1. Introduction

Resurgence of artificial intelligence and advancements in it has led to automation of many cyber-physical systems [1, 2, 3]. Such systems have multiple operating states and they evolve based on what the underlying intelligent agent or the controller decides to do. The prime factor for efficient system operation lies in the control decisions of the agent.

Cyber-physical systems having multiple operating states that can be controlled by an intelligent agent are often modeled using the framework of Markov decision processes [4, 5]. Informally, a Markov decision process (MDP) is characterized by states, actions and their evolution. The states correspond to the operating states of the physical system. The agent chooses *actions* to control the system by exploring various actions in an iterative manner. The iterative learning of the agent is facilitated by a *feedback* from the system where the agent receives *reward* or *cost* for the choice of actions. This feedback is a sort of "reinforcement" to the agent which indicates whether the agent is learning in the desired manner or not. This approach of learning to choose the optimal (or right) actions based on indirect feedback is known as reinforcement learning (RL) [6].

The success of the RL approach has been demonstrated on a broad range of applications [7, 8, 9]. Further, recent advances in deep learning have also improved the scalability of RL algorithms [10]. However, a grave concern arises when we speak of these potential applications of RL, which is that of *safety* [11]. In this paper, we focus on issues which arise in safety-critical systems [12], controlled by RL autonomous learning algorithms. This problem is of profound importance in systems like self-driving cars, where during learning, any decision/action applied to the car has to guarantee safe driving or at least minimize the frequency of unsafe behaviour. RL algorithms [13] learn by exploring actions, which leads to the evolution of the system to different states. The states of the system reached by taking random actions may not be safe, or even the entire trajectory may be unsafe. How do we build new RL algorithms which minimize risky/unsafe operation during learning and learn a safe policy?

We bring in the aspect of safety into RL algorithms using two important ideas. The first idea is to utilize the constrained RL framework [14]. Using this, the RL agent explores decisions whose average cost violations are well within a specified budget. The second idea is that of off-policy learning [6]. Using off-policy learning, the learning agent learns to take optimal decisions in the operating states based on decisions that have earlier been taken. This is often referred to as the behaviour policy (the decision function or rule from states to actions, see Section II). This policy is typically specified by an expert or known to be a safe policy.

In our work, we formulate the problem of choosing safe decisions as a constrained MDP problem. Further, while solving this constrained MDP, we impose an additional requirement of learning only from existing or historical data (off-policy learning). To achieve this goal, we develop an RL algorithm that utilizes the cross-entropy method (CEM). The CEM can handle constraints effectively

and at the same time is amenable to off-policy learning. Further, our RL algorithm can utilize the non-linear function approximations provided by neural networks for large scale problems. We now summarize our contributions:

- We propose a sample-efficient RL algorithm for safe exploration based on the cross entropy method.
- Given safety threshold, i.e., a quantity which indicates the level of safety violations that can be tolerated by the system, our algorithm utilizes the available data to find a policy that is feasible. Also the safety constraints are kept under check during the learning process as well.
- The algorithm utilizes limited data samples for policy improvement during learning.

1.1. Organization of the Paper

The next section provides a brief background on MDPs and the mathematical formulations employed in safe RL works. Section 3 describes our problem formulation and the safe RL algorithm we propose. In Section 4, we experimentally validate and analyze our proposed algorithm. Finally, in Section 5, we provide concluding remarks and interesting future directions.

2. Background

A Markov decision process (MDP) [5] is formally defined as a tuple $M = \langle S, A, P, R, \gamma \rangle$, where S is the set of states of the system, A is the set of actions (or decisions) that can be taken in these states (assume $|A| < \infty$ and all actions are feasible in every state), $P : S \times A \times S \to [0, 1]$ is the transition probability function that governs the dynamics of the system based on the choice of actions in states, $R : S \times A \to \mathbb{R}$ is a real-valued reward feedback for choosing actions in states, $0 \leq \gamma < 1$ denotes the discount factor and $\beta : S \to \mathbb{R}$ denotes the initial distribution over states.

A stationary policy $d : S \to A$ is a state-dependent decision rule that prescribes the action to be taken in state s, $\forall t \geq 0$. For an infinite horizon MDP, the often used performance measure is the expected sum of discounted rewards (value function) of a stationary policy $\pi = (d, d, \ldots)$ defined as:

$$V^{\pi} = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, d(s_t)) | s_0 \sim \beta(s)\right]. \tag{1}$$

Here, the discount factor γ measures the present value of a unit reward that is received one epoch in the future. A policy π^* is optimal w.r.t the long-run discounted criterion if it maximizes (1) over all stationary policies Π. The dynamic programming (DP) [5] techniques iteratively solve (1) and obtain an optimal policy π^* and the optimal value function for a given MDP if the model information specified via P and R is known. Reinforcement learning (RL) [6] algorithms, on the other hand, are model-free methods to obtain the optimal policy when model information is not available and only samples are available.

2.1. Safety Formulations

As discussed in Section 1, an RL agent has to learn safe policies for safety-critical applications and not mere policies that maximize (1). The agent additionally also has to ensure that the learnt policy does not violate the prescribed safety measures. This brings in a clear case of trade-off in the learning process, i.e., the agent needs to maximize discounted reward return, however at the same time can only explore actions that does not jeopardize the system functioning. Hence system performance and safety can become competing objectives in many real-world scenarios.

For safe reinforcement learning, multiple formulations have been attempted in the literature as described below:

(a) Risk-based objective functions: To assess the average risk of a policy, it's performance criterion is specified in terms of measures like conditional value at risk (CVaR) [15] and via risk-sensitive MDPs [16].

(b) Safety constraints: The basic formulation of MDPs is extended to the constrained MDP formulation [14]. The objective in a constrained MDPs is the following:

$$\begin{aligned} \max_{\pi \in \Pi} \quad & V^{\pi} \\ \text{s.t.} \quad & S_i(\pi) \leq \alpha_i, \quad 1 \leq i \leq n, \end{aligned}$$

where V^{π} is defined by (1) and Π is the set of all stationary policies. The functions S_i, $1 \leq i \leq n$ are expected long-term safety constraints that need to be strictly satisfied, i.e., value of each constraint S_i must be lower than the threshold value α_i.

(c) Exploration strategies: The learning algorithms follow varied prior exploration strategies to collect more information about the state and action spaces of the MDP model. The collected data of the underlying model is leveraged to compute safe policies.

(d) Modelling safety constraint: A safety constraint $f(\mathbf{x})$ is modeled as a Gaussian process (GP) [17]. So the safety function is characterized by mean and covariance functions, denoted as $m(\mathbf{x})$ and $k(\mathbf{x}, \mathbf{x}')$ respectively. Using experience data from the environment, these properties of the safety function are learned. The learnt safety function can be further optimized to get an optimal safe policy.

2.2. Related Work on Safe RL

Prior works on safe RL (see [18]) utilizes one of the aforementioned safety formulations. Constrained policy optimization approaches [19, 20] utilize problem formulation (a) and (b) as described in Section 2.1. A trust-region [19] based method proposes surrogate objective and constraint functions to suit model-free optimization. This method is suited for the case when the objective and

constraint functions are hard to estimate. However, the method is data intensive and during learning, the algorithm needs to explore all actions, making it ineffective for avoiding unsafe actions. [20] proposes a constrained RL optimization algorithm adapted from cross-entropy method [21]. Using trajectory data samples, the objective function as well as the safety constraints are computed. Based on the computed values, the policy parameters are updated. As with [19], this method also requires exploration of all actions (including unsafe) for evaluating a policy.

Gaussian process (GP) [22, 23, 24] based approaches use mathematical formulation (b) and (d) as described in Section 2.1. In these approaches, the exploration is guided by a constraint on the acceptable safety values. Every state is characterized by this safety value. The safety function is apriori unknown and the idea is to find a set of "safe" states [22], i.e those states whose safety level is above a certain pre-fixed threshold. The initial set of safe states is expanded by exploration. Though [22] does not consider reward optimization, its extensions [23, 24] do consider reward optimization which also is modeled as a GP. In addition, [24] experiments with a stopping rule for safe states exploration.

3. Proposed Approach

In this section, we describe our safe RL formulation utilizing safety formulations (b) and (c) of Section 2.1.

3.1. Problem Formulation

A safety-constrained problem is modeled as a constrained MDP (CMDP) $M = \langle S, A, P, R, c_1, \ldots, c_K \rangle$. In this notation, S, A, P and R are same as the corresponding quantities in the specification of MDP in Section 2. However, in addition to the reward function, there are K safety constraint functions $c_i : S \times A \to \mathbb{R}$, $1 \leq i \leq K$. For $n \geq 0$, let $r(n)$, $c_1(n), c_2(n), \ldots, c_K(n)$ denote the single-stage reward and constraint values at instant n. We assume that reward and constraints are (uniformly) bounded and independent random variables. The state evolution of the Markov process under any policy π is denoted $\{X_n\}$ and the corresponding action sequence is $\{Z_n\}$. Thus, at instant n, when state X_n is observed and action Z_n is taken, a reward $r(n)$ and constraint costs $c_1(n), \ldots, c_K(n)$ are obtained. The process also moves to the next state X_{n+1}. Let $R(s,a) = \mathbb{E}[r(n)|X_n = s, Z_n = a]$ and $c_i(s,a) = \mathbb{E}[c_i(n)|X_n = s, Z_n = a]$, $i \leq i \leq K$.

In our problem formulation, the set of policies we consider is specified by a state-dependent, randomized decision rule $u : S \to \mathcal{P}(A)$. The map u outputs a probability distribution over the action space A for every state in S. A randomized policy $\pi = (u, u, \ldots)$ belongs to a large class of policies, which is known [14] to contain feasible policies for the problem we consider (see (2)). Using a randomized policy π, we define $U^\pi(s) = \mathbb{E}\left[\sum_{m=0}^{\infty} \gamma^m r(m)|X_0 = s, \pi\right]$ and $W_i^\pi(s) = \mathbb{E}\left[\sum_{m=0}^{\infty} \gamma^m c_i(m)|X_0 = s, \pi\right]$. Let β be the initial distribution over

states. Given this specification, our mathematical problem formulation is the following:

$$\begin{aligned} \max_{\pi \in \Pi} \quad & V^{\beta}(\pi) \\ \text{s.t} \quad & G_i^{\beta}(\pi) \leq \alpha_i, \end{aligned} \tag{2}$$

where $V^{\beta}(\pi) = \sum_{s \in S} \beta(s) U^{\pi}(s)$, $G_i^{\beta}(\pi) = \sum_{s \in S} \beta(s) W_i^{\pi}(s)$ for $1 \leq i \leq K$. G_i^{β} is the long-term measure w.r.t the i^{th} constraint function and the initial state distribution β.

If $\pi = (u, u, , \ldots)$ is a randomized policy, and $u(s)$ is the probability distribution over actions in state s, the probability of picking action a in state s is denoted as $\pi(a|s)$. Additionally, when two policies are involved (say μ and π), we differentiate between them using this notation as $\mu(a|s)$ and $\pi(a|s)$, which give the respective probabilities of taking action a under both policies in state s.

The key problem in (2) is that the RL agent needs to find a randomized policy π^* which not only maximizes V^{β} but also results in each of the safety functions G_i to be lower than a threshold value α_i. A policy which respects the threshold on the constraint functions is said to be *feasible*. Thus, π^* needs to be a feasible policy which also maximizes V^{β}. We restrict our attention to the randomized policy space which is parameterized by $\theta \in \mathbb{R}^{\omega}$, where $\omega > 1$. This space is denoted as $\Pi_{\Theta} = \{\pi^{\theta}(a|s) : s \in S, a \in A, \theta \in \mathcal{C} \subset \mathbb{R}^{\omega}\}$.

Even though the constrained formulation incorporates safety in the policy learnt by RL algorithm, it does not restrict the kind of actions the RL agent can explore during learning. This can potentially jeopardize the system. Therefore, to impose restriction on the exploration of actions, we consider off-policy learning as opposed to on-policy learning. In off-policy learning, the agent is provided with an existing (historic) dataset or a behaviour policy from which the agent is allowed to choose actions. The goal of the agent is to learn an optimal policy meeting the safety constraints based on experience samples from the behaviour policy. This is a hard learning problem when compared to the on policy learning. This is because the agent gets only indirect feedback when using the behaviour policy, as opposed to executing and evaluating the current policy in the on policy setting.

The RL agent has the following input dataset to compute a good feasible policy: a sample trajectory $\mathcal{D}$, which is a collection of *experience* tuples, i.e., tuples of the form $e_n = (s(n), a(n), r(n), c_1(n), \ldots, c_K(n))$. So, $\mathcal{D} = \{e_0, e_1, \ldots\}$ is generated from a behaviour policy μ (a randomized policy). Further, the RL agent also has access to the threshold values α_i, $1 \leq i \leq K$. The agent has no model information concerning P, R and c_i.

3.2. Constrained RL Optimization: Lagrangian and Cross Entropy Methods

The classical theory underlying a constrained optimization problem is that of Lagrange multipliers [25]. Prior algorithms in RL also utilize the Lagrangian theory to derive actor critic type algorithms [26]. More recent algorithms in

the same line of work are [19, 27] that utilise neural network architectures for function approximation of large complexity problems. However, it is observed that [19] does not satisfy feasibility of constraints during learning. Additionally, these are on-policy methods, and demand a large dataset in terms of the number of experience samples for training.

To find a good policy, [26, 27] first relax the problem into finding optimal policy of a relaxed unconstrained MDP for a given vector of Lagrange multipliers. The objective of this unconstrained MDP is obtained through the standard procedure of relaxation [25]. In the next stage, with perturbed policy parameters, two parallel simulations of the CMDP are obtained. The objective functions corresponding to the two different policies are estimated using TD errors. The Lagrange parameters are also updated using these parallel simulations. [26] shows that this algorithm converges to a good feasible policy. While [26] uses linear function approximation for objective and constraint function value estimation, [27] utilizes neural networks for the same and is a multi-agent formulation. The neural network function approximators in [27] are tuned using loss functions based on TD errors and are adpated from [26] for multiple agent setting.

The main problem that we see in the above works is that in many real-world applications, the RL agent can obtain samples for only one policy. So the idea of using a generative model for obtaining two parallel simulations is not practically feasible. Additionally, in many cases, only limited experience data samples (from some unknown policy) of the physical system may be available. Hence, when there is a limitation on obtaining experience data samples, using an algorithm based on a generative model is not a reasonable choice. Another issue is that the algorithm proposed in [26] requires the tuning of multiple weight parameters. These problems motivate us to adapt other optimization solution techniques for the problem posed in (2) and we consider cross-entropy method for the same.

Cross-entropy method [21] (CEM) is a zero-order optimization method and does not use gradient or other higher-order derivatives of the objective function. Hence, it can be used in scenarios where the objective function does not possess smoothness properties and computation of higher order derivatives is intensive. The basic idea of CEM is simple: in each iteration, samples of objective function are generated. A set of elite samples is chosen. These elite samples correspond to inputs where the objective function value is high among the samples obtained. Then, based on these samples, the distribution from which the inputs are sampled is updated.

CEM has been used in RL previously [21, 20, 28] as a policy iteration algorithm. In this setting, each sample of the objective function V corresponds to a particular policy, sampled from a distribution. Based on the elite samples of V, the distribution from which the policies are obtained are updated. [21] is an initial attempt at improving policy over multiple iterations for a finite-horizon, finite MDP setting. The policy updation is based on finding frequency of occurrences of actions and states in the elite samples. This sort of updation does not allow for approximations in policy space and hence is not scalable to complex

MDPs like continuous space MDPs.

In contrast, [28, 20] use approximations for the policy space with CEM. [28] proposes a stochastic approximation variant of CEM with policy and value function approximation for a simple MDP setting, whereas, [20] proposes CEM for a constrained setting. The constrained setting in [20] is for episodic RL problems, while [28] deals with discounted unconstrained RL problems. Another feature in [20] is that unlike (2), the value function as well as constraint cost functions are not defined for every decision of the RL agent. Instead objective and cost functions are defined for the complete trajectory. This simple setting enables comparison of objective and constraint function values, but is highly sample inefficient when objective and constraint function estimation for every policy sample is involved.

The stochastic approximation variant of CEM proposed in [28] is defined for the maximization of the objective function. This algorithm iteratively predicts the objective function of a policy sample. The elite samples are monitored using the concept of quantiles. The $(1-\rho)$ quantile of a function $f(\cdot)$ under the probability distribution P is a number $q = \sup\{l : P(\{f(x) \geq l\}) \geq \rho\}$. For the RL setting, f is the objective function and in CEM, one would like to estimate the $(1-\rho)$ quantile of this function from the obtained samples. For a pre-fixed value of ρ, the stochastic approximation variant of CEM proposed in [28] iteratively improves the policy distribution by tuning this quantile. It is easy to observe that for constraint funtions the same logic can be used (with minimization instead of maximization) and we can optimize the constraint function based on the policy distribution.

However, there is a problem with this approach. The problem is that this method involves uni-objective optimization, i.e., we can either maximize the value function of a policy or minimize the constraint cost functions, *but not both*. This method cannot find policy distribution which achieves a sort of trade-off between the objective and constraint cost functions. The attainment of such a policy is the crux of the problem specified in (2), wherein the threshold values specify the trade-off that we care about. For e.g., if we consider a MDP which has say 10 policies $\pi_1, \ldots, \pi_{10}$. The objective functions of each of these policies take values from 1 to 10, i.e., $V^{\beta}(\pi_j) = j$, $1 \leq j \leq 10$ and constraint cost G_1^{β} takes values such that $G_1^{\beta}(\pi_j) = (j-10)$. Here β is an arbitrarily chosen initial-state distribution. With a $\rho = 0.2$ value, the CEM variant proposed in [28] will iteratively improve policy to pick distributions which favour π_1, π_2 when constraint cost function needs to be minimized. However, it will concentrate the policy distribution to favour policies π_9, π_{10} when objective function V needs to be maximized with the same value of ρ. Clearly, this method cannot achieve a trade-off between V and G_1 as seen in this simple example.

3.3. Our Approach

The method we propose (shown in Algorithm 1) is also based on CEM, but alleviates the problems described above. The procedure is shown for one safety constraint, i.e., $K = 1$ in Algorithm 1, but the same holds for any $K > 1$ also.

The algorithm takes as input the state and action space dimension, the number of constraint functions K, the quantile sequence $\rho_j,\ 1 \leq j \leq M$ and a sample trajectory generated using a policy μ with initial state distribution $\beta \in \mathcal{P}(S)$. Here M is the total number of updates of the policy parameters (Line 1 in Algorithm 1).

3.3.1. Approximation in Policy Space, Value and Constraint Estimation

The algorithm utilizes a dataset $\mathcal{D}$ consisting of experience tuples generated by following a behaviour policy μ, where μ is also a SRP. The RL agent has no access to any other dataset, forcing it to learn good feasible policies using samples from $\mathcal{D}$. This scenario is very common in robotic applications where certain trajectory samples are collected using expert supervision and the RL agent needs to learn optimal behaviour strategy from the available samples only, since collecting robot samples for each policy may be extremely cumbersome.

The policy search space is defined by the parameter $\theta \in \mathcal{C}$. This set $\mathcal{C}$ is defined as the support of a distribution from the natural exponential family (NEF) class [21]. This class consists of Gaussian, Poisson, Exponential and Binomial distributions. In Algorithm 1, we consider Gaussian distributed policy parameters with covariance being the identity matrix. This class is denoted as $\mathcal{F}_{\mathcal{V}} = \{f_v : f_v = \mathcal{N}(v, \mathcal{I})\}$. We sample θ from a distribution $f_v \in \mathcal{F}_{\mathcal{V}}$. For a given parameter vector θ, the policy analytical form π^θ is defined (usually this form is either the softmax function or the Gaussian distribution) and a policy is obtained. For this policy, using the samples from $\mathcal{D}$, the RL agent computes the expected objective and constraint function values. In keeping with the terminology used in policy evaluation works [29], we call π^θ as the *target* policy, since the objective and constraint functions corresponding to π^θ are to be estimated. It is easy to observe that behaviour and target policies are not same and so, the RL agent needs off-policy value estimation as denoted by the PREDICT subroutine in Algorithm 1 (see Line 6).

The objective and constraint function values are approximated using linear function approximation. For a state $s \in S$, the features are represented as $\phi(s) \in \mathbb{R}^m$ and the state-action features denoted as $\psi(s,a) \in \mathbb{R}^l$.

3.3.2. Objective and safety constraint evaluation

The PREDICT subroutine tunes the weights $x, y \in \mathbb{R}^m$, such that $\phi x \approx V^\beta(\pi^\theta)$ and $\phi y \approx G^\beta(\pi^\theta)$. In the experiments with discrete action space, $\pi^\theta(a|s) = \frac{\exp(\theta^\top \psi(s,a))}{\sum_{b \in A} \exp(\theta^\top \psi(s,b))}$. In the experiments, we mostly adapt Emphatic TD [30] (ETD) and Perturbed TD [31] (PTD) off-policy policy evaluation algorithms for the PREDICT subroutine (Line 6). Let $\eta_t = \frac{\pi^\theta(a_t|s_t)}{\mu(a_t|s_t)}$ be the importance sample estimate at epoch t. Similarly, let x_t be the weight of V^β and y_t the weight vector of G^β in iteration t. The ETD and PTD update rules are:

- Emphatic TD (ETD): We define the follow-on trace $F_0 = 1$ and eligibility trace $b_{-1} = 0$. Let $\lambda = 0.1$. The temporal difference error w.r.t reward is

defined as $\delta_t^x = R(s_t, a_t) + \gamma x_t^\top \phi(s_{t+1}) - x_t^\top \phi(s_t)$. Using this error, the weight vector x_t is updated as follows:

$$x_{t+1} = x_t + stepsize * \delta_t^x b_t \tag{3}$$
$$b_t = \eta_t(\gamma\lambda b_{t-1} + M_t\phi(s_t)) \tag{4}$$
$$M_t = \lambda + (1-\lambda)F_t \tag{5}$$
$$F_t = \eta_{t-1}\gamma F_{t-1} + 1. \tag{6}$$

While w.r.t constraint i, temporal difference error is defined as $\delta_t^i = c_i(s_t, a_t) + \gamma x_t^\top \phi(s_{t+1}) - x_t^\top \phi(s_t)$. The update for y_t is similar where in (3), x_t is replaced by y_t and δ_t^x is replaced by δ_t^i.

- Perturbed TD (PTD): Define perturbation parameter $pt \in (0, 1)$. The temporal difference error term w.r.t reward is defined as $\delta_t^x = R(s_t, a_t) + \gamma x_t^\top \phi(s_{t+1}) - (1+pt)x_t^\top \phi(s_t)$. Using this error, the weight vector x is updated as follows:

$$x_{t+1} = x_t + stepsize * \delta_t^x b_t \phi(s_t). \tag{7}$$

While w.r.t constraint i, temporal difference error is defined as $\delta_t^i = c_i(s_t, a_t) + \gamma x_t^\top \phi(s_{t+1}) - (1+pt)x_t^\top \phi(s_t)$. The update for y_t is similar where in (7), x_t is replaced by y_t and δ_t^x is replaced by δ_t^i.

More details about ETD and PTD can be found in [31].

3.3.3. Updation of Policy Parameter Distribution

The policy parameter distribution is updated using the output of the PREDICT subroutine. A number of policy parameter samples are utilized and their values estimated (Lines 4-7). These samples are collected and the POLICY_UPDATE subroutine updates this distribution in a manner that the trade-off between value and constraint functions is maintained (Line 8). This takes place over a number of iterations (Lines 3-10). The number of θ samples for policy iteration j is denoted as n_j. For each such sample, the approximate values of $V^\beta(\pi^\theta)$ and $G_i^\beta(\pi^\theta)$ are computed using $\mathcal{D}$. Further, the computed pairs are sorted and the number of samples satisfying the constraint threshold α_1 are observed. Let this number be denoted as m_j. If $m_j < \lfloor \rho n_j \rfloor$, then this implies we have deficient samples. So, θ from the remaining $(n_j - m_j)$ samples whose $V_i^\beta(\pi^\theta)$ values are high are used for updating the policy parameter distribution. If otherwise $m_j > \lfloor \rho n_j \rfloor$, then from the m_j samples of θ, the ones having highest values of $V^\beta(\pi^\theta)$ are selected for updating the policy parameter distribution. This forms the POLICY_UPDATE subroutine (Lines 1-12 in Algorithm 2).

Algorithm 1 Off-Policy based Constrained Safe RL (OffPol_CCE)

Input: M, $\{\rho_j,\ 1 \leq j \leq M\}$, $\beta \in \mathcal{P}(S)$, $K = 1$, $\Phi = \{\phi(s)|s \in S\}$ and $\Psi = \{\psi(s,a)|s \in S, a \in A\}$, α_1 and $\mathcal{D} = \{e_0, e_1, \ldots, e_T\}$, where T is fixed apriori. Also given is the class parameterized policies Π_Θ and a NEF family $F_\mathcal{V}$ of distributions.
Initialize: $j = 1$, $v_1 \in \mathcal{V}$, $s_p \in (0,1)$, $\{n_l : 1 \leq l \leq M, n_l > 10\}$
for $j = 1$ to M **do**
  Sample $\theta_1, \ldots, \theta_{n_j} \sim f_{v_j} \in F_\mathcal{V}$ i.i.d
  for $i = 1$ to n_j **do**
    $V^\beta(\pi^{\theta_i})$, $G_1^\beta(\pi^{\theta_i})$ = PREDICT($\mathcal{D}$, θ_i, Π_Θ, Φ, Ψ)
  end for
  H_j = POLICY_UPDATE($\{(V^\beta(\pi^{\theta_i}), G_1^\beta(\pi^{\theta_i})) : 1 \leq i \leq n_j\}$, ρ_j, α_1)
  $v_{j+1} = s_p \sum\limits_{k \in H_j} \frac{V^\beta(\pi^{\theta_k})\theta_k}{\sum\limits_{k \in H_j} V^\beta(\pi^{\theta_k})} + (1 - s_p)v_j$
end for

Algorithm 2 POLICY_UPDATE($\{(V^\beta(\pi^{\theta_i}), G^\beta(\pi^{\theta_i})) : 1 \leq i \leq n_j\}$, ρ_j, α_1)

Sort $\{(V^\beta(\pi^{\theta_i}), G^\beta(\pi^{\theta_i}))\}$ in ascending order of G^β. Let sorted order be denoted Υ.
Let m_j be the number of (V^β, G^β) tuples such that $G^\beta < \alpha_1$. Denote $\Lambda = \{\theta_i : G^\beta(\pi^{\theta_i}) < \alpha_1\}$.
B = Null Set
if $m_j < \lfloor \rho_j n_j \rfloor$ **then**
  Sort $\{\theta_i : G^\beta(\pi^{\theta_i}) \geq \alpha_1\} \subset \Upsilon$ in descending order of V^β. Denote this ordered set as Ξ.
  Pick the first $(\lfloor \rho_j n_j \rfloor - m_j)$ samples from Ξ.
  $B = \Lambda \cup \Xi$
else
  Sort Λ in descending order of V^β. Pick the first $(\lfloor \rho_j n_j \rfloor)$ elements of Λ and Ξ be the set of θ corresponding to this selection.
  $B = \Xi$
end if
return B

3.3.4. Behaviour Policy and Features

Clearly, by re-using the limited dataset $\mathcal{D}$, the requirement of a large number of on-policy experience tuples is eliminated. However, for reducing the error in off-policy prediction, RL agent needs samples from a policy which covers all regions of the state-space. Thus, the choice of the behaviour policy μ is very important. As can be observed in the experiments (Section 4), the behaviour policy is selected in a problem dependent manner in addition to facilitating exploration of all actions. Algorithm 1 gives the pseudocode of the proposed method. It does not require parallel simulations of the system, neither a large

number of on-policy experience samples. In the next section, we show numerical results for Algorithm 1 on benchmark problems where constraints and safety specifications arise.

4. Experimental Results

The performance of our algorithm is evaluated numerically on different benchmark problems as listed below. We also compare the results of our algorithm with the constrained actor critic algorithm [26] described in Section 3.

4.1. Benchmarks

The following benchmark applications are carefully designed to highlight the crucial safety aspect in autonomous systems[1]. All of these are inspired from current complex systems like for e.g., mobile robots, self-driving cars and exploration robots, although simplified for ease of exposition.

4.1.1. Random MDP

This basic benchmark is designed to investigate the learning behaviour of our proposed algorithm and to assess whether it is capable of learning safe optimal policies. The system model (P, R and the safety constraint c_1) is generated using Python MDPtoolbox. We consider a single safety constraint, which simplifies our analysis of Algorithm 1. The number of states (i.e., $|S|$) is fixed at 200, while the number of actions for each state is 5. The goal of the agent is to compute a policy π that maximizes V^π, but also satisfies $G_1^\beta(\pi) < \alpha_1$.

4.1.2. Chain Walk

The chain walk MDP has 450 states, i.e. $|S| = 450$, and each state has two actions. The action indicates the movement direction, i.e., the agent can either move left or move right at every state. Thus, $|A| = 2$ and $A = \{L, R\}$. The transition probability information is as shown in Table 1, where s is the current state and a denotes the action taken at state s. The transitions are grouped based on the probability of moving to the next state, these being either 0.1 or 0.9. The next state varies based on the current state s and the action taken. For e.g., when in state $s = 1$ if left (L) action is taken, the MDP evolves to state 2 with probability 0.1 and remains in the same state with probability 0.9. Other rows in Table 1 are to be interpreted in a similar manner. See Fig. 1 for a graphical illustration.

The reward function is as follows: $R(\cdot, \cdot, 150) = -10$, $R(\cdot, \cdot, 300) = 100$ and -1 for all other transitions. Hence, only transitions to state 300 will yield a positive payoff, while every other transition yields a negative reward. The single safety constraint values are randomly assigned to the states. The agent

[1]Paper code is available at `https://bit.ly/3ipj4Yi`

State and Action	Probability = 0.1	Probability = 0.9
$1 < s < \|S\|$, $a = L$	$s + 1$	$s - 1$
$1 < s < \|S\|$, $a = R$	$s - 1$	$s + 1$
$s = 1$, $a = L$	2	1
$s = 1$, $a = R$	1	2
$s = \|S\|$, $a = L$	$\|S\|$	$\|S\| - 1$
$s = \|S\|$, $a = R$	$\|S\| - 1$	$\|S\|$

Table 1: Transition probability information for Chain Walk MDP.

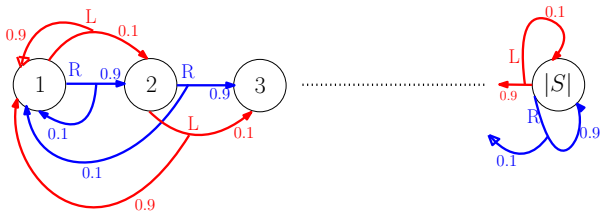


Figure 1: Chain walk MDP: Shows only the transition probabilities for each state and for both actions $\{L, R\}$.

gets the constraint value on each step and the task is to find a policy such that $G_1^{\beta}(\pi)$ as defined in Section 3 is maintained below a threshold value α_1. This benchmark is a simplified version of the problem faced by a mobile robot which needs to clean a hallway. The safety constraint represents the physical damage to the robot due to collision with obstacles in the hallway. The hallway area cleaned by the robot is indicated by the reward per step. A good feasible policy helps the robot clean maximum portion of the hallway by avoiding obstacles. In this benchmark we have avoided incorporating the mechanical dynamics of the robot since this allows us to focus on the learning algorithm.

4.1.3. Threshold Type MDP

The threshold type MDP has 400 states, i.e., $|S| = 400$ and each state has 4 actions ($|A| = 4$). The transition probability structure is as given in Table 2. The reward and constraint functions follow a threshold structure as also indicated in the same table. The threshold structure allows for clear policy behaviour as these policies allow the agent to operate in a subset of the states depending on what is being optimized.

This benchmark is a simplified scenario where exploration robots move in a small geographical area. To limit the movement of these robots, safety constraints are added so that they explore in a safe area.

4.1.4. Straight Path Navigation

This benchmark is a close approximation to real-world vehicle maneouvre problems, often encountered in autonomous driving car systems. A race car

State and Action	Next State	Reward	Constraint
$0 \leq s < 100,\ a \in A$	$100a + s$	10	5
$100 \leq s < 200,\ a \in A$	$100a + (s - 100)$	3	-1
$200 \leq s < 300,\ a \in A$	$100a + (s - 200)$	0	-4
$300 \leq s < 400,\ a \in A$	$100a + (s - 300)$	-3	-8

Table 2: Transition probability, reward and constraint value information for Threshold Type MDP. Here $A = \{0, 1, 2, 3\}$.

Figure 2: Straight Path Navigation: Illustrates a vehicle with four wheels with a direction vector (in blue) and example of a path traced (in green) till the current position.

needs to be controlled by the agent in a manner that it moves on a straight path with minimum deviations on a slippery road. The road conditions bring in an added aspect of environment response to the agent's actions. Slippery road conditions cause varying levels of tyre friction forces and the agent needs to adjust the velocity and steering direction accordingly. Further, safety restrictions arising out of road slippery conditions require the car to be navigating close to the $y = 0$ line.

The state s consists of the position coordinates of the vehicle, denoted as (x, y), inertial heading (yaw) of the centre of mass ψ and the corresponding velocities of these components. Hence, $s = (x, y, \psi, \dot{x}, \dot{y}, \dot{\psi})$. The control inputs available to the agent are velocity v and steering angle δ. Thus, $a = (v, \delta)$. The dynamics of the vehicle and the tyre models are adapted from [32]. The high level path planner requires the car to move on the line $y = 0$ and any other line can be suitably adjusted with translation. In the experiments we require the vehicle to follow this straight line at a target velocity v_f. Based on this requirement, the reward function is as follows: $R(s, a) = -|y| - \lambda_1(v - v_f)^2$. The first term is a distance penalty, and the second term is a velocity penalty. The constraint is to not drift too far from the actual pre-planned straight path. This drifting behaviour is gauged from the y coordinate as well as the heading angle ψ of the vehicle. We have $H(s, a) = \delta(\text{sgn}(y) + \text{sgn}(\psi))$ and $c_1(s, a) = H(s, a)$ if $H(s, a) > 0$ and is 0 otherwise. Hence, a positive value of $H(s, a)$ indicates that the vehicle is drifting away from $y = 0$ and this behaviour is captured in c_1.

4.2. Experimental Setup

In vehicle navigation benchmark, we discretize the action space with a resolution of $v = 1$ and $\delta = \frac{\pi}{6}$. The maximum velocity is 10 and the steering angle is varied from $\frac{-5\pi}{6}$ to π. This results in the vehicle moving in both positive and negative X directions. With this discretization, the action space is

large. For all benchmarks the randomized policy space Π_Θ consists of softmax policies with a temperature parameter. In the vehicle navigation case, a deep neural network (DNN) architecture represents the policy network. With the other benchmark problems, Chebyshev polynomials are used for state-action features, as described in [33]. For value and constraint prediction, the features range from simple radial basis features to polynomial basis functions.

4.3. Analysis of Results

Off-policy constrained cross entropy method is tested on the Chain Walk MDP and the value function learnt by our method is as shown in Fig. 3. This plot shows the normalized value function when safety constraints also need to be satisfied. The normalizing constant is the number of states. The threshold was pre-fixed at 8 for constraint value function. It can be seen in Fig. 3 that even though initial policy is not safe, the agent quickly learns and moves towards safe policies. During later updates of cross entropy, the agent finds safe policies with higher objective value function.

The objective and constraint functions for the Threshold Type MDP is shown in Fig. 4. The threshold value for the constraint function was fixed at 0 and the off-policy algorithm to evaluate the policies in every update of cross entropy algorithm is Emphatic TD [30] (labelled as ETD-OffPol-CCE). This figure plots the norm of the objective and constraint values as the agent training proceeds. The variance of the norm of these vectors is also observed which depends on the policy samples at every update. It can be observed that the initial policy is safe with low value of the constraint, but its value gradually improves which is required for improving the objective value. The policy learnt after 50 CE updates is safe and also yields a high objective value.

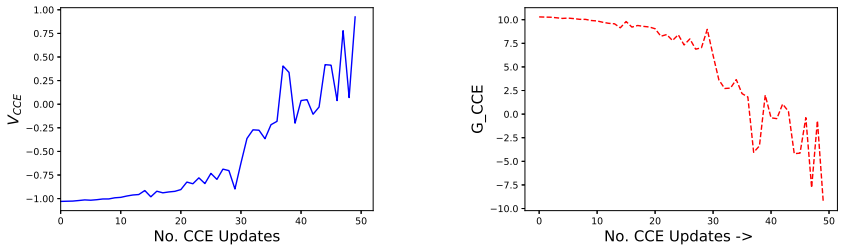


Figure 3: Chain Walk: Objective (labelled V_{CCE}) and Constraint value function (labelled G_{CCE}) during training of agent using off-policy constrained cross entropy method.

Fig. 5 plots the proportion of visits to all states by the policies learnt by our off-policy constrained cross entropy method in comparison to Q-learning (QL) [6]. The number of learning iterations of QL matched with the total number of samples used in the training of off-policy CCE method. QL learns a policy that maximizes the discounted return ignoring the constraint values. Thus, based on the problem structure of Threshold Type MDP, it is easy to observe that the QL policy must facilitate the agent to operate in the states

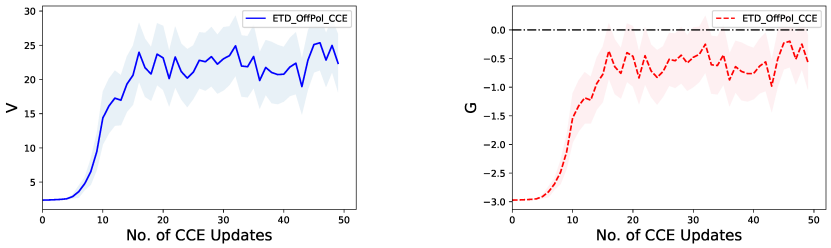


Figure 4: The objective function and the safety constraint of the policy learnt by the off-policy constrained cross entropy method for the Threshold Type MDP.

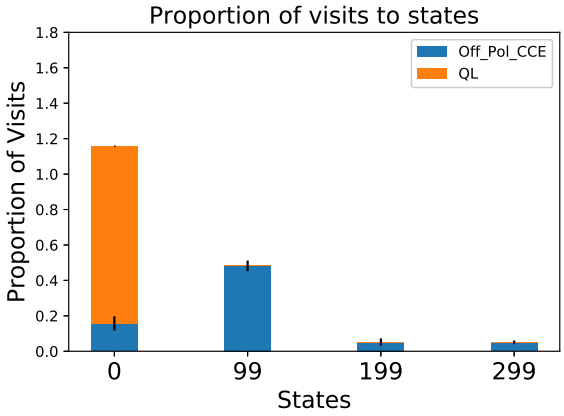


Figure 5: QL vs. Off policy Constrained cross entropy method: Proportion of visits to states. Computed by evaluating the learnt policy for 4000 steps.

0-99 for the maximum time, since these states yield high rewards. However, with off-policy constrained cross entropy method, it is observed that the agent learns to operate in states 100-199 for the maximum time, as, in these states the constraint values are low and rewards are also better when compared to states 200-400. Both the policies made the agent to operate in only 4 states - 0, 99, 199 and 299 and hence we plot the proportion of visits to these states only. The proportion of visits to all other states was close to zero under both policies.

In Algorithm 1, the PREDICT subroutine performs off-policy evaluation. In our experiments, we compared two different off-policy evaluation algorithms on the Random MDP benchmark problem - they are *Emphatic TD* [30] (ETD) and *Perturbed TD* [31] (PTD) algorithms. Both are importance-sampling based prediction algorithms. In Fig. 6, the objective function and constraint values for the constrained cross entropy method with ETD and PTD off-policy evaluation algorithms is shown. The respective plots are labelled as ETD-OffPol-CCE and PTD-OffPol-CCE. The behaviour policy sample paths were obtained with initial state being 0 always. Hence, here we plot the objective and constraint functions with initial state 0. With ETD, the policy improvement algorithm

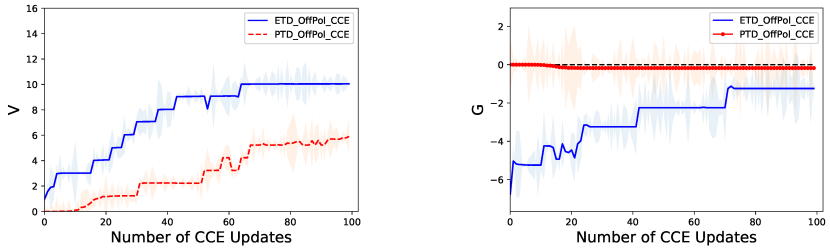


Figure 6: The objective function and the constraint of the policy learnt by the off-policy constrained cross entropy method for the Random MDP.

clearly shows a steady rise when the initial policy is feasible. In contrast, there is no significant change in the constraint value evaluated by PTD. The number of cross entropy iterations is 100. The threshold for the constraint value is 0. The same comparison for the Threshold Type MDP is shown in Fig. 7.

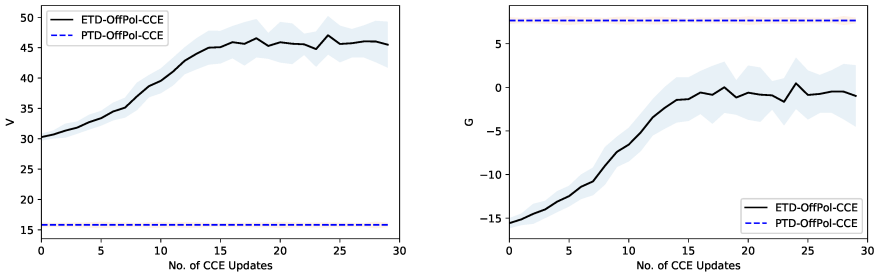
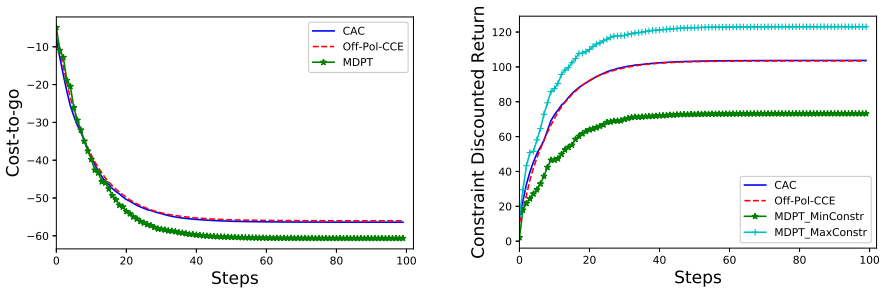


Figure 7: The objective function and the constraint of the policy learnt by the off-policy constrained cross entropy method for the Threshold Type MDP.

As described in Section 3, the other major line of works in safety-constrained RL build on Lagrangian formulation. Actor-critic type of algorithms [26, 27] have been proposed previously for constrained optimization in RL. We compare our algorithm with the constrained actor-critic algorithm proposed in [26]. We understand that the multi-agent constrained optimization algorithm in [27] is an extension of [26] and hence we compare our algorithm performance with the base work, i.e., [26]. Fig. 8 presents comparison of policies learned by constrained actor critic (abbv. CAC) and our safe RL algorithm. (abbv. Off-Pol-CCE) on Random MDP benchmark. These plots show the performance after the policy is learnt, in contrast to the plots shown earlier which are performance during learning. CAC does not allow for learning evaluation as the policy weights and prediction weights change every decision step, while in our algorithm, these weights remain the same for T number of steps, which is the length of the dataset $\mathcal{D}$. Moreover, none of the weights being tuned in CAC correspond to the actual CMDP - they are weights of a "relaxed" MDP. Thus, *CAC does not track safety during learning.*

The Random MDP has a cost objective instead of a reward objective. Thus,



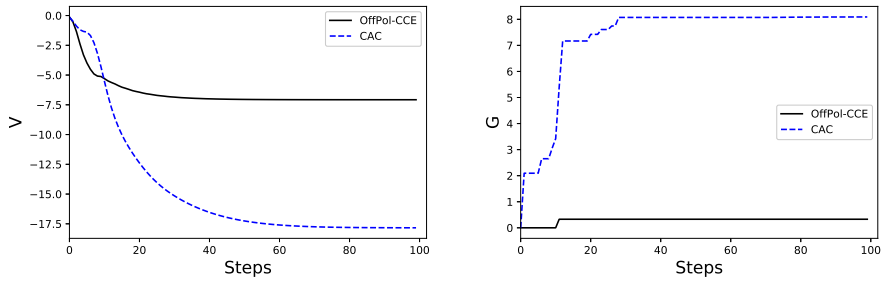
(a) Cost-to-go

(b) Constrainted long-term return

Figure 8: Comparison of performance of Constrained Actor Critic (CAC) and Algorithm 1 on Random MDP benchmark.

the objective function needs to be minimized, as opposed to maximization method described in Section 3. However, the same sorting logic in Algorithm 2 holds good in this case. Fig. 8(a) shows the cost-to-go (i.e. V^β) of policies learned by CAC, Off-Pol-CCE and value iteration (labelled MDPT). Value iteration has access to all model information (see Section 2) unlike CAC and Off-Pol-CCE. The cost-to-go is averaged over 50 sample paths for 100 decision steps with the initial state distribution β. As we can observe, the policies learnt by CAC and Off-Pol-CCE are comparable in performance - even though Off-Pol-CCE utilizes much less information compared to CAC. The value iteration learns a policy which finds the optimal cost-to-go for given transition probability and cost functions. Fig. 8(b) shows the constrainted return of policies learnt by CAC, Off-Pol-CCE and value iteration. The threshold is fixed at 100, i.e., $\alpha_1 = 100$ and the initial state distribution is concentrated on state 0. In this plot, we show two policies learnt in the full information case - one that minimizes the constrainted return (i.e., G^β) and the other which maximizes G^β. These are labelled as MDPT_MinConstr and MDPT_MaxConstr respectively in Fig. 8(b). We can see that these two policies are at the opposite ends of the spectrum and hence shows us the spread of the various G^β values that can be attained by policies for the CMDP model concerned. Here too we observe that the performance of CAC and Off-Pol-CCE are comparable and the G^β value of this policy is midway between the performance of the policies learnt in the full information case.

For the straight-line following vehicle navigation benchmark, we compare Constrained Actor Critic (CAC) and Algorithm 1. Fig. 9 plots V^β and G^β with β concentrated on one initial state of the vehicle. The vehicle specifications that we model are those of 'RC Car', as described in [32]. The policies learnt by both algorithms are evaluated for 400 steps, however only the plot for the first 100 steps is shown in Fig. 9. The threshold α_1 is fixed at 1 during learning. The results are averaged for 10 Monte Carlo runs. The number of learning iterations for both algorithms is 1×10^6 and the discount factor used is 0.9. The structure of R for this benchmark is such that it penalizes the drift from



(a) Cost-to-go

(b) Constraint long-term return

Figure 9: Comparison of performance of Constrained Actor Critic (CAC) and Algorithm 1 on vehicle straight-line following benchmark.

$y = 0$ line, while g_1 penalizes whenever the steering angle and y-coordinate distance from $y = 0$ are not favourable. It is observed that for the same number of learning iterations, Algorithm 1 learn better safe policies when compared to CAC. Such policies attain high V^β and low G^β. The above experiments suugest that our proposed method Off-Policy based Safe RL, adapted from cross-entropy algorithm is well suited for safety-constrained systems. Also by using neural network architectures as described here, the algorithm can be scaled to large and complex autonomous systems.

5. Conclusions and Future Work

In this work we proposed a safe RL algorithm utilizing cross entropy method. Our algorithm handles constraints effectively and also enables off policy learning. Unlike the previous safe RL algorithms, our method is sample efficient owing to the off-policy value prediction. Further, the experimental results demonstrate the practical applicability of the proposed algorithm.

An interesting research direction would be to utilize off policy prediction algorithm in [34] instead of importance sampling based off-policy prediction to reduce the variance. Further, it would be interesting to utilize model reference adaptive search (MRAS) method [35] for safe policy optimization as MRAS exhibits global convergence property.

References

[1] K. J. Prabuchandran, H. Kumar, and S. Bhatnagar, "Multi-agent Reinforcement Learning for Traffic Signal Control," in *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, 2014.

[2] X. Gu and A. Easwaran, "Towards Safe Machine Learning for CPS: Infer Uncertainty from Training Data," in *Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems*, ser. ICCPS '19. Association for Computing Machinery, 2019, p. 249–258.

[3] L. Waymo, "On the road to fully self-driving," *Waymo Safety Report*, pp. 1–43, 2017.

[4] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, 2nd ed. New York, NY, USA: John Wiley & Sons, Inc., 2005.

[5] D. Bertsekas, *Dynamic Programming and Optimal Control*, 4th ed. Belmont,MA: Athena Scientific, 2013, vol. II.

[6] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[7] K. Malialis, S. Devlin, and D. Kudenko, "Resource abstraction for reinforcement learning in multiagent congestion problems," in *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, ser. AAMAS '16, 2016, p. 503–511.

[8] S. Padakandla, K. J. Prabuchandran, and S. Bhatnagar, "Energy Sharing for Multiple Sensor Nodes with Finite Buffers," *IEEE Transactions on Communications*, vol. 63, no. 5, pp. 1811–1823, 2015.

[9] F. Niroui, K. Zhang, Z. Kashino, and G. Nejat, "Deep reinforcement learning robot for search and rescue applications: Exploration in unknown cluttered environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 610–617, April 2019.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[11] P. Kohli and A. Chadha, "Enabling Pedestrian Safety Using Computer Vision Techniques: A Case Study of the 2018 Uber Inc. Self-driving Car Crash," in *Advances in Information and Communication*, K. Arai and R. Bhatia, Eds. Cham: Springer International Publishing, 2020, pp. 261–279.

[12] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete Problems in AI Safety," *arXiv*, 2016.

[13] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[14] E. Altman, "Constrained markov decision processes with total cost criteria: Lagrangian approach and dual linear program," *Mathematical Methods of Operations Research*, vol. 48, no. 3, pp. 387–417, 1998.

[15] Y. Chow and M. Ghavamzadeh, "Algorithms for CVaR Optimization in MDPs," in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'14. MIT Press, 2014, p. 3509–3517.

[16] V. Borkar and R. Jain, "Risk-Constrained Markov Decision Processes," *IEEE Transactions on Automatic Control*, vol. 59, no. 9, pp. 2574–2579, Sep. 2014.

[17] C. E. Rasmussen, "Gaussian Processes in Machine Learning," *Lecture Notes in Computer Science*, p. 63–71, 2004.

[18] J. García and F. Fernández, "A Comprehensive Survey on Safe Reinforcement Learning," *Journal of Machine Learning Research*, vol. 16, no. 42, pp. 1437–1480, 2015.

[19] J. Achiam, D. Held, A. Tamar, and P. Abbeel, "Constrained Policy Optimization," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17, 2017, p. 22–31.

[20] M. Wen and U. Topcu, "Constrained Cross-Entropy Method for Safe Reinforcement Learning," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., 2018, pp. 7450–7460.

[21] S. Mannor, R. Rubinstein, and Y. Gat, "The Cross Entropy Method for Fast Policy Search," in *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ser. ICML'03. AAAI Press, 2003, p. 512–519.

[22] M. Turchetta, F. Berkenkamp, and A. Krause, "Safe Exploration in Finite Markov Decision Processes with Gaussian Processes," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, ser. NIPS'16, Red Hook, NY, USA, 2016, p. 4312–4320.

[23] A. Wachi, Y. Sui, Y. Yue, and M. Ono, "Safe Exploration and Optimization of Constrained MDPs Using Gaussian Processes," in *AAAI Conference on Artificial Intelligence*, 2018.

[24] A. Wachi and Y. Sui, "Safe Reinforcement Learning in Constrained Markov Decision Processes," in *Proceedings of Machine Learning and Systems 2020*, 2020, pp. 1493–1502.

[25] J. Nocedal and S. J. Wright, *Numerical Optimization*, ser. Springer Series in Operations Research and Financial Engineering. Springer New York, 2006.

[26] S. Bhatnagar, "An actor–critic algorithm with function approximation for discounted cost constrained Markov decision processes," *Systems & Control Letters*, vol. 59, no. 12, pp. 760 – 766, 2010.

[27] R. B. Diddigi, D. S. K. Reddy, P. K.J., and S. Bhatnagar, "Actor-Critic Algorithms for Constrained Multi-Agent Reinforcement Learning," in *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, ser. AAMAS '19. International Foundation for Autonomous Agents and Multiagent Systems, 2019, p. 1931–1933.

[28] A. G. Joseph and S. Bhatnagar, "An incremental off-policy search in a model-free Markov decision process using a single sample path," *Machine Learning*, vol. 107, no. 6, p. 969–1011, Feb 2018.

[29] S. Ghiassian, A. Patterson, M. White, R. S. Sutton, and A. White, "Online Off-policy Prediction," *arXiv*, 2018.

[30] R. S. Sutton, A. R. Mahmood, and M. White, "An Emphatic Approach to the Problem of Off-Policy Temporal-Difference Learning," *Journal of Machine Learning Research*, vol. 17, no. 1, p. 2603–2631, Jan 2016.

[31] R. B. Diddigi, C. Kamanchi, and S. Bhatnagar, "A convergent off-policy temporal difference algorithm," pp. 1103–1110, 2020.

[32] E. Ahn, "Towards safe reinforcement learning in the real world," Master's thesis, Pittsburgh, PA, July 2019.

[33] I. Palunko, A. Faust, P. Cruz, L. Tapia, and R. Fierro, "A reinforcement learning approach towards autonomous suspended load manipulation using aerial robots," in *2013 IEEE International Conference on Robotics and Automation*, 2013, pp. 4896–4901.

[34] Q. Liu, L. Li, Z. Tang, and D. Zhou, "Breaking the curse of horizon: Infinite-horizon off-policy estimation," *arXiv preprint arXiv:1810.12429*, 2018.

[35] J. Hu, M. C. Fu, and S. I. Marcus, "A model reference adaptive search method for global optimization," *Operations Research*, vol. 55, no. 3, pp. 549–568, 2007.