Long-Term Visitation Value for Deep Exploration in Sparse Reward Reinforcement Learning

S. Parisi[*1], D. Tateo[1], M. Hensel[1], C. D'Eramo[1], J. Peters[1,2], and J. Pajarinen[1,3]

[1] *Technische Universität Darmstadt, Hochschulstr. 10, 64289 Darmstadt, Germany*
[2] *Max Planck Institute for Intelligent Systems, Spemannstr. 41, 72076 Tübingen, Germany*
[3] *Tampere University, Korkeakoulunkatu 10, 33720 Tampere, Finland*

Preprint (under review)

Abstract

Reinforcement learning with sparse rewards is still an open challenge. Classic methods rely on getting feedback via extrinsic rewards to train the agent, and in situations where this occurs very rarely the agent learns slowly or cannot learn at all. Similarly, if the agent receives also rewards that create suboptimal modes of the objective function, it will likely prematurely stop exploring. More recent methods add auxiliary intrinsic rewards to encourage exploration. However, auxiliary rewards lead to a non-stationary target for the Q-function. In this paper, we present a novel approach that (1) plans exploration actions far into the future by using a long-term visitation count, and (2) decouples exploration and exploitation by learning a separate function assessing the exploration value of the actions. Contrary to existing methods which use models of reward and dynamics, our approach is off-policy and model-free. We further propose new tabular environments for benchmarking exploration in reinforcement learning. Empirical results on classic and novel benchmarks show that the proposed approach outperforms existing methods in environments with sparse rewards, especially in the presence of rewards that create suboptimal modes of the objective function. Results also suggest that our approach scales gracefully with the size of the environment. Source code is available at `https://github.com/sparisi/visit-value-explore`

Keywords: reinforcement learning, sparse reward, exploration, upper confidence bound, off-policy

1 Introduction

Reinforcement learning (RL) is a process where an agent learns how to behave in an environment by trial and error. The agent performs actions and, in turn, the environment may provide a *reward*, i.e., a feedback assessing the quality of the action. The goal of RL is then to learn a *policy*, i.e., a function producing a sequence of actions yielding the maximum cumulative reward. Despite its simplicity, RL achieved impressive results, such as learning to play Atari videogames from pixels [Mnih et al., 2013, Schulman et al., 2017], or beating world-class champions at Chess, Go and Shogi [Silver et al., 2017]. However, a high-quality reward signal is typically necessary to learn the optimal policy, and without it RL algorithms may perform poorly even in small environments [Osband et al., 2019].

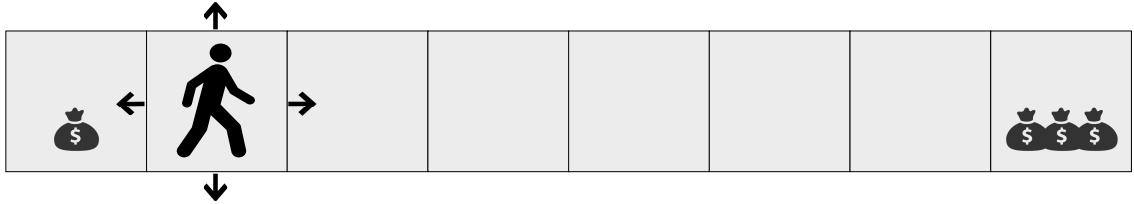[*]Corresponding author: `simone@robot-learning.de`

Figure 1: A simple problem where naive exploration performs poorly. The agent starts in the second leftmost cell and is rewarded only for finding a treasure. It can move up / down / left / right, and if it tries to move out of the chain or if it finds a treasure its position resets. Selecting the action randomly at each step takes on average 4^N attempts to find the rightmost and most valuable reward, where N is the number of cells to the right of the agent. As no reward is given in any intermediate cell, ϵ-greedy exploration is likely to converge to the local optimum represented by the leftmost reward if ϵ decays too quickly.

Characteristics of high-quality rewards. One important factor that defines the quality of the reward signal is the *frequency* at which rewards are emitted. Frequently emitted rewards are called "dense", in contrast to infrequent emissions which are called "sparse". Since improving the policy relies on getting feedback via rewards, the policy cannot be improved until a reward is obtained. In situations where this occurs very rarely, the agent learns slowly or cannot learn at all. Furthermore, reinforcement signals should encourage the agent to find the best actions to solve the given task. However, the environment may also provide *distracting* rewards, i.e., rewards that create suboptimal modes of the objective function. In this case, the agent should be able to extensively explore the environment without prematurely converge to locally optimal solutions caused by distracting rewards.

Reward engineering. In the presence of sparse distracting rewards, efficiently exploring the environment to learn the optimal policy is challenging. Therefore, many RL algorithms rely on well-shaped reward functions such as quadratic costs. For example, in a collect-and-deliver task, we could design an additional reward function that rewards proximity to the items to be collected. These well-shaped functions help to guide the agent towards good solutions and avoid bad local optima. However, from the perspective of autonomous learning, this so-called reward engineering is unacceptable for three reasons. First, the commonly utilized reward functions heavily restrict the solution space and may prevent the agent from learning optimal behavior (especially if no solution to the task is known). Second, it is easy to misspecify the reward function and cause unexpected behavior. For instance, by getting excessively high rewards for proximity to the items the agent may never learn to deliver them. Third, reward engineering requires manual tuning of the reward function and the manual addition of constraints. This process requires expertise and the resulting reward function and constraints may not transfer to different tasks or environments.

In this paper, we address the problem of reinforcement learning with sparse rewards without relying on reward engineering, or requiring prior knowledge about the environment or the task to solve. In particular, we highlight the problem of learning in the presence of rewards that create suboptimal modes of the objective function, which we call "distractors".

The exploration-exploitation dilemma. Learning when rewards are sparse and possibly distracting challenges classic RL algorithms. Initially, the agent has no knowledge about the environment and needs to explore it in search for feedback. As it gathers rewards, the agent develops a better understanding of the environment and can estimate which actions are "good", i.e., yield positive feedback, and which are not. Then, at any point, it needs to decide whether it should explore the environment in search for better rewards by executing new actions, or exploit its current knowledge and execute known good actions. This "exploration-exploitation dilemma" is frequently addressed naively by dithering [Mnih et al., 2013, Lillicrap et al., 2016]. In continuous action spaces Gaussian noise is added to the action, while in discrete spaces actions are chosen ϵ-greedily, i.e., optimally with probability $1-\epsilon$ and randomly with probability ϵ. Typically, the initial noise and ϵ are large and then decay over time. These approaches work in environments

where random sequences of actions are likely to cause positive rewards. However, if rewards are sparse or are given only for specific sequences of actions, it is very unlikely that the agent will receive any feedback. In this case, the worst-case sample complexity for learning the optimal policy with dithering exploration is exponential in the number of states and actions [Kakade, 2003, Szepesvari, 2010, Osband et al., 2016b]. Furthermore, if distractor rewards are easily reachable, the agent may prematurely converge to local optima. An example of this problem is depicted in Figure 1.

Deep exploration. Well-performing exploration strategies should be able to solve long-horizon problems with sparse and possibly distracting rewards in large state-action spaces while remaining computationally tractable. This can be achieved if the agent takes several coherent actions to explore unknown states instead of just locally choosing the most promising actions. These behaviors are usually referred as "deep exploration" and "myopic exploration", respectively [Osband et al., 2016a]. RL literature has a long history of algorithms for deep exploration, even though the term "deep" became popular only in recent years. The first algorithms to guarantee full exploration of tabular environments date back to Kearns and Singh [2002] and Brafman and Tennenholtz [2002]. These algorithms learn a model of the environment, i.e., the transition and reward functions, and plan actions accordingly to a state-action visitation count in order to favor less visited states. Since then, other model-based [Jaksch et al., 2010, Hester and Stone, 2013] as well as model-free algorithms [Strehl et al., 2006, Bellemare et al., 2016, Jin et al., 2018, Dong et al., 2019] have been proposed. However, despite their strong guarantees, these algorithms either require to learn the complete model of the environment, rely on optimistic initialization, or have impractically high sample complexity.

In continuous environments, intrinsic motivation and bootstrapping are the most prominent approaches. The former defines an additional *intrinsic* reward added to the environment *extrinsic* reward. If the extrinsic reward is sparse, the intrinsic reward can fill the gaps between the sparse signals, possibly giving the agent quality feedback at every timestep. This approach can be used in conjunction with any RL algorithm by just providing the modified reward signal. However, the quality of exploration strongly depends on the intrinsic reward, which may not scale gracefully with the extrinsic reward and therefore needs hand-tuning. Furthermore, combining exploration and exploitation by summing intrinsic and extrinsic reward can result in undesired behavior due to the non-stationarity of the augmented reward function. Moreover, convergence is harder to guarantee [Kolter and Ng, 2009], and in general, there is no agreement on the definition of the best intrinsic reward [Houthooft et al., 2016, Pathak et al., 2017]. Bootstrapping, instead, is used to estimate the actions value posterior distribution over the environment, which then drives exploration. Because actions are selected with regard to the level of uncertainty associated with their value estimates, bootstrapping incentivizes experimentation with actions of highly uncertain value and, thus, induces exploration [Osband et al., 2019]. However, these methods rely on approximated posteriors and usually lack guarantees of convergence, unless either the environment model is learned or a time-dependent policy is used [Osband et al., 2019].

Deep exploration via long-term visitation value. *In this paper, we present a novel approach that (1) plans exploration actions far into the future by using a long-term visitation count, and (2) decouples exploration and exploitation by learning a separate function assessing the exploration value of the actions.* Contrary to existing methods which use models of reward and dynamics, our approach is off-policy and model-free.

We further comprehensively benchmark our approach against existing algorithms on several environments, stressing the challenges of learning with sparse and distracting rewards. Empirical results show that the proposed approach outperforms existing algorithms, and suggest that it scales gracefully with the size of the environment.

2 Preliminaries

We start with a description of the RL framework, providing the reader with the notation used in the remainder of the paper. Subsequently, we review the most prominent exploration strategies in literature, discuss their shortcomings, and identify open challenges.

2.1 Reinforcement Learning and Markov Decision Processes

We consider RL in an environment governed by a Markov Decision Process (MDP). An MDP is described by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \mu_1 \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}(s'|s,a)$ defines a Markovian transition probability density between the current s and the next state s' under action a, $\mathcal{R}(s,a)$ is the reward function, and μ_1 is initial distribution for state s_1. The state and action spaces could be finite, i.e., $|\mathcal{S}| = d_s$ and $|\mathcal{A}| = d_A$, or continuous, i.e., $\mathcal{S} \subseteq \mathbb{R}^{d_s}$ and $\mathcal{A} \subseteq \mathbb{R}^{d_A}$. The former case is often called *tabular* MDP, as a table can be used to store and query all state-action pairs if the size of the state-action space is not too large. If the model of the environment is known, these MDPs can be solved exactly with dynamic programming. The latter, instead, requires function approximation and algorithms typically guarantee convergence to local optima. In the remainder of the paper, we will consider only tabular MDPs, and will consider only model-free RL, i.e., the model of the environment is neither known nor learned.

Given an MDP, the goal of RL is to learn to act. Formally, we want to find a *policy* $\pi(a|s)$ to take an appropriate action a when the agent is in state s. By following such a policy starting at initial state s_1, we obtain a sequence of states, actions and rewards $(s_t, a_t, r_t)_{t=1...H}$, where $r_t = \mathcal{R}(s_t, a_t)$ is the reward at time t, and H is the total timesteps also called *horizon*, which can possibly be infinite. We refer to such sequences as *trajectories* or *episodes*.

Our goal is to find a policy that maximizes the expected return

$$Q^*(s,a) := \max_\pi \ \mathbb{E}_{\mu_\pi(s)\pi(a|s)}[Q^\pi(s,a)], \tag{1}$$

$$\text{where} \ \ Q^\pi(s_t, a_t) := \mathbb{E}_{\prod_{i=t}^{H} \pi(a_{i+1}|s_{i+1})\mathcal{P}(s_{i+1}|s_i,a_i)}\left[\sum_{i=t}^{H} \gamma^{i-t} r_i\right], \tag{2}$$

where $\mu_\pi(s)$ is the (discounted) state distribution under π, i.e., the probability of visiting state s under π, and $\gamma \in [0,1)$ is the discount factor which assigns weights to rewards at different timesteps. $Q^\pi(s,a)$ is the action-state value function (or Q-function) which is the expected return obtained by executing a in state s and then following π afterwards. In tabular MDPs, the Q-function can be stored in a table, commonly referred as Q-table, and in stationary infinite-horizon MDPs the greedy policy $\pi^*(a|s) = \arg\max_a Q^*(s,a)$ is always optimal [Puterman, 1994]. However, the Q-function of the current policy is initially unknown and has to be learned.

Q-learning. For tabular MDPs, Q-learning by Watkins and Dayan [1992] was one of the most important breakthroughs in RL, and it is still at the core of recent successfull algorithms. At each training step i, the agent chooses an action according to a *behavior* policy $\beta(a|s)$, i.e., $a_t \sim \beta(\cdot|s_t)$, and then updates the Q-table according to

$$Q_{i+1}^\pi(s_t, a_t) = Q_i^\pi(s_t, a_t) + \eta\delta(s_t, a_t, s_{t+1}) \tag{3}$$

$$\delta(s_t, a_t, s_{t+1}) = \begin{cases} r_t + \gamma \max_a Q_i^\pi(s_{t+1}, a) - Q_i^\pi(s_t, a_t) & \text{if } s_t \text{ is non-terminal} \\ r_t - Q_i^\pi(s_t, a_t) & \text{otherwise} \end{cases} \tag{4}$$

where η is the learning rate and $\delta(s,a,s')$ is the *temporal difference (TD) error*. The blue term is often referred to as *TD target*. This approach is called *off-policy* because the policy $\beta(s,a)$ used to explore the environment is different from the target greedy policy $\pi(a|s)$.

The behavior policy and the "exploration-exploitation dilemma". As discussed in Section 1, an important challenge in designing the behavior policy is to account the exploration-exploitation dilemma. *Exploration* is a long-term endeavour where the agent tries to maximize the possibility of finding better rewards in states not yet visited. On the contrary, *exploitation* maximizes the expected rewards according to the current knowledge of the environment. Typically, in continuous action spaces Gaussian noise is added to the action, while in discrete spaces actions are chosen ϵ-greedily, i.e., optimally with probability $1-\epsilon$ and randomly with probability ϵ. In this case, the exploration-exploitation dilemma is simply addressed by having larger noise and ϵ at the beginning, and then decaying them over time. This naive dithering exploration is

extremely inefficient in MDPs with sparse rewards, especially if some of them are "distractors", i.e., rewards that create suboptimal modes of the objective function. In the the remainder of this section, we review more sophisticatd approaches addressing exploration with sparse rewards. We first show theoretically grounded ideas for tabular or small MDP settings which are generally computationally intractable for large MDPs. We then give an overview of methods which try to apply similar principles to large domains by making approximations.

2.2 Related Work

Since finding high value states is crucial for successful RL, a large body of work has been devoted to efficient exploration in the past years. We begin by discussing an optimal solution to exploration, then continue with confidence-bound-based approaches. Then, we discuss methods based on intrinsic motivation, and describe posterior optimality value-distribution-based methods. We conclude by stating the main differences between our approach and these methods.

Optimal solution to exploration. In this paper, we consider model-free RL, i.e., the agent does not know the MDP dynamics. It is well-known that in model-free RL the optimal solution to exploration can be found by a Bayesian approach. First, assign an initial (possibly uninformative) prior distribution over possible unknown MDPs. Then, at each time step the posterior belief distribution over possible MDPs can be computed using the Bayes' theorem based on the current prior distribution, executed action and made observation. The action that optimally explores is then found by planning over possible future posterior distributions, e.g., using tree search, sufficiently far forward. Unfortunately, optimal exploration is intractable even for very small tasks [Szepesvari, 2010, Strens, 2000, Poupart et al., 2006]. The worst-case computational effort is exponential with respect to the planning horizon in tabular MDPs, and can be even more challenging with continuous state-action spaces.

Optimism. In tabular MDPs, many of the provably efficient algorithms are based on optimism in the face of uncertainty (OFU) [Lai and Robbins, 1985]. In OFU, the agent acts greedily with respect to an optimistic action value estimate composed of the value estimate and a bonus term that is proportional to the uncertainty of the value estimate. After executing an optimistic action, the agent then either experiences a high reward learning that the value of the action was indeed high, or the agent experiences a low reward and learns that the action was not optimal. After visiting a state-action pair, the exploration bonus is reduced. This approach is superior to naive approaches in that it avoids actions where low value and low information gain are possible. Under the assumption that the agent can visit every state-action pair infinitely many times, the overestimation will decrease and almost optimal behavior can be obtained [Kakade, 2003]. Most algorithms are optimal up to a polynomial amount of states, actions, or horizon length. RL literature provides many variations of these algorithms which use bounds with varying efficacy or different simplifying assumptions, such as [Kearns and Singh, 2002, Brafman and Tennenholtz, 2002, Kakade, 2003, Auer and Ortner, 2007, Jaksch et al., 2010, Dann and Brunskill, 2015].

Upper confidence bound. Perhaps the best well-known OFU algorithm is the upper confidence bound (UCB) algorithm [Auer et al., 2002]. UCB chooses actions based on a bound computed from visitation counts: the lower the count, the higher the bonus. Recently, Jin et al. [2018] proved that episodic Q-learning with UCB exploration converges to a regret proportional to the square root of states, actions, and timesteps, and square root of the cube of the episode length. Dong et al. [2019] gave a similar proof for infinite horizon MDPs with sample complexity proportional to the number of states and actions. In our experiments, we compare our approach to Q-learning with UCB exploration as defined by Auer et al. [2002].

Intrinsic motivation. The previously discussed algorithms are often computationally intractable, or their guarantees no longer apply in continuous state-action spaces, or when the state-action spaces are too large. Nevertheless, these provably efficient algorithms provided inspiration for more practical algorithms. Commonly, exploration is conducted by adding a bonus reward, also called auxiliary reward, to interesting states. This is often referred to as intrinsic motivation [Ryan and Deci, 2000]. For example, the bonus can be similar to the UCB [Strehl and Littman, 2008, Bellemare et al., 2016], thus encouraging actions of high uncertainty or of low visitation count.

Other forms of bonus are based on the prediction error of some quantity. For example, the agent may learn a model of the dynamics and try to predict the next state [Stadie et al., 2015, Houthooft et al., 2016, Pathak et al., 2017, Schmidhuber, 1991, 2006]. By giving a bonus proportional to the prediction error, the agent is incentivized to explore unpredictable states. Unfortunately, in the presence of noise, unpredictable states are not necessarily interesting states. Recent work addresses this issue by training an ensemble of models [Pathak et al., 2019] or predicting the outputs of a neural network [Burda et al., 2019].

As we will discuss in Section 3.2, our approach has similarities with the use of auxiliary reward. However, the use of auxiliary rewards can be inefficient for long-horizon exploration. Our approach addresses this issue by using a decoupled long-horizon exploration policy and, as the experiments in Section 4 show, it outperforms auxiliary-reward-based approaches [Strehl and Littman, 2008].

Thompson sampling. Another principled well-known technique for exploration is Thompson sampling [Thompson, 1933]. Thompson sampling samples actions from a posterior distribution which specifies the probability for each action to be optimal. Similarly to UCB-based methods, Thompson sampling is guaranteed to converge to an optimal policy in multi-armed bandit problems [Kaufmann et al., 2012, Agrawal and Goyal, 2013], and has shown strong empirical performance [Scott, 2010, Chapelle and Li, 2011]. For a discussion of known shortcomings of Thompson sampling, we refer to [Russo and Van Roy, 2014, Russo et al., 2017, 2018].
Inspired by these successes, recent approaches have tried to follow the principle of Thompson sampling in Q-learning [Osband et al., 2013, 2019, D'Eramo et al., 2019]. These approaches (implicitly) assume that an empirical distribution over Q-functions – an ensemble of randomized Q-functions – is similar to the distribution over action optimalities used in Thompson sampling. Actions are therefore sampled from such ensemble. Since the Q-functions in the ensemble become similar when updated with new samples, there is a conceptual similarity with the action optimality distribution used in Thompson sampling for which variance also decreases with new samples. While there are no general proofs for the randomized Q-function approaches, Osband et al. [2019] proved a bound on the Bayesian regret of an algorithm based on randomized Q-functions in a tabular time-inhomogeneous MDP with a transition kernel drawn from a Dirichlet prior, providing a starting point for more general proofs.
Other exploration methods approximating a distribution over action optimalities include for example the work of Fortunato et al. [2017] and Plappert et al. [2018], who apply noise to the parameters of the model. This may be interpreted as approximately inferring a posterior distribution [Gal and Ghahramani, 2016], although this is not without contention [Osband, 2016]. Osband et al. [2016a] more directly approximates the posterior over Q-functions through bootstrapping [Efron, 1982]. The lack of a proper prior is an issue when rewards are sparse, as it causes the uncertainty of the posterior to vanish quickly. Osband et al. [2019] and Osband et al. [2018] try to address this by enforcing a prior via regularization or by adding randomly initialized but fixed neural network on top of each ensemble member.
Methods based on posterior distributions have yielded high performance in some tasks. However, because of the strong approximations needed to model posterior optimality distributions instead of maintaining visitation counts or explicit uncertainty estimates, these approaches may often have problems in assessing the state uncertainty correctly. In our experiments, the proposed approach outperforms state-of-the-art methods based on approximate posterior distributions, namely the algorithms proposed by Osband et al. [2016a, 2019] and D'Eramo et al. [2019].

Above, we discussed state-of-the-art exploration methods that (1) use confidence bounds or a distribution over action optimalities to take exploration into account in a principled way, (2) modify the reward by adding an intrinsic reward signal to encourage exploration, and (3) approximate a posterior distribution over value functions. The main challenge that pertains to all these methods is that they do not take long-term exploration into account explicitly. *Contrary to this, similarly to how value iteration propagates state values, we propose an approach that uses dynamic programming to explicitly propagate visitation information backwards in time.* This allows our approach to efficiently find the most promising parts of the state space that have not been visited before, and avoid getting stuck in suboptimal regions that at first appear promising.

3 Long-Term Visitation Value for Deep Exploration

In this section, we present our novel off-policy method for improving exploration 1with sparse and distracting rewards. We start by motivating our approach with a toy example, showing the limitation of current count-based algorithms and the need for "long-term visitation counts". Subsequently, we formally define the proposed method, and show how it solves the toy example.
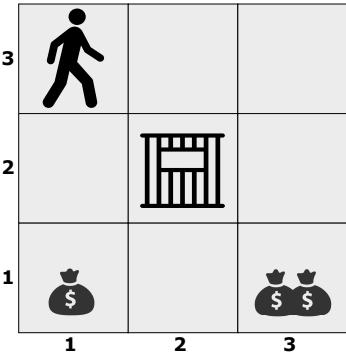
3.1 Example of the Limitations of Existing Immediate Count Methods

Figure 2: Toy problem domain.

Consider the toy problem of an agent moving in a grid, shown in Figure 2. Reward is 0 everywhere except in $(1,1)$ and $(3,1)$, where it is 1 and 2, respectively. The reward is given for executing an action in the state, i.e., on transitions from reward states. The agent initial position is fixed in $(1,1)$ and the episode ends when a reward is collected or after five steps. Any action in the prison cells $(2,2)$ has only an arbitrarily small probability of success. If it fails, the agent stays in $(2,2)$. In practice, the prison cell almost completely prevents any further exploration. Despite its simplicity, this domain highlights two important challenges for exploration in RL with sparse rewards. First, there is a locally optimal policy collecting the lesser reward, which acts as distractor. Second, the prison state is particularly dangerous as it severely hinders exploration.

Assume that the agent has already explored the environment for four episodes performing the following actions: (1) left, down, left, down, down (reward 1 collected); (2) up, down, right (fail), right (fail), right (fail); (3) right, left, right, down, down (fail); (4) right, down, left (fail), up (fail), right (fail). The resulting state-action visitation count is shown in Figure 3a. Given this data, we initialize $Q^\pi(s,a)$ to zero, train it with Q-learning with $\gamma = 0.99$ until convergence, and then derive three greedy policies. The first simply maximizes $Q^\pi(s,a)$. The second maximizes the behavior $Q^\beta(s,a)$ based on UCB1 [Auer et al., 2002], defined as

$$Q^\beta(s,a) = Q^\pi(s,a) + \kappa\sqrt{\frac{2\log\sum_{a_j} n(s_t, a_j)}{n(s_t, a)}}, \tag{5}$$

where $n(s,a)$ is the state-action count, and κ is a scaling coefficient[1]. The third maximizes an augmented Q-function $Q^+(s,a)$ trained by adding the intrinsic reward based on the immediate count [Strehl and Littman, 2008], i.e.,

$$r_t^+ = r_t + \alpha\, n(s_t, a_t)^{-1/2}, \tag{6}$$

where α is a scaling coefficient set to $\alpha = 0.1$ as in [Strehl and Littman, 2008, Bellemare et al., 2016]. Figure 3a shows the state-action count after the above trajectories, and Figures 3b, 3c, and 3d depict the respective Q-functions (colormap) and policies (arrows). Given the state-action count, consider what the agent has done and knows so far.

a) It has executed all actions at least once in (1,3) and (2,2) (the prison cell),
b) It still did not execute "up" in (1,2),
c) It visited (1,3) once and experienced the reward of 1, and
d) It still did not execute "up" and "right" in (2,1).

In terms of exploration, the best decisions would be to select actions not executed yet, such as "up" in (1,2). *But what should the agent do in states where it has already executed all actions at least once? Ideally, it should select actions driving it to unvisited states, or to states where it has not executed all actions yet.* However, none of the policies above exhibits such behavior.

---

[1]In classic UCB, the reward is usually bounded in $[0,1]$ and no coefficient is needed. This is not usually the case for MDPs, where Q-values can be larger/smaller than 1/0. We thus need to scale the square root with $\kappa = (r_{\max} - r_{\min})/(1-\gamma)$, which upper-bounds the difference between the largest and smallest possible Q-value.

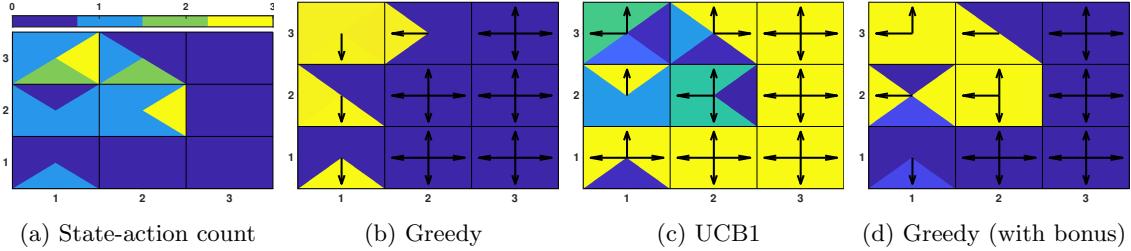|(a) State-action count|(b) Greedy|(c) UCB1|(d) Greedy (with bonus)|

Figure 3: Toy example (part 1). Visitation count (a) and behavior policies (b, c, d) for the toy problem below. Cells are divided into four triangles, each one representing the count (a) or the behavior Q-value (b, c, d) for the actions "up / down / left / right" in that cell. Arrows denote the action with the highest Q-value. None of the behavior policies exhibits long-term deep exploration. In (1,3) both UCB1 (c) and the augmented Q-table (d) myopically select the action with the smallest immediate count. The latter also acts badly in (1,2) and (2,3). There, it selects "left" which has count one, instead of "up" which has count zero.

a) The greedy policy (Figure 3b), as expected, points to the only reward collected so far.

b) The UCB1 policy (Figure 3c) yields Q-values of much larger magnitude. Most of the actions have been executed few times, thus their UCB is larger than their Q-value. The policy correctly selects unexecuted actions in (1,2) and (2,3), but fails (1,3). There it also selects the least executed actions ('up" and "left") which, however, have already been executed once. The agent should know that these actions do not bring it anywhere (the agent hits the environment boundaries and does not move) and should avoid them. However, it selects them because the policy $Q^\beta(s, a)$ is based on the *immediate* visitation count. Only when the count will be equal for all actions, the policy will select a random action. This behavior is clearly myopic and extremely inefficient.

c) The policy learned using the auxiliary bonus (Figure 3d) is even worse, as it acts badly not only in (1,3), but also in (1,2) and (2,3). There, it chooses "left", which was already selected once, instead of "up" (or "right in (2,3)), which instead has not been selected yet. The reason of such behavior is that this auxiliary reward requires optimistic initialization [Strehl and Littman, 2008]. The agent, in fact, is positively rewarded for simply executing any action, with value proportional to the visitation count. However, if the Q-function is initialized to zero, the positive feedback will make the agent believe that the action is good. This mechanism encourages the agent to execute the same action again and hinders exploration. This problem is typical of all methods using an auxiliary reward based on visitation count, unless optimistic initialization of the Q-table is used [Strehl and Littman, 2008]. More recent methods use more sophisticated rewards [Jin et al., 2018, Dong et al., 2019]. However, despite their strong theoretical convergence guarantees, for large size problems these methods may require an impractical number of samples and are often outperformed by standard algorithms.

3.2 The Long-Term Visitation Value Approach

Despite its simplicity, the above toy problem highlights the limitations of current exploration strategies based on the immediate count. More specifically, (1) considering the immediate count may result in shallow exploration, and (2) directly augmenting the reward to train the Q-function, i.e., coupling exploration and exploitation, can have undesired effects due to the non-stationarity of the augmented reward function. To address these limitations, we propose an approach that (1) plans exploration actions far into the future by using a *long-term visitation count*, and (2) decouples exploration and exploitation by learning a separate function assessing the *exploration value* of the actions. Contrary to existing methods which use models of reward and dynamics [Hester and Stone, 2013], which can be hard to come by, our approach is off-policy and model-free.

Visitation value. The key idea of our approach is to train a visitation value function using the temporal difference principle. The function, called W-function and denoted by $W^\beta(s, a)$, approximates the cumulative sum of an intrinsic rewards r^W based on the visitation count, i.e.,

$$W^\beta(s_t, a_t) := \mathbb{E}_{\prod_{i=t} \beta(a_{t+1}|s_{t+1})\mathcal{P}(s_{i+1}|s_i,a_i)} \left[\sum_{i=t}^{H} \gamma_w^{i-t} r_i^W \right], \tag{7}$$

where γ_w is the visitation count discount factor. The higher the discount, the more in the future the visitation count will look. In practice, we estimate $W^\beta(s, a)$ using TD learning on the samples produced using the behavior policy β. Therefore, we use the following update

$$W_{i+1}^\beta(s_t, a_t) = W_i^\beta(s_t, a_t) + \eta \delta^W(s_t, a_t, s_{t+1}), \tag{8}$$

where η is the learning rate, and $\delta^W(s_t, a_t, s_{t+1})$ is the W-function TD error, which depends on the reward r^W (can be either maximized or minimized). Subsequently, following the well-known upper confidence bound (UCB) algorithm [Lai and Robbins, 1985], the behavior policy is greedy with respect to the sum of the Q-function and an upper confidence bound, i.e.,

$$\beta(a|s) = \arg\max_a \left\{ Q^\pi(s, a) + \kappa U_W(s, a) \right\}, \tag{9}$$

where $U_W(s, a)$ is an upper confidence bound based on $W^\beta(s, a)$, and κ is the same exploration constant used by UCB1 in Eq. (5).

This proposed approach (1) considers long-term visitation count by employing the W-function, which is trained to maximize the cumulative – not the immediate – count. Intuitively, the W-function encourages the agent to explore state-actions which have not been visited before. Given the current state s, in fact, $W^\beta(s, a)$ specifies how much exploration we can expect on (discount-weighted) average in future states if we choose action a. Furthermore, since the Q-function is still trained greedily with respect to the extrinsic reward as in Q-learning, while the W-function is trained greedily on the intrinsic reward and favors less visited states, our approach (2) effectively decouples exploration and exploitation. This allows us to more efficiently prevent underestimation of the visitation count exploration term. Below, we propose two versions of this approach based on two different rewards r^W and upper bounds $U_W(s, a)$.

W-function as long-term UCB. The visitation reward is the UCB of the state-action count at the current time, i.e.,

$$r_t^W = \begin{cases} \sqrt{\dfrac{2 \log \sum_{a_j} n(s_t, a_j)}{n(s_t, a_t)}} & \text{if } s_t \text{ is non-terminal} \\ \dfrac{1}{1-\gamma_w} \sqrt{\dfrac{2 \log \sum_{a_j} n(s_t, a_j)}{n(s_t, a_t)}} & \text{otherwise.} \end{cases} \tag{10}$$

Notice how we distinguish between terminal and non-terminal states. As terminal state-action pairs prevent exploration, we need to avoid them after the first visit. Therefore, we assume that for terminal state-action pairs the agent gets the same visitation reward at each time step forever. This W-function represents the discount-weighted average UCB along the trajectory, i.e.,

$$W_{\text{UCB}}^\beta(s_t, a_t) = \sum_{i=t}^{H} \gamma_w^{i-t} \sqrt{\frac{2 \log \sum_{a_j} n(s_i, a_j)}{n(s_i, a_i)}}, \tag{11}$$

and its TD error is

$$\delta^W(s_t, a_t, s_{t+1}) = \begin{cases} r_t^W + \gamma_w \max_a W_{\text{UCB},i}^\beta(s_{t+1}, a) - W_{\text{UCB},i}^\beta(s_t, a_t) & \text{if } s_t \text{ is non-terminal} \\ r_t^W - W_{\text{UCB},i}^\beta(s_t, a_t) & \text{otherwise.} \end{cases} \tag{12}$$

9

This TD error has two important consequences. First, the target of non-terminal states will be higher than the one of terminal states despite the different visitation reward (Eq. (10)), given proper initialization of W_{UCB}^{β}. We will discuss this in more detail later in this section. Second, since we can only update the W-function approximately based on a limited number of samples, it is beneficial to overestimate its target with the max operator. While it is known that in highly stochastic environments the overestimation can degrade the performance of value-based algorithm [van Hasselt, 2010, D'Eramo et al., 2017], it has been shown that underestimation does not perform well when the exploration is challenging [Tateo et al., 2017], and for a wide class of environments, overestimation allows finding the optimal solution due to self-correction [Sutton and Barto, 2018]. Assuming identical visitation over state-action pairs, the W-function becomes the discounted immediate UCB, i.e.,

$$W_{\text{UCB}}^{\beta}(s_t, a_t) = \frac{1}{1 - \gamma_w} \sqrt{\frac{2 \log \sum_{a_j} n(s_t, a_j)}{n(s_t, a_t)}}, \tag{13}$$

thus, we can use $W_{\text{UCB}}^{\beta}(s, a)$ directly in the behavior policy

$$\beta(a_t | s_t) = \arg \max_a \left\{ Q^{\pi}(s_t, a) + \kappa \underbrace{(1 - \gamma_w) W_{\text{UCB}}^{\beta}(s_t, a)}_{U_W^{\text{UCB}}(s, a)} \right\}, \tag{14}$$

In Section 3.3, we discuss how to initialize $W_{\text{UCB}}^{\beta}(s, a)$. Finally, notice that when $\gamma_w = 0$, Eq. (13) is the classic UCB, and Eq. (14) is equivalent to the UCB1 policy in Eq. (5).

W-function as long-term count. This W-function approximates the total count over a path. Its visitation reward and TD error are

$$r_t^W = \begin{cases} n(s_t, a_t) & \text{if } s_t \text{ is non-terminal} \\ \frac{1}{1 - \gamma_w} \max_{(s,a)} n(s, a) & \text{otherwise,} \end{cases} \tag{15}$$

$$\delta^W(s_t, a_t, s_{t+1}) = \begin{cases} r_t^W + \gamma_w \min_a W_{\text{N},i}^{\beta}(s_{t+1}, a) - W_{\text{N},i}^{\beta}(s_t, a_t) & \text{if } s_t \text{ is non-terminal} \\ r_t^W - W_{\text{N},i}^{\beta}(s_t, a_t) & \text{otherwise.} \end{cases} \tag{16}$$

Similarly to the UCB-based reward discussed above, we distinguish between terminal and non-terminal states. Since terminal states prevent any further exploration, we penalize them by using an upper bound of the state-action count as visitation reward. Regarding the TD target, the min operator now replaces the max operator of Eq. (12), because to encourage exploration we need to prioritize low-count state-action pairs. This yields an optimistic estimate of the W-function, because the lower the W-function the more optimistic the exploration is. Assuming identical visitation over state-action pairs, the W-function becomes the discounted cumulative count, i.e.,

$$W_{\text{N}}^{\beta}(s_t, a_t) = \frac{1}{1 - \gamma_w} n(s_t, a_t). \tag{17}$$

We then compute the pseudocount $\hat{n}(s_t, a_t) = (1 - \gamma_w) W_{\text{N}}^{\beta}(s_t, a_t)$ and use it in the UCB policy

$$\beta(a_t | s_t) = \arg \max_a \left\{ Q^{\pi}(s_t, a) + \kappa \underbrace{\sqrt{\frac{2 \log \sum_{a_j} \hat{n}(s_t, a_j)}{\hat{n}(s_t, a)}}}_{U_W^{\text{N}}(s, a)} \right\}. \tag{18}$$

When the counts are zero, $W_{\text{N}}^{\beta}(s, a) = 0$, thus we need to initialize it to zero. We discuss how to avoid numerical problem in the square root in Section 3.3. Finally notice that when $\gamma_w = 0$, Eq. (18) is equivalent to the UCB1 policy in Eq. (5).

(a) State-action count   (b) UCB W-function policy   (c) Count W-function policy
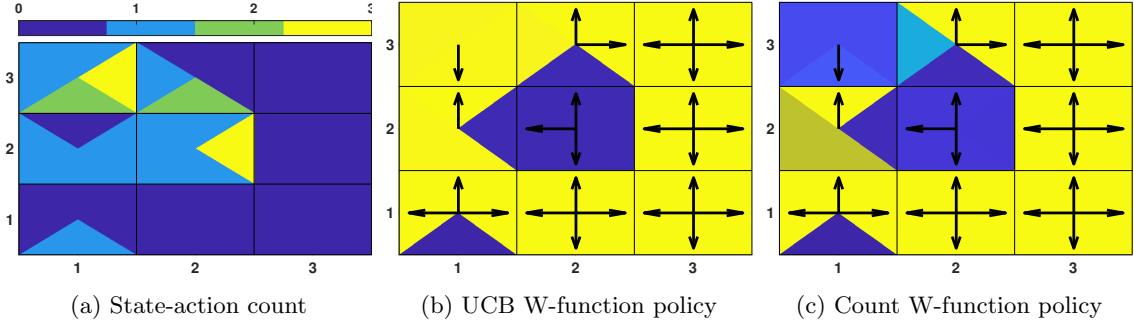
Figure 4: Toy example (part 2). Visitation count (a) and behavior policies derived from the proposed W-functions (b, c). Due to the low count, the exploration upper bound dominates the greedy Q-value, and the proposed behavior policies successfully achieve long-term exploration. Both policies prioritize unexecuted actions, and avoid terminal and prison states.

Example of Long-Term Exploration with the W-function. Consider again the toy problem presented in Section 3.1. Figure 4 shows the behavior policies of Eq. (14) and Eq. (18) with $\gamma_w = 0.99$. The policies are identical and both drive the agent to unvisited states, or to states where it has not executed all actions yet, successfully exhibiting long-term exploration. The only difference is the magnitude of the state-action pair values (the colormap, especially in unvisited states) due to their different initialization.

a) In (1,3), they both select "down" despite "up" and "left" having lower count. The agent, in fact, knows that from (1,2) it can go "down" to (1,1) where it executed only one action. Thus, it knows that there it can try new actions. On the other hand, the only thing it knows about (2,3) is that it leads to the prison state, where the agent already executed all actions. Thus, (1,2) has higher long-term exploration value than (2,3).

b) In (1,2) they both select "up", overturning the greedy action "up" (Figure 3b). This is an important aspect, because in order to explore the environment we always need to prioritize unexecuted actions.

c) The same happens in (2,3), where actions with count zero are selected, rather than the greedy ones.

d) None of the policies lead the agent to the prison state, because it has already been visited and all actions have been executed in there.

e) They both do not select "right" in (2,2) because it has been executed one more time than other actions. Nonetheless, the difference in the action value is minimal, as shown by the action colormap (almost uniform).

3.3 W-function Initialization and Bounds

In UCB bandits, it is assumed that each arm/action will be executed once before actions are executed twice. In order to enforce this, we need to make sure that the upper confidence bound $U_W(s, a)$ is high enough so that an action cannot be executed twice before all other actions are executed once. Below, we discuss how to initialize the W-functions and to set bounds in order achieve the same behavior. For the sake of simplicity, below we drop subscripts and superscripts, i.e., we write $Q(s, a)$ in place of $Q^\pi(s, a)$, $W(s, a)$ in place of $W_{\text{UCB}}^\beta(s, a)$ in the first part, and $W(s, a)$ in place of $W_{\text{N}}^\beta(s, a)$ in the second part.

W_{ucb}^β **initialization.** To compute the initialization value we consider the worst-case scenario, that is, in state s all actions a but $\bar{a}$ have been executed. In this scenario, we want that $W(s, \bar{a})$ is an upper bound of $W(s, a)$. Following Eq. (14) and (12), we have

$$Q_{\max} + \kappa(1 - \gamma_w)W(s, \bar{a}) = Q_{\max} + \kappa(1 - \gamma_w)\left(\sqrt{2\log(|\mathcal{A}| - 1)} + \gamma_w \max_a W(s', a)\right), \quad (19)$$

11

where the blue term is the TD target for non-terminal states, and $\sqrt{2\log(|\mathcal{A}|-1)}$ is the immediate visitation reward for executing $\bar{a}$ in the worst-case scenario (all other $|\mathcal{A}|-1$ actions have been executed, and $\hat{a}$ has not been executed yet). In this scenario, $W(s,\bar{a})$ is an upper bound of $W(s,a)$ under the assumption of uniform count, i.e.,

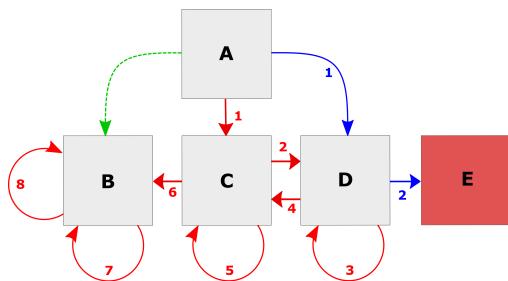$$\forall (s,a) \neq (s,\bar{a}), \quad n(s,a) = \bar{n}(s) \Rightarrow W(s,\bar{a}) \geq W(s,a). \tag{20}$$

We can then write Eq. (19) as

$$Q_{\max} + \kappa(1-\gamma_w)W(s,\bar{a}) = Q_{\max} + \kappa(1-\gamma_w)\left(\sqrt{2\log(|\mathcal{A}|-1)} + \gamma_w W(s,\bar{a})\right). \tag{21}$$

In order to guarantee to select $\bar{a}$, which has not been select yet, we need to initialize $W(s,\bar{a})$ such that the following is satisfied

$$Q_{\min} + \kappa(1-\gamma_w)W(s,\bar{a}) > Q_{\max} + \kappa(1-\gamma_w)\left(\sqrt{2\log(|\mathcal{A}|-1)} + \gamma_w W(s,\bar{a})\right), \tag{22}$$

where $Q_{\min}$ denotes the smallest possible Q-value. We get

$$W(s,\bar{a}) > \frac{Q_{\max} - Q_{\min}}{\kappa(1-\gamma_w)} + \sqrt{2\log(|\mathcal{A}|-1)} + \gamma_w W(s,\bar{a})$$

$$W(s,\bar{a}) > \frac{Q_{\max} - Q_{\min}}{\kappa(1-\gamma_w)^2} + \frac{\sqrt{2\log(|\mathcal{A}|-1)}}{(1-\gamma_w)}. \tag{23}$$

Initializing the W-function according to Eq. (23) guarantees to select $\bar{a}$.

W_n^{β} **upper bound.** Since this W-function represents the discounted cumulative count, it is initialized to zero. However, numerical problems may occur in the square root of Eq. (18) because the pseudocount can be smaller than one or even zero. In these cases, the square root of negative numbers and the division by zero are not defined. To address these issues, we add +1 inside the logarithm and replace the square root with an upper bound when $\hat{n}(s,a) = 0$. Similarly to the previous section, the bound needs to be high enough so that an action cannot be executed twice before all other actions are executed once. Following Eq. (18), we need to ensure that

$$Q_{\min} + \kappa U_W(s,\bar{a}) > Q_{\max} + \kappa U_W(s,a),$$

$$U_W(s,\bar{a}) > \frac{Q_{\max} - Q_{\min}}{\kappa} + U_W(s,a). \tag{24}$$

However, this time assuming uniform count does not guarantee a uniform *pseudocount*, i.e.,

$$\forall (s,a) \neq (s,\bar{a}), \quad n(s,a) = \bar{n}(s) \nRightarrow W(s,\bar{a}) \geq W(s,a). \tag{25}$$

Figure 5: Chainworld with uniform count.

To illustrate this issue, consider the chain MDP in Figure 5. The agent always starts in A and can move "left", "right", and "down". The episode ends after eight steps, or when terminal state E is reached. The goal of this example is to explore all states uniformly. In the first episode, the agent follows the blue trajectory (A, D, E). In the second episode, it follows the red trajectory (A, C, D, D, C, B, B, B). In both trajectories, an action with count zero was always selected first. At the beginning of the third episode, all state-action pairs have been executed once, except for "left" in A, i.e., $n(s,a) = 1 \ \forall (s,a) \neq (A, left), \ n(A, left) = 0$. Thus, we need to enforce the agent to select it. However, the discounted *cumulative* count $\hat{n}(s,a) = (1-\gamma_w)W(s,a)$ is *not* uniform. For example with $\gamma_w = 0.9$, $\hat{n}(A, right) = 3.8$ and $\hat{n}(A, down) = 4.52$.
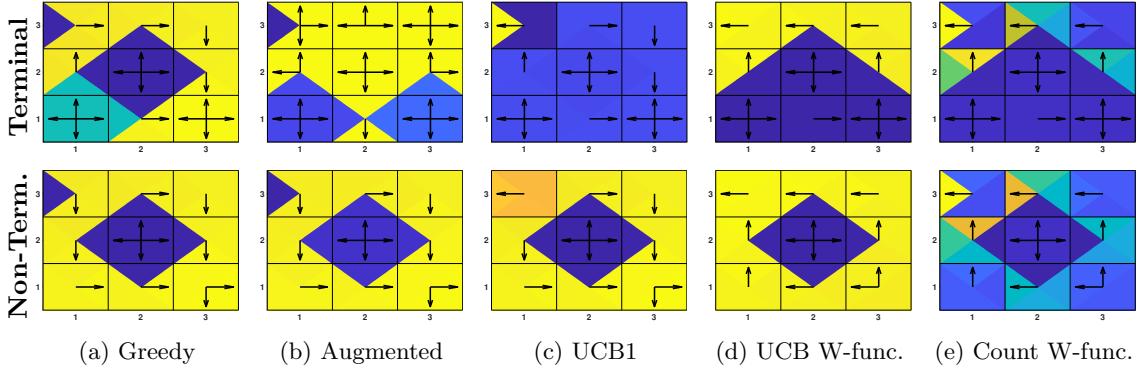
Figure 6: Toy example (part 3). Behavior policies under uniform visitation count. In the top row, reward states (1,1) and (3,1) are terminal. In the bottom row, they are not and the MDP has infinite horizon. All state-action pairs been executed once, except for "left" in (1,3). Only the proposed methods (d, e) correctly indentify such state-action pair as the most interesting for exploration, and propagate this information to other states thanks to TD learning. When the action is executed, all policies converge to the greedy one (a).

More in general, the worst-case scenario we need to consider to satisfy Eq. (24) is the one where (1) action $\bar{a}$ has not been executed yet, (2) an action $\dot{a}$ has been executed once and has the smallest pseudocount, and (3) all other actions have been executed once and have the highest pseudocount. In this scenario, to select $\bar{a}$ we need to guarantee $U_W(s,\bar{a}) > U_W(s,\dot{a})$. Since we assume that counts are uniform and no action action has been executed twice, the smallest pseudocount is $\hat{n}(s,\dot{a}) = 1 - \gamma_w$, while the highest is $\hat{n}(s,a) = 1$. Then, Eq. (24) becomes

$$U_W(s,\bar{a}) > \frac{Q_{\max} - Q_{\min}}{\kappa} + U_W(s,\dot{a})$$

$$U_W(s,\bar{a}) > \frac{Q_{\max} - Q_{\min}}{\kappa} + \sqrt{\frac{2\log\left((1-\gamma_w) + |\mathcal{A}| - 2\right)}{1 - \gamma_w}}. \tag{26}$$

Replacing the square root in Eq. (18) with the above upper bound when $n(s,\bar{a}) = 0$ guarantees to select $\bar{a}$.

Example of Behavior Under Uniform Count. Consider our toy problem again. We distinguish two cases: (1) all state-action pairs have been executed once except for "left" in (1,3), and (2) the count is one everywhere. The Q-table is optimal for visited state-action pairs, thus the greedy policy simply brings the agent to (3,1) where the highest reward lies. Figure 6 shows baseline and proposed behavior policies in case (1). In case (2), all policies look like the greedy one (Figure 6a).

a) The augmented reward policy (Figure 6b) has no interest in trying "left" in (1,3), and its value is the lowest. The agent, in fact, cannot get the visitation bonus without first executing the action. This shows once more that this passive way of exploration is highly inefficient.

b) UCB1 policy (Figure 6c) acts greedily everywhere except in (1,3), where it correctly selects the unexecuted action. However, since UCB1 considers only the *immediate* visitation, it has no way of propagating this information to other state-action pairs.

c) By contrast, for the proposed W-function policies (Figure 6d and 6e) the unexecuted state-action pair is the most interesting, and the policy brings the agent there from any state. Thanks to the separate W-function, we can propagate the value of unexceted actions to other states, always prioritizing the exploration of new state-action pairs. Under uniform count $n(s,a) = 1 \ \forall (s,a)$, then the agent acts greedily with respect to the Q-function.

13

3.4 Final Remarks

In this section, we discuss the uniform count assumption used to derive the bounds, the differences between the proposed W-functions and intrinsic motivation, the benefit of using the count-based one, and potential issues in stochastic environments.

Uniform count assumption. In Section 3.3 we derived an initialization for W_{UCB}^{β} and a bound for W_{N}^{β} to guarantee that an action is not executed twice before all other actions are executed once. For that, we assumed the visitation count to be uniform for all state-action pairs. We acknowledge that this assumption is not true in practice. First, it may be necessary to revisit old states in order to visit new ones. Second, when an episode is over the state is reset and a new episode starts. This is common practice also for infinite horizon MDPs, when usually the environment is reset after some steps. Thus, the agent initial state will naturally have higher visitation count. Nonetheless, in Section 4.2.1, we will empirically show that our approach allows the agent to explore the environment as uniformly as possible.

W-function vs auxiliary rewards. Using an auxiliary reward such as an exploration bonus or an intrinsic motivation signal has similarities with the proposed approach, especially with W_{UCB}^{β}, but the two methods are not equivalent. Consider augmenting the reward with the same visitation reward used to train the W-function, i.e., $r_t^+ = r_t + \alpha\, r_t^W$. Using TD learning, the augmented Q-function used for exploration can be rewritten as[2]

$$Q^+(s_t, a_t) = \textcolor{red}{r_t} + \textcolor{blue}{\alpha r_t^W} + \textcolor{blue}{\gamma \max_a Q^+(s_{t+1}, a)}. \tag{27}$$

Then consider the behavior policy derived from W_{UCB}^{β} in Eq. (14), and decompose it as well

$$Q^\pi(s_t, a_t) + \alpha W_{\text{UCB}}^{\beta}(s_t, a_t) = \textcolor{red}{r_t} + \textcolor{blue}{\gamma \max_a Q^\pi(s_{t+1}, a)} + \textcolor{blue}{\alpha r_t^W} + \alpha\gamma_w \max_a W_{\text{UCB}}^{\beta}(s_{t+1}, a), \tag{28}$$

where we considered $\alpha = \kappa(1 - \gamma_w)$. The red terms are equivalent, but the blue terms are not because of the max operator, since $\max_x\{f(x) + g(x)\} \neq \max_x\{f(x)\} + \max_x\{g(x)\}$.

This explicit separation between exploitation (Q) and exploration (W) has two important consequences. First, it is easy to implement the proposed behavior policy for any off-policy algorithm, as the exploration term is separated and independent from the Q-function estimates. This implies that we can propagate the exploration value independently from the action-value function estimate. With this choice we can select a discount factor γ_w which differs from the MDP discount γ. By choosing a high exploration discount factor γ_w, we do long-term exploration allowing us to find the optimal policy when exploration is crucial. By choosing a low exploration discount factor γ_w, instead, we perform short-term exploration which may converge faster to a greedy policy. In the evaluation in Section 4.2, we show experiments for which the discount factors are the same, as well as experiments where they differ. Furthermore, we investigate the effect of changing the W-function discount while keeping the Q-function discount fixed.

Second, auxiliary rewards are non-stationary, as the visitation count changes at every step. This leads to a non-stationary target for the Q-function. With our approach, by decoupling exploration and exploitation, we have stationary target for the Q-function while moving all the non-stationarity to the W-function.

Benefits of zero-initialization. Because of its initialization (Eq. (23)), the proposed W_{UCB}^{β} gives an optimistic estimate which is needed for efficient exploration. By contrast, W_{N}^{β} does not require any kind of optimistic initialization. This can be crucial for continuous domains where function approximation is necessary, and for which proper optimistic initialization is not always feasible [Osband et al., 2019]. In the experiments of Section 4, we show that the two W-functions attain almost equal results, making W_{N}^{β} very promising for solving non-tabular MDPs with sparse rewards. However, the evaluation in continuous domain is out of the scope of this paper, and will be pursued in future work.

---

[2]For the sake of simplicity, we consider only non-terminal states.

Terminal states and stochasticity. The visitation rewards for training the W-functions penalize terminal state-action pairs (Eq. (10) and (15)). Consequently, once visited, the agent will avoid visiting them again. One may think that this would lead to poor exploration in stochastic environments, where the same state-action pair can lead to both terminal and non-terminal states. In this scenario, trying the same action again in the future may yield better exploration depending on the stochastic transition function. However, as we empirically show in Section 4.2.5, stochasticity in the transition function does not harm our approach.

4 Evaluation

In this section, we evaluate the proposed method against state-of-the-art methods for exploration in model-free RL on several domains. The experiments are designed to highlight the challenges of learning with sparse rewards, and are split into two parts. In the first, we present the environments and we address (1) learning when only few states give a reward, (2) learning when a sequence of correct actions is needed to receive a reward, (3) exploring efficiently when few steps are allowed per training episode, (4) avoiding local optima and distracting rewards, and (5) avoiding dangerous states that stop the agent preventing exploration. In the second, we further investigate (6) the visitation count at convergence, (7) the empirical sample complexity, (8) the impact of the visitation value hyperparameter γ_w, (9) the performance in the infinite horizon scenario, and (10) with stochastic transition function.

To ensure reproducibility, the experiments are performed over fixed random seeds. For deterministic MDPs, the seeds are $1, 2, \ldots, 20$. For stochastic MDPs, the seeds are $1, 2, \ldots, 50$. Pseudocode and further details of the hyperparameters are given in Appendix A. Source code is available at `https://github.com/sparisi/visit-value-explore`

Baseline Algorithms. The evaluated algorithms all use Q-learning [Watkins and Dayan, 1992] and share the same hyperparameters, e.g., learning rate and discount factor. In most of the experiments, we use infinite replay memory as presented by Osband et al. [2019]. In Appendix A.5 we also present an evaluation without replay memory on some domains. The algorithms all learn two separate Q-tables, a behavior one for exploration, and a target one for testing the quality of the greedy policy. The difference among the algorithms is how the behavior policy performs exploration. In this regard, we compare against the following exploration strategies: random, ϵ-greedy, bootstrapped, count-based.

For bootstrapping, we evaluate the original version proposed by Osband et al. [2016a], as well as the more recent versions of D'Eramo et al. [2019] and Osband et al. [2019]. These methods all share the core idea, i.e., to use an ensemble of Q-tables, each initialized differently and trained on different samples, to approximate a distribution over Q-values via bootstrapping. The differences are the following. Osband et al. [2016a] select a random Q-table at the beginning of each episodes, and use it until the episode is over. Osband et al. [2019] further regularize the Q-tables using the squared ℓ_2-norm to "prior tables" representing a prior over the Q-function. D'Eramo et al. [2019] instead select the Q-table randomly within the ensemble at each step to approximate Thompson sampling, but do not use any prior.

For count-based algorithms, we compare against UCB1 exploration [Auer et al., 2002] on the immediate state-action count as in Eq. (5), and against augmenting the reward with the exploration bonus proposed by Strehl and Littman [2008] as in Eq. (6). Notice that Strehl and Littman [2008] apply the exploration bonus to a model-based algorithm, but the same bonus has since then been used by many model-free algorithms [Bellemare et al., 2016, Dong et al., 2019]. More recent methods use more sophisticated bonus to derive strong convergence guarantees [Jin et al., 2018, Dong et al., 2019]. However, for large size problems these methods may require an impractical number of samples and are often outperformed by standard algorithms.

4.1 Part 1: Discounted Return and States Discovered

We use the following environments: deep sea [Osband et al., 2018, 2019], taxi driver [Asadi and Littman, 2017], and four novel benchmark gridworlds. All the environments have sparse discounted rewards, and each has peculiar characteristics making it challenging to solve.

Evaluation Criteria. For each environment, we evaluate the algorithms on varying two different conditions: optimistic vs zero initialization of the behavior Q-tables, and short vs long horizon, for a total of four scenarios. For each of them, we report the expected discounted return of $\pi(a|s)$, which is greedy over $Q^\pi(s, a)$, and the number of visited states over training samples. In all environments the initial state is fixed.

Optimistic initialization [Lai and Robbins, 1985] consists of initializing the Q-tables to a large value. After visiting a state-action pair, either the agent experiences a high reward confirming his belief about its quality, or the agent experiences a low reward and learns that the action was not optimal and will not execute it again in that state. By initializing the Q-tables to the upper bound $r_{\max}/(1 - \gamma)$ we are guaranteed to explore all the state-action pair even with just an ϵ-greedy policy. However, this requires prior knowledge of the reward bounds, and in the case of continuous states and function approximation the notion of an universally "optimistic prior" does not carry over from the tabular setting [Osband et al., 2019]. Thus, it is important that an algorithm is able to explore and to learn even with a simple zero or random initialization.

The length of the horizon is also an important factor for exploration. If the agent can perform a limited number of actions during an episode, it has to carefully choose them in order to explore as efficiently as possible. Prominent example in this regard are real-world tasks such as videogames, where the player has limited time to fulfill the objective before the game is over, or robotics, where the autonomy of the robot is limited by the hardware. Thus, it is important that an algorithm is able to explore and to learn even when the training episode horizon is short. In the "short horizon" scenario, we allow the agent to explore for a number of steps slightly higher than the ones needed to find the highest reward. For example, if the highest reward is eight steps away from the initial position, the "short horizon" is ten steps. The "long horizon" scenario is instead always twice as long.

Results are first presented in plots showing number of states discovered and discounted return against training steps averaged over 20 seeds. Due to the large number of evaluated algorithms, confidence intervals are not reported in plots. At the end of the section, a recap table summarizes the results and reports the success of an algorithm, i.e., the number of seeds the algorithm learned the optimal policy, and the percentage of states visited at the end of the learning, both with 95% confidence interval.

4.1.1 Deep Sea
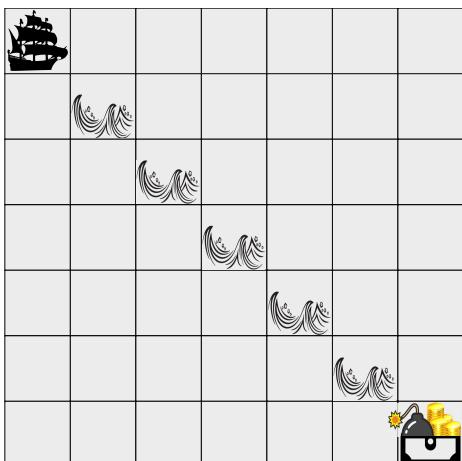
Figure 7: The deep sea.

In this environment (Figure 7), the agent (the ship) always starts at the top-left corner of a gridworld of size $N \times N$, and has to descend through the grid to open a chest, located in the bottom-right corner. At the beginning of the episode, either a treasure or a bomb appear inside the chest, with a 50-50. The outcome is known to the agent, as the state consists of the ship position and a flag denoting if the bomb or the treasure has appeared. At every step, the ship descends by one step, and must chose to turn left or right. If it hits the left boundary, it will descend without turning. For example, from the starting position $(1, 1)$, doing "left" will move the ship to $(2, 1)$, while doing "right" will move it to $(2, 2)$. The episode ends when the ship reaches the bottom of the grid, and thus the horizon is fixed to N. The reward is 1 if the ship finds the treasure, -1 if it finds the bomb, and $-0.01/N$ if the
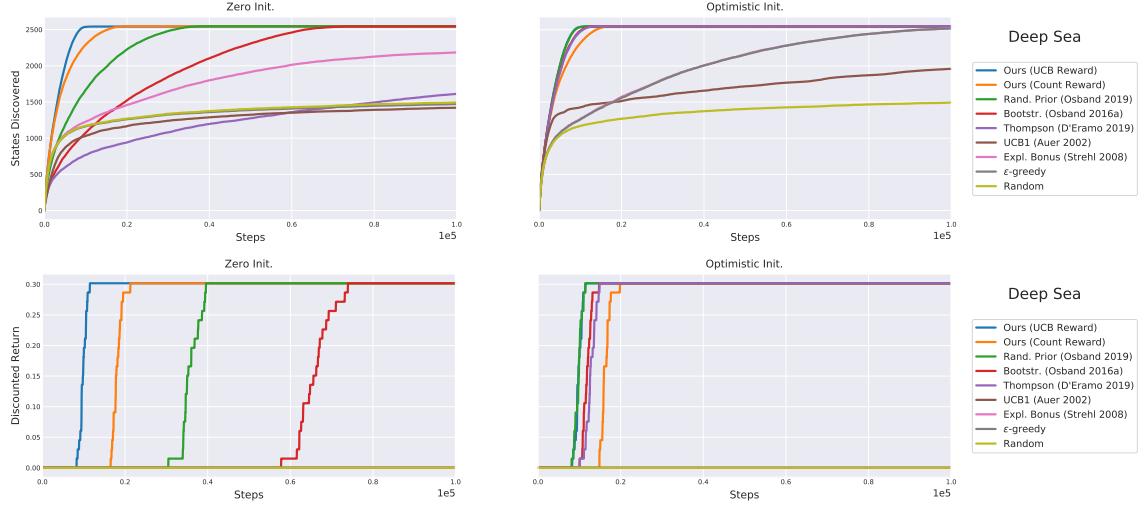
Figure 8: Results on the deep sea domain averaged over 20 seeds. The proposed exploration achieves the best results, followed by bootstrapped exploration.

ship does "right" in any cell along the diagonal (denoted by sea waves). The optimal policy selects always "right" if the treasure has appeared, otherwise "left" at the initial state and then can act randomly.

The challenge of this environment lies in having to select the same action ("right") N times in a row in order to receive either 1 or -1. Doing "left" even only once prevents the ship from reaching the chest. At the same time, the agent is discouraged from doing "right", since doing so along the diagonal yields an immediate penalty. Consequently, an ϵ-greedy policy without optimistic initialization would perform poorly, as the only short-term reward it receives would discourage it from doing "right". A random policy is also highly unlikely to reach the chest. In particular, the probability such a policy reaches it in any given episode is $(1/2)^N$. Thus, the expected number of episodes before observing either the bomb or the treasure is 2^N. Even for a moderate value of $N = 50$, this is an impractical number of episodes [Osband et al., 2019].

Results. Figure 8 shows the results on a grid of depth $N = 50$. Since this domain has a fixed horizon, the evaluation does not include the "short vs long horizon" comparison. With zero initialization, only our algorithms and bootstrapping by Osband et al. [2016a] and Osband et al. [2019] discover all states and converge to the optimal policy within steps limit. Both proposed visitation-count-based (blue and orange) outperform the other two, converging twice or more as faster. Bootstrapped exploration with random prior (green) comes second, reproducing the results reported by Osband et al. [2018] and Osband et al. [2019]. Results with optimistic inizialization are similar, with the main difference being that also approximate Thompson sampling by D'Eramo et al. [2019] (purple) converges. Boostrapping by Osband et al. [2018] (green) matches the performance of visitation-count-based with UCB (blue), and the two lines almost overlap. For more details about the percentage of successful trials and the percentage of states discovered by each algorithm with confidence intervals, we refer to Table 1.

This evaluation shows that the proposed exploration outperform state-of-the-art boostrapping on domain with fixed horizon and no local optima. In the next sections, we investigate domains with absorbing states, i.e., that can prematurely end the episode, and distractor reward. In Section 4.2.3, we also present an empirical study on sample complexity on varying the deep sea depth N. Furthermore in Section 4.2.1, we show the state visitation count at the end of the learning for the deep sea and other domains, discussing why the proposed visitation-value-based exploration achieves new state-of-the-art results.

17
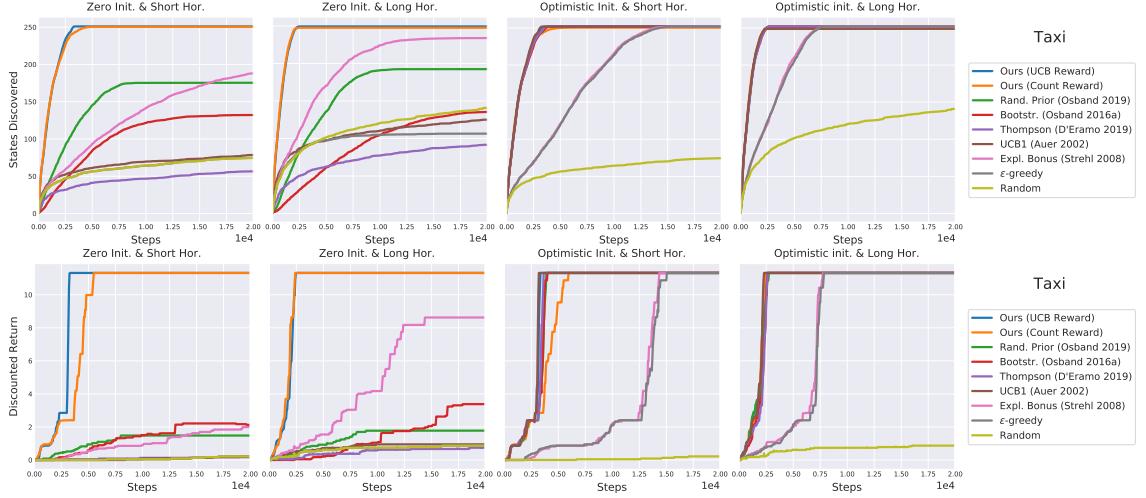
Figure 10: Results on the taxi domain averaged over 20 seeds. The proposed algorithms outperform all others, being the only solving the task without optimistic initialization.

4.1.2 Multi-Passenger Taxi Driver

Figure 9: The taxi driver.

In this environment (Figure 9), the agent (the taxi) starts at the top-left corner of a grid, and has to pick up the passengers and drop them off at the destination (the flag). It can carry multiple passengers at the same time. If one, two, or all three passengers reach the destination, the agent is rewarded with 1, 3, or 15, respectively. Everywhere else the reward is 0. The state consists of the taxi position and three booleans denoting if a passenger has been picked up or not. The agent can turn left, right, up, or down. Black cells cannot be visited. The taxi ends its ride upon visiting the destination, regardless of how many passengers it is carrying. The optimal policy picks up all passengers and drops them off in 29 steps[3].

This domain is challenging for two reasons. First, it admits many locally optimal policies, depending on how many passengers reach the destination. Second, with a short horizon learning to pick up and drop off all passengers is difficult and requires effective exploration.

Results. In this domain, the "short horizon" consists of 33 steps[4], while the "long" of 66. As shown in Figure 10, bootstrapped algorithms here perform significantly worse than before. None of them, in fact, learned to pick all three passengers if Q-tables are initialized to zero, and the "long horizon" does not help. In particular, random prior bootstrapping (green) clearly converged prematurely due to the presence of local optima (the algorithm does not discover new states after 750-1,000 steps). The proposed algorithms (blue and orange), instead, perform extremly well, quickly discovering all states and then learning the optimal policy. Other algorithms learned to pick one or two passengers, and only the auxiliary visitation bonus (pink) learned to pick all passengers in some seeds but only with long horizon. With optimistic initialization, most of the algorithms match the performance of our proposed ones. Among them, ϵ-greedy (gray) and bonus-based exploration (pink) learn slowly. This is not surprising for the former, while for the latter it may be due to the small bonus coefficient (see Eq. (6)) or to the issues discussed in Section 3.1. Only random exploration (light green) still cannot learn to pick all passengers. For more details

---

[3]The agent needs 28 steps to pick all passengers and reach the destination. However, it then needs to execute an additional action to get the reward.

[4]Our algorithms could learn also with a horizon of only 29 steps. However, with zero initialization other algorithms performed extremely poorly. We thus decided to give the agent 33 steps.
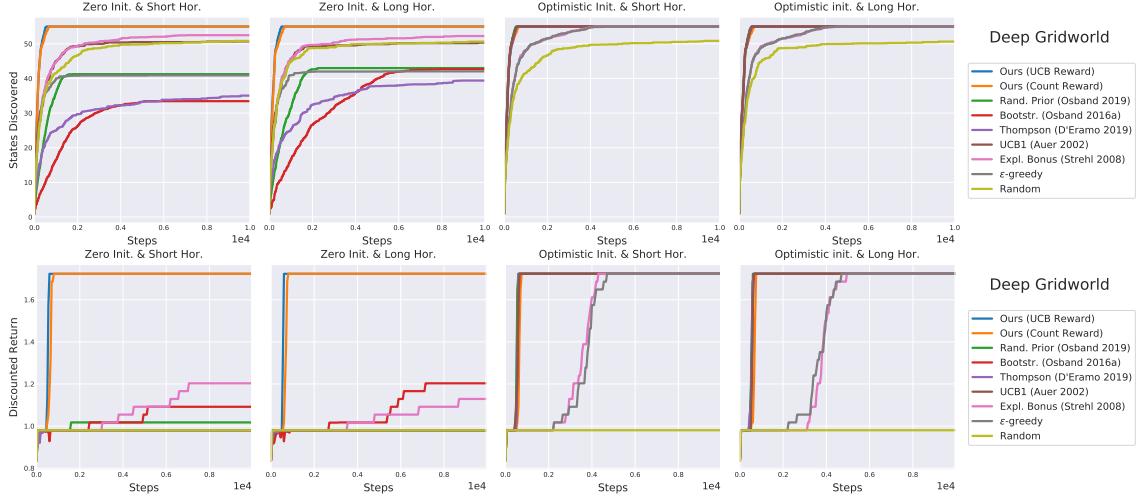
Figure 12: Results on the deep gridworld averaged over 20 seeds. Once again, the proposed algorithms are the only solving the task without optimistic initialization.

about the percentage of successful trials and the percentage of states discovered by each algorithm with confidence intervals, we refer to Table 1.

This evaluation shows that the proposed exploration outperforms existing algorithms in the presence of local optima, and that its performance is not affected by the length of horizon. Next, we investigate what happens when we combine the challenges of the deep sea and the taxi.

4.1.3 Deep Gridworld
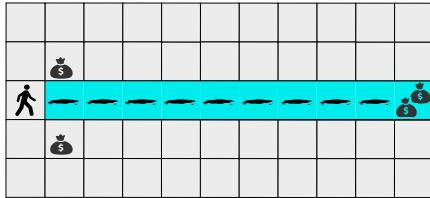
Figure 11: The deep gridworld.

The first of the four gridworlds we propose is inspired by the deep sea domain. In this environment (Figure 11), the agent navigates in a 5×11 grid by moving "left", "right", "up", or "down". All states can be visited, but the blue corridor can be accessed only from its left side. The agent can exit the corridor anytime by moving either "up" or "down". A treasure of value 2 lies at the end of the corridor, while two treasures of value 1 serve as distractors next to the corridor entrance. The corridor is filled with puddles, each giving a penalty of -0.01. The agent always starts next to the entrance and the episode ends when the agent collects any of the treasures. The optimal policy consists of doing always "right".

The deep gridworld combines the challenges of both the deep sea and the taxi domains. Like the former, to discover the highest reward the agent needs to execute the action "right" multiple times in a row, receiving negative immediate rewards due to the puddles. However, doing a different action does not preclude reaching the end of the corridor, because the agent can still go back to the entrance and try again within the same episode. At the same time, the presence of the two distractor treasures results in local optima, as in the taxi driver domain.

Results. Because it is possible to exit the corridor without ending the episode, we set a longer horizon compared to the previous domains, i.e., 55 steps for the "short" scenario and 110 for the "long" one. Results shown in Figure 12 confirm previous results. The proposed exploration quickly discoves all states and learn to navigate through the corridor. By contrast, other algorithms, including bootstrapping, get stuck in local optima and learn to collect one of the two lesser treasures. Similarly to the taxi domain, with optimistic initialization all algorithms but random exploration (light green) learn to navigate through the corridor.

This evaluation shows that distractor rewards are challenging for existing algorithms but not for ours. Next, we increase the difficulty by adding more distractors and new special states.

The Gridworlds Environments
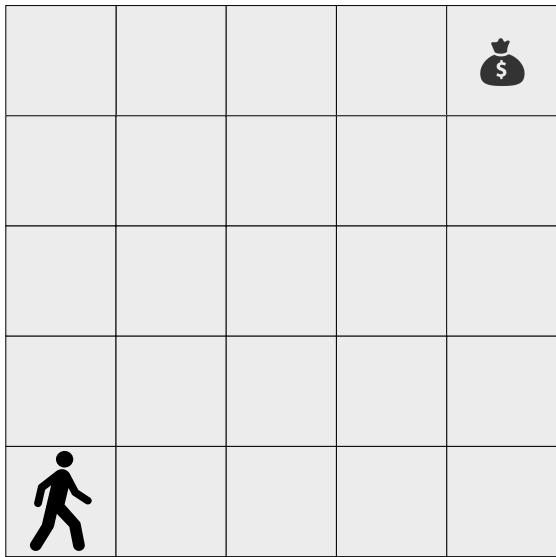
Figure 13: The "toy" gridworld.
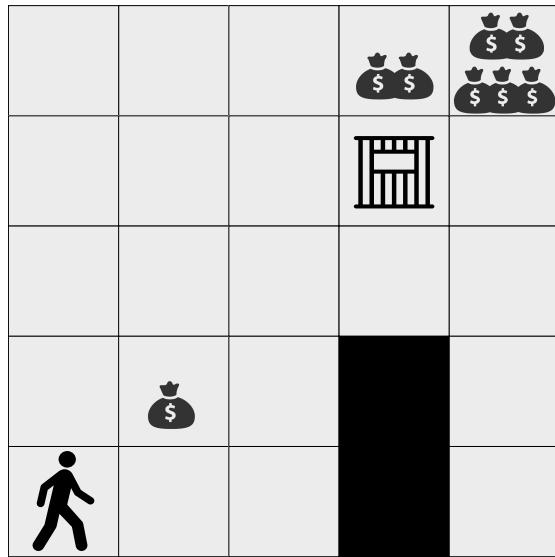
Figure 14: The "prison" gridworld.
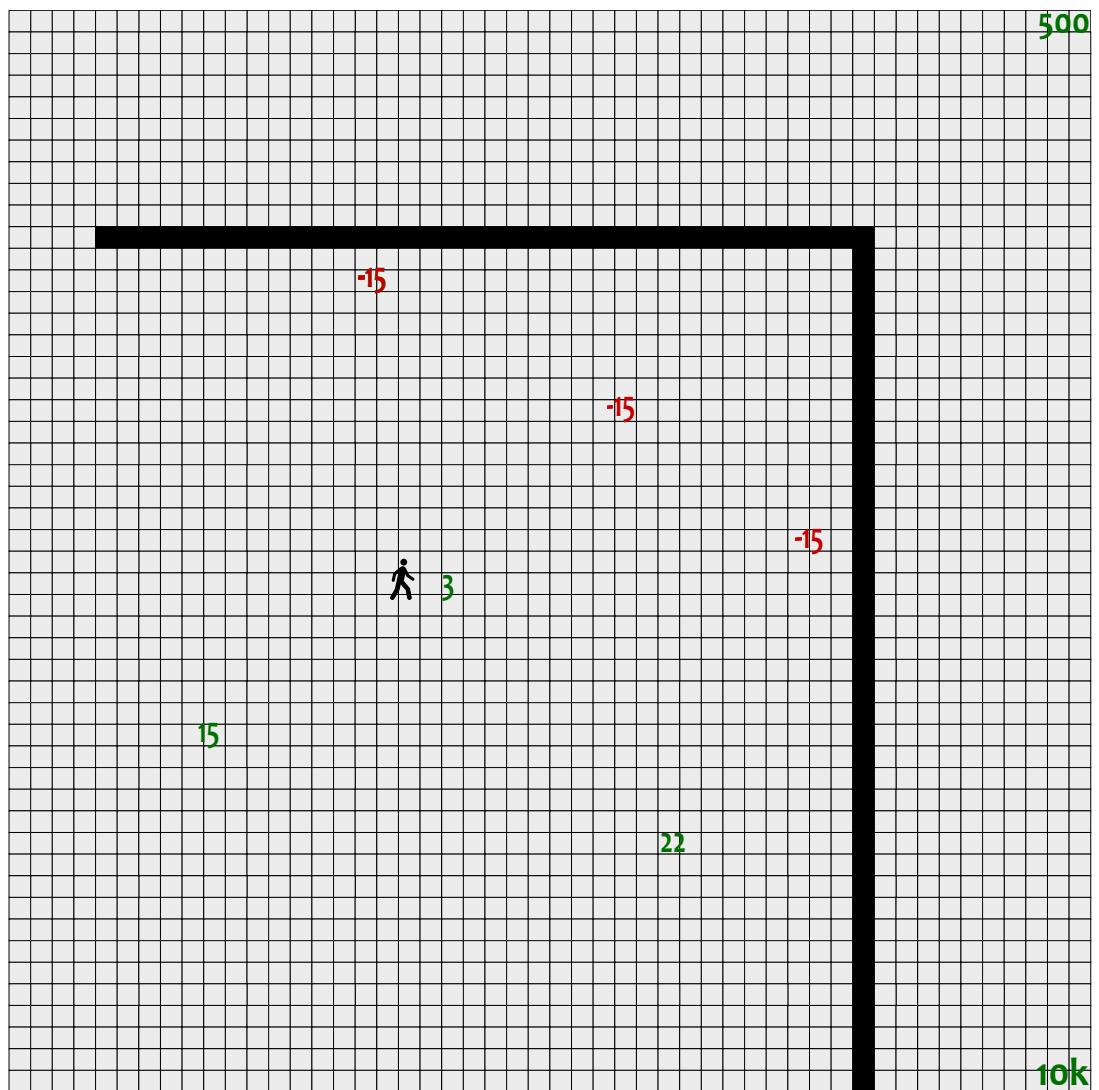
Figure 15: The "wall" gridworld.

4.1.4 Gridworlds

These environments (Figure 13, 14, and 15) share the same structure. The agent navigates in a grid by moving "left", "right", "up", or "down". Black cells cannot be visited, while any action in the prison has only an arbitrarily small probability of success. If it fails, the agent does not move. In practice, the prison almost completely prevents any further exploration. The reward is 0 everywhere except in treasure or penalty cells, and is given for executing an action in the state, i.e., on transitions. Treasure cells (denoted by a money bag or a green number) give a bonus reward of different magnitude and end the episode. Penalty cells (denoted by a red number) give a penalty reward and end the episode. The agent also gets a small penalty of -0.01 at each step. The goal, thus, is to find the biggest treasure using as few steps as possible. The initial position is fixed such that it is far from the biggest treasure and close to the smallest one.

Similarly to the deep gridworld, these domains are designed to highlight the difficulty of learning with many distractors (the smaller treasures) and, thus, of many local optima. The addition of the constant penalty at each step further discourages exploration and makes the distractors more appealing, since ending the episode will stop receiving penalties. Each domain has increasing difficulty and introduces additional challenges, as we explain below.

The "toy" gridworld. We start with a simple 5×5 gridworld with one single reward of value 1, as shown in Figure 13. The focus of this domain is learning with sparse reward, without distractors or additional difficulties. The optimal policy takes only nine steps to reach the treasure, and the "short horizon" scenario ends after eleven steps.

The "prison" gridworld. Shown in Figure 14, this domain increases the value of the furthest reward to 5, adds two distractors of value 1 and 2, and adds a prison cell. The first distractor is close to the initial position, while the second is close to the goal reward. The prison cell is also close to the goal. The optimal policy navigates around the first distractor, then right below the prison, and finally up to the goal. The prison highlights the importance of seeking states with low long-term visitation count, rather than low immediate count. As discussed in Section 3.1, in fact, current count-based algorithms cannot deal with this kind of states efficiently.

The "wall" gridworld. Shown in Figure 15, the last gridworld is characterized by increased grid size (50×50) and reward magnitude, and by the wall separating the grid into two areas. The first area, where the agent starts, has six small rewards (both treasures and penalties). The second area has two bigger treasures in the upper-right and bottom-right corners, of value 500 and 10,000, respectively. The optimal policy brings the agent beyond the wall and then to the bottom right corner, where the highest reward lies. The wall significaly increases the difficulty, due to the narrow passage which the agent needs to find in order to visit the second area of the grid. Learning is even harder when the horizon is short, as the agent cannot afford to lose time randomly looking for new states. To increase the chance of success of all algorithms, we set the "short horizon" to 330 steps, and 135 are needed to reach the reward of 10,000.

Results. The gridworld environments confirm previous results. Without distractors (toy gridworld, Figure 16), bootstrapped algorithms (green and red) perform well, and so does using the auxiliary visitation bonus (pink). Neither, however, match ours (blue and orange, almost overlapping). Increasing the horizon helps the remainder algorithms, including random exploration (light green), except for approximate Thompson sampling (purple). In the next gridworld, however, the distractors and the prison cell substantially harm all algorithms except ours (Figure 17). Without optimistic initialization, in fact, existing algorithms cannot find the highest reward even with long horizon, and all converge to local optima. This behavior was expected, given the study of Section 3.1. Finally, the wall gridworld results emphasize even more the superiority of our algorithms (Figure 18). With zero initialization, in fact, every other algorithm cannot go beyond the wall and find even the reward of 500. The results also stress that using the UCB reward visitation value (blue) over the count reward (orange) performs slightly best.

This evaluation strenghtens the findings of previous experiments. First, it stresses how difficult it is for existing algorithms to learn with distracting rewards and short horizon. Second, it shows that our proposed approach overcomes these challenges. Next, we show how efficiently the algorithms explore in terms of final visitation count and sample complexity.
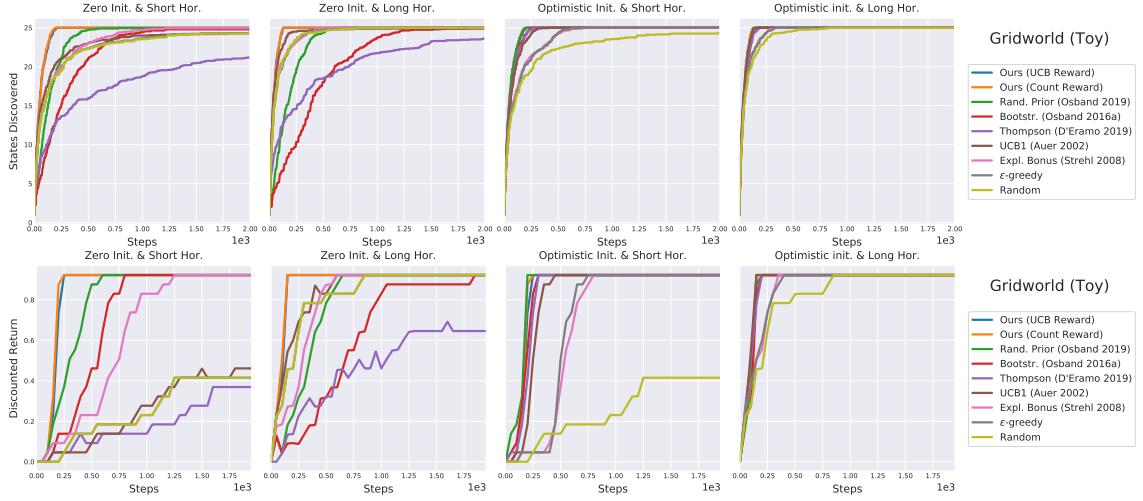
Figure 16: Results on the toy gridworld averaged over 20 seeds. Bootstrapped and bonus-based exploration perform well, but cannot match the proposed one (blue and orange line overlap).
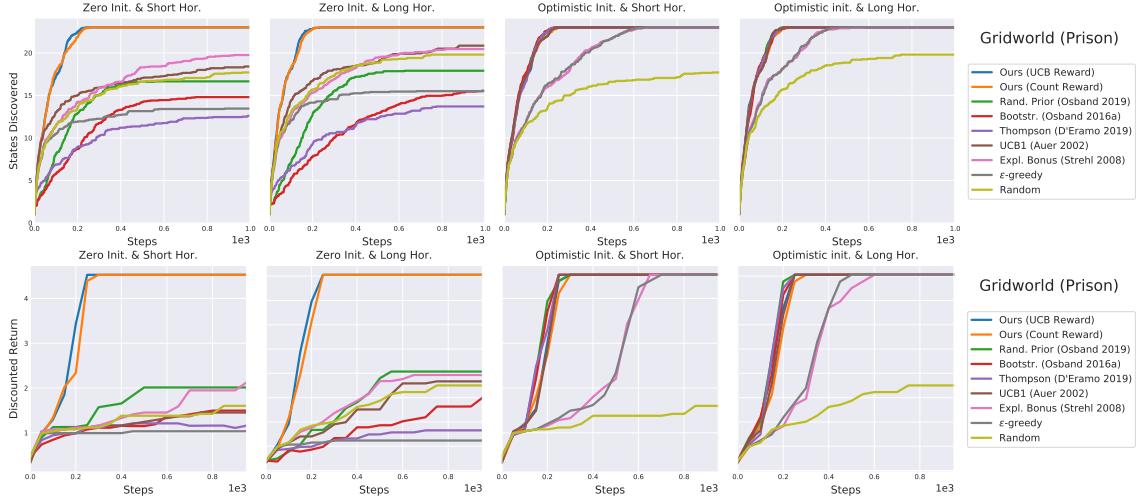
Figure 17: Distractors and the "prison" affect the performance of all algorithms but ours.
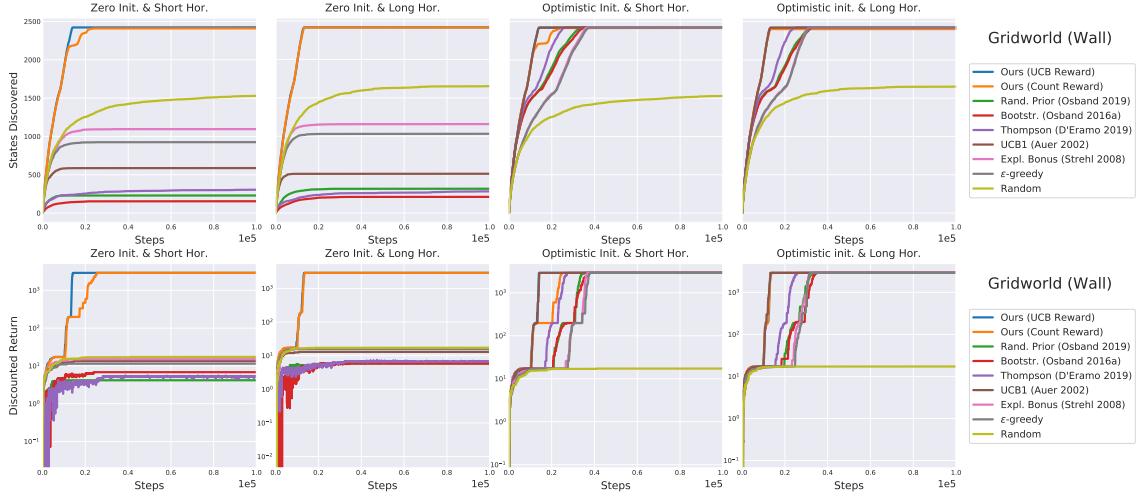
Figure 18: The final gridworld emphasizes the better performance of the proposed algorithms, especially the one using UCB reward. Return plot is in log scale.

Table 1: Results recap for the "zero initialization short horizon" scenarios. Only the proposed exploration strategy is able to always discover all states and solve the task in all 20 seeds.

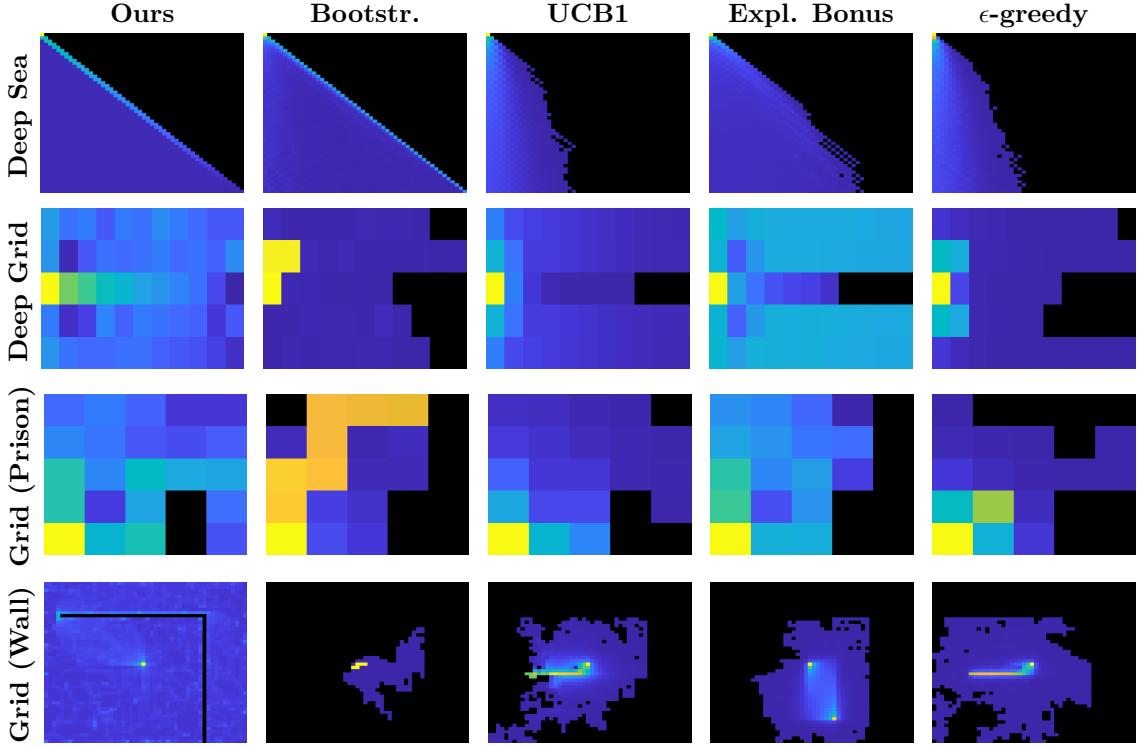|	Algorithm	Discovery (%)	Success (%)
Deep Sea	**Ours (UCB Reward)**	**100 ± 0**	**100 ± 0**
	Ours (Count Reward)	**100 ± 0**	**100 ± 0**
	Rand. Prior (Osband 2019)	99.90 ± 0.01	100 ± 0
	Bootstr. (Osband 2016a)	99.77 ± 0.05	100 ± 0
	Bootstr. (D'Eramo 2019)	63.25 ± 3.31	0 ± 0
	UCB1 (Auer 2002)	55.72 ± 0.34	0 ± 0
	Expl. Bonus (Strehl 2008)	85.65 ± 1.0	0 ± 0
	ϵ-greedy	57.74 ± 1.11	0 ± 0
	Random	58.59 ± 1.35	0 ± 0
Taxi	**Ours (UCB Reward)**	**100 ± 0**	**100 ± 0**
	Ours (Count Reward)	**100 ± 0**	**100 ± 0**
	Rand. Prior (Osband 2019)	69.60 ± 2.96	13.07 ± 2.96
	Bootstr. (Osband 2016a)	52.44 ± 7.55	18.77 ± 9.24
	Bootstr. (D'Eramo 2019)	22.44 ± 1.81	1.9 ± 1.47
	UCB1 (Auer 2002)	31.17 ± 0.70	1.53 ± 1.37
	Expl. Bonus (Strehl 2008)	74.62 ± 2.24	17.6 ± 2.56
	ϵ-greedy	29.64 ± 0.98	1.92 ± 1.49
	Random	29.56 ± 0.98	1.92 ± 1.49
Deep Gridworld	**Ours (UCB Reward)**	**100 ± 0**	**100 ± 0**
	Ours (Count Reward)	**100 ± 0**	**100 ± 0**
	Rand. Prior (Osband 2019)	75 ± 9.39	59.03 ± 4.2
	Bootstr. (Osband 2016a)	60.82 ± 11.78	63.35 ± 6.89
	Bootstr. (D'Eramo 2019)	63.73 ± 10.35	56.85 ± 0.06
	UCB1 (Auer 2002)	92.18 ± 0.36	56.88 ± 0
	Expl. Bonus (Strehl 2008)	95.45 ± 1.57	69.81 ± 8.84
	ϵ-greedy	74.36 ± 4.42	56.88 ± 0
	Random	92.45 ± 0.64	56.88 ± 0
Gridworld (Toy)	**Ours (UCB Reward)**	**100 ± 0**	**100 ± 0**
	Ours (Count Reward)	**100 ± 0**	**100 ± 0**
	Rand. Prior (Osband 2019)	99.8 ± 0.39	100 ± 0
	Bootstr. (Osband 2016a)	99.2 ± 0.72	100 ± 0
	Bootstr. (D'Eramo 2019)	84.8 ± 4.33	40 ± 21.91
	UCB1 (Auer 2002)	97.4 ± 0.99	50 ± 22.49
	Expl. Bonus (Strehl 2008)	99.8 ± 0.39	100 ± 0
	ϵ-greedy	97.4 ± 1.17	45 ± 22.25
	Random	97.2 ± 1.15	45 ± 22.25
Gridworld (Prison)	**Ours (UCB Reward)**	**100 ± 0**	**100 ± 0**
	Ours (Count Reward)	**100 ± 0**	**100 ± 0**
	Rand. Prior (Osband 2019)	72.39 ± 6.9	44.44 ± 14.69
	Bootstr. (Osband 2016a)	64.35 ± 11.1	33.03 ± 8.54
	Bootstr. (D'Eramo 2019)	54.78 ± 6.69	25.61 ± 8.39
	UCB1 (Auer 2002)	80.78 ± 2.56	32.31 ± 4.11
	Expl. Bonus (Strehl 2008)	85.87 ± 3	46.96 ± 12.35
	ϵ-greedy	58.38 ± 5.27	22.84 ± 2.48
	Random	76.96 ± 3.08	35.36 ± 10.37
Gridworld (Wall)	**Ours (UCB Reward)**	**100 ± 0**	**100 ± 0**
	Ours (Count Reward)	**100 ± 0**	**100 ± 0**
	Rand. Prior (Osband 2019)	9.44 ± 3.14	0.14 ± 0.06
	Bootstr. (Osband 2016a)	6.35 ± 2.74	0.23 ± 0.08
	Bootstr. (D'Eramo 2019)	12.55 ± 4.45	0.18 ± 0.08
	UCB1 (Auer 2002)	24.7 ± 4.45	0.46 ± 0.03
	Expl. Bonus (Strehl 2008)	45.16 ± 2.29	0.52 ± 0.04
	ϵ-greedy	38.15 ± 1.95	0.39 ± 0.08
	Random	65.28 ± 0.45	0.59 ± 0

Figure 19: Visitation count at the end of the learning. The seed is the same across all images. Initial states naturally have higher count than other states. Recall that the upper portion of the deep sea and some gridworlds cells cannot be visited. Only the proposed methods explore the environment uniformly[5]. Other algorithms myopically focus on distractors.

4.2 Part 2: In-Depth Investigation

In this section, we present an in-depth analysis of issues associated with learning with sparse rewards, explaining in more detail why our approach outperformed existing algorithms in the previous evaluation. We start by reporting the visitation count at the end of the learning, showing that our algorithms explore the environment uniformly. We continue with a study on the impact of the visitation discount γ_w, showing how it affects the performance of the W-function. Next, we present the empirical sample complexity analysis of all algorithms on the deep sea domain, showing that our approach scales gracefully with the number of states. Finally, we evaluate the algorithms in the infinite horizon setting and in stochastic MDPs, evaluating the accuracy of the Q-function and the empirical sample complexity. Even in these scenarios, our approach outperforms existing algorithms.

4.2.1 Visitation Count at the End of the Learning

In Section 3.3 and 3.4, we discussed that the proposed behavior policies guarantee that an action is not executed twice before all other actions are executed once, under the uniform count assumption. We also acknowledged that this assumption cannot hold in practice because of state resets, or becuase the agent may need to revisit the same state to explore new ones. For example, in the deep sea the agent needs to traverse diagonal states multiple times to visit every state. Nonetheless, in this section we empirically show that our approach allows the agent to explore the environment as uniformly as possible.

Figure 19 reports the state visitation count at the end of learning in the "short-horizon zero-initialization" scenario for some domains. Only the proposed method (first column) uniformly explores all states. Recall, in fact, that episodes reset after some steps or when terminal states are

---

[5]Figures show the count of UCB-based W-function. Count-based W-function performed very similarly.
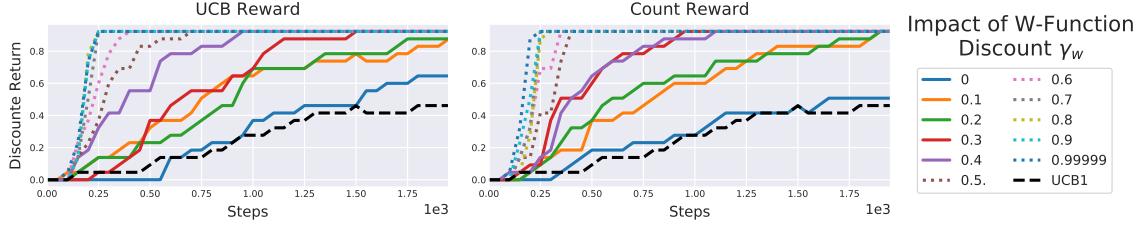
Figure 20: Performance of the proposed algorithms on varying of γ_w on the "toy" gridworld (Figure 14). The higher γ_w, the more the agent explore, allowing to discover the reward faster.
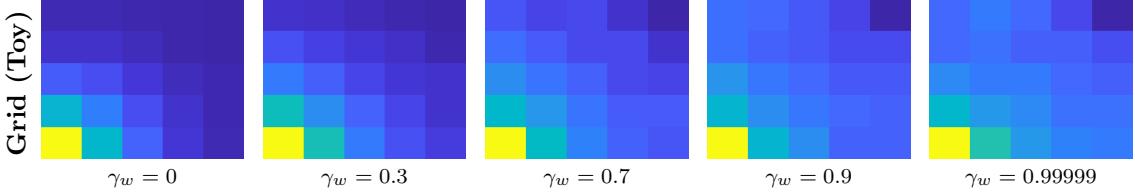
Figure 21: Visitation count for W_{N}^{β} at the end of the learning on the same random seed (out of 20). The higher the discount, the more uniform the count is. W_{UCB}^{β} performed very similarly.

reached. Thus initial states (which are fixed) are naturally visited more often. The count in our method uniformly decreases proportionally to the distance from the initial states. This denotes that at each episode, starting from the same state, the agent followed different paths, exploring new regions of the environment uniformly.

Other algorithms suffer from distractors and local optima.

In particular, UCB1 (third column) explores very slowly. The reason is that the agent myopically selects actions based on the *immediate* count, which makes exploration highly inefficient. By selecting the action with the lowest immediate count, the agent does not take into account where the action will lead it, i.e., if future states have already been visited or not. By contrast, our approach achieves deep exploration by taking into account the long-term visitation count of future states.

The performance of bootrstrapping (second column) is peculiar. It immediately converges to the first reward found in the gridworlds, barely exploring afterwards, but performs very well on the deep sea. This hints that bootstrapping may be sensitive to any kind of reward, including negative penalties. In the deep sea, in fact, diagonal states provide the only intermediate feedback beside the final reward/penalty. Coincidentially, traversing the diagonal also leads to the only positive reward. The agent may thus be guided by the reward received on the diagonal, even if it is a negative penalty. This behavior can be explained by recalling that Osband et al. [2019] regularize TD learning with the ℓ_2-distance from a prior Q-table. The agent may therefore be attracted to any state providing some kind of feedback in order to minimize the regularization.

4.2.2 Impact of Visitation Value Discount Factor γ_w

Here, we investigate the effect of the visitation discount factor γ_w on the "toy" gridworld (Figure 14). Figure 20 shows that the higher the discount, the better. As expected, with $\gamma_w = 0$ the algorithms behave very similarly to UCB1. In this case, in fact, the proposed approach and UCB1 are equivalent. However, the learning curves are not exactly the same because the W-function is updated with TD learning at every step.

The better exploration of our approach is also confirmed by Figure 21. The colormaps show the count at the end of the learning, and the higher γ_w, the more uniform the exploration is. With smaller γ_w, in fact, the agent rarely discovers states far from the initial position.
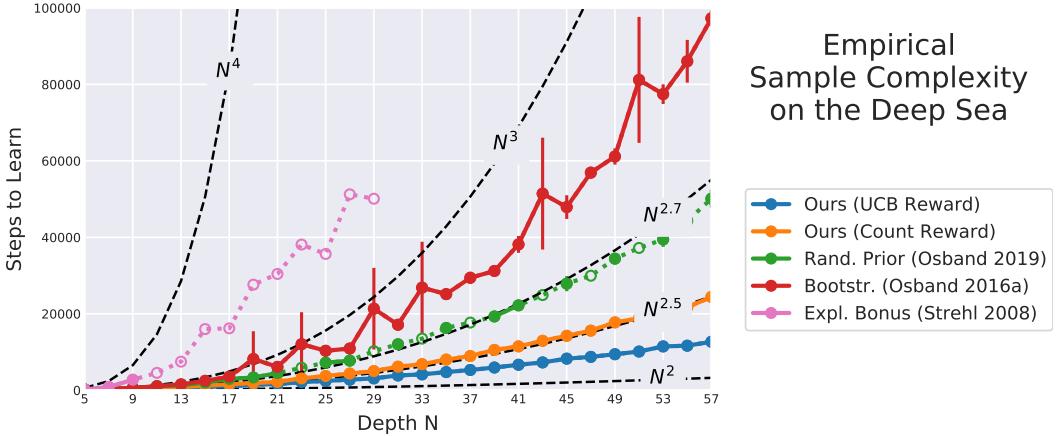
Figure 22: Steps to learn on varying the deep sea size $N = 5, 7, \ldots, 59$. Each filled dot denotes the average over ten runs with 500,000 steps limit. Error bars denote 95% confidence interval. In empty dots connected with dashed lines, the algorithm did not learn within steps limit at least once. In this case, the average is over converged runs only. Missing algorithms (random, ϵ-greedy, UCB1, Thompson sampling) performed poorly and are not reported. Only our algorithms learn with confidence interval close to zero, and attain the lowest sample complexity.

4.2.3 Empirical Sample Complexity on the Deep Sea

Here, we propose the same evaluation presented by Osband et al. [2018] to investigate the empirical sample complexity of the proposed algorithms, and to assess how they scale to large problems. Figure 22 plots training steps to learn the optimal policy as function of the environment size N. Only our approach (blue and orange) scales gracefully to large problem sizes. Results suggest an empirical scaling of $\mathcal{O}(N^{2.5})$ for the visitation-value-based with count reward, and even smaller for UCB reward. Bootstrapping attains an empirical complexity of $\mathcal{O}(N^3)$, confirming the findings of Osband et al. [2019]. However, in many cases ($N = 23, 27, 33, 37, 43, 47, 51, 55$) there was one seed for which bootstrapping either did not learn within the steps limit (green dashed line, empty dots) due to premature convergence, or learned after substantially more steps than the average (red dots, large error bar). Missing algorithms (random, ϵ-greedy, UCB1, approximate Thompson sampling) performed extremely poorly, often not learning within 500,000 steps even for small N, and are thus not reported.

4.2.4 Infinite Horizon Stochastic Chainworld

This evaluation investigates how the algorithms perform in a stochastic MDP with infinite horizon. In particular, we are interested in (1) the mean squared value error (MSVE) at each timestep, (2) the sample complexity when varying the number of states, and (3) how behavior policies explore and if they converge to the greedy policy.

Evaluation Criteria. The MSVE is computed between the true value function of the optimal policy $V^*(s)$ and the learned value function of the current policy $V^{\pi_t}(s)$, i.e.,

$$\text{MSVE} = \frac{1}{N} \sum_{i=1}^{N} (V^*(s_i) - V^{\pi_t}(s))^2. \tag{29}$$

The value function is computed according to the Bellman expectation equation in matrix form, i.e., $V^\pi = (I - \gamma \mathcal{P}^\pi)^{-1} R^\pi$, where I is the identity matrix, and $\mathcal{P}^\pi$ and R^π are the transition and reward functions induced by the policy, respectively [Sutton and Barto, 2018]. This error indicates how much the learned greedy policy π_t deviates from the optimal policy.

Similarly, the sample complexity is defined as the number of timesteps t such that the non-stationary policy π_t at time t is not ε-optimal for current state s_t, i.e, $V^*(s_t) - V^{\pi_t}(s_t) > \varepsilon$ [Strehl and Littman, 2008, Dong et al., 2019].

(a) Ours (UCB Reward) (b) Ours (Count Reward) (c) UCB1

(d) Expl. Bonus (e) Approx. Thompson Sampl. (f) ϵ-greedy
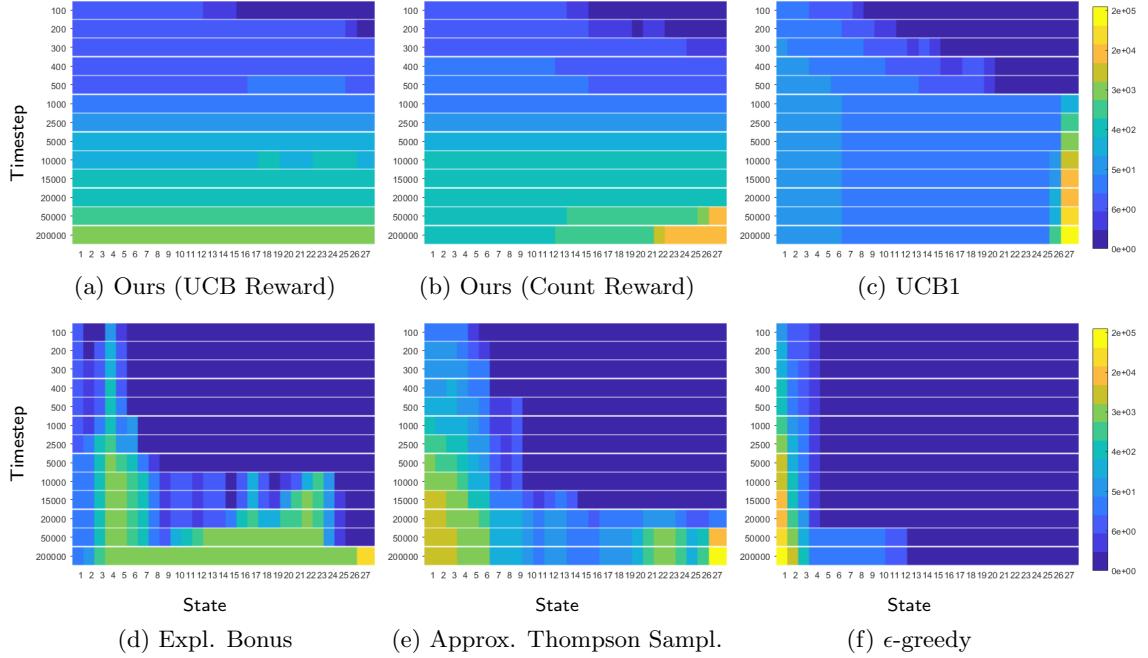
Figure 23: Visitation counts for the chainworld with 27 states (x-axis) over time (y-axis). As time passes (top to bottom), the visitation count grows. Only our algorithms and UCB1 explore uniformly, producing a uniform colormap. However, UCB1 finds the last state (with the reward) much later. Once UCB1 finds the reward (Figure 23c, step 1,000) the visitation count increses only in the last state. This suggests that when UCB1 finds the reward contained in the last state it effectively stops exploring other states. By contrast, our algorithms keep exploring the environment for longer. This allows the agent to learn the true Q-function for all states, and explains why our algorithms achieve lower sample complexity and MSVE in Figures 25 and 26.

For the behavior of exploration policies and their empirical convergence, we show how the visitation count changes over time.
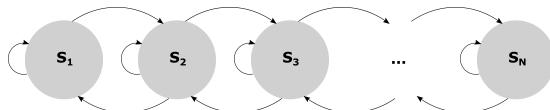
Figure 24: The ergodic chainworld.

MDP Characteristics. The MDP, shown in Figure 24, is a chainworld with $s = 1, \ldots, N$ states, three actions, and stochastic transition defined as follows. The first action moves the agent forward with probability p, backward otherwise. The second action moves the agent backward with probability p, forward otherwise. The third action keeps the agent in the current state with probability p, and randomly moves it backward or forward otherwise. The initial state is the leftmost state, and no state is terminal. This MDP is ergodic, i.e., all state are transient, positive recurrent and aperiodic for any deterministic policy. In our experiments, we set $p = 0.99$. The reward is 10^{-8} for doing "stay" in the initial state s_1, 1 for doing "stay" in the last state s_N, and 0 everywhere else.

Unlike previous MDPs, this has no terminal state, the horizon is infinite, and the state is never reset. Instead, to simulate infinite horizon, the agent explores the environment for one episode of 200,000 steps. To comply with the classic sample complexity definition [Strehl and Littman, 2008, Dong et al., 2019], we use classic Q-learning without replay memory. We do not compare against bootstrapping algorithms, because they are designed to select a different behavior Q-function at every episode, and this setup has only one episode. Approximate Thompson sampling by D'Eramo et al. [2019], instead, selects the Q-function at every step, and it is included in the comparison (without memory as well). All algorithms use zero Q-function initialization. Due to the stochasticity of the MDP, we increased the number of random seeds to 50.
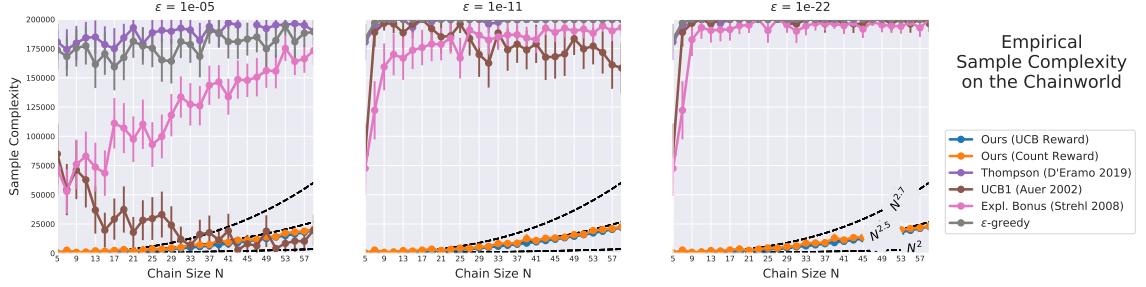
Figure 25: Sample complexity on varying the chain size $N = 5, 7, \ldots, 59$. Error bars denote 95% confidence interval. Only the proposed algorithms (blue and orange lines almost overlap) show little sensitivity to ε, as their sample complexity only slightly increases as ε decreases. By contrast, other algoritms are highly influenced by ε. Their estimate complexity has large confidence interval, and for smaller ε they almost never learn an ε-optimal policy.
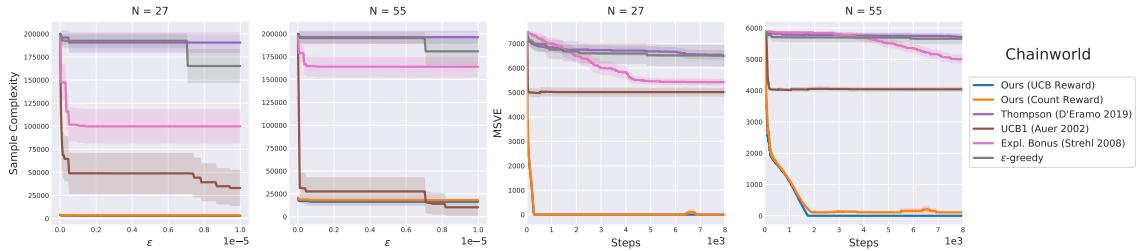
Figure 26: Sample complexity against ε, and MSVE over time for 27- and 55-state chainworlds. Shaded areas denote 95% confidence interval. As seen in Figure 25, the sample complexity of our approach is barely influenced by ε. Even for $\varepsilon = 0$, our algorithms attain low sample complexity, whereas other algorithms complexity is several orders of magnitude higher. Similarly, only our algorithms learn an almost perfect Q-function, achieving a MSVE close to zero.

Results. Figure 23 shows how the algorithms explore over time. Only ours (23a and 23b) and UCB1 (23c) explore uniformly, but UCB1 finds the last state (with the reward) later. The auxiliary rewards (23d) and approximate Thompson sampling (23e) also perform poorly, since the exploration is not uniform and the reward is found only late. ϵ-greedy exploration (23f), instead, is soon stuck in the local optimum represented by the small reward in the initial state.

The visitation count also shows that all exploration policies but ours act greedily once the reward is found. For example, UCB1 finds the reward around step 1,000, and after that its visitation count increses only in the last state (and nearby states, because of the stochastic transition). This suggests that when these algorithms find the reward contained in the last state they effectively stops exploring other states. By contrast, our algorithms explore the environment for longer, yielding a more uniform visitation count. This allows the agent to learn the true Q-function for all states, achieving lower sample complexity and MSVE as shown in Figures 25 and 26.

Figures 25 shows the sample complexity on varying the chain size over three values of ε. In all three cases, our algorithms scale gracefully with the number of states, suggesting a complexity of $\mathcal{O}(N^{2.5})$ and strengthening the findings of Section 4.2.3. By contrast, other algorithms performed poorly. Their sample samplexity has large confidence interval, and as ε decreases the complexity increases to the point that they almost never learn an ε-optimal policy.

These results are confirmed by Figure 26 (left plots), where only our algorithms attain low sample complexity even for $\varepsilon = 0$. Our algorithms are also the only ones to learn an almost perfect Q-function, with a MSVE close to zero (right plots). As anticipated, these results are explained by the uniform visitation count in Figure 23. By not converging to the greedy policy too quickly after discovering the reward, the agent keeps visiting old states and propagate to them the information about the reward, thus learning the true optimal Q-function for all states.
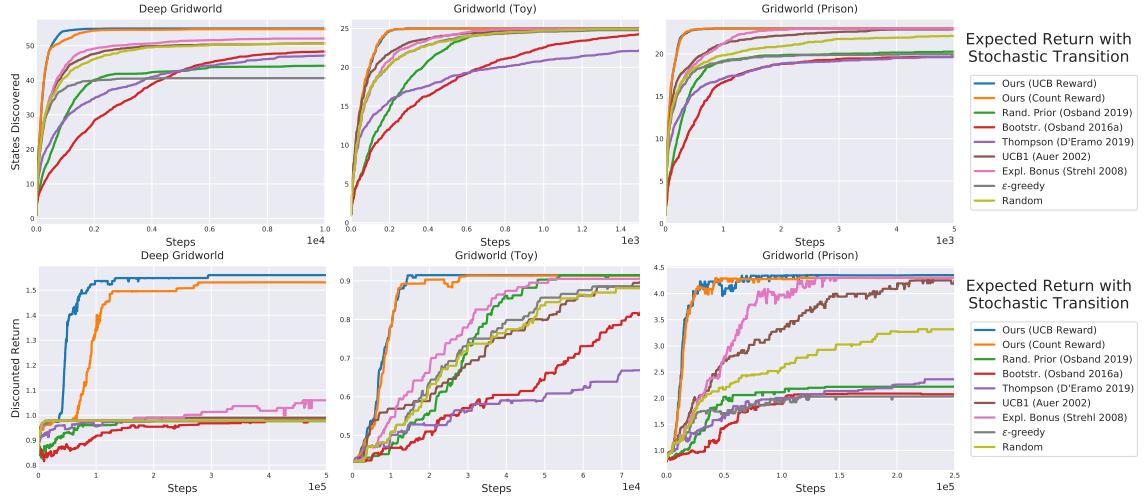
Figure 27: Results on three of the previous MDPs, but this time with stochastic transition function. Once again, only our algorithms visit all states and learn the optimal policy within steps limit in all MDPs, and their performance is barely affected by the stochasticity.

4.2.5 Stochastic Gridworlds

As we discussed in Section 3.4, the visitation rewards for training the W-functions penalize terminal state-action pairs. One may think that this would lead to poor exploration in stochastic environments, where the same state-action pair can lead to both terminal and non-terminal states. Here, we evaluate all algorithms again on some of the previous environments but this time with stochastic transition, and show that the proposed W-functions still solve the MDPs and outperform existing algorithms. Each transition (s, a) has probability p of succeeding, probability $1 - p/2$ of not moving the agent, and probability $1 - p/2$ of moving the agent to a random adjacent state. In our experiments, we set $p = 0.9$. Due to the stochasticity of the transition function, we increased the number of random seeds from 20 to 50.

Figure 27 shows the number of states discovered, and the expected discounted return computed in closed form as $V^\pi(s_0)$, where V^π is defined according to the Bellman expectation equation as $V^\pi = (I - \gamma \mathcal{P}^\pi)^{-1} R^\pi$. Similarly to the deterministic setting (Figures 12, 16, and 17, leftmost plots), our algorithms (blue and orange) outperform all baseline algorithms, being the only ones to solve all three MPDs within steps limit[6]. With the UCB reward (blue), our approach always learns the optimal policy. With count reward (orange), our approach almost always learns the optimal policy, and converges to a distractor only two times (out of 50) in the deep gridworld, and once in the "prison" gridworld. However, they both quickly discovered all states, as shown by top plots. The better performance of UCB-based reward is due to the optimistic initialization of the W-function. As seen in Section 4.2.4, Figure 23a, this version of the algorithm explores the environment for longer, and converges to the greedy policy much later. Therefore, the agent will visit the high-reward state more often, and it is less likely to learn suboptimal policies.
Bootstrapping with a prior (green) is not affected by the stochasticity, and it performs as in the deterministic setting (it solves the "toy" gridworld but not the other two). Vanilla bootstrapping (red), instead, is heavily affected by the stochasticity, and its performance is substantially worse compared to the deterministic setting (it cannot learn even the "toy" gridworld within steps limit). UCB1 (brown) and bonus-based exploration (pink), instead, benefit from the stochasticity. In the deterministic setting they could not solve these MDPs, while here they solve the two gridworlds (even though much later than ours). This improvement is explained by the larger number of visitated states (top plots), which denotes an overall better exploration. This is not surprising, if we consider that a stochastic transition function naturally helps exploration.

---

[6]Notice that the steps limit is higher compared to the deterministic setting. Because of the stochasticity, in fact, we decreased the learning rate from 0.5 to 0.1.

5 Conclusion and Future Work

Effective exploration with sparse rewards is an important challenge in RL, especially when sparsity is combined with the presence of "distractors", i.e., rewards that create suboptimal modes of the objective function. Classic algorithms relying on dithering exploration typically perform poorly, often converging to poor local optima or not learning at all. Methods based on immediate counts have strong guarantees but are empirically not sample efficient, while we showed that methods based on intrinsic auxiliary rewards require hand-tuning and are prone to suboptimal behavior. In this paper, we presented a novel approach that (1) plans exploration actions far into the future by using a long-term visitation count, and (2) decouples exploration and exploitation by learning a separate function assessing the exploration value of the actions. Contrary to existing methods which use models of reward and dynamics, our approach is off-policy and model-free. Empirical results showed that the proposed approach outperforms existing methods in environments with sparse and distracting rewards, and suggested that our approach scales gracefully with the size of the environment.

The proposed approach opens several avenues of research. First, in this work we focused on empirical results. In the future, we will investigate the theoretical properties of the proposed approach in more detail. Second, in this work we considered model-free RL. In the future, we will extend the proposed approach and combine it with model-based RL. Third, the experimental evaluation focused on identifying the challenges of learning with sparse and distracting rewards. In the future, we will consider more diverse tasks with continuous states and actions, and extend the proposed exploration to actor-critic methods.

References

Shipra Agrawal and Navin Goyal. Further optimal regret bounds for Thompson sampling. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2013.

Kavosh Asadi and Michael L Littman. An alternative softmax operator for reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2017.

Peter Auer and Ronald Ortner. Logarithmic online regret bounds for undiscounted reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 49–56, 2007.

Peter Auer, Nicoló Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002.

Marc G Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1471–1479, 2016.

Ronen I Brafman and Moshe Tennenholtz. R-MAX - A general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research (JMLR)*, 3(Oct): 213–231, 2002.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *International Conference on Learning Representations (ICLR)*, 2019.

Olivier Chapelle and Lihong Li. An empirical evaluation of Thompson sampling. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2249–2257, 2011.

Christoph Dann and Emma Brunskill. Sample complexity of episodic fixed-horizon reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2818–2826, 2015.

C. D'Eramo, A. Cini, and M. Restelli. Exploiting Action-Value uncertainty to drive exploration in reinforcement learning. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, 2019.

Carlo D'Eramo, Alessandro Nuara, Matteo Pirotta, and Marcello Restelli. Estimating the maximum expected value in continuous reinforcement learning problems. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017.

Kefan Dong, Yuanhao Wang, Xiaoyu Chen, and Liwei Wang. Q-learning with UCB exploration is sample efficient for Infinite-Horizon MDP, 2019.

Bradley Efron. *The Jackknife, the bootstrap and other resampling plans*. SIAM, 1982.

Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, et al. Noisy networks for exploration. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the International Conference on Machine learning (ICML)*, 2016.

Todd Hester and Peter Stone. TEXPLORE: Real-time sample-efficient reinforcement learning for robots. *Machine Learning*, 90(3):385–429, 2013.

Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. VIME: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1109–1117, 2016.

Thomas Jaksch, Ronald Ortner, and Peter Auer. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research (JMLR)*, 11(Apr):1563–1600, 2010.

Chi Jin, Zeyuan Allen-Zhu, Sebastien Bubeck, and Michael I Jordan. Is Q-learning provably efficient? In *Advances in Neural Information Processing Systems (NIPS)*, pages 4863–4873, 2018.

S.M. Kakade. *On the sample complexity of reinforcement learning*. PhD thesis, University College London, 2003.

Emilie Kaufmann, Nathaniel Korda, and Rémi Munos. Thompson sampling: An asymptotically optimal finite-time analysis. In *Proceedings of the International Conference on Algorithmic Learning Theory (ALT)*, 2012.

Michael Kearns and Satinder Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2-3):209–232, 2002.

J Zico Kolter and Andrew Y Ng. Near-Bayesian exploration in polynomial time. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 513–520. ACM, 2009.

T.L Lai and Herbert Robbins. Asymptotically efficient adaptive allocation rules. *Adv. Appl. Math.*, 6(1):4–22, Mar 1985.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. In *Proceedings of the NIPS Workshop on Deep Learning*, 2013.

Ian Osband. Risk versus uncertainty in deep learning: Bayes, bootstrap and the dangers of dropout. In *Proceedings of the NIPS Workshop on Bayesian Deep Learning*, 2016.

Ian Osband, Daniel Russo, and Benjamin Van Roy. (More) efficient reinforcement learning via posterior sampling. In *Advances in Neural Information Processing Systems (NIPS)*, 2013.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped DQN. In *Advances In Neural Information Processing Systems (NIPS)*, 2016a.

Ian Osband, Benjamin Van Roy, and Zheng Wen. Generalization and exploration via randomized value functions. In *Proceedings of the International Conference on Machine learning (ICML)*, 2016b.

Ian Osband, John Aslanides, and Albin Cassirer. Randomized prior functions for deep reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 8617–8629, 2018.

Ian Osband, Benjamin Van Roy, Daniel J. Russo, and Zheng Wen. Deep exploration via randomized value functions. *Journal of Machine Learning Research (JMLR)*, 20(124):1–62, 2019.

Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2017.

Deepak Pathak, Dhiraj Gandhi, and Abhinav Gupta. Self-Supervised exploration via disagreement. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the International Conference on Machine Learning (ICML)*, 2019.

Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y. Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz. Parameter space noise for exploration. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

Pascal Poupart, Nikos Vlassis, Jesse Hoey, and Kevin Regan. An analytic solution to discrete Bayesian reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 697–704, 2006.

Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, 1994. ISBN 0471619779.

Daniel Russo and Benjamin Van Roy. Learning to optimize via posterior sampling. *Mathematics of Operations Research*, 39(4):1221–1243, 2014.

Daniel Russo, David Tse, and Benjamin Van Roy. Time-sensitive bandit learning and satisficing Thompson sampling, 2017.

Daniel J Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, Zheng Wen, et al. A tutorial on Thompson sampling. *Foundations and Trends in Machine Learning*, 11(1):1–96, 2018.

Richard M Ryan and Edward L Deci. Intrinsic and extrinsic motivations: Classic definitions and new directions. *Contemporary Educational Psychology*, 25(1):54–67, 2000.

Jürgen Schmidhuber. A possibility for lmplementing curiosity and boredom in Model-Building neural controllers. In *Proceedings of the International Conference on Simulation of Adaptive Behavior (SAB)*, pages 222–227, 1991.

Jürgen Schmidhuber. Developmental robotics, optimal artificial curiosity, creativity, music, and the fine arts. *Connection Science*, 18(2):173–187, 2006.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

Steven L Scott. A modern Bayesian look at the multi-armed bandit. *Applied Stochastic Models in Business and Industry*, 26(6):639–658, 2010.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, 2017.

Bradly C Stadie, Sergey Levine, and Pieter Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. In *Proceedings of the NIPS Workshop on Deep Reinforcement Learning*, 2015.

Alexander L Strehl and Michael L Littman. An analysis of model-based interval estimation for Markov decision processes. *Journal of Computer and System Sciences (JCSS)*, 74(8):1309–1331, 2008.

Alexander L Strehl, Lihong Li, Eric Wiewiora, John Langford, and Michael L Littman. PAC model-free reinforcement learning. In *Proceedings of the International Conference on Machine learning (ICML)*, pages 881–888, 2006.

Malcolm Strens. A Bayesian framework for reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 943–950, 2000.

Richard S Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine Learning*, pages 216–224. Elsevier, 1990.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 2018.

Csaba Szepesvari. *Algorithms for Reinforcement Learning*, volume 4. Morgan & Claypool Publishers, 01 2010.

Davide Tateo, Carlo D'Eramo, Alessandro Nuara, Marcello Restelli, and Andrea Bonarini. Exploiting structure and uncertainty of Bellman updates in Markov decision processes. In *Proceedings of the Symposium Series on Computational Intelligence (SSCI)*, 2017.

William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

Hado van Hasselt. Double Q-learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2010.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.

A Experiment Details

Below, we present the pseudocode and the hyperparameters of the algorithms used in Section 4. In all of them, we kept two separate Q-tables for the behavior policy $\beta(a|s)$ and the target (greedy) policy $\pi(a|s)$. The former can be either initialized to zero or optimistically, while the latter always to zero. With the optimistic initialization, all entries of $Q(s,a)$ are set to $r_{\max}/(1-\gamma)$. The reason why we do not initialized $Q^\pi(s,a)$ optimistically is that if the agent does not visit all the state-action pairs, and thus never updated the corresponding Q-table entries, it would still had an optimistic belief over some states. The performance of the target policy can therefore be poor until all state-action pairs are visited. For example, in our experiments in the "prison" gridworld, the ϵ-greedy policy was finding some low-value treasures, but during the evaluation the greedy policy was not trying to collect them because it still had an optimistic belief over unvisited empty states.

In most of the pseudocode, we explicitly distinguish between $Q^\beta(s,a)$ and $Q^\pi(s,a)$. If simply $Q(s,a)$ appears, it means that both Q-tables are updated using the same equation.

In all algorithms, we evaluate $\pi(a|s)$, i.e., the greedy policy over $Q^\pi(s,a)$, every 50 training steps. Each evaluation episode has a horizon 10% longer than the training one. The learning rate is $\eta = 0.5$ and the discount factor $\gamma = 0.99$. For MDPs with stochastic transition function (Sections 4.2.4 and 4.2.5) we used $\eta = 0.1$. The ϵ-greedy initially has $\epsilon_0 = 1$, then it decays at step according to $\epsilon_{i+1} = \xi\epsilon_i$, where ξ is chosen such that $\epsilon_{end} = 0.1$ when the learning is over.

Finally we break ties with random selection, i.e., if two or more actions have the same max Q-value the winner is chosen randomly.

A.1 The Generic Q-Learning Scheme

In all experiments we used Q-learning with infinite replay memory. Classic Q-learning by Watkins and Dayan [1992] updates the Q-tables using only the current transition. The so-called *experience replay*, instead, keeps past transitions and use them multiple times for successive updates. This procedure became popular with deep Q-learning [Mnih et al., 2013] and brought substantial improvement to many RL algorithms.

In our experiments, we followed the setup proposed by Osband et al. [2019] and used an infinite memory, i.e., we stored all transitions, since it allowed much faster learning than classic Q-learning. This can also be seen as Dyna-Q [Sutton, 1990] without random memory sampling. In our experiments, the size of the domains does not pose any storage issue. When storage size is an issue, it is possible to fix the memory size and sample random mini-batches as in DQN [Mnih et al., 2013]. More details in Appendix A.5.

Algorithm 1 describes the generic scheme of Q-learning with infinite replay memory. TD learning is used on both $Q^\pi(s,a)$ and $Q^\beta(s,a)$, with the only difference being the initialization ($Q^\pi(s,a)$ is always initialized to zero).

Algorithm 1: Tabular Q-Learning with Replay Memory

---

1 Initialize $Q_0^\beta(s,a)$, $Q_0^\pi(s,a), i = 0$

2 While $i < i_{\text{BUDGET}}$ do

3   Reset environment to state s_1

4   For $t = 1...H$ or until s_t is terminal

5   Select action according to $a_t \sim \beta(\cdot|s_t)$

6   Transition to $s_{t+1} = \mathcal{P}(s_{t+1}|s_t, a_t)$ and receive reward $r_t = \mathcal{R}(s_t, a_t)$

7   Store tuple (s_t, a_t, r_t, s_{t+1})

8   For all tuples (s, a, s', r) in the replay memory

9 $\delta(s,a,s') = \begin{cases} r_t - Q_i(s,a) & \text{if } s \text{ is terminal} \\ r_t + \gamma \max_a Q_i(s',a) - Q_i(s,a) & \text{otherwise} \end{cases}$

10 $Q_{i+1}(s,a) \leftarrow Q_i(s,a) + \eta\delta(s,a,s')$

11   Update budget counter: $i \leftarrow i + 1$

Depending on $\beta(a|s)$, we have different exploration strategies, i.e.,

$$\beta(a_t|s_t) \sim \text{unif}\{\mathcal{A}\}, \qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{random} \qquad (30)$$

$$\beta(a_t|s_t) \begin{cases} = \arg\max_a \left\{ Q^\beta(s_t, a) \right\} & \text{with probability } 1-\epsilon, \\ \sim \text{unif}\{\mathcal{A}\} & \text{with probability } \epsilon, \end{cases} \qquad \epsilon\text{-greedy} \qquad (31)$$

$$\beta(a_t|s_t) = \arg\max_a \left\{ Q^\beta(s_t, a) + \kappa \sqrt{\frac{2\log \sum_{a_j} n(s_t, a_j)}{n(s_t, a)}} \right\}. \qquad \text{UCB1} \qquad (32)$$

Numerical stability and worst-case scenario. For UCB1 [Auer et al., 2002], a visitation count $n(s,a)$ is increased after every transition (see Algorithm 2, line 7). In classic bandit problems, UCB1 is initialized by executing all actions once, i.e., with $n(s,a) = 1$. In MDPs we cannot do that, i.e., we cannot arbitrarily set the agent in any state and execute all actions, thus $n(s,a) = 0$. Following the W-function bound in Section 3.3, we always add +1 inside the logarithm, and bound the square root to $(Q_{\max} - Q_{\min})/\kappa + \sqrt{2\log|\mathcal{A}|}$ when $n(s,a) = 0$. This correspond to the case where all actions but $\bar{a}$ has been executed once, and enforces the policy to choose $\bar{a}$. In our experiments, we set $Q_{\max} = r_{\max}/(1-\gamma)$ and $Q_{\min} = 0$.

A.2 Q-Learning with Augmented Reward

This version follows the same scheme of Algorithm 1 with ϵ-greedy exploration. The only difference is that the reward used to train the behavior policy is augmented with the exploration bonus proposed by Strehl and Littman [2008]. In our experiments, $\alpha = 0.1$, as used by Strehl and Littman [2008] and Bellemare et al. [2016].

Algorithm 2: Tabular Q-Learning with Replay Memory and Augmented Reward

---

1   Initialize $Q_0^\beta(s,a)$, $Q_0^\pi(s,a) = 0, n(s,a) = 0, i = 0$

2   While $i < i_{\text{BUDGET}}$ do

3   Reset environment to state s_1

4   For $t = 1...H$ or until s_t is terminal

5   Select action according to the ϵ-greedy policy of $Q_i^\beta(s,a)$

6   Transition to $s_{t+1} = \mathcal{P}(s_{t+1}|s_t, a_t)$ and receive reward $r_t = \mathcal{R}(s_t, a_t)$

7   Update visitation count: $n(s_t, a_t) \leftarrow n(s_t, a_t) + 1$

8   Store tuple (s_t, a_t, r_t, s_{t+1})

9   For all tuples (s, a, s', r) in the replay memory

10            $\delta^\pi(s,a,s') = \begin{cases} r_t - Q_i^\pi(s,a) & \text{if } s \text{ is terminal} \\ r_t + \gamma \max_a Q_i^\pi(s',a) - Q_i^\pi(s,a) & \text{otherwise} \end{cases}$

11            $Q_{i+1}^\pi(s,a) \leftarrow Q_i^\pi(s,a) + \eta \delta^\pi(s,a,s')$

12   Augment reward: $r_t^+ = r_t + \alpha\, n(s_t, a_t)^{-1/2}$

13            $\delta^\beta(s,a,s') = \begin{cases} r_t^+ - Q_i^\beta(s,a) & \text{if } s \text{ is terminal} \\ r_t^+ + \gamma \max_a Q_i^\beta(s',a) - Q_i^\beta(s,a) & \text{otherwise} \end{cases}$

14            $Q_{i+1}^\beta(s,a) \leftarrow Q_i^\beta(s,a) + \eta \delta^\beta(s,a,s')$

15   Update budget counter: $i \leftarrow i + 1$

A.3 Q-Learning with Visitation Value

Algorithms 3 and 4 describe the novel method proposed in this paper. In our experiments $\kappa = r_{\max}/(1-\gamma)$ and $\gamma_w = 0.99$. For the chainworld in Section 4.2.4 we set $\gamma_w = 0.999$. With infinite horizon, in fact, this yielded better results by allowing the agent to explore for longer. For the

stochastic gridworld in Section 4.2.5 we set $\gamma_w = 0.9$. As discussed in the results, in fact, a stochastic transition function naturally improves exploration, thus a smaller visitation discount was sufficient and yielded better results.

In Algorithm 3, $W_{\text{UCB}}^{\beta}(s, a)$ is initialized as Eq. (23). In Algorithm 4, line 6, we bound the square root to Eq. (26) when $n(s, a) = 0$. In our experiments, with $Q_{\max} = r_{\max}/(1 - \gamma)$ and $Q_{\min} = 0$.

Algorithm 3: Tabular Q-Learning with Replay Memory and Visit. Value (UCB)

1 Initialize $Q_0^{\beta}(s, a)$, $Q_0^{\pi}(s, a) = 0, W_{\text{UCB},0}^{\beta}(s, a), n(s, a) = 0, i = 0$

2 While $i < i_{\text{BUDGET}}$ do

3   Reset environment to state s_1

4   For $t = 1...H$ or until s_t is terminal

5   Select action according to $\beta(a_t|s_t) = \arg\max_a \left\{ Q_i^{\beta}(s_t, a) + \kappa(1 - \gamma_w) W_{\text{UCB},i}^{\beta}(s_t, a) \right\}$

6   Transition to $s_{t+1} = \mathcal{P}(s_{t+1}|s_t, a_t)$ and receive reward $r_t = \mathcal{R}(s_t, a_t)$

7   Update visitation count: $n(s_t, a_t) \leftarrow n(s_t, a_t) + 1$

8   Store tuple (s_t, a_t, r_t, s_{t+1})

9   For all tuples (s, a, s', r) in the replay memory

10 $\delta(s, a, s') = \begin{cases} r_t - Q_i(s, a) & \text{if } s \text{ is terminal} \\ r_t + \gamma \max_a Q_i(s', a) - Q_i(s, a) & \text{otherwise} \end{cases}$

11 $Q_{i+1}(s, a) \leftarrow Q_i(s, a) + \eta\delta(s, a, s')$

12   Compute visitation reward according to Eq. (10)

13 $\delta^W(s, a, s') = \begin{cases} r_t^W - W_{\text{UCB},i}^{\beta}(s, a) & \text{if } s \text{ is terminal} \\ r_t^W + \gamma \max_a W_{\text{UCB},i}^{\beta}(s', a) - W_{\text{UCB},i}^{\beta}(s, a) & \text{otherwise} \end{cases}$

14 $W_{\text{UCB},i+1}^{\beta}(s, a) \leftarrow W_{\text{UCB},i}^{\beta}(s, a) + \eta\delta^W(s, a, s')$

15   Update budget counter: $i \leftarrow i + 1$

Algorithm 4: Tabular Q-Learning with Replay Memory and Visit. Value (Count)

1 Initialize $Q_0^{\beta}(s, a)$, $Q_0^{\pi}(s, a) = 0, n(s, a) = 0, W_{\text{N},0}^{\beta}(s, a) = 0, i = 0$

2 While $i < i_{\text{BUDGET}}$ do

3   Reset environment to state s_1

4   For $t = 1...H$ or until s_t is terminal

5   Compute pseudocount: $\hat{n}(s_t, a) = (1 - \gamma_w) W_{\text{N},i}^{\beta}(s_t, a)$

6   Select action according to $\beta(a_t|s_t) = \arg\max_a \left\{ Q_i^{\beta}(s_t, a) + \kappa \sqrt{\frac{2 \log \sum_{a_j} \hat{n}(s_t, a_j)}{\hat{n}(s_t, a)}} \right\}$

7   Transition to $s_{t+1} = \mathcal{P}(s_{t+1}|s_t, a_t)$ and receive reward $r_t = \mathcal{R}(s_t, a_t)$

8   Update visitation count: $n(s_t, a_t) \leftarrow n(s_t, a_t) + 1$

9   Store tuple (s_t, a_t, r_t, s_{t+1})

10   For all tuples (s, a, s', r) in the replay memory

11 $\delta(s, a, s') = \begin{cases} r_t - Q_i(s, a) & \text{if } s \text{ is terminal} \\ r_t + \gamma \max_a Q_i(s', a) - Q_i(s, a) & \text{otherwise} \end{cases}$

12 $Q_{i+1}(s, a) \leftarrow Q_i(s, a) + \eta\delta(s, a, s')$

13   Compute visitation reward according to Eq. (15)

14 $\delta^W(s, a, s') = \begin{cases} r_t^W - W_{\text{N},i}^{\beta}(s, a) & \text{if } s \text{ is terminal} \\ r_t^W + \gamma \min_a W_{\text{N},i}^{\beta}(s', a) - W_{\text{N},i}^{\beta}(s, a) & \text{otherwise} \end{cases}$

15 $W_{\text{N},i+1}^{\beta}(s, a) \leftarrow W_{\text{N},i}^{\beta}(s, a) + \eta\delta^W(s, a, s')$

16   Update budget counter: $i \leftarrow i + 1$

A.4 Bootstrapped Q-Learning

Bootstrap algorithms have an ensemble of Q-tables $\{Q^b(s,a)\}_{b=1...B}$. The behavior policy is greedy over one Q-table, randomly chosen from the ensemble either at the beginning of the episode or at every step.

The behavior Q-tables are initialized randomly. After initializing each $Q^b(s,a)$ either optimistically or to zero, we add random noise drawn from a Gaussian $\mathcal{N}(0,1)$. Then, each one is trained on a random mini-batch from the replay memory. In our experiments, we used batches of size 1,024 and an ensemble of $B = 10$ behavior Q-tables. The target Q-table is still updated as in previous algorithms, i.e., using the full memory.

Algorithm 5: Tabular Bootstrapped Q-Learning with Replay Memory

---

1 Initialize $Q_0^b(s,a)$ for $b = 1 \ldots B$, $Q_0^\pi(s,a) = 0, i = 0$

2 While $i < i_{\text{BUDGET}}$ do

3   Reset environment to state s_1

4   Select random Q-table: $Q_i^\beta(s,a) \sim \text{unif}\{Q_i^1(s,a), \ldots, Q_i^B(s,a)\}$

5   For $t = 1...H$ or until s_t is terminal

6   Select action according to the ϵ-greedy policy of $Q_i^\beta(s,a)$

7   Transition to $s_{t+1} = \mathcal{P}(s_{t+1}|s_t, a_t)$ and receive reward $r_t = \mathcal{R}(s_t, a_t)$

8   Store tuple (s_t, a_t, r_t, s_{t+1})

9   For all tuples (s,a,s',r) in the replay memory

10 $\delta^\pi(s,a,s') = \begin{cases} r_t - Q_i^\pi(s,a) & \text{if } s \text{ is terminal} \\ r_t + \gamma \max_a Q_i^\pi(s',a) - Q_i^\pi(s,a) & \text{otherwise} \end{cases}$

11 $Q_{i+1}^\pi(s,a) \leftarrow Q_i^\pi(s,a) + \eta \delta^\pi(s,a,s')$

12   For all behavior Q-tables $Q_i^b(s,a)$

13   Sample mini-batch from the replay memory

14   For all tuples (s,a,s',r) in the mini-batch

15 $\delta^b(s,a,s') = \begin{cases} r_t - Q_i^b(s,a) & \text{if } s \text{ is terminal} \\ r_t + \gamma \max_a Q_i^b(s',a) - Q_i^b(s,a) & \text{otherwise} \end{cases}$

16 $Q_{i+1}^b(s,a) \leftarrow Q_i^b(s,a) + \eta \delta^b(s,a,s')$

17   Update budget counter: $i \leftarrow i + 1$

The above pseudocode defines the generic bootstrap approach proposed by Osband et al. [2016a]. In Section 4 we also compared to two slightly different versions. The first uses approximate Thompson sampling and randomly selects the behavior Q-table at every step instead of at the beginning of the episode [D'Eramo et al., 2019]. The second keeps the sampling at the beginning of the episode, but further regularizes the TD error [Osband et al., 2018, 2019]. The regularization is the squared ℓ_2-norm of the distance of the Q-tables from "prior Q-tables" $Q^p(s,a)$, resulting in the following regularized TD update

$$\delta_{reg}^b(s,a,s') = \delta^b(s,a,s') + \nu \left(Q^p(s,a) - Q^b(s,a) \right), \tag{33}$$

wehere ν is the regularization coefficient which we set to $\nu = 0.1$ (in the original paper $\nu = 1$, but dividing it by ten worked best in our experiments).

In theory, $Q^p(s,a)$ should be drawn from a distribution. In practice, the same authors fix it at the beginning of the learning, and for the deep sea domain they set it to zero. This configuration also worked best in our experiments.

Horizons. All environments in Section 4 are infinite horizon MDPs, except for the deep sea which has a finite horizon equal to its depth. However, for practical reason we end the episode after H steps and reset the agent to the initial state. Table 2 summarizes the training horizon H for each environment, as well as the steps needed for the optimal policy to find the highest reward. Also, recall that an episode can end prematurely if a terminal state (e.g., a reward state) is reached.

Finally, notice that the agent receives the reward on state-action transition, i.e., it needs to execute an additional action in the state with a treasure to receive the reward. This is why, for instance, the agent needs 9 steps instead of 8 to be rewarded in the 5×5 gridworlds.

Table 2: Time horizons H for the tabular MDPs presented in Section 4.

	Deep Sea	Taxi	Deep Grid.	Grid. (Toy)	Grid. (Prison)	Grid. (Wall)
Optimal	Depth	29	11	9	9	135
Short H.	Depth	33	55	11	11	330
Long H.	Depth	66	110	22	22	660
Stochastic	-	-	55	15	25	-

A.5 Infinite Memory vs No Memory

Except for the chainworld in Section 4.2.4, the evaluation in Section 4 was conducted with Q-learning with infinite replay memory, following the same setup of Osband et al. [2019] as described in Section A.1. Here, we present additional results without replay memory, i.e., using classic Q-learning by Watkins and Dayan [1992] which updates the Q-tables using only the current transition. We show that the proposed method benefits the most from the replay memory, but can learn even without it while other algorithms cannot. We compare ϵ-greedy exploration, our proposed visitation-value-based exploration, and bootstrapped exploration. Nevertheless, the latter needs a replay memory to randomize the Q-tables update, as each Q-table is updated using different random samples. Thus, for bootstrapped Q-learning, we compare the use of finite memory and small batches to infinite memory and large batches..

Algorithm 6 describes the generic scheme of Q-learning without replay memory. The only difference from Algorithm 1 is the absence of the loop over the memory. The same modification is done to Algorithms 4 and 5 to derive the version of our algorithms without infinite memory. Boostrapped Q-learning is the same as Algorithm 3 but with finite memory. The finite memory keeps at most $20H$ steps, where H is the episode horizon, and uses mini-batches of 32 samples instad of 1,024.

Algorithm 6: Classic Tabular Q-Learning

---

1 Initialize $Q_0^\beta(s,a)$, $Q_0^\pi(s,a)$, $i = 0$

2 While $i < i_{\text{BUDGET}}$ do

3   Reset environment to state s_1

4   For $t = 1...H$ or until s_t is terminal

5   Select action according to $a_t \sim \beta(\cdot|s_t)$

6   Transition to $s_{t+1} = \mathcal{P}(s_{t+1}|s_t, a_t)$ and receive reward $r_t = \mathcal{R}(s_t, a_t)$

7   Store tuple (s_t, a_t, r_t, s_{t+1})

8 $\delta(s_t, a_t, s_{t+1}) = \begin{cases} r_t - Q_i(s_t, a_t) & \text{if } s_t \text{ is terminal} \\ r_t + \gamma \max_a Q_i(s_{t+1}, a) - Q_i(s_t, a_t) & \text{otherwise} \end{cases}$

9 $Q_{i+1}(s_t, a_t) \leftarrow Q_i(s_t, a_t) + \eta\delta(s_t, a_t, s_{t+1})$

10   Update budget counter: $i \leftarrow i + 1$
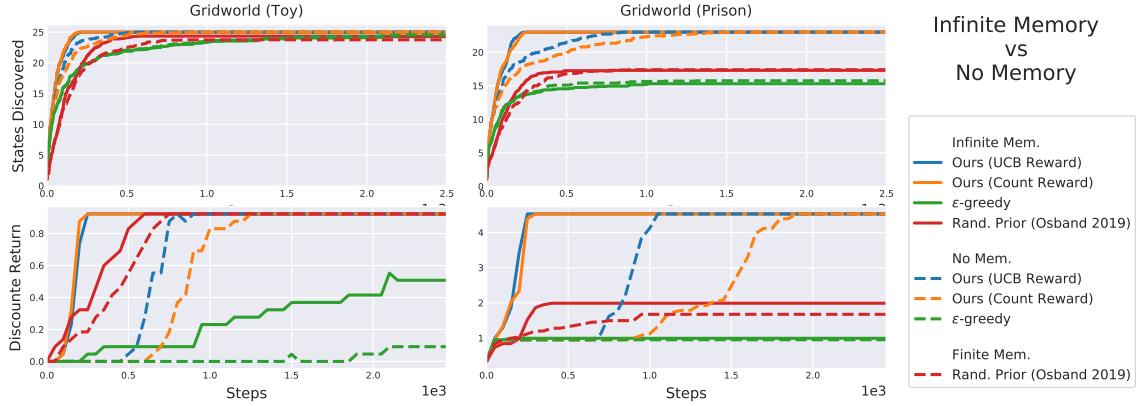
Figure 28: Comparison of the proposed exploration against ϵ-greedy and bootstrapped exploration with and without replay memory. Each line denotes the average over 20 seeds. Only exploration based on the visitation value always learns in both domains, i.e., both with and withou local optima (distractor rewards), regardless of the memory.

Results. Figure 28 shows the results on two gridworlds. The "toy" one has a single reward, while the "prison" one has also distractors. We refer to Section 4.1 for their full description. In the first domain, all algorithms perform well except for ϵ-greedy exploration (green), which missed the reward in some trials. The use of infinite memory helped all algorithms, allowing boostrap and visitation-value-based to learn substantially faster. For example on average, without replay memory our algorithms (blue and orange) converged in 1,000 steps, while with replay memory they took only 250 steps, i.e., $\times 4$ times faster. Bootstrapped Q-learning (red), performance does not change substantially with finite and infinite memory.

However, in the second domain only visitation-value-based exploration always learns, regardless of the memory. As already discussed in Section 4.1, bootstrap performs poorly due to distractor rewards, and the infinite memory does not help it. For visitation-value-based exploration, the speed up gained from the infinite memory is even larger than before, due to the higher complexity of the domain. In this case, in fact, with infinite memory learn $\times 4$ and $\times 8$ times faster than without memory (blue and orange, respectively).

This evaluation shows that the proposed method benefits the most from the replay memory, but can learn even without it while other algorithms cannot.

A.6   Recap Tables

Here, we report tables summarizing the results of Section 4.1 in terms of discovery (percentage of states discovered during learning) and success (number of times the algorithm learned the optimal policy within steps limit). These are extended version of Table 1, which reported results only for the "zero initialization short horizon" scenario.

Table 3: Deep sea results recap.

	Algorithm	Discovery (%)	Success (%)
	Ours (UCB Reward)	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Ours (Count Reward)	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Rand. Prior (Osband 2019)	99.90 ± 0.01	100 ± 0
	Bootstr. (Osband 2016a)	99.77 ± 0.05	100 ± 0
Zero Init.	Bootstr. (D'Eramo 2019)	63.25 ± 3.31	0 ± 0
	UCB1 (Auer 2002)	55.72 ± 0.34	0 ± 0
	Expl. Bonus (Strehl 2008)	85.65 ± 1.0	0 ± 0
	ϵ-greedy	57.74 ± 1.11	0 ± 0
	Random	58.59 ± 1.35	0 ± 0
	Ours (UCB Reward)	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Ours (Count Reward)	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Rand. Prior (Osband 2019)	100 ± 0	100 ± 0
	Bootstr. (Osband 2016a)	100 ± 0	100 ± 0
Opt. Init.	Bootstr. (D'Eramo 2019)	100 ± 0	100 ± 0
	UCB1 (Auer 2002)	76.85 ± 0.3	0 ± 0
	Expl. Bonus (Strehl 2008)	98.79 ± 0.21	0 ± 0
	ϵ-greedy	98.79 ± 0.20	0 ± 0
	Random	58.59 ± 1.35	0 ± 0

Table 4: Taxi results recap.

	Algorithm	Discovery (%)	Success (%)
	Ours (UCB Reward)	**100 ± 0**	**100 ± 0**
	Ours (Count Reward)	**100 ± 0**	**100 ± 0**
	Rand. Prior (Osband 2019)	69.60 ± 2.96	13.07 ± 2.96
	Bootstr. (Osband 2016a)	52.44 ± 7.55	18.77 ± 9.24
Zero Init. & Short Hor.	Bootstr. (D'Eramo 2019)	22.44 ± 1.81	1.9 ± 1.47
	UCB1 (Auer 2002)	31.17 ± 0.70	1.53 ± 1.37
	Expl. Bonus (Strehl 2008)	74.62 ± 2.24	17.6 ± 2.56
	ϵ-greedy	29.64 ± 0.98	1.92 ± 1.49
	Random	29.56 ± 0.98	1.92 ± 1.49
	Ours (UCB Reward)	**100 ± 0**	**100 ± 0**
	Ours (Count Reward)	**100 ± 0**	**100 ± 0**
	Rand. Prior (Osband 2019)	100 ± 0	100 ± 0
	Bootstr. (Osband 2016a)	100 ± 0	100 ± 0
Opt. Init. & Short Hor.	Bootstr. (D'Eramo 2019)	100 ± 0	100 ± 0
	UCB1 (Auer 2002)	99.94 ± 0.08	100 ± 0
	Expl. Bonus (Strehl 2008)	100 ± 0	100 ± 0
	ϵ-greedy	100 ± 0	100 ± 0
	Random	29.56 ± 0.98	1.92 ± 1.49
	Ours (UCB Reward)	**100 ± 0**	**100 ± 0**
	Ours (Count Reward)	**100 ± 0**	**100 ± 0**
	Rand. Prior (Osband 2019)	76.38 ± 2.31	15.69 ± 2.93
	Bootstr. (Osband 2016a)	53.80 ± 6.63	29.85 ± 12.72
Zero Init. & Long Hor.	Bootstr. (D'Eramo 2019)	36.34 ± 3.16	6.54 ± 2.62
	UCB1 (Auer 2002)	49.66 ± 2.45	8.35 ± 1.32
	Expl. Bonus (Strehl 2008)	92.83 ± 2.26	76.18 ± 16.09
	ϵ-greedy	42.26 ± 2.52	7.66 ± 0.03
	Random	55.89 ± 0.98	2.06 ± 1.49
	Ours (UCB Reward)	**100 ± 0**	**100 ± 0**
	Ours (Count Reward)	**100 ± 0**	**100 ± 0**
	Rand. Prior (Osband 2019)	100 ± 0	100 ± 0
	Bootstr. (Osband 2016a)	100 ± 0	100 ± 0
Opt. Init. & Long Hor.	Bootstr. (D'Eramo 2019)	95.45 ± 0	100 ± 0
	UCB1 (Auer 2002)	98.74 ± 0.09	100 ± 0
	Expl. Bonus (Strehl 2008)	100 ± 0	100 ± 0
	ϵ-greedy	100 ± 0	100 ± 0
	Random	55.89 ± 0.98	2.06 ± 1.49

Table 5: Deep gridworld results recap.

	Algorithm	Discovery (%)	Success (%)
	Ours (UCB Reward)	**100 ± 0**	**100 ± 0**
	Ours (Count Reward)	**100 ± 0**	**100 ± 0**
	Rand. Prior (Osband 2019)	75 ± 9.39	59.03 ± 4.2
	Bootstr. (Osband 2016a)	60.82 ± 11.78	63.35 ± 6.89
Zero Init. & Short Hor.	Bootstr. (D'Eramo 2019)	63.73 ± 10.35	56.85 ± 0.06
	UCB1 (Auer 2002)	92.18 ± 0.36	56.88 ± 0
	Expl. Bonus (Strehl 2008)	95.45 ± 1.57	69.81 ± 8.84
	ϵ-greedy	74.36 ± 4.42	56.88 ± 0
	Random	92.45 ± 0.64	56.88 ± 0
	Ours (UCB Reward)	**100 ± 0**	**100 ± 0**
	Ours (Count Reward)	**100 ± 0**	**100 ± 0**
	Rand. Prior (Osband 2019)	100 ± 0	100 ± 0
	Bootstr. (Osband 2016a)	100 ± 0	100 ± 0
Opt. Init. & Short Hor.	Bootstr. (D'Eramo 2019)	100 ± 0	100 ± 0
	UCB1 (Auer 2002)	100 ± 0	100 ± 0
	Expl. Bonus (Strehl 2008)	100 ± 0	100 ± 0
	ϵ-greedy	100 ± 0	100 ± 0
	Random	92.45 ± 0.64	56.88 ± 0
	Ours (UCB Reward)	**100 ± 0**	**100 ± 0**
	Ours (Count Reward)	**100 ± 0**	**100 ± 0**
	Rand. Prior (Osband 2019)	78.18 ± 8.97	56.88 ± 0
	Bootstr. (Osband 2016a)	77.64 ± 10.02	69.81 ± 8.84
Zero Init. & Long Hor.	Bootstr. (D'Eramo 2019)	71.64 ± 10.35	56.85 ± 0.06
	UCB1 (Auer 2002)	95.54 ± 0.46	56.4 ± 3.12
	Expl. Bonus (Strehl 2008)	95 ± 1.66	65.5 ± 7.71
	ϵ-greedy	76.45 ± 4.25	56.88 ± 0
	Random	92.09 ± 0.69	56.88 ± 0
	Ours (UCB Reward)	**100 ± 0**	**100 ± 0**
	Ours (Count Reward)	**100 ± 0**	**100 ± 0**
	Rand. Prior (Osband 2019)	100 ± 0	100 ± 0
	Bootstr. (Osband 2016a)	100 ± 0	100 ± 0
Opt. Init. & Long Hor.	Bootstr. (D'Eramo 2019)	100 ± 0	100 ± 0
	UCB1 (Auer 2002)	100 ± 0	100 ± 0
	Expl. Bonus (Strehl 2008)	100 ± 0	100 ± 0
	ϵ-greedy	100 ± 0	100 ± 0
	Random	92.09 ± 0.69	56.88 ± 0

Table 6: Gridworld (toy) results recap.

	Algorithm	Discovery (%)	Success (%)
	Ours (UCB Reward)	**100 ± 0**	**100 ± 0**
	Ours (Count Reward)	**100 ± 0**	**100 ± 0**
	Rand. Prior (Osband 2019)	99.8 ± 0.39	100 ± 0
	Bootstr. (Osband 2016a)	99.2 ± 0.72	100 ± 0
Zero Init. & Short Hor.	Bootstr. (D'Eramo 2019)	84.8 ± 4.33	40 ± 21.91
	UCB1 (Auer 2002)	97.4 ± 0.99	50 ± 22.49
	Expl. Bonus (Strehl 2008)	99.8 ± 0.39	100 ± 0
	ϵ-greedy	97.4 ± 1.17	45 ± 22.25
	Random	97.2 ± 1.15	45 ± 22.25
	Ours (UCB Reward)	**100 ± 0**	**100 ± 0**
	Ours (Count Reward)	**100 ± 0**	**100 ± 0**
	Rand. Prior (Osband 2019)	100 ± 0	100 ± 0
	Bootstr. (Osband 2016a)	100 ± 0	100 ± 0
Opt. Init. & Short Hor.	Bootstr. (D'Eramo 2019)	100 ± 0	100 ± 0
	UCB1 (Auer 2002)	100 ± 0	100 ± 0
	Expl. Bonus (Strehl 2008)	100 ± 0	100 ± 0
	ϵ-greedy	100 ± 0	100 ± 0
	Random	97.2 ± 1.15	45 ± 22.25
	Ours (UCB Reward)	**100 ± 0**	**100 ± 0**
	Ours (Count Reward)	**100 ± 0**	**100 ± 0**
	Rand. Prior (Osband 2019)	100 ± 0	100 ± 0
	Bootstr. (Osband 2016a)	99.8 ± 0.39	100 ± 0
Zero Init. & Long Hor.	Bootstr. (D'Eramo 2019)	94.6 ± 1.9	70 ± 20.49
	UCB1 (Auer 2002)	100 ± 0	98.5 ± 0.22
	Expl. Bonus (Strehl 2008)	100 ± 0	100 ± 0
	ϵ-greedy	100 ± 0	100 ± 0
	Random	100 ± 0	100 ± 0
	Ours (UCB Reward)	**100 ± 0**	**100 ± 0**
	Ours (Count Reward)	**100 ± 0**	**100 ± 0**
	Rand. Prior (Osband 2019)	100 ± 0	100 ± 0
	Bootstr. (Osband 2016a)	100 ± 0	100 ± 0
Opt. Init. & Long Hor.	Bootstr. (D'Eramo 2019)	100 ± 0	100 ± 0
	UCB1 (Auer 2002)	100 ± 0	100 ± 0
	Expl. Bonus (Strehl 2008)	100 ± 0	100 ± 0
	ϵ-greedy	100 ± 0	100 ± 0
	Random	100 ± 0	100 ± 0

Table 7: Gridworld (Prison) results recap.

	Algorithm	Discovery (%)	Success (%)
	Ours (UCB Reward)	**100 ± 0**	**100 ± 0**
	Ours (Count Reward)	**100 ± 0**	**100 ± 0**
	Rand. Prior (Osband 2019)	72.39 ± 6.9	44.44 ± 14.69
	Bootstr. (Osband 2016a)	64.35 ± 11.1	33.03 ± 8.54
Zero Init. & Short Hor.	Bootstr. (D'Eramo 2019)	54.78 ± 6.69	25.61 ± 8.39
	UCB1 (Auer 2002)	80.78 ± 2.56	32.31 ± 4.11
	Expl. Bonus (Strehl 2008)	85.87 ± 3	46.96 ± 12.35
	ϵ-greedy	58.38 ± 5.27	22.84 ± 2.48
	Random	76.96 ± 3.08	35.36 ± 10.37
	Ours (UCB Reward)	**100 ± 0**	**100 ± 0**
	Ours (Count Reward)	**100 ± 0**	**100 ± 0**
	Rand. Prior (Osband 2019)	100 ± 0	100 ± 0
	Bootstr. (Osband 2016a)	100 ± 0	100 ± 0
Opt. Init. & Short Hor.	Bootstr. (D'Eramo 2019)	100 ± 0	100 ± 0
	UCB1 (Auer 2002)	100 ± 0	100 ± 0
	Expl. Bonus (Strehl 2008)	100 ± 0	100 ± 0
	ϵ-greedy	100 ± 0	100 ± 0
	Random	76.96 ± 3.08	35.36 ± 10.37
	Ours (UCB Reward)	**100 ± 0**	**100 ± 0**
	Ours (Count Reward)	**100 ± 0**	**100 ± 0**
	Rand. Prior (Osband 2019)	77.83 ± 6.93	56.04 ± 14.79
	Bootstr. (Osband 2016a)	67.61 ± 9.75	44.12 ± 13.02
Zero Init. & Long Hor.	Bootstr. (D'Eramo 2019)	59.57 ± 9.92	29.31 ± 4.11
	UCB1 (Auer 2002)	90.17 ± 2.55	51.55 ± 10.67
	Expl. Bonus (Strehl 2008)	88.91 ± 3.1	54.48 ± 11.63
	ϵ-greedy	67.39 ± 5.16	24.69 ± 3.31
	Random	86.09 ± 3.77	49.73 ± 11.5
	Ours (UCB Reward)	**100 ± 0**	**100 ± 0**
	Ours (Count Reward)	**100 ± 0**	**100 ± 0**
	Rand. Prior (Osband 2019)	100 ± 0	100 ± 0
	Bootstr. (Osband 2016a)	100 ± 0	100 ± 0
Opt. Init. & Long Hor.	Bootstr. (D'Eramo 2019)	100 ± 0	100 ± 0
	UCB1 (Auer 2002)	100 ± 0	100 ± 0
	Expl. Bonus (Strehl 2008)	100 ± 0	100 ± 0
	ϵ-greedy	100 ± 0	100 ± 0
	Random	86.09 ± 3.77	49.73 ± 11.5

Table 8: Gridworld (wall) results recap.

	Algorithm	Discovery (%)	Success (%)
	Ours (UCB Reward)	**100 ± 0**	**100 ± 0**
	Ours (Count Reward)	**100 ± 0**	**100 ± 0**
	Rand. Prior (Osband 2019)	9.44 ± 3.14	0.14 ± 0.06
	Bootstr. (Osband 2016a)	6.35 ± 2.74	0.23 ± 0.08
Zero Init. & Short Hor.	Bootstr. (D'Eramo 2019)	12.55 ± 4.45	0.18 ± 0.08
	UCB1 (Auer 2002)	24.7 ± 4.45	0.46 ± 0.03
	Expl. Bonus (Strehl 2008)	45.16 ± 2.29	0.52 ± 0.04
	ϵ-greedy	38.15 ± 1.95	0.39 ± 0.08
	Random	65.28 ± 0.45	0.59 ± 0
	Ours (UCB Reward)	**100 ± 0**	**100 ± 0**
	Ours (Count Reward)	**100 ± 0**	**100 ± 0**
	Rand. Prior (Osband 2019)	100 ± 0	100 ± 0
	Bootstr. (Osband 2016a)	100 ± 0	100 ± 0
Opt. Init. & Short Hor.	Bootstr. (D'Eramo 2019)	100 ± 0	100 ± 0
	UCB1 (Auer 2002)	100 ± 0	100 ± 0
	Expl. Bonus (Strehl 2008)	100 ± 0	100 ± 0
	ϵ-greedy	100 ± 0	100 ± 0
	Random	65.28 ± 0.45	0.59 ± 0
	Ours (UCB Reward)	**100 ± 0**	**100 ± 0**
	Ours (Count Reward)	**100 ± 0**	**100 ± 0**
	Rand. Prior (Osband 2019)	13.05 ± 3.83	0.2 ± 0.08
	Bootstr. (Osband 2016a)	8.69 ± 3.53	0.2 ± 0.08
Zero Init. & Long Hor.	Bootstr. (D'Eramo 2019)	11.64 ± 5.24	0.23 ± 0.08
	UCB1 (Auer 2002)	21.44 ± 3.64	0.44 ± 0.03
	Expl. Bonus (Strehl 2008)	47.75 ± 1.68	0.55 ± 0.03
	ϵ-greedy	42.53 ± 1.89	0.54 ± 0.03
	Random	70.51 ± 0.75	0.59 ± 0
	Ours (UCB Reward)	**100 ± 0**	**100 ± 0**
	Ours (Count Reward)	**100 ± 0**	**100 ± 0**
	Rand. Prior (Osband 2019)	100 ± 0	100 ± 0
	Bootstr. (Osband 2016a)	100 ± 0	100 ± 0
Opt. Init. & Long Hor.	Bootstr. (D'Eramo 2019)	100 ± 0	100 ± 0
	UCB1 (Auer 2002)	100 ± 0	100 ± 0
	Expl. Bonus (Strehl 2008)	100 ± 0	100 ± 0
	ϵ-greedy	100 ± 0	100 ± 0
	Random	70.51 ± 0.75	0.59 ± 0