Safe Reinforcement Learning

Eloise Berthier
eloise.berthier@polytechnique.edu

Julien Chhor
julien.chhor@polytechnique.edu

Abstract

In order to be used for potentially critical applications, reinforcement learning algorithms must prove that they will not propose harmful policies. Hence they will be required to meet some safety conditions. We explore and compare different approaches dealing with this issue. For instance, safety can be defined in terms of performance monotonicity, surpassing a performance baseline, or constraints satisfaction. We summarize the main theoretical results and simple algorithms meeting safety requirements for these approaches, with a particular focus on continuous domain applications.

1 Introduction

1.1 Approaches to Safety

Safety is one of the bottlenecks preventing reinforcement learning (RL) from being in use in real-life applications. While a RL agent in a virtual environment can afford to perform arbitrary policies, as soon as it interacts with physical or human-related processes, it must avoid dangerous behaviors. This is the case for online recommendation systems, physical robotics, medical applications or self-driving cars.

A recent survey [5] on safe RL defined two categories of approaches to safe RL: one is based on modifying the optimization criterion, the other one on modifying the exploration process. The latter involves the integration of external knowledge in the exploration process, for instance providing advice from a teacher or demonstrations. In this report, we only focus on the first category, keeping the same learning framework as in standard RL.

Such methods include a notion of worst-case, risk or constraint criterion. We follow three approaches to safety. The first one defines safety as a guarantee to monotically increase the performance of a policy. The second one is close to the first one, but it requires the policy to be better than a user-chosen performance baseline. The last approach requires policies to satisfy some constraints, in expectation. We give the main theoretical foundations of the three approaches, and explore how safety can be integrated in RL algorithms. The most recent articles emphasize approximate and continuous domain algorithms.

1.2 Common Notations

We consider a discrete-time Markov decision process (MDP) defined by $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, D)$. $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces, $\mathcal{P}$ defines the transition density of going to a state given an action taken in a previous state, and $\mathcal{R}$ defines the density of the rewards given a state-action pair. D is the initial state distribution. The rewards are bounded in $[0, 1]$, and we consider an infinite time horizon with discount factor $0 \leq \gamma < 1$. Depending on the articles considered, we will use finite or continuous MDPs, so we write the following definitions most generally in the continuous case.

Consider a stationary policy π. The unnormalized γ-discounted future state distribution starting from D and following policy π is $d_D^\pi(s) = \sum_{t=0}^{\infty} \gamma^t \mathbb{P}(s_t = s|\pi, D)$. The value function is $V^\pi(s) = \mathbb{E}_{a\sim\pi}\big[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t)|s_0 = s\big]$ and the state-action value function is $Q^\pi(s,a) = \mathcal{R}(s,a) + \gamma\mathbb{E}_{s'\sim\mathcal{P}(s',s,a)}\big[V_\pi(s')\big]$. The advantage function is $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$. We also define the advantage of a policy π' over another policy π as: $A_\pi^{\pi'}(s) = \int_{\mathcal{A}} \pi'(a|s)A^\pi(s,a)\mathrm{d}a$ and its expected value with respect to an initial state distribution μ as $\mathbb{A}_{\pi,\mu}^{\pi'} = \int_{\mathcal{S}} d_\mu^\pi(s)A_\pi^{\pi'}(s)\mathrm{d}s$. This will help us compare performances between different policies.

Policies are usually ranked by their expected discounted reward starting from the initial state distribution D: $J_D^\pi = \mathbb{E}_{s\sim D}\big[V^\pi(s)\big] = \int_{\mathcal{S}} d_D^\pi(s)\int_{\mathcal{A}} \pi(a|s)\mathcal{R}(s,a)\mathrm{d}a\mathrm{d}s$. Solving an MDP is finding a policy $\pi^\star$ maximizing J_D^π. For any MDP, there exists a deterministic policy simultaneously maximizing $V^\pi(s)$ for all states.

2 Increasing Performance at Each Step

2.1 General Principle

When an MDP is known exactly, policy iteration monotonically improves the policy at each step. However, as soon as we make approximations on the behavior of the MDP, we must use approximate policy iteration algorithms that are no longer monotonically increasing. It may result in oscillations, and possibly propose dangerous policies. Hence we may state a first definition of safety: *a safe algorithm provides a sequence of policies with non decreasing performances*. This definition was first stated in [4], which defined Safe Policy Iteration algorithms (SPI) but some of the main ideas were already present in [2], which defined Conservative Policy Iteration (CPI).

One schematic solution to this safety problem is the following. Consider a current policy π that we want to safely improve. To ensure theoretical guarantees on the performance, we shall be conservative and look at new policies that are not too far from π. Somehow, for instance maximizing an approximate Q function, we found a new policy π' that we expect to perform better. However, we do not know yet its true performance $J(\pi')$ and we can only estimate it. If this estimation returns an increase in performance that is *statistically significant*, then we can adopt a new policy. It will not be directly π', but rather a *convex combination* of π and π'. Else we stop and return the current policy π. Statistical inequalities provide good choices for the stopping criterion and for the parameter of this convex combination that guarantees a performance increase with high probability. This is the general sketch of the CPI and SPI algorithms.

2.2 Lower Bounds on the Performance Gap

CPI and SPI are based on finding appropriate lower bound on the performance gap between two policies π and π'. We have this first result ([2], lemma 6.1):

Lemma 2.1 *For any policies π and π' and any initial state distribution μ,*

$$J_\mu^{\pi'} - J_\mu^\pi = \int_{\mathcal{S}} d_\mu^{\pi'}(s)A_\pi^{\pi'}(s)\mathrm{d}s$$

Computing improvements of J can then be reduced to computing the advantage function. However, it also requires to estimate $d_\mu^{\pi'}$ for each candidate policy π', which is intractable. Hence we will try to replace $d_\mu^{\pi'}$ by d_μ^π which is easier to estimate. To achieve this we may first upper-bound the difference between the two future state distributions ([4], corollary 3.2):

Lemma 2.2 *For any policies π and π' and any initial state distribution μ,*

$$\int_{\mathcal{S}} |d_\mu^{\pi'}(s) - d_\mu^\pi(s)|\mathrm{d}s \leq \frac{\gamma}{(1-\gamma)^2}\sup_{s\in\mathcal{S}}\int_{\mathcal{A}} |\pi'(a|s) - \pi(a|s)|\mathrm{d}a$$

For simplicity, we will denote the supremum term as $||\pi' - \pi||_\infty$. The two previous results allow to derive a lower bound on the performance gap that depends only on quantities that we can estimate (simplified version of corollary 3.6 in [4]):

Theorem 2.3 *For any policies π and π' and any initial state distribution μ,*

$$J_\mu^{\pi'} - J_\mu^\pi \geq \mathbb{A}_{\pi,\mu}^{\pi'} - \frac{\gamma}{2(1-\gamma)^3} ||\pi' - \pi||_\infty^2$$

were $\mathbb{A}_{\pi,\mu}^{\pi'} = \int_\mathcal{S} d_\mu^\pi(s) A_\pi^{\pi'}(s)\mathrm{d}s$. In the case of a discrete MDP, $||\pi' - \pi||_\infty$ is known as soon as π and π' are. In the case of a continuous MDP where we learn parametric policies with policy gradient, we will lower-bound this distance with a polynomial function of the learning rate (see section 2.5).

$\mathbb{A}_{\pi,\mu}^{\pi'}$ can be approximated with accuracy ε by an estimate $\hat{\mathbb{A}}_{\pi,\mu}^{\pi'}$. [2] explains how to produce this approximation using trajectory sampling, rejection and importance sampling. This assumes that we have access to μ as a restart distribution (at any time, we can draw a state from μ), which can be D or not. [2] extensively discusses the importance of choosing μ as uniform as possible to implicitly allow exploration, but this is not our main focus here.

There is clearly a trade-off between choosing a new policy with a large advantage (first term) and choosing a policy not too far from the previous one (second term).

2.3 Conservative Policy Updates

We want to *conservatively* update our policy π towards a target policy $\bar{\pi}$, for instance the greedy policy. We make an update like: $\pi' = \alpha\bar{\pi} + (1-\alpha)\pi$, for $\alpha \in [0, 1]$. The choice of α is given by the maximization of the lower bound in a slightly stronger version of theorem 2.3. We get the following performance guarantee (corollary 4.1 in [4]):

Corollary 2.3.1 *If $\mathbb{A}_{\pi,\mu}^{\pi'} \geq 0$, then with $\alpha = \min\left(1, \frac{(1-\gamma)^2 \mathbb{A}_{\pi,\mu}^{\bar{\pi}}}{\gamma ||\bar{\pi}-\pi||_\infty \Delta A_\pi^{\bar{\pi}}}\right)$, we have the improvement*

$$J_\mu^{\pi'} - J_\mu^\pi \geq \begin{cases} \frac{(1-\gamma)^2 {\mathbb{A}_{\pi,\mu}^{\bar{\pi}}}^2}{2\gamma ||\bar{\pi}-\pi||_\infty \Delta A_\pi^{\bar{\pi}}} & \text{if } \alpha < 1 \\ \mathbb{A}_{\pi,\mu}^{\bar{\pi}} - \frac{\gamma}{2(1-\gamma)^2} ||\bar{\pi} - \pi||_\infty \Delta A_\pi^{\bar{\pi}} & \text{if } \alpha = 1 \end{cases}$$

where $\Delta A_\pi^{\bar{\pi}} = \sup_{s,s' \in \mathcal{S}} |A_\pi^{\bar{\pi}}(s) - A_\pi^{\bar{\pi}}(s')|$.

In all cases, this is a non-negative improvement, and so it satisfies our safety criterion.

We may signal here that the bounds from CPI were also independently extended in [6] which defined trust region policy iteration (TRPO). They provide lower bounds on the performance improvement between two general stochastic policies, hence replacing the mixture parameter α for conservative policy updates by a constraint on the Kullback-Leibler divergence between two policies. The exact version of TRPO is a safe algorithm, unlike the approximate one. Yet TRPO generally performs near monotonic policy improvements and proved very efficient on several tasks.

2.4 Safe Policy Iteration Algorithm

Conservative policy iteration (CPI) is a simple algorithm defined in [2], based on the maximization of the former lower bound at each policy improvement step. It has been refined in [4] with slightly better bounds, hence defining safe policy iteration (SPI) algorithms. We will focus on the approximate unique-parameter safe policy improvement algorithm (aUSPI in [4]). A variant with one update parameter per state in the policy update is also derived, but we focus on the simplest version, described in Algorithm 1.

SPI produces, with high probability, a sequence of improving policies (for the metric J_μ^π), and stops when no improvement can be guaranteed. If the target policy is fixed along the iterations, then the algorithm stops after $O(\frac{1}{(1-\gamma)^2\varepsilon})$ iterations.

However, the absolute quality of the policy produced by SPI is not really discussed in the articles. [2] relates the distribution μ to the performance loss (for J_D^π) between π produced by the exact CPI algorithm and the optimal policy: it is bounded by a constant times $\left|\left|\frac{d_{\pi^\star,D}}{\mu}\right|\right|_\infty$. If we have prior knowledge on the states most likely to be visited under the optimal policy, then it suggest to take μ close to $d_{\pi^\star,D}$. Else this bound is not very informative to quantify the performance of the algorithm.

```
Data: access to samples from the MDP through a restart distribution $\mu$
Result: a policy $\pi$ that cannot be improved by SPI
choose an initial policy $\pi$ at random;
while true do
    select the target policy $\bar{\pi}$ by maximizing a sample-based version of the $Q$-function;
    produce $\hat{\mathbb{A}}^{\bar{\pi}}_{\pi,\mu}$ an $\frac{\varepsilon}{3(1-\gamma)}$-accurate estimate of $\mathbb{A}^{\bar{\pi}}_{\pi,\mu}$ ;
    if $\hat{\mathbb{A}}^{\bar{\pi}}_{\pi,\mu} \geq \frac{2\varepsilon}{3(1-\gamma)}$ then
        set $\alpha \leftarrow \frac{(1-\gamma)^3(\hat{\mathbb{A}}^{\bar{\pi}}_{\pi,\mu} - \frac{\varepsilon}{3(1-\gamma)})}{\gamma||\bar{\pi}-\pi||_\infty}$ according to corollary 2.3.1;
        compute the new policy by a conservative policy update $\pi \leftarrow \alpha\bar{\pi} + (1-\alpha)\pi$ ;
    else
        stop and return the current policy;
    end
end
```

Algorithm 1: Safe Policy Iteration

The major computational bottleneck of SPI is the computation of the target greedy policy at each step. Hence the algorithm needs to enumerate all the states, which could be intractable for large or continuous state spaces. One approach could be to consider a set of parametric policies, and use the bounds to choose a safe step size in the policy gradient algorithm.

2.5 Extension to Continuous MDPs

There is a lot of literature on how to estimate the gradient direction in policy gradient algorithms, but not much on how to choose the learning rate. Unlike for classical optimization algorithms, we cannot do line search, as it would take too much policy evaluations.

Indeed, the previous results are extended in [3] to policy gradient algorithms for continuous MDPs. Instead of choosing a convex combination towards the greedy policy, *in the space of policies*, the previous bounds also help choosing a convex combination towards a gradient, *in the space of policy parameters*.

When specialized to policy gradient, the lower bound is polynomial in the learning rate α, so it can be optimized in α. We consider parametric Gaussian policies with fixed standard deviation σ and with mean a linear combination of a state vector $\phi(s)$ for each state s, so that $\pi(.|s,\theta) \sim \mathcal{N}(\theta^\top\phi(s), \sigma^2)$. The parameter for each policy is a vector θ. A closed form solution for α can be derived in the case of Gaussian policies (corollary 4.4 in [3]):

Theorem 2.4 *For any starting state distribution μ, if there exists M_ϕ uniformly bounding all basis functions, that is, $\forall s, ||\phi(s)||_\infty \leq M_\phi$, then there exists an optimal step size $\alpha^\star$ maximizing the lower bound that can be written in closed form. It is such that the performance improvement between two Gaussian policies $\pi_\theta \sim \mathcal{N}(\theta^\top\phi(s))$ and $\pi_\theta \sim \mathcal{N}({\theta'}^\top\phi(s))$, with $\theta' = \theta + \alpha^\star\nabla_\theta J_\mu(\theta)$, is lower bounded as follows:*

$$J_\mu(\theta') - J_\mu(\theta) \geq \frac{1}{2}\alpha^\star||\nabla_\theta J_\mu(\theta)||_2^2$$

As for theorem 2.3, the choice of $\alpha^\star$ relies a quantity that we can only approximate when the MDP is not known: $\nabla J_\mu(\theta)$. The policy gradient can be estimated by sampling, for instance using the REINFORCE algorithm, with N trajectories of length H. Using $N = \frac{M_\phi^2 H(1-\gamma^H)^2}{\delta\varepsilon^2\sigma^2(1-\gamma)^2}$ trajectories guarantees that all components of the gradient estimate have accuracy ε, with probability at least $1 - \delta$. In that case an approximate version of theorem 2.4 holds and guarantees a positive performance increase with high probability.

3 Surpassing a Performance Baseline

Another definition of safety is examined in [7]. The definition is the following one: *a policy is considered safe if its return is greater than a threshold J_-, with high probability* $1-\delta$. The idea is to avoid executing a policy unless we are sure it will pass such a safety test. The methods introduced in this article also allow ensuring a further notion of safety: indeed, they do not involve any hyper-parameter needing to be optimized before learning (such as a learning rate for example). This is extremely advantageous in such cases as medical applications (or in any case where the execution of a bad policy could be dangerous), where one can't afford to tune parameters by hand while executing policies until the optimal ones are found. Therefore, the notion of safety examined in this article is actually twofold.

To achieve this safety goal, one estimates *offline* the performance $J(\pi_e)$ of a new trajectory π_e using a set of trajectories $\mathcal{D}$ which have already been executed in the past. Each sample trajectory $(\tau_i, \theta_i) \in \mathcal{D}$ was executed using the policy parameter θ_i. Indeed, under mild assumptions, importance sampling allows computing unbiased estimates of the return of a given policy with parameter θ. The assumption is as follows:

$$\forall a, s, \theta_i : \pi(a|s, \theta_i) = 0 \Rightarrow \pi(a|s, \theta) = 0$$

By contraposition, this means that it is obviously impossible to evaluate $J(\pi_e)$ in an unbiased way if this policy can choose actions which can't be drawn by any of the already observed policies. However, if we relax this assumption, it is possible to show that the estimates of the returns are underestimated, which is still well suited for our setting since it eventually results in the lower bound on $J(\pi_e)$ being underestimated too, hence more conservative. The importance sampling estimates of $J(\pi_e)$ are $(\hat{J}(\theta, \tau_i, \theta_i))_{(\tau_i,\theta_i)\in\mathcal{D}}$ where $\hat{J}(\theta, \tau_i, \theta_i) = \mathcal{R}(\tau_i)\frac{Pr(\tau_i|\theta)}{Pr(\tau_i|\theta_i)}$. Based on these estimates, a $1-\delta$ confidence lower bound on $J(\pi_e)$ can be found.

This article introduces a batch algorithm which searches for safe policies, and can rely on three specific methods for computing confidence intervals, hence dealing with high confidence off-policy evaluation (HCOPE). The following inequality results are stated for a set $\mathbf{X}$ of random variables $(X_i)_i$, and will be applied to $\mathbf{X} = \mathcal{D}$ and $X_i = \hat{J}(\pi_e, \tau_i, \theta_i)$, where $\mathcal{D} = \{(\tau_i, \theta_i), i = 1, ..., n\}$.

3.1 Different Methods for Confidence Intervals

3.1.1 First method: concentration inequality (CI)

The first method comes from the following concentration inequality, proved in [8].

Lemma 3.1 *Let* $\boldsymbol{X} = (X_i)_{i=1,\dots,n}$ *be* n *independent positive and bounded random variables such that* $\forall\ 0 \le i \le n, \mathbb{E}[X_i] \le \mu$*. Let* $(c_i)_{0\le i\le n} \in \mathbb{R}^n$*. Let* $\delta > 0$ *and* $Y_i := \min(X_i, c_i)$*. Then with probability at least* $1-\delta$ *we have*

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

This inequality holds for n random variables that are not necessarily iid and doesn't depend directly on their range, as long as their mean μ is identical. The properties of importance sampling guarantee that these conditions are fulfilled by the $\hat{J}(\theta, \tau, \theta_i)$. An interesting result is that it is possible to deduce what a $1-\delta$ confidence lower bound on $\overline{x}$ would be if it were computed thanks to m random variables rather than n.

Theorem 3.2 *Let* $J_-(\boldsymbol{X}, \delta, n, c)$ *be a* $1-\delta$ *confidence lower bound on* $\overline{x}$ *computed thanks to* $\boldsymbol{X}$ *from the lemma with* $c_i = c$*, where* $\boldsymbol{X}$ *contains* n *random variables. If we had made the computation thanks to a set* $\boldsymbol{X}'$ *containing* m *random variables, where* $\boldsymbol{X}'$ *has the same variance as* $\boldsymbol{X}$*, then the lower bound would have been:*

$$J_-(\boldsymbol{X}, \delta, m, c) = \frac{1}{n}\sum_{i=1}^{n}Z_i - \frac{7c\log(2/\delta)}{3(m-1)} - \sqrt{\frac{2\log(2/\delta)}{mn(n-1)}\Big(n\big(\sum_{i=1}^{n}Z_i^2\big) - \big(\sum_{i=1}^{n}Z_i\big)^2\Big)}$$

where $Z_i = \min(X_i, c)$*.*

The idea is the following: we separate $\mathcal{D}$ into two disjoints sets $\mathcal{D}_{pre}$ (accounting for 1/20 th of $\mathcal{D}$) and $\mathcal{D}_{post}$ (accounting for the rest of $\mathcal{D}$). To compute $J_-(\mathcal{D}, \delta, m, c)$, the constant c is optimized to maximize $J_-(\mathcal{D}_{pre}, \delta, m, c)$, and then we set $J_-(\mathcal{D}, \delta, m, c^*) = J_-(\mathcal{D}_{post}, \delta, m, c^*)$.

3.1.2 Second method: Student's t-test for HCOPE (TT)

As $n \to \infty$, [7] states that $\hat{X} := \frac{1}{n}\sum_{i=1}^{n} X_i$ approximates a normal distribution "under mild assumptions", which allows computing the desired $1-\delta$ confidence lower bound, given by $\hat{X} - \frac{\hat{\sigma}}{\sqrt{m}} t_{1-\delta,m-1}$, where $\hat{\sigma}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \hat{X})$ and $t_{1-\delta,m-1}$ is the $1-\delta$ quantile of the Student distribution with $m-1$ degrees of freedom.

Remark: However the $(X_i)_i$ do not satisfy the assumptions of the strong law of large numbers (they are not iid) nor the weak one (they have the same mean but not the same variance). Moreover, the state-action space can be continuous so that no argument involving its finiteness can be invoked to account for that result.

3.1.3 Third method: Bootstrap confidence interval for HCOPE (BCa)

The last method for computing confidence intervals is to estimate the distribution of $\hat{X}$ using *Bias Corrected and Accelerated Bootstrapping* (BCa). It aims at estimating the true distribution of $\hat{X}$ with bootstrap, instead of assuming it is normally distributed like in t-tests. Hence BCa can produce bounds which are not too overly conservative.

In what follows, we will denote by $J_{-}^{\dagger}(\pi, \mathcal{D}, \delta, m)$, the $1-\delta$ confidence lower bound computed using each method associated to $\dagger = CI, TT, BCa$ respectively. A *semi-safe* algorithm will denote an algorithm which would be safe except that it makes a false but reasonable assumption (for example assuming that $\hat{X}$ is normally distributed). Such algorithms are reasonable because the whole method assumes that the environment is a partially observed MDP (POMDP) and is stationary, which both are less reasonable assumptions than what semi-safe algorithms could assume.

3.2 Policy Improvement Algorithm

The goal is to return the policy expected to perform best among the policies deemed safe: $\pi' = \arg\max_{\text{safe } \pi} g(\pi|\mathcal{D})$, where $g(\pi|\mathcal{D})$ denotes the importance sampling estimate of $J(\pi)$. We first split $\mathcal{D}$ into $\mathcal{D}_{train}$ (accounting for 1/5 of $\mathcal{D}$) and $\mathcal{D}_{test}$. On $\mathcal{D}_{train}$ we find the optimal safe policy, and then we perform a safety test for this policy on $\mathcal{D}_{test}$ to double-check that it is secure to use. If no solution is found, the algorithm returns NSF (No Solution Found).

```
PolicyImprovement$^{\dagger}_{\ddagger}(\mathcal{D}_{train}, \mathcal{D}_{test}, \delta, J_{-})$ ;
Result: Either returns a (semi-)safe policy or NSF
$\pi_c \leftarrow$ GetCandidatePolicy$^{\dagger}_{\ddagger}(\mathcal{D}_{train}, |\mathcal{D}_{test}|, \delta, J_{-})$;
if $J_{-}^{\dagger}(\pi_c, \mathcal{D}_{test}, \delta, |\mathcal{D}_{test}|) \geq J_{-}$ then
    return $\pi_c$
else
    return NSF
end
```

Algorithm 2: Policy improvement

For the GetCandidatePolicy$^{\dagger}_{\ddagger}$ algorithm we have two choices, either a first method which does nothing to avoid overfitting (which is well suited for a large $\mathcal{D}$) and a second one, which uses cross-validation to prevent overfitting especially on small datasets, but requires more computational complexity. In this case, π is searched in the space $\mu_{\alpha,\pi_0,\pi}$ of mixed policies with a cross-validated regularization parameter α. We define the following function:

$$f^{\dagger}(\pi, \mathcal{D}, J_{-}, \delta, m) = \begin{cases} g(\pi|\mathcal{D}) & \text{if } J_{-}^{\dagger}(\pi, \mathcal{D}, \delta, m) \geq J_{-} \\ J_{-}^{\dagger}(\pi, \mathcal{D}, \delta, m) & \text{otherwise.} \end{cases}$$

```
GetCandidatePolicy$^{\dagger}_{None}(\mathcal{D}, m, \delta, J_{-})$:
return $\arg\max_{\pi} f^{\dagger}(\pi, \mathcal{D}, J_{-}, \delta, m)$
```

```
GetCandidatePolicy$^{\dagger}_{k-fold}(\mathcal{D}, m, \delta, J_{-})$:
$\alpha^{*} \leftarrow \arg\max_{\alpha \in [0,1]}$ CrossValidate$(\alpha, \mathcal{D}, m, \delta, J_{-})$
$\pi^{*} \leftarrow \arg\max_{\pi} f^{\dagger}(\mu_{\alpha^{*},\pi_0,\pi}, \mathcal{D}, J_{-}, \delta, m)$
return $\mu_{\alpha^{*},\pi_0,\pi^{*}}$
```

The latter algorithm makes use of the following one:

```
CROSSVALIDATE($\alpha, \mathcal{D}, m, \delta, J_-$);
Partition $\mathcal{D}$ into $\mathcal{D}_1, ..., \mathcal{D}_k$;
result $\leftarrow 0$;
for $i = 1, ..., n$ do
    $\hat{\mathcal{D}} \leftarrow \cup_{i \neq j} \mathcal{D}_j$ ;
    $\pi^* = \arg\max_{\pi} f^\dagger(\mu_{\alpha,\pi_0,\pi}, \hat{\mathcal{D}}, J_-, \delta, m)$ ;
    result $\leftarrow$ result $+ f^\dagger(\mu_{\alpha,\pi_0,\pi^*}, \hat{\mathcal{D}}, J_-, \delta, m)$
end
return result$/k$
```

3.3 Incremental Algorithm: Daedalus

The previous methods allow defining an incremental algorithm which searches for a sequence of policies ensuring increasing performances with high probability. Hence, the methods developed so far bring back to the very first definition of safety. The algorithm starts with the policy π_0 given by the GETCANDIDATEPOLICY$^\dagger_{k-fold}$ algorithm. It takes a parameter β denoting the number of trajectories drawn at each step, a fifth of which are appended in $\mathcal{D}_{train}$, the rest put into $\mathcal{D}_{train}$. If it finds a policy that performs better than the current best one, it adds it to the set $\mathcal{C}$ containing the incremental sequence of safe policies and sets $\mathcal{D}_{test}$ back to $\emptyset$.

4 Constraining the Set of Possible Policies

Another way to define safety is requiring a policy to enforce some explicit constraints. For instance, we would want a robot not to make too fast or wide movements. This can be modeled by constrained MDPs (CMDPs).

4.1 Constrained MDPs

CMDPs are an extension of MDPs, extensively studied by E. Altman, where we require constraints to be enforced in expectation. Hence the constraints are defined along whole trajectories. Consider an MDP $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, D)$, we may define a CMDP $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, D, (C_j, d_j)_{1 \leq j \leq m})$, where each C_j is a function from state-actions to real values (like reward functions), and d_j is an upper bound constraint on the expected value of C_j. The constraint functions are very similar to the reward function and we can define $J^\pi_{C_j} = \mathbb{E}_{s \sim D, a \sim \pi}\left[\sum_{t=0}^{\infty} \gamma^t C_i(s_t, a_t) | s_0 = s\right]$. We may also define corresponding V, Q and A functions and other similar notations for each constraint.

Solving a CMDP is finding $\pi^\star$ maximizing J^π_D in the feasible set $\Pi_C = \left\{\pi : \forall j, J^\pi_{C_j} \leq d_j\right\}$. Under mild conditions, there exists a stationary policy that is optimal for the CMDP.

4.2 Lagrangian Optimization

A simple way to transform a constrained problem into an unconstrained one is to introduce Lagrange multipliers. Hence we may add to the expected reward objective J^π_D a penalty on constraint violations $J^\pi_{C_j} - d_j$. The Lagrangian is $\mathcal{L}(\pi, (\lambda_j)) = -J^\pi_D + \sum_{j=1}^m \lambda_j (J^\pi_{C_j} - d_j)$. The CMDP problem is equivalent to solving $\inf_\pi \sup_{\lambda \geq 0} \mathcal{L}(\pi, \lambda)$.

This inf-sup problem on the Lagrangian is studied in [1]. Under convexity assumptions and when Slater's conditions hold, the inf and sup can be exchanged and some optimal policies and multipliers exist, satisfying the Kuhn-Tucker conditions. For fixed Lagrange multipliers $\lambda_j \geq 0$, the optimal policy can be found with standard RL algorithms. Yet solving the whole CMDP problem means finding *learnable* multipliers. It requires to find a saddle point of the Lagrangian.

This can be challenging, as for instance first order optimization methods have stability issues when searching saddle points. Furthermore, this method is only safe asymptotically and gives no safety guarantees during training.

4.3 Constrained Policy Optimization

Constrained Policy Optimization (CPO) [10] is an adaptation of trust region methods, first proposed in [6], to solve CMDPs with parametric policies. It is a policy improvement algorithm that guarantees both an increase in reward at each step, and satisfaction of the constraints. Hence it provides *double safety*, according to two different definitions: increasing performance and constraints feasibility. The approach is again inspired by performance improvement bounds derived in [2].

We consider the problem of constrained local policy search, among parametric policies Π_θ, that is:

$$\pi_{k+1} = \arg\max_{\pi \in \Pi_\theta} J_D^\pi \text{ s.t. } \mathbf{d}(\pi, \pi_k) \leq \delta, \text{ and } \forall j, J_{C_j}^\pi \leq d_j$$

If **d** is the euclidean distance, this corresponds to standard policy gradient. Checking feasibility of a policy π requires to evaluate the constraints, for instance with off-policy sampling, and can be computationally expensive in high dimension. CPO replaces this problem by an approximate one, using functions that are easier to sample, in the wake of trust region methods.

The bounds from CPI can be adapted to this particular problem when **d** is the Kullback-Leibler divergence. Furthermore, we get similar bounds for the performance measure J_D^π and for the constraints $J_{C_j}^\pi$. [10] proves the following result (corollaries 1, 2, 3 in [10]):

Theorem 4.1 *For any policies π and π', write $\epsilon^{\pi'} = \max_{s\in\mathcal{S}} \left|\mathbb{E}_{a\sim\pi'}\left[A^\pi(s,a)\right]\right|$ and $\forall j, \epsilon_{C_j}^{\pi'} = \max_{s\in\mathcal{S}} \left|\mathbb{E}_{a\sim\pi'}\left[A_{C_j}^\pi(s,a)\right]\right|$. The following bounds hold:*

$$J_D^{\pi'} - J_D^\pi \geq \mathbb{A}_{\pi,D}^{\pi'} - \frac{2\gamma\epsilon^{\pi'}}{(1-\gamma)^2} \int_{\mathcal{A}} \pi'(a|s) \sqrt{\frac{1}{2}\mathbb{E}_{s\sim(1-\gamma)d_D^\pi}\left[D_{KL}(\pi'||\pi)(s)\right]} \mathrm{d}a$$

$$J_{C_j}^{\pi'} - J_{C_j}^\pi \leq \mathbb{A}_{\pi,C_j}^{\pi'} + \frac{2\gamma\epsilon_{C_j}^{\pi'}}{(1-\gamma)^2} \int_{\mathcal{A}} \pi'(a|s) \sqrt{\frac{1}{2}\mathbb{E}_{s\sim(1-\gamma)d_D^\pi}\left[D_{KL}(\pi'||\pi)(s)\right]} \mathrm{d}a$$

Note that these bounds are very similar to the ones of CPI and SPI, having ensured consistency between the notations of the articles (one must be careful with the normalization of the future state distribution function).

This theorem is used in [10] to give worst-case performance degradation guarantees for trust-region methods, and hence CPO, that search policy updates in a *trust region* around the current policy: $\{\pi : \mathbb{E}_{s\sim\pi_k}[D_{KL}(\pi||\pi_k)(s)] \leq \delta\}$. It helps justify the heuristics of trust regions and explain the near monotonicity of TRPO.

Similarly, the bounds are used to bound the worst-case constraint violation of CPO and show that they are always approximately enforced. Since the raw adaptation of TRPO to CMDPs can be hard to solve in high dimension, [10] provides an approximate algorithm for CPO. Each policy update is:

$$\begin{aligned}\theta_{k+1} = \arg\max_\theta \nabla J_D(\theta_k)^\top(\theta - \theta_k) \\ \text{s.t. } \frac{1}{2}(\theta-\theta_k)^\top H(\theta-\theta_k) \leq \delta \text{ and } \forall j, J_{C_j}(\theta_k) - d_j + \nabla J_{C_j}(\theta_k)^\top(\theta-\theta_k) \leq 0\end{aligned}$$

where H is the Hessian of the KL-divergence, a.k.a the Fisher information matrix. This is a second order approximation of the constrained local policy search previously defined. This roughly defines the CPO algorithm, which solves the dual which is a convex program, and then recovers the solution of the primal at each policy update. This algorithm is approximately safe, in the sense that it approximates an algorithm for which, with high probability, the constraints are enforced and the performance is guaranteed to increase at each iteration.

4.4 Lyapunov Functions and Safety

Yet CPO only applies to local policy gradient algorithms, and cannot easily generalize to other algorithms. [12] recently proposed an approach using Lyapunov functions to solve CMDPs. [1] had already considered Lyapunov functions associated to MDPs to prove results on the Lagrangian approach to CMDPs.

Lyapunov functions on a space $\mathcal{X}$ are continuously differentiable functions $L : \mathcal{X} \to \mathbb{R}_+$, with $L(0) = 0$ and $L(x) > 0$ for $x \neq 0$. They are tools that help estimate the stability of dynamic systems. In the context of CMDPs with only one constraint C_0, [12] defines Lyapunov functions associated to a CMDP as $L : \mathcal{S} \to \mathbb{R}_+$, with $L(s_0) \leq d_0$ (s_0 is the initial state and d_0 the threshold on the constraint), and $\forall s \in \mathcal{S}$, $L(s) \geq T_{\pi,C_0}[L](s)$, where T_{π,C_0} is the Bellman operator with respect to a policy π and constraint C_0.

Beginning from an initial safe policy satisfying the constraint, one can ensure safety of new policies searched in the set of L-induced policies at state s: $\{\pi : L(s) \geq T_{\pi,C_0}[L](s)\}$. [12] explains how to build a convenient Lyapunov function, such that the optimal policy can be reached by this iterative method. Then standard dynamic programming and reinforcement learning algorithms, namely policy iteration, value iteration, Q-learning and policy improvement, can be adapted to handle CMDPs with this Lyapunov approach.

Originally, Lyapunov functions have been used in the field of control to study stability properties. Following this approach, [11] defined safety in reinforcement learning in terms of *stability*. This is quite natural if we think of the inverted pendulum problem. Guaranteeing stability through learning helps finding a policy that avoids falls of the pole. The idea is to progressively extend an estimated *safe attraction region* over states. Attraction regions are subsets of the state space such that any trajectory starting from there stays in this region. Hence safety is no longer considered in terms of policies, but rather of states. One builds regions of attraction as level sets of Lyapunov functions. This allows to derive a model-based policy optimization algorithm that strictly decreases a Lyapunov function, hence gradually increasing the region of attraction. It is safe according to the safety definition related to *stability*.

5 Discussion

We explored three categories of methods to ensure safety in RL: (1) increasing performance at each step, (2) surpassing a performance baseline and (3) constraining the set of possible policies. Obviously, each one relies on a different definition of safety. The first two definitions are close, the second one being more general. Yet the algorithms that we considered have similar architectures. In fact, the Daedalus algorithm acts as if the user resets the performance baseline requirement at each policy iteration step to match the performance of the previous policy. Hence we get back to the first definition of safety as monotonicity. However, keeping this second definition, we could imagine a different kind of algorithm: the user once and for all sets a performance baseline, preventing the agent from performing worse policies. Thus the user imposed a constraint on the expected reward. This can be modeled by a CMDP, choosing minus the reward function as a unique constraint function. It means that when the performance baseline is fixed along the whole experiment, we get back to the third definition of safety.

So in a way, the second definition interpolates between the first and the third ones. The first definition is local, in the sense that the performance requirement changes during the execution of the algorithm. At each time step, the performance requirement is increasingly restrictive. With the third definition, we set a global constraint on the policies. This constraint is never modified and is part of the model. When the constraint is directly imposed on the expected reward, then this can be as well modeled by the second definition. But the constraints can apply to other functions. For instance, we might measure the performance of a self-driving car by the fact that it reaches a given destination. Setting constraints on the reward function can do little to curb dangerous behaviors. For instance we could set constraints on the velocity or the acceleration, that are not directly related to the reward function.

Most of the presented methods are not specific to a particular RL algorithm and have some generality. For instance, the bounds from CPI, originally developed for policy iteration are naturally extended to the policy gradient algorithm. The results from high confidence policy improvement are as well applied to discrete or continuous state space problems. CPO is an adaptation of TRPO, a policy gradient algorithm achieving state-of-the-art results. We have mainly focused on approaches applied to model-free algorithms. We may also consider model-based approaches. For instance, [9] uses uncertainty measures on the model to define a safe algorithm. When such uncertainties are accessible, this approach makes use of robust MDPs and is less conservative than [7]. The advantage of this method is that it takes into account the fact that the uncertainties are evenly distributed among states.

References

[1] Eitan Altman. Constrained markov decision processes with total cost criteria: Lagrangian approach and dual linear program. *Mathematical methods of operations research*, 48(3):387–417, 1998.

[2] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. 2002.

[3] Matteo Pirotta, Marcello Restelli, and Luca Bascetta. Adaptive step-size for policy gradient methods. In *Advances in Neural Information Processing Systems*, pages 1394–1402, 2013.

[4] Matteo Pirotta, Marcello Restelli, Alessio Pecorino, and Daniele Calandriello. Safe policy iteration. In *International Conference on Machine Learning*, pages 307–315, 2013.

[5] Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

[6] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

[7] Philip Thomas, Georgios Theocharous, and Mohammad Ghavamzadeh. High confidence policy improvement. In *International Conference on Machine Learning*, pages 2380–2388, 2015.

[8] Philip S Thomas, Georgios Theocharous, and Mohammad Ghavamzadeh. High-confidence off-policy evaluation. In *AAAI*, pages 3000–3006, 2015.

[9] Mohammad Ghavamzadeh, Marek Petrik, and Yinlam Chow. Safe policy improvement by minimizing robust baseline regret. In *Advances in Neural Information Processing Systems*, pages 2298–2306, 2016.

[10] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *International Conference on Machine Learning*, pages 22–31, 2017.

[11] Felix Berkenkamp, Matteo Turchetta, Angela Schoellig, and Andreas Krause. Safe model-based reinforcement learning with stability guarantees. In *Advances in Neural Information Processing Systems 30*, pages 908–918, 2017.

[12] Yinlam Chow, Ofir Nachum, Edgar Duenez-Guzman, and Mohammad Ghavamzadeh. A lyapunov-based approach to safe reinforcement learning. *arXiv preprint arXiv:1805.07708*, 2018.