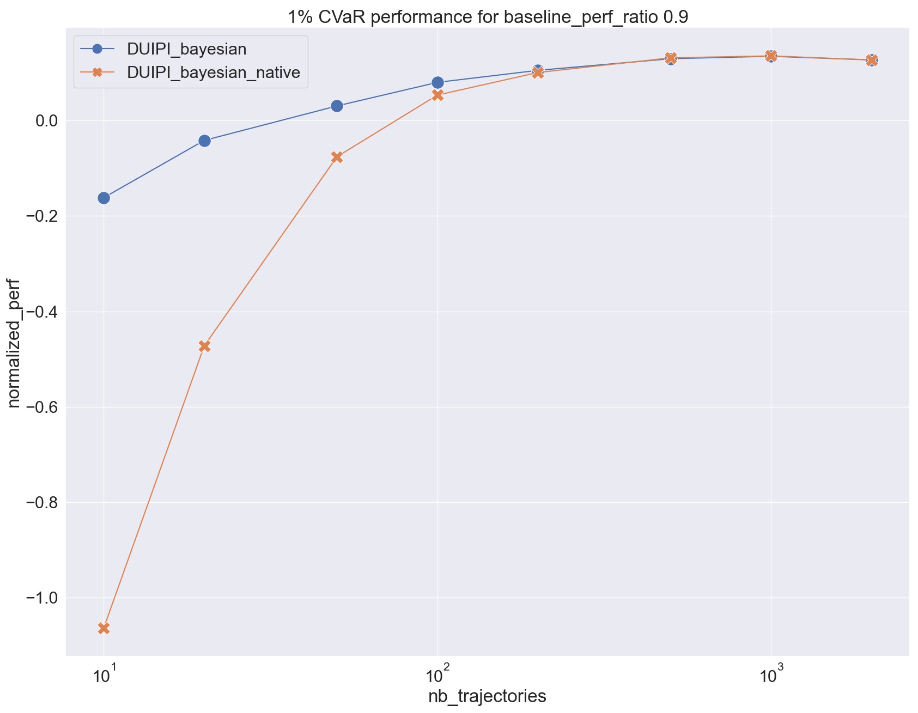


(m) *RaMDP*

Fig. 23. *ECDF for different hyper-parameter values for various algorithms on the Wet Chicken benchmark.*

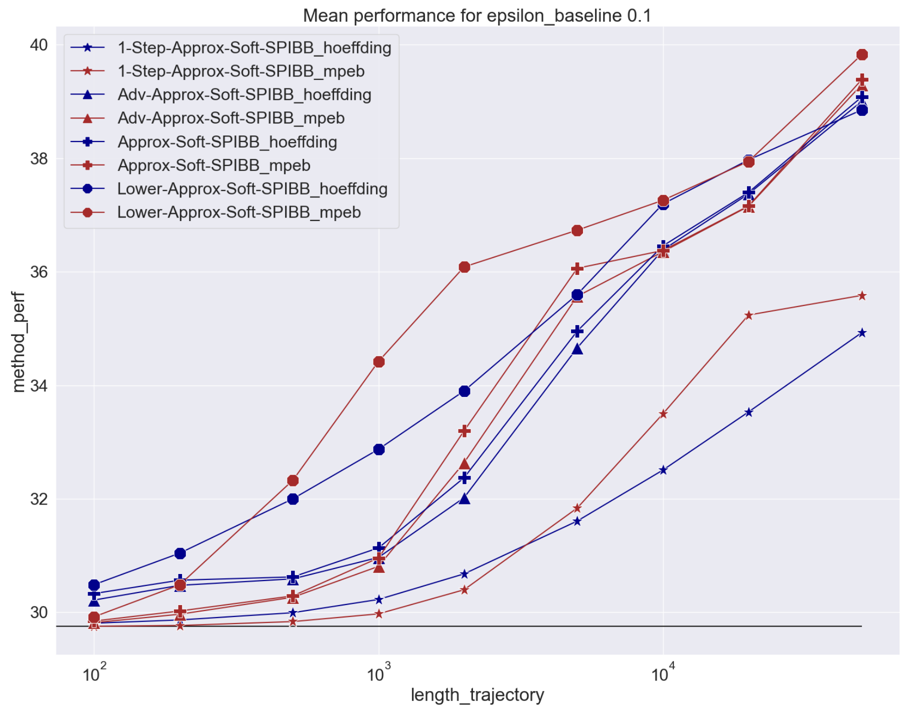


(a) *Mean*

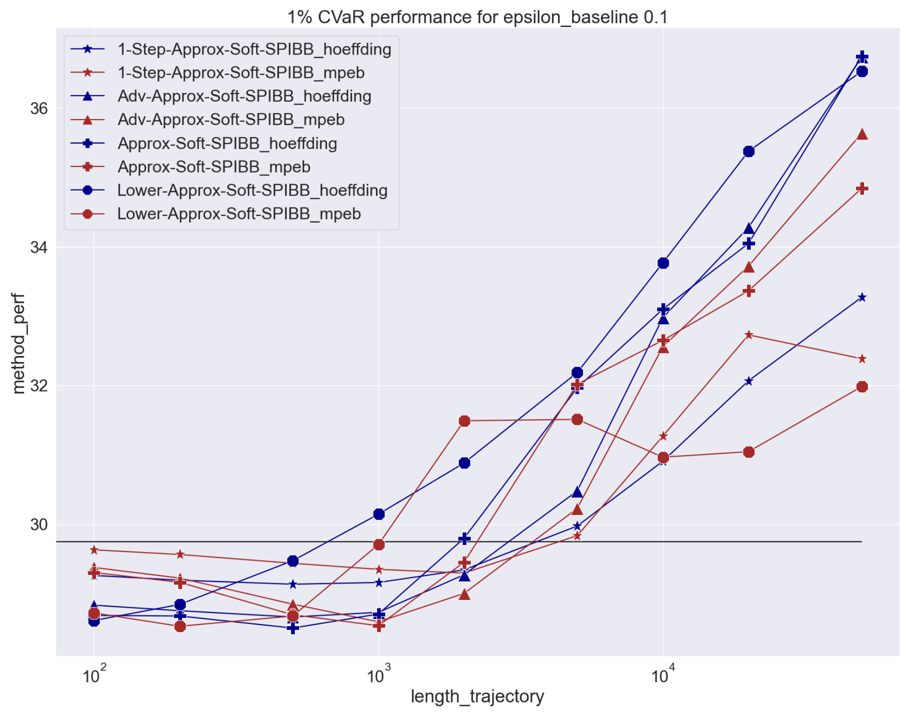


(b) *1%-CVaR*

Fig. 24. *Mean (a) and 1%-CVaR (b) performance over 10,000 trials on the Random MDPs benchmark for the target performance ratio of 0.9 of the behavior policy. Comparison between the original version of DUIPI (DUIPI_bayesian_native) and my adaption which avoids none visited state-action pairs (DUIPI_bayesian).*

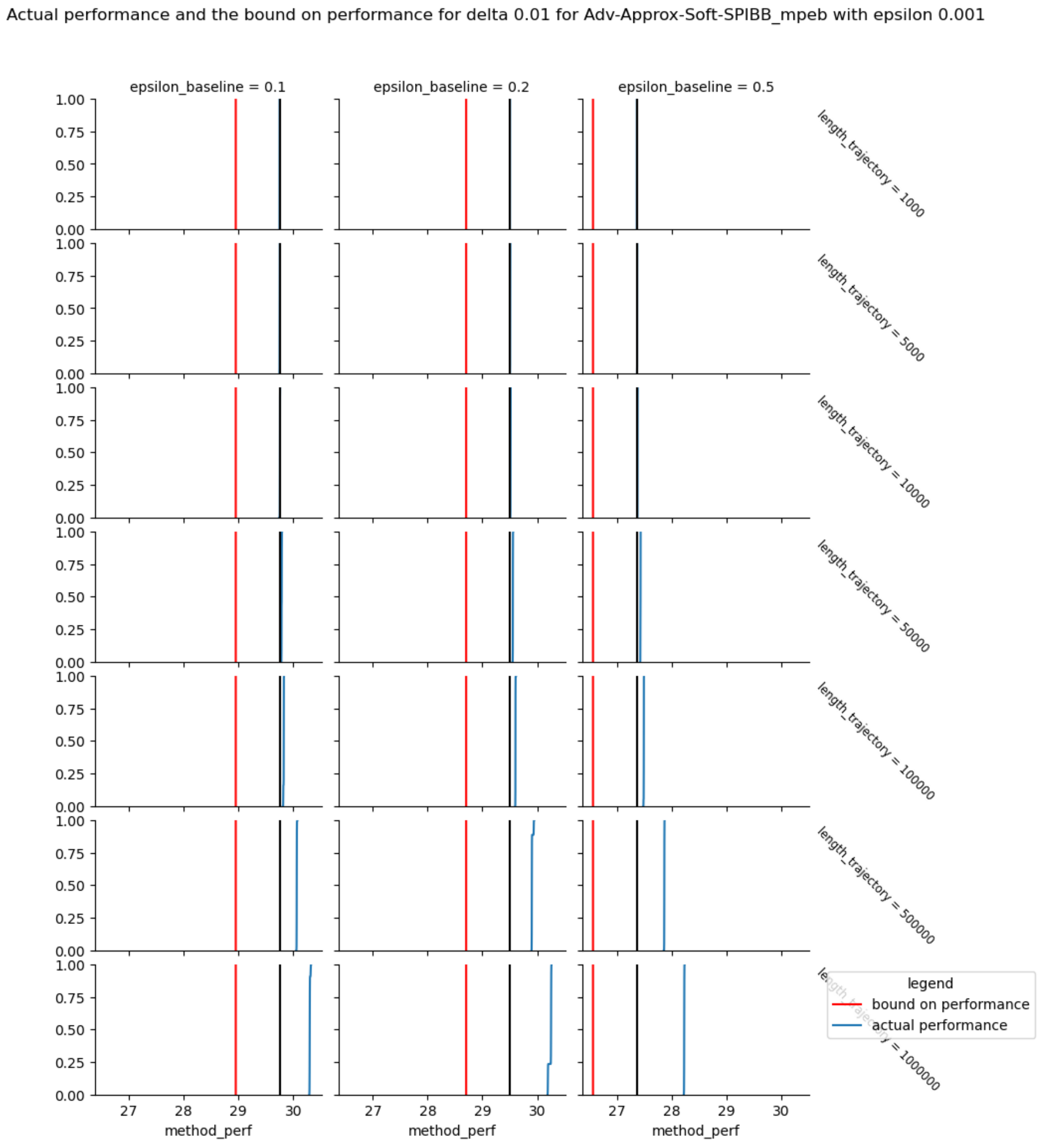


(a) *Mean*



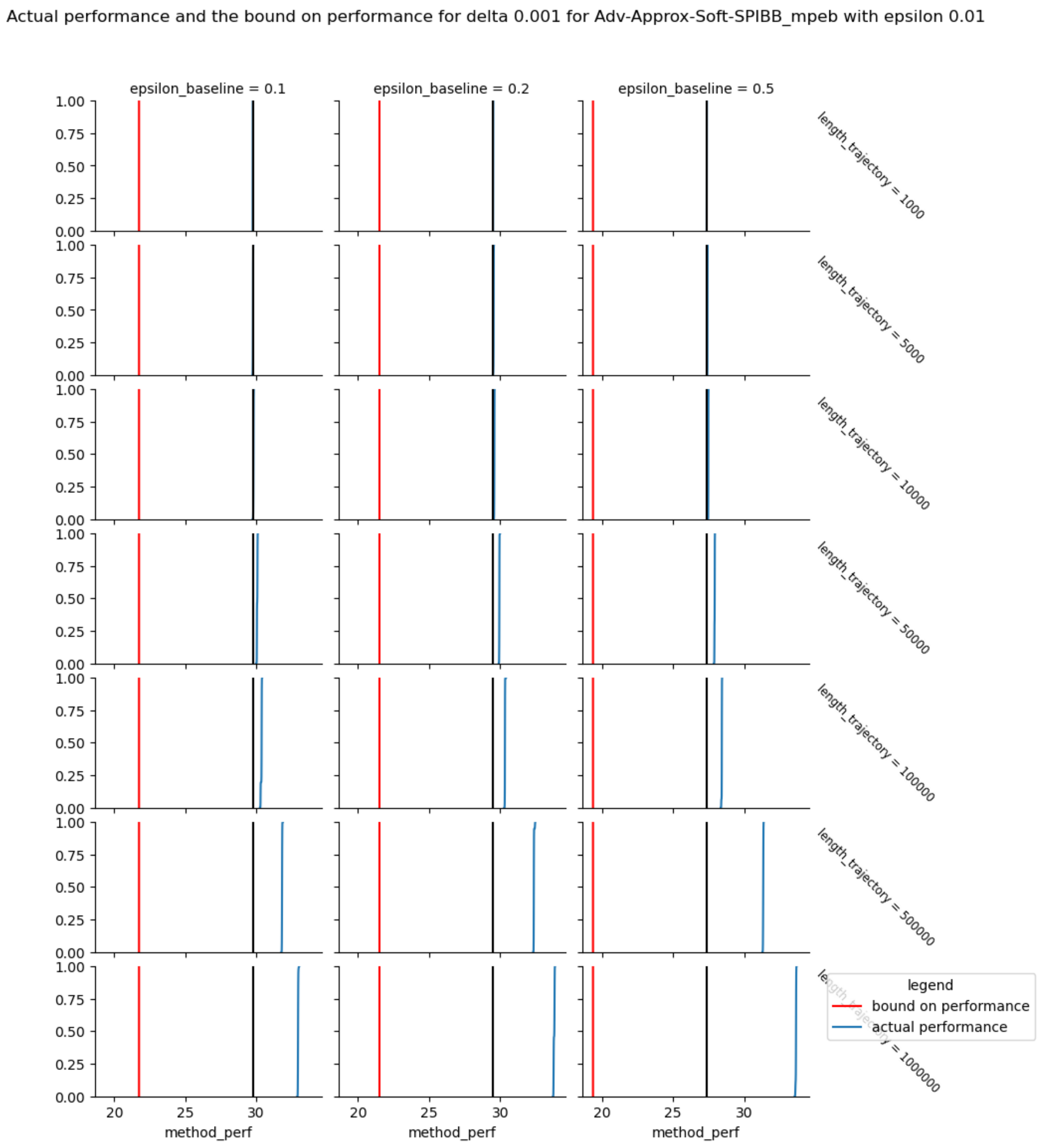
(b) *1%-CVaR*

Fig. 25. *Mean (a) and 1%-CVaR (b) performance on the Wet Chicken benchmark for a 0.1-greedy behavior policy. Comparison between the Soft-SPIBB algorithms using an error function relying on Hoeffding's bound (blue) vs. Soft-SPIBB algorithms using an error function relying on the Maurer and Pontil bound (brown).*



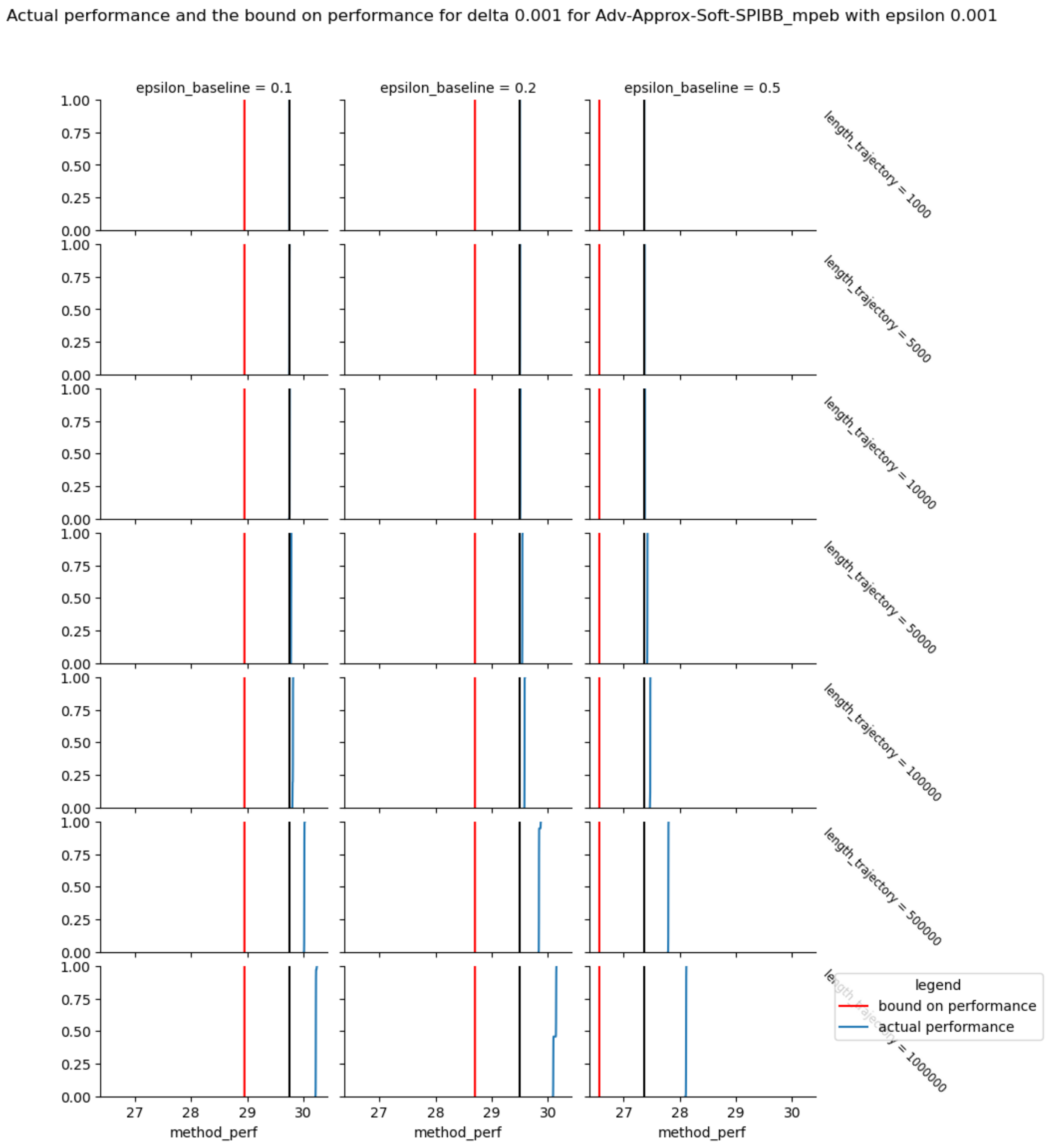
(a) $\delta = 0.01$, $\epsilon = 0.001$

Fig. 26. *ECDF plots of the performance of Adv-Approx-Soft-SPIBB-MPeB and the lower bound on its performance on the Wet Chicken benchmark with various* ϵ *and* δ*. The approximate independence of the returns was ensured by applying Algorithm 3 with* $\kappa = 0.01$*.*



(b) $\delta = 0.001$, $\epsilon = 0.01$

Fig. 26. *ECDF plots of the performance of Adv-Approx-Soft-SPIBB-MPeB and the lower bound on its performance on the Wet Chicken benchmark with various ϵ and δ. The approximate independence of the returns was ensured by applying Algorithm 3 with $\kappa = 0.01$.*



(c) $\delta = 0.001$, $\epsilon = 0.001$

Fig. 26. *ECDF plots of the performance of Adv-Approx-Soft-SPIBB-MPeB and the lower bound on its performance on the Wet Chicken benchmark with various* ϵ *and* δ*. The approximate independence of the returns was ensured by applying Algorithm 3 with* $\kappa = 0.01$*.*

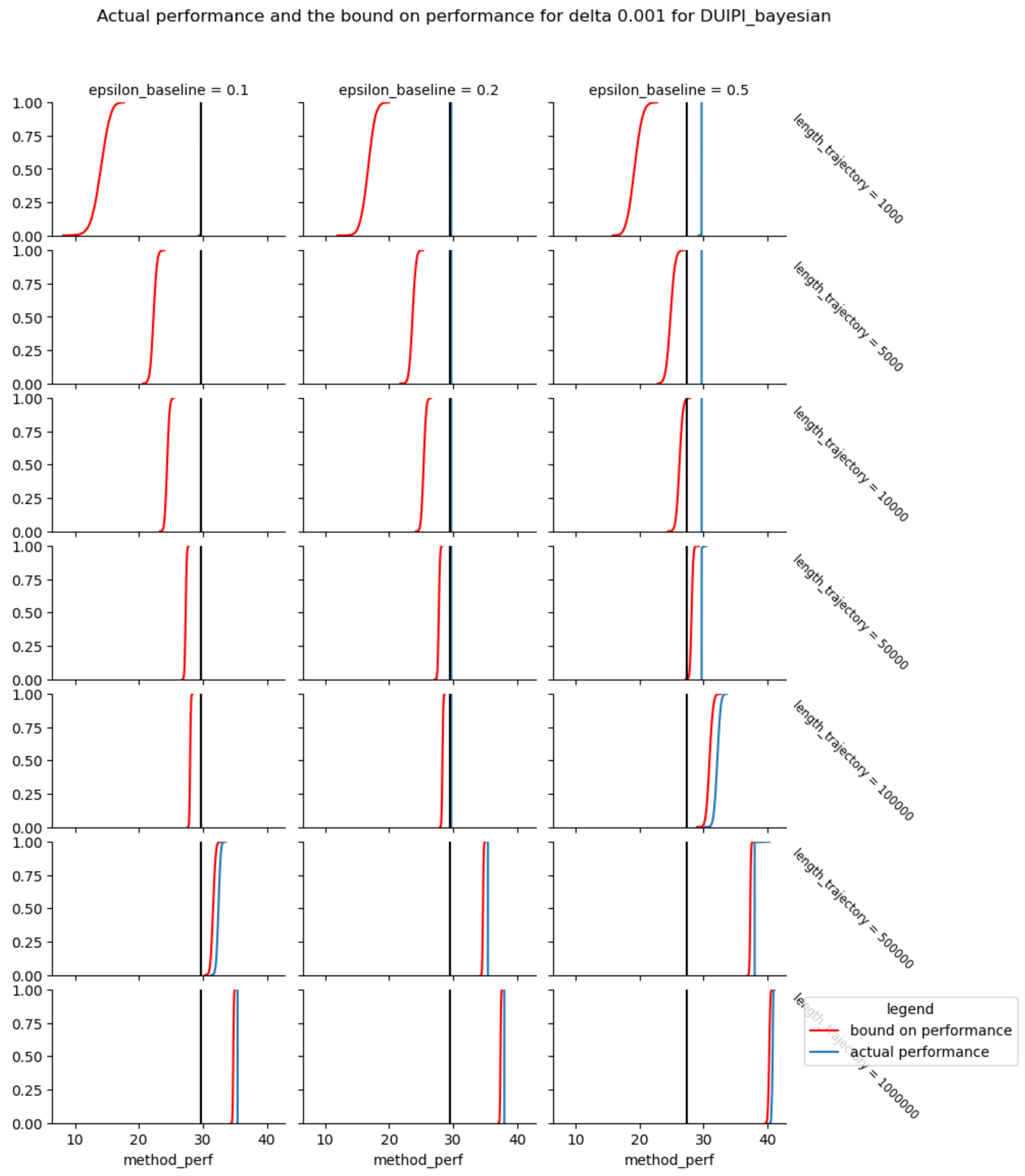


Fig. 27. *ECDF plot of the performance of DUIPI and the lower bound on its performance on the Wet Chicken benchmark with ξ such that the lower bound holds with probability 0.999.*

References

[1] M. Sonka, V. Hlavac, R. Boyle. *Image Processing, Analysis, and Machine Vision.* Fourth Edition. Cengage Learning. 2015.

[2] L. Deng, X. Li. *Machine Learning Paradigms for Speech Recognition: An Overview.* IEEE Transactions on Audio, Speech, and Language Processing. 2013.

[3] S. Benjamens, P. Dhunnoo, B. Mesko. *The state of artificial intelligence-based FDA-approved medical devices and algorithms: an online database.* NPJ Digital Medicine. 2020.

[4] H.-Z. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, R. Anil, Z. Haque, L. Hong, V. Jain, X. Liu, H. Shah. *Wide and Deep Learning for Recommender Systems.* Association for Computing Machinery. New York. 2016.

[5] S. Shalev-Schwartz, S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms.* Cambridge University Press. USA. 2014.

[6] Y. Li. *Reinforcement Learning Applications.* Computing Research Repository. 2019.

[7] Alexander Hans and Steffen Udluft. *Efficient Uncertainty Propagation for Reinforcement Learning with Limited Data.* Proceeding International Conference on Artificial Neural Networks, 70–79. Springer. Cyprus. 2009.

[8] R. S. Sutton, A. G. Barto. *Reinforcement Learning - An Introduction.* Second Edition. The MIT Press. Massachusetts. 2018.

[9] A. M. Schaefer, D. Schneegass, V. Sterzing and S. Udluft. *A Neural Reinforcement Learning Approach to Gas Turbine Control.* Proceedings of International Joint Conference on Neural Networks. Orlando, Florida, USA. 2007.

[10] W. Hauptmann, A. Hentschel, C. Otte, V. Sterzing, M. Tokic, S. Udluft, H.-G. Zimmermann. *ALICE: Autonomes Lernen in komplexen Umgebungen.* Siemens AG. Munich, Germany. 2015.

[11] K. Nadjahi, R. Laroche, R. Tachet des Combes. *Safe Policy Improvement with Soft Baseline Bootstrapping.* Proceedings of the 2019 European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD). 2019.

[12] R. Laroche, P. Trichelair, R. Tachet des Combes. *Safe Policy Improvement with Baseline Bootstrapping.* Proceedings of the 36th International Conference on Machine Learning (ICML). 2019.

[13] S. Lange, T. Gabel, M. Riedmiller. *Batch Reinforcement Learning.* In: M. Wiering, M. van Otterlo (eds) Reinforcement Learning. Adaptation, Learning, and Optimization, vol 12. Springer, Berlin, Heidelberg. 2012.

[14] J. Garcia, F. Fernandez. *A Comprehensive Survey on Safe Reinforcement Learning.* Journal of Machine Learning Research. Spain. 2015.

[15] W. Hoeffding. *Probability inequalities for sums of bounded random variables.* Journal of the American Statistical Association. 1963.

[16] E. D. Andersen, K. D. Andersen. *The MOSEK interior point optimizer for linear programming: an implementation of the homogeneous algorithm.* High performance optimization. Springer US, 2000. 197-232.

[17] P. S. Thomas. *Safe reinforcement learning.* PhD thesis, Stanford university. 2015.

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

[19] G. N. Iyengar. *Robust dynamic programming.* Mathematics of Operations Research. 2005.

[20] I. Goodfellow, Y. Bengio, A. Courville. *Deep Learning.* MIT Press. 2016.

[21] P. J. Huber. *Robust Estimation of a Location Parameter.* The Annals of Mathematical Statistics. 1964.

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

[23] H. v. Hasselt, A. Guez, D. Silver. *Deep Reinforcement Learning with Double Q-learning.* AAAI. 2016.

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

[25] H. Amann, J. Escher. *Analysis III.* Springer. 2001.

[26] T. Popoviciu. *Sur les équations algébriques ayant toutes leurs racines réelles.* Mathematica (Cluj). 1935.

[27] A. Maurer, M. Pontil. *Empirical Bernstein Bounds and Sample Variance Penalization.* 2009.

[28] R. I. Brafman, M. Tennenholtz. *$R-MAX$ - A Genearl Polynomial Time Algorithm for Near-Optimal Reinforcement Learning.* Journal of Machine Learning Research 3, 213-231. 2002.

[29] D. Schnegass, A. Hans, S. Udluft. *Uncertainty in Reinforcement Learning - Awareness, Quantisation, and Control.* 2010.

[30] D. Shafer, Z. Thang. *Introductory Statistics.* 2012.

[31] B. Efron. *Bootstrap Methods: Another Look at the Jackknife.* The Annals of Statistics. 1979.

[32] X. Glorot, A. Bordes, Y. Bengio. *Deep sparse rectifier neural networks.* Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics. 2011.

[33] K. He, X. Zhang, S. Ren, J. Sun. *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification.* IEEE International Conference on Computer Vision. 2015.

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

[35] S. Fujimoto, E. Conti, M. Ghavamzadeh, J. Pineau. *Benchmarking Batch Deep Reinforcement Learning Algorithms.* https://arxiv.org/abs/1910.01708. 2019.

[36] C. Pelekis, J. Ramon. *Hoeffding's inequality for sums of weakly dependent random variables.* Mediterranean Journal of Mathematics. 2017.