Dead-ends and Secure Exploration in Reinforcement Learning

Mehdi Fatemi [1] Shikhar Sharma [1] Harm van Seijen [1] Samira Ebrahimi Kahou [2]

Abstract

Many interesting applications of reinforcement learning (RL) involve MDPs that include numerous "dead-end" states. Upon reaching a dead-end state, the agent continues to interact with the environment in a dead-end trajectory before reaching an undesired terminal state, regardless of whatever actions are chosen. The situation is even worse when existence of many dead-end states is coupled with distant positive rewards from any initial state (we term this as Bridge Effect). Hence, conventional exploration techniques often incur prohibitively many training steps before convergence. To deal with the bridge effect, we propose a condition for exploration, called security. We next establish formal results that translate the security condition into the learning problem of an auxiliary value function. This new value function is used to cap "any" given exploration policy and is guaranteed to make it secure. As a special case, we use this theory and introduce secure random-walk. We next extend our results to the deep RL settings by identifying and addressing two main challenges that arise. Finally, we empirically compare secure random-walk with standard benchmarks in two sets of experiments including the Atari game of Montezuma's Revenge.

1. Introduction and Motivation

Theoretically, if a Markov Decision Process (MDP) corresponding to an environment is fully specified, then its (optimal) value function can be computed without any empirical interaction with the actual environment (Sutton & Barto, 1998; Bertsekas & Tsitsiklis, 1996). However, in many real-world problems, transition probabilities and/or rewarding transitions are not known in advance and the agent has to interact with the environment and learn from its experience. There always exists a (greedy) policy corresponding to the learned value function, which we call *exploitation policy* and is desired to converge to the optimal policy that maximises the expected return. One main challenge for the agent is then to decide when to *exploit* its current knowledge by sticking to the exploitation policy and when/how to *explore* other actions to increase its knowledge by following other policies, which we call *exploration* policies.

Exploration has always been a core problem in RL. Perhaps the oldest and still the most widely-used methods are the ones which perturb the exploitation policy, for example, ϵ-greedy, Boltzmann, and count-based explorations. In tabular settings, several techniques have been suggested, mostly on the basis of regret minimization, which are hardly scalable. However, methods such as count-based techniques have recently been extended to deep RL settings (Bellemare et al., 2016). In this paper, we highlight that these existing methods fail to secure basic properties in certain environments, causing the learning progress to be extremely slow. For example, exploring uninteresting portions of an environment (uninteresting in the context of achieving the desired task) may result in wasted training efforts that generate little to no relevant information. Furthermore, exploring unsafe portions of the environment may be costly and/or raise safety concerns. Thus, the agent's needs to generate new information must be balanced with considerations associated with limited training time, cost, and safety concerns.

In particular, one may note that, in many MDPs of interest, at many states, a large portion of the available actions are likely to result in an undesired terminal state. Upon reaching an undesired terminal state, the environment is forced to terminate/restart before the goal is achieved. As an example, consider the Atari game of Montezuma's Revenge, shown in Figure 1. A first task to be done is to go all the way to the key position and capture it. On playing the game, one may immediately notice that at most of the states many actions result in an imminent death, which causes the agent to lose one life and be re-positioned to the initial state. Furthermore, most of such applications include *dead-end* states (formally defined in Section 2). Having reached a dead-end state, the agent *will* reach an undesired terminal state with probability one, but after some —possibly random— number of steps, regardless of whatever action it chooses. Returning to Mon-

[1]Microsoft Research, 2000 McGill College Avenue, Suite 550, Montréal, QC H3A 3H3, Canada [2]McGill University, 845 Sherbrooke Street West, Montréal, QC H3A 0G4, Canada. Correspondence to: Mehdi Fatemi <mehdi.fatemi@microsoft.com>.

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

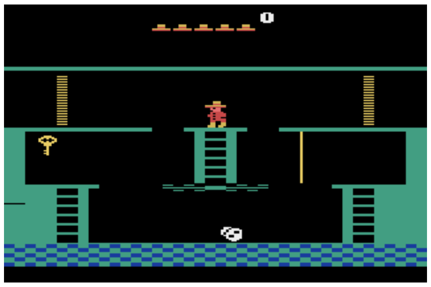


Figure 1. Atari game of Montezuma's Revenge. The first goal is to get the key after passing through a dangerous environment, where at most states, many of the actions cause an undesired termination after a sequence of uncontrollable and inescapable steps.

tezuma's Revenge, a death happens only after a trajectory of unknown length, which is both inescapable and uncontrollable (even worse, there is often an animation segment, in which the agent should still continue interacting with the game console). This inherently produces a tremendously large number of training steps with no information gain about the goal. Additionally, such undesired terminations, and dead-ends by extension, prohibit the agent to easily move far from its initial position in order to find any source of positive rewards (e.g., picking up the key).

Importantly, undesired terminations (but not dead-ends) are often directly signalled from the environment. However, considering a negative reward in the case of undesired termination is not a generic solution. The reason is that, in particular, the negative rewards will then become ubiquitous in certain parts of the environment. Hence, once function approximation is used, the resulting negative values are generalised and the agent can easily become scared of passing through certain parts of the environment, even though it may be necessary to do so in order to reach a positive reward. This is a serious issue and it is often for this very reason that designers avoid using negative rewards in their implementations (see for example Ecoffet et al. (2019) for a discussion on the adverse effect of negative rewards).

In this paper, we consider a different approach. We start with the necessity that any action that is likely to result in entering a dead-end should similarly assume less probability in the exploration policy; a property that is called *security*. We then formalize a learning problem to nicely remove "all" the assumptions regarding domain knowledge, except for an undesired-termination signal (e.g., losing a life), which is a fair assumption. We finally derive algorithms for both tabular and deep RL settings.

The rest of this paper is organized as the following. After the formal definitions are presented, we establish our first main result, which enables us to precisely introduce secure exploration in a generic way and with no specific assumption. Next, we present secure random walk in the tabular settings with the proof of convergence for Q-learning under security. We then extend the results for deep RL settings by identifying and addressing two main challenges. Next, we present experimental results for two interesting domains of the Bridge game and Atari game of Montesuma's Revenge, and conclude the paper with further remarks and future research directions.

2. Problem Formalism

We assume standard reinforcement learning settings (Sutton & Barto, 1998) of an MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, r, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ are the discrete sets of states and actions; $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$ is a transition function that defines the transition probability from state s to s' if action a is taken; $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [r_{min}, r_{max}]$ is a reward function and $\gamma \in [0,1]$ denotes a scalar discount factor. A *trajectory* is a sequence of $< s_t, a_t, r_t, s'_t >$ with $s'_t = s_{t+1}$ and s_0 being the first state. A policy $\pi(s,a) = \mathbb{P}[A_t = a \mid S_t = s]$ defines how an action is selected in a given state. Therefore, selecting actions according to a stationary policy results in a trajectory. A trajectory starting from any state induces *return* as $G = \sum_{j=0}^{\infty} \gamma^j r_j$. We assume that all returns are finite and bounded. A trajectory is therefore called *optimal* if it induces maximum return. A value function $q(s,a) = \mathbb{E}^{\pi}[G \mid S_0 = s, A_0 = a]$ is defined in conjunction with a policy π to evaluate the expected return of taking action a at state s and following π thereafter. Additionally, the optimal value function is defined as $q^*(s,a) = max_{\pi} q(s,a)$, which is the maximum expected return of all trajectories starting from (s,a). We further augment $\mathcal{M}$ with a non-empty termination set $\mathcal{S}_T \subset \mathcal{S}$, which is the set of all terminal states. A terminal state is by definition a state, at which the environment terminates its dynamics. All terminal states are by definition zero-valued. We require that, from all states, there exist at least one trajectory with non-zero probability to a terminal state.

Definition 1 (Undesired Terminal State)**.** *A terminal state $s_u \in \mathcal{S}_U \subset \mathcal{S}_T$ is called an* undesired *terminal state if reaching s_u prevents achieving maximum return.*

Definition 2 (Dead-end)**.** *A state $s' \in \mathcal{S}_D$ is called a dead-end if all the trajectories starting from s' reach an undesired terminal state with probability 1 (w.p.1) in some finite (possibly random) number of steps.*

The set of all the dead-ends and all the undesired terminal states are denoted by $\mathcal{S}_D$ and $\mathcal{S}_U$, respectively. The environment defines all $s_u \in \mathcal{S}_U$; however, we do not augment $\mathcal{S}_U$ to the MDP's tuple to avoid uncommon notation. Definition 2 implies that, after entering a dead-end state s', regardless of taken actions afterwards, an undesired termination will occur in some (finite but possibly random) number of inescapable steps. A direct consequence of Definition 2 is the following result, which can be proved by induction.

Proposition 1. *If a state is dead-end, then so are all the states after that on all the possible trajectories.*

Consequently, although *dead-end* is a state by definition, we also conveniently use the term to refer to a trajectory starting from a dead-end state and *length* of a dead-end to refer to the (random) length of that trajectory. In practical situations, undesired terminal states are flagged once entered; however, this assumption hardly exists for entrance states to the dead-end trajectories (they instead should be learned). We can now state the following qualitative definition.

Definition 3 (Bridge Effect)**.** *Bridge effect exists in an environment if under uniform policy, the probability of reaching a positive reward in N steps is less than that of transitioning into a dead-end, for all $N > 1$.*

When the probability of falling into a dead-end is *much* higher than that of reaching any positive reward, bridge effect becomes a severe learning barrier. It often happens when there is a non-zero and non-negligible probability of entering a dead-end in most states (at least in all trajectories from initial state to any positively-rewarded transition), and that all the positively-rewarded transitions are relatively far from the initial state. The term "bridge" is inspired by the challenge of passing a bridge, where most actions are likely to cause falling. The bridge effect is a fundamental issue in practice, which can easily prohibit effective learning in reasonable time, regardless of the learning method in use.

2.1. Security

Given an MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, r, \gamma)$ and in the course of training, let us formally distinguish between the *exploration* policy $\eta : \mathcal{S} \times \mathcal{A} \to [0, 1]$, and the *exploitation* policy π. As mentioned before, through training, π is then meant to converge to the optimal policy, which maximizes the expected return. In light of the discussion above, we therefore require the following condition to secure the exploration:

Property 1 (Security)**.** *A policy $\eta(\cdot, \cdot)$ is secure if for any $\lambda \in [0, 1]$ the following condition holds:*

$$\sum_{s' \in \mathcal{S}_D} T(s, a, s') \geq 1 - \lambda \Longrightarrow \eta(s, a) \leq \lambda \qquad (1)$$

for all $s \in \mathcal{S} \backslash \mathcal{S}_D$.

In words, if (s, a) enters a dead-end with some level of certainty, then a should be excluded from $\eta(s, \cdot)$ with same level of certainty. This property say nothing about when s' is not a dead-end. One may therefore say it only *secures* exploration with respect to the dead-ends. Any exploration policy that holds Property 1 is then called *secure exploration*.

We note that at a given state s, Property 1 can be enforced on a given policy $\eta(s, \cdot)$ by cutting and re-weighting the probabilities in $\eta(s, \cdot)$ so that it satisfies Property 1. This is possible only if it is mathematically allowed, due to $\sum_a \eta(s, a) = 1$. For example, if all the actions transition to some dead-end with sufficiently large probability, then η may not be reduced sufficiently for all the actions. Another trivial example is when there is only one action, which leaves no choice. Such policies cannot become secure.

2.2. Secure Exploration

As much as Property 1 is desired in practice, it is not easy to enforce it. The drawback is twofold: having information about dead-ends as well as accessing the transition function T, both of which are not likely to exist. In this section, we establish our first main result, which intrinsically removes both drawbacks.

Returning to the core idea of separating π and η, the goal here is to characterize η with a proper value function, so that the value function can be learned algorithmically. More generally, η can *be related* to the value function of a relevant MDP $\mathcal{M}_e = (\mathcal{S}, \mathcal{A}, T, r_e, \gamma_e)$. We call it *exploration MDP* and it has the same state and action spaces as well as transition function as $\mathcal{M}$. This allows for considering a different scheme for expected return, thereby satisfying different goals or conditions during exploration. We should emphasize that $\mathcal{M}_e$ directly induces an optimal value function q_e^*; however, η is not an optimal policy of $\mathcal{M}_e$. Specifically, η is guided/corrected by q_e^* but is not greedy to q_e^*.

To be able to properly define r_e and γ_e, let us start with a basic observation. Assume an MDP with reward of -1 for entering an undesired terminal state and 0 otherwise. The optimal value of a given dead-end state is therefore negative; however, its magnitude depends on the length of the dead-end trajectory (i.e., number of steps from the dead-end state to an undesired termination) and the discount factor. Hence, although the optimal value of dead-end states are negative, they can be arbitrarily close to zero. As a result, it is impossible in general to identify a dead-end trajectory's *entrance* based only on its value. It implies that in such settings, knowing when undesired terminations happen seems insufficient to identify dead-ends. To overcome this issue, recall that the undesired terminal states are well-defined (signalled when get in). But, we still need to be able to formally discern if a state is dead-end. To this end, let us state the following properties for $\mathcal{M}_e$.

Property 2. *$\mathcal{M}_e$ is characterized as follows*

P2.1 $r_e = -1$ *if enter an undesired terminal state and* $r_e = 0$ *otherwise.*

P2.2 *No discount:* $\gamma_e = 1$.

It can be seen that the optimal value of all the dead-ends (and only the dead-ends) will be exactly -1. We then establish the following main result (all the proofs of our theorems are presented in the supplementary materials, Section S1).

Theorem 1. *Let q_e^* be the optimal value function of $\mathcal{M}_e$ under P2.1 and P2.2. Let further η be any arbitrary policy that satisfies the following:*

$$\eta(s,a) \le 1 + q_e^*(s,a), \quad \forall (s,a) \in \mathcal{S} \times \mathcal{A} \tag{2}$$

where $q_e^*(s,\cdot) \neq -1$ at least for one action. Then η is secure.

The value of η on dead-end states (where $q_e^*(s,a) = -1$ $\forall a \in \mathcal{A}$) is not important since all trajectories from a dead-end will reach an undesired termination anyways. It is again worth mentioning that in Theorem 1, η is different from the optimal solution of $\mathcal{M}_e$, namely $argmax\ q_e^*(s,\cdot)$, and is stochastic in general, as desired. We call $\kappa(s,a) = 1 + q_e(s,a)$ and $\kappa^*(s,a) = 1 + q_e^*(s,a)$ *security cap* and *optimal security cap* functions, respectively.

Theorem 1 is an important theoretical result, since it translates Property 1 to a learning problem by removing the explicit appearance of T as well as the explicit knowledge about dead-end states, and replacing them with q_e^*, which is learnable. Hence, it makes it possible to move forward to model-free methods. In practical contexts, Theorem 1 provides the means for securing any given exploration policy. One should keep in mind that the original exploration policy η may be designed with other considerations, such as curiosity and information gain. At a given state s, the security cap can then be enforced by iteratively cutting and re-weighting $\eta(s,\cdot)$ until (2) is satisfied. In this paper, we consider securing ϵ-greedy exploration, which is arguably the most basic exploration technique and it uses random walk for exploration. Hence, the result is called secure random walk (SRW).

We finish this section with a complementary result. In practice, it may also be favourable to set a security threshold on q_e, under which the action is not allowed to be taken. This helps to mechanically remove fully non-secure (or susceptible) actions. The following result provides the guarantee that such a threshold may be found that excludes non-secure actions without touching others. Let $\mathcal{A}_D(s)$ denote the set of actions at state s that transition to a dead-end w.p.1.

Theorem 2. *Under P2.1 and P2.2, let v_e^* and q_e^* be the optimal state and state-action value functions of $\mathcal{M}_e$. Then there exists a gap between $v_e^*(s')$ and $q_e^*(s,a)$ for all $a \in \mathcal{A}\backslash\mathcal{A}_D(s)$, $s' \in \mathcal{S}_D$, and $s \in \mathcal{S}\backslash\mathcal{S}_D$. Furthermore, the gap is independent of dead-end's possible length.*

Theorem 2 allows us to set a single threshold that does not need any domain knowledge about possible length of dead-ends. Any threshold in the open interval

$$I = (-1, \max_{s\in\mathcal{S}\backslash\mathcal{S}_D} \max_{a'\in\mathcal{A}\backslash\mathcal{A}_D(s)} \sum_{s'\in\mathcal{S}_D} T(s,a',s')) \tag{3}$$

will separate actions, which result in irrefutable transition (w.p.1) to dead-ends from the rest. Nevertheless, the maximum probability in equation (3) has to be known or guessed. In any case, if it is decided to use such a threshold, it is easy to append it to the main algorithm.

3. Tabular Settings

We first discuss the implementation of SRW in tabular settings. In the next section, we will extend it to deep RL settings as well.

Before going further, let us study if Property 1 influences the convergence. The result may be summarized with the following theorem.

Theorem 3. *If the followings hold:*

1. *States and actions are finite (tabular settings).*
2. *Policy η exists that satisfies* Property 1*, and η is used as the sole behavioural policy. Furthermore, η visits all the* non-dead-end *states infinitely often.*
3. *$q_\pi(\cdot,\cdot)$ is initialized pessimistically and standard conditions are applied on the step-size α_π.*

then, Q-learning of $q_\pi(s,a)$ converges to $q_\pi^*(s,a)$ for $s \in \mathcal{S}\backslash\mathcal{S}_D$ and $a \in \mathcal{A}\backslash\mathcal{A}_D(s)$.

Theorem 3 asserts that holding Property 1 does not endanger convergence of π to the optimal policy using Q-learning. That follows from pessimistic initialization of q_π, which assures that the value of excluded state-actions does not change a greedy policy. It is a strong result in that it holds for the fully off-policy case. It can simply be extended to the looser case of ϵ-greedy between π and η. Of note, in an environment that involves bridge effect, only the exclusion of $\mathcal{A}_D(s)$ may improve the convergence rate dramatically; rest assured Property 1 serves its purpose.

In order to implement SRW, we need to simply enforce the security cap on the uniform distribution over the action-space. That is, actions are equiprobable under security cap. We therefore keep the equality in equation (2) and normalise the result:

$$\eta(s,a) \doteq \frac{1 + q_e(s,a)}{\sum_{a'} 1 + q_e(s,a')} = \frac{1 + q_e(s,a)}{n_a + \sum_{a'} q_e(s,a')} \tag{4}$$

Additionally, we require $q_e(\cdot,\cdot)$ to be initialized at zero. Because all the q_e's are between -1 and 0, the resulting η is well-defined (except when all q_e's are -1, which happens on dead-end trajectories). This way, η starts from a uniform distribution and the probability of taking an action quickly decreases when (and only when) it proves to be insecure. Importantly, it is not affected by γ of $\mathcal{M}$, which means that it is equally sensitive to all the dead-end states, thereby also to all the transitions entering a dead-end trajectory. Notice that the initialization is important. In the course of training, after the dead-end's values converge to -1, one may write:

$$q_e(s,a) = -\sum_{s' \in \mathcal{S}_D} T(s,a,s') + \sum_{s' \notin \mathcal{S}_D} T(s,a,s') \max_{a'} q_e(s',a') \quad (5)$$

Zero-initialization induces $\max_{a'} q_e(s',a') \leq 0$. Hence, (5) implies $q_e(s,a) \leq -\sum_{s'\in\mathcal{S}_D} T(s,a,s') \leq -(1-\lambda)$. It yields $\eta(s,a) \leq 1 + q_e(s,a) \leq \lambda$. Therefore, η is secure and the security cap is reliable even if q_e has not yet fully converged. One may be tempted to initialize q_e at -1 to assure security even before convergence of dead-ends. It can be seen that $q_e(s,a) \leq -\sum_{s'\in\mathcal{S}_D} T(s,a,s')$ always holds, which implies $\eta(s,a) \leq \lambda$. However, under -1 initialization, (4) is undefined at the beginning and a uniform policy should be used. The moment any $q_e(s,a)$ is updated to a value slightly bigger than -1, the exploration policy *always* selects a at s. This follows from (4) that will assume probability one for a and zero for all other actions. It clearly forfeits exploration. In practice, any value in the semi-open interval $(-1,\ 0]$ may be used for initialization, which introduces a trade-off between earlier security versus less exploration. We use zero-initialization, which assumes maximum exploration and decays only if (s,a) proves dangerous. SRW is therefore summarized as follows:

Definition 4 (Secure Random-Walk)**.** *Let q_e be the value function (under training) with* zero-initialization*, corresponding to $\mathcal{M}_e$ under P2.1 and P2.2. The* secure random-walk *is defined as the following:*

$$\eta(s,\cdot) = \begin{cases} 1/n_a & \textit{if } q_e(s,a') = -1, \forall a' \in \mathcal{A} \\ \frac{1+q_e(s,\cdot)}{n_a + \sum_{a'} q_e(s,a')} & \textit{otherwise} \end{cases}$$

This definition manifests certain interesting properties: As it learns, it guarantees Property 1 (due to Theorem 1 and the discussion above) as well as convergence of Q-learning when it is used as behavioural policy (due to Theorem 3). If $n_a + \sum_{a'} q_e(s,a') < 1$, the security cap can be reapplied until η holds (2), though it is often not required in practice.

Definition 4 together with P2.1 and P2.2 provide sufficient technical requirements and the stage is then set to define the algorithm. In Algorithm 1, we present a generic SRW, where Q-learning is used to learn both q_e and q_π side-by-side. This is a fully off-policy version of the algorithm. However, it is also an option to slowly switch from η to π with an ϵ-greedy mechanism, for instance.

If it is decided to use a security threshold according to Theorem 2, it suffices to simply limit the selected action (at the two *select* lines) to also have its q_e be larger than the given threshold.

Finally, it is worth noting that SRW is theoretically guaranteed not to hurt performance if no dead-end is present. It is immediate from Definition 4 that in the absence of dead-ends, η reduces to a uniform policy; hence, ϵ-greedy exploration.

Algorithm 1 Q-learning with secure random-walk.

```
Parameters: step sizes $\alpha_e$, $\alpha_\pi$.
set $q_e(s,a) = 0,\ \forall(s,a) \in \mathcal{S}\times\mathcal{A}$
set $q_\pi(s,a)$ pessimistically $\forall(s,a) \in \mathcal{S}\times\mathcal{A}$
set $s \leftarrow$ initial state
repeat
  if $q_e(s,a) = -1$ for all $a \in \mathcal{A}$ then
    select $a \sim \mathcal{U}(\mathcal{A})$
  else
    compute $\eta(s,\cdot)$ from Equation (4)
    select $a \sim \eta(s,\cdot)$
  end if
  observe $(r, s')$
  if undesired termination then
    $q'_e = 0$
    $r_e = -1$
  else
    $q'_e = \max_{a'} q_e(s',a')$
    $r_e = 0$
  end if
  $q_e(s,a) \leftarrow (1-\alpha_e)q_e(s,a) + \alpha_e(r_e + q'_e)$
  $q_\pi(s,a) \leftarrow (1-\alpha_\pi)q_\pi(s,a) +$
        $\alpha_\pi(r_\pi + \gamma_\pi \max_{a'\in\mathcal{A}} q_\pi(s',a'))$
  $s \leftarrow s'$
until $q_\pi$ converges
```

4. Deep Reinforcement Learning

Moving forward, it is desirable to extend Algorithm 1 to a DQN (Mnih et al., 2015) format with two networks for q_e and q_π. However, there are two main challenges, which influence a deep RL implementation: 1) cascading value-overflow, and 2) catastrophic forgetting.

4.1. Cascading Value-overflow in Deep RL

There is an inherent problem when using DQN (and similar algorithms). Due to the quadratic loss function, value overflow is bound to happen. In particular, quadratic loss allows for the estimated q to be bigger than Bellman q if it is positive (the negative side is not a big issue due to the

MAX operator in Bellman target). This can be severely destructive because the estimated q is used for bootstrapping, which means q is always pushed toward some larger value. The max operator forces the value function to almost surely result in an overestimation. However, it does not stop there. The overestimation will exponentially cascade like a chain effect and the q values quickly become too large. Note here that the root of this problem is quadratic loss and not the use of same Q-table (or network) for both bootstrapping and action-selection or estimation error, both of which may be addressed by Double Q-learning and Double DQN (van Hasselt et al., 2016). Note also that the use of small-enough γ effectively hides the issue. The reason is that the bootstrapped q is weighted by γ and is therefore exponentially decayed by γ. Additionally, reward-clipping also diminishes the issue by making the return small. We experimentally show that commonly-used $\gamma = 0.99$ is often sufficient to prevent the overflow, but perhaps not larger γ (see Figure 4). In the case of security, $\gamma = 1$ is required. As a result, the cascading overflow issue will become a large obstacle. To address cascading overflow, recall that under P2.1 and P2.2, $r_e = 0$ everywhere except when an undesired termination happens, where $r_e = -1$. As a result, the return is always between -1 and 0. This allows us to clip the *bootstrapping* value in Bellman target to stay inside $[-1, 0]$. This technique in principle stops cascading of overflowed values and should fix training of q_e.

4.2. Catastrophic Forgetting

If a is not secure at s, it will not be selected and will be soon removed from the experience replay buffer. As a result, in just a few epochs, the q_e-network will forget the value of (s, a). More critically, as η learns to stay secure, q_e-network may saturate on the abundance of zero-rewarding transitions and a catastrophic forgetting happens. There are different approaches that may be considered to partially deal with this issue. A simple method is to stop training q_e-network after a specific number of epochs (which also introduces a new hyper-parameter for when to stop training of q_e). However, stopping training of q_e inherently involves a dilemma: early-stopping causes not learning the value of farther dead-ends, while late stopping causes forgetting the value of closer dead-ends. The forgetting issue is a general problem in deep learning community. We therefore stick to the naive method of early-stopping for now and leave more advanced approaches for the future research.

Aside from these challenges, we should also highlight that in the lack of dead-ends, the "exploration network" is trained with zero rewards and will learn to output zero values. Hence, as it is trained, its effect reduces to ϵ-greedy, similarly to the tabular case.

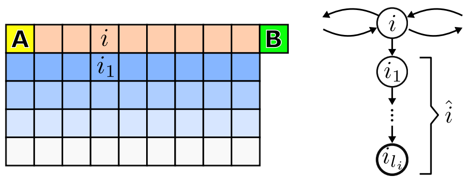


Figure 2. The bridge problem. Left: Agent starts from **A** and should reach **B**. Each vertical blue path is an inescapable and uncontrollable fall trajectory with a random length l_i (demonstrated by the gradient blue). Right: The corresponding part of MDP for state i on the bridge.

5. Experimental Results

5.1. Tabular: Bridge Game

We begin with a variation of the classical cliff-walking problem (Sutton & Barto, 1998), which we call *bridge game*. Consider a bridge of length L, shown in Figure 2. The agent starts at the left end of the bridge and its goal is to reach the right end, resulting in some very large reward $r_B \gg +1$. At each step, n_a actions are available, where action a_n goes right with probability (w.p.) x_n, goes left w.p. y_n, and falls down w.p. z_n. Once on the bridge, assume the following:

- For a_0: x_0 close to 1, $y_0 = 0$ and z_0 close to 0 ($z_0 = 1 - x_0$);
- For a_1: $x_1 = z_1 = 0$ and $y_1 = 1$;
- For all other actions, a_j, $j \geq 2$: $x_j = y_j = 0$, and $z_j = 1$.

Once the agent falls, all actions cause a transition to another non-bridge state until reaching an undesired terminal state. It therefore takes random (but finite) number of uncontrollable steps to terminate and return to the initial position. Hence, the agent is always at the risk of falling into a random-length inescapable trajectory before the episode *undesirably* terminates and it can restart. The reward signal is $+r_B$ if reach **B** and zero everywhere. Direct evaluation of Bellman equation yields the following result.

Proposition 2. *If r_B is sufficiently large and $\gamma < 1$, the optimal policy is $\pi(s, a_0) = 1$, for all the bridge states s.*

In the supplementary materials, Section S2, we present a concrete theoretical discussion on why it takes huge number of training steps for tabular Q-learning to converge under both cases of rewards. Furthermore, in Section S3, we provide a formal discussion on why Boltzmann policy in also not secure in general.

Figure 3 presents experimental results for different exploration methodologies. In particular, we examined ϵ-greedy, Boltzmann, and count-based methods as the baselines. As for the count-based method, we add a motivation term $1/(1 + \sqrt{N(s,a)})$ to $q(s,a)$, where $N(s,a)$ is the number of times (s,a) has been visited (Bellemare et al., 2016). To ensure stability, a small enough step-size has to be used due to stochasticity of the environment. We use $\alpha = 0.1$, 0.01, and 0.001 for Boltzmann, count-based, and ϵ-greedy, respectively, all without annealing. We use a considerably simple problem of $z_0 = 1\%$ and only one fall action ($n_a = 3$). As it can be expected, the convergence time increases considerably with the bridge length. Furthermore, Figure 3-(a)–(c) reveals that even for such a simple problem and a surprisingly small bridge of $L = 15$, which corresponds to a tiny state-space, the learning still needs tremendously large number of episodes to converge even under count-based exploration; the fact that is not directly expected.

In order to demonstrate the power of SRW, Figure 3-(d) illustrates that Algorithm 1 solves the bridge problem orders of magnitude faster than Q-learning with ϵ-greedy, Boltzmann, or count-based exploration. Indeed, in the case of $L = 15$, which is also illustrated in Figure 3-(a)–(c), bridge problem can be solved three orders of magnitude faster. The cases of $L = 20$ and $L = 25$ require (tens of) millions of episodes to converge without secure exploration. Figure 3-(e) depicts different cases of switching between η and π using an ϵ-greedy mechanism with different fix ϵ. As it can be expected, the best case happens when $\epsilon = 1$ (fully secure) and as ϵ decreases, it quickly deteriorates. In general, in the environments involving bridge effect, one should expect considerable improvement when securing the exploration policy. In more complex environments of practical interest, we conjecture that securing a smarter exploration technique than random-walk (such as (pseudo-)count-based) should obtain better results than the secure random-walk.

5.2. Deep RL: Montezuma's Revenge

Surprisingly, several Atari 2600 games in the ALE suit (Bellemare et al., 2013), which look nearly unsolvable using DQN and other similar methods are environments that indeed suffer from the bridge effect. In specific, at the bottom of the score list in (Mnih et al., 2015), 5 out of 9 games may receive better results by using secure random-walk exploration. Most notably is of course Montezuma's Revenge.

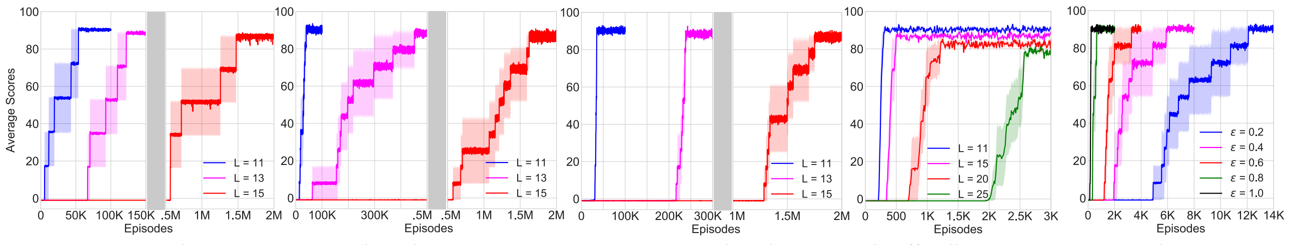
The game of Montezuma's Revenge is commonly considered as the most difficult game among the Arcade Atari Environment suite (Bellemare et al., 2013). We limit the game to only pick the key on the first screen; something that is still almost impossible using DQN without the help of intrinsic rewards (for example in forms of pseudo-counts (Bellemare et al., 2016)). We show that using SRW can optimally achieve this goal with some quite basic implementation. Combination of more sophisticated techniques such as recurrent networks, dueling architecture, generic prioritized experience replay, or using bonus-based exploration may be used to improve the results even more. However, we deliberately stick to some very basic implementation in order to showcase the role of security. As for the baseline, we use DQN with identical structure and same hyper-parameters. The details of implementation and the used hyper-parameters are described in the supplementary materials, Section S4. The results are illustrated in Figure 5. To avoid forgetting issue, we stop training of exploration network after 2M steps, which only guarantees security at the top two levels of the screen. Even with that, the considerable boost in performance is evident. We also tried having bootstrapping correction when a positive reward is observed for the exploitation policy. It clearly has a large effect on the performance, and together with security, the agent always achieves the goal very robustly[1].

6. Related Work

The term *secure exploration* and the concept of security in RL is new, to the best of our knowledge. The concept of dead-end may also be viewed as a safety issue, specially in physical scenarios. However, we particularly use a different term to distinguish between safe RL and the present paradigm, mainly because secure exploration essentially concerns only the exploration. It is important to also note that secure exploration is, in its nature, an additional process on top of any given policy η to secure it. This makes it easy to fulfill other criterion, such as information gain (or curiosity) and safety as well. Nevertheless, to highlight particular differences, we explain safe RL literature in more details.

Safety in RL (García & Fernández, 2015) is an overloaded term, as it may be considered with respect to parametric uncertainty (Thomas, 2015; Petrik et al., 2016; Laroche et al., 2019), internal uncertainty (Altman, 1999; Carrara et al., 2019), or interruptibility (Orseau & Armstrong, 2016; Guerraoui et al., 2017). It may also be divided into two main approaches (for a survey see García & Fernández (2015)). The first approach tries to assert an optimization criterion, such as worst case scenario (Heger, 1994), risk-sensitive criterion by transforming the TD error (Mihatsch & Neuneier, 2002), and call-time criterion by defining the safety in terms the probability of returning to the initial state (Moldovan & Abbeel, 2012). A considerably longer tale of history exists in the robust control literature, which designs the controller on the basis of worst-case scenario (see for example Başar & Bernhard (1995) that provides a nice game-theoretic approach). These approaches are inherently different from ours in that they basically solve an optimization problem with additional constraints to account for some sort of safety. The second set of safe exploration approaches tries to de-

[1]Code is available at https://github.com/Maluuba/srw.



(a) ϵ-greedy (b) Boltzmann (c) Count-based (d) Off-policy Secure (e) ϵ-greedy Secure L=11

Figure 3. Learning the bridge game with various length, using Q-learning with (a) ϵ-greedy ($\epsilon = 0.1$), (b) Boltzmann, and (c) count-based policies, respectively. Both Boltzmann and count-based are the learning curves averaged over 10 seeds. For ϵ-greedy, after each learning episode we test π greedily with 10 seeds (due to stochasticity) and repeat the experiment with 5 different seeds. (d) learning optimal policy with secure exploration. After each learning episode, the target policy is used 10 times greedily for evaluation. The experiments are then repeated for 10 different seeds. (Compare the results with (a)–(c), specifically, for $L = 15$.) Interestingly, $L = 25$ is solved in less than 3k episodes, while for the common algorithms to converge, it needs tens of millions of episodes. (e) same as (d) but with different levels of ϵ-greedy switching between π and η.

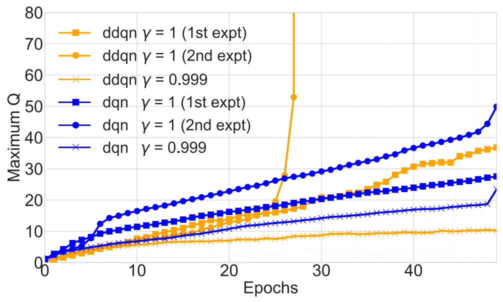


Figure 4. Cascading overflow in Montezuma Revenge. All the rewards are removed; hence, the maximum values should remain around zero all times. The issue is worse for larger γ. After each epoch, the network is used in 100 episodes, each of which consists of fully random action-selection until all lives are lost. Each data point is the $\max_{s,a} Q$, obtained from the Q-network, evaluated at all the state-actions seen.

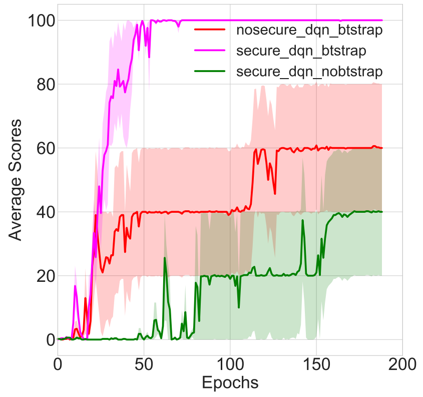


Figure 5. Getting the key ($r_{key} = 100$) in Montezuma's Revenge. The line for vanilla DQN has been removed (it is all zero). The results are averaged over five random seeds.

fine an exploration process by asserting external knowledge in terms of demonstrations or teacher advice, for example see García & Fernández (2015) for several recent and old references. These methods are often suitable where safety is more important than learning *tabula rasa*. Secure exploration, in contrast, does not involve any external teacher.

7. Concluding Remarks

In this paper, we focused on the problem of *dead-ends*, which are inescapable and uncontrollable trajectories with undesired end and no positive reward. In particular, we described the *bridge effect* as a high probability of entering a dead-end in most states, coupled with distant positive rewards. Bridge effect, by its nature, prohibits exploration. Our formal treatment and core contributions include: 1) defining a desired property for the exploration policy to hold, called *security*. 2) Formal results that enable us to properly define the secure exploration, and quite generically secure any given exploration policy. Specifically, our theoretical results translate the security property into a learning problem with required guarantees. 3) The SRW algorithm with extension to deep RL architectures.

As an experimental evidence, we particularly show that a given bridge game that takes tens of millions of episodes to learn by Q-learning and any of ϵ-greedy, Boltzmann, or count-based exploration, can easily be solved quite robustly more than three orders of magnitude faster, using secure exploration. Finally, we tested our algorithm on the infamous Atari game of Montezuma's Revenge and we solved its first task (getting the key) quite robustly and much faster that other basic methods.

One strength of the present methodology is that it can be combined with other exploration techniques because security is inherently orthogonal to the main exploration strategy in use. Moving forward, we believe that an approach similar to ours may also be used to achieve different desired properties for the exploration.

Acknowledgements

We would like to thank the anonymous reviewers for their fruitful comments on the manuscript of this paper. We are grateful to R. Tachet des Combes for his help with *philly tools*, which enabled us to use the GPU clusters. We are also grateful to R. Laroche for his valuable comments on the manuscript as well as the rebuttal.

Author Contributions

MF conceived the research, developed all the theoretical concepts and results, developed the single-machine codebase and wrote the manuscript. SS developed and administered the cluster version of the code for hyper-parameter search and helped revising the manuscript. HVS contributed to fruitful discussions in the course of the project. SEK made the suggestion for value clipping and using red-channel in Montezuma's Revenge.

References

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

Başar, T. and Bernhard, P. *H-∞ Optimal Control and Related Minimax Design Problems: A Dynamic Game Approach*. Birkhauser, 1995. ISBN 978-0-8176-4756-8.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. Unifying count-based exploration and intrinsic motivation. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 29*, pp. 1471–1479. Curran Associates, Inc., 2016.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47(1):253–279, May 2013. ISSN 1076-9757.

Bertsekas, D. P. and Tsitsiklis, J. N. *Neuro-Dynamic Programming*. Athena Scientific, 1st edition, 1996. ISBN 1886529108.

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

Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., and Clune, J. Go-explore: a new approach for hard-exploration problems. *CoRR*, abs/1901.10995, 2019. URL `http://arxiv.org/abs/1901.10995`.

García, J. and Fernández, F. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, January 2015. ISSN 1532-4435.

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

Heger, M. Risk and reinforcement learning: Concepts and dynamic programming. *ZKW-Bericht No. 8/94*, 1994.

Laroche, R., Trichelair, P., and Tachet des Combes, R. Safe policy improvement with baseline bootstrapping. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, 2019.

Mihatsch, O. and Neuneier, R. Risk-sensitive reinforcement learning. *Machine Learning*, 49(2):267–290, Nov 2002. ISSN 1573-0565.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529, 2015.

Moldovan, T. M. and Abbeel, P. Safe exploration in markov decision processes. In *Proceedings of the 29th International Conference on Machine Learning*, ICML '12, pp. 1711–1718. Omnipress, July 2012. ISBN 978-1-4503-1285-1.

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

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

Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. MIT Press, 1998. ISBN 0262193981.

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

van Hasselt, H., Guez, A., and Silver, D. Deep Reinforcement Learning with Double Q-Learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pp. 2094–2100. AAAI Press, 2016.