RECONNAISSANCE FOR REINFORCEMENT LEARNING WITH SAFETY CONSTRAINTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Practical reinforcement learning problems are often formulated as constrained Markov decision process (CMDP) problems, in which the agent has to maximize the expected return while satisfying a set of prescribed safety constraints. In this study, we consider a situation in which the agent has access to the generative model which provides us with a next state sample for any given state-action pair, and propose a model to solve a CMDP problem by decomposing the CMDP into a pair of MDPs; *reconnaissance* MDP (R-MDP) and *planning* MDP (P-MDP). In R-MDP, we train *threat function*, the Q-function analogue of danger that can determine whether a given state-action pair is safe or not. In P-MDP, we train a reward-seeking policy while using a fixed threat function to determine the safeness of each action. With the help of generative model, we can efficiently train the threat function by preferentially sampling rare dangerous events. Once the threat function for a baseline policy is computed, we can solve other CMDP problems with different reward and different danger-constraint without the need to re-train the model. We also present an efficient approximation method for the threat function that can greatly reduce the difficulty of solving R-MDP. We will demonstrate the efficacy of our method over classical approaches in benchmark dataset and complex collision-free navigation tasks.

1 INTRODUCTION

With recent advances in reinforcement learning (RL), it is becoming possible to learn complex reward-maximizing policy in an increasingly more complex environment (Mnih et al., 2015; Silver et al., 2016; Andrychowicz et al., 2018; James et al., 2018; Kalashnikov et al., 2018). However, it is difficult in general to assess whether the policies found by a given RL algorithm is physically safe when applied to real world situations. This has long been one of the greatest challenges in the application of reinforcement learning to mission-critical systems. In a popular setup, one assumes a Markovian system together with a predefined measure of danger, and formulates the problem as a type of constrained Markov decision process (CMDP) problem. That is, based on the classical RL notations in which π represents a policy of the agent, we aim to solve

$$\max_{\pi} E^{\pi}[R(h)] \quad \text{s.t.} \quad E^{\pi}[D(h)] \leq c, \tag{1}$$

where h is a trajectory of state-action pairs, $R(h)$ is the total return that can be obtained by h, and $D(h)$ is the measure of how dangerous the trajectory h is. To solve this problem, one must monitor the value of $E^{\pi}[D(h)]$ throughout the training. Methods like (Altman, 1999; Geibel & Wysotzki, 2005; Geibel, 2006; Achiam et al., 2017a; Chow et al., 2018; 2019) uses sampling to approximate $E^{\pi}[D(h)]$ or its Lyapunov function at every update. However, the sample-based evaluation of the $E^{\pi}[D(h)]$ is particularly difficult when the system involves "rare" catastrophic accidents, because an immense number of samples will be required to collect information about the cause of such accident.

This problem can be partially resolved if we can use a generative model to predict the outcome of any given sequence of actions and initial state. Model Predictive Control (MPC) (Maciejowski, 2002; Falcone et al., 2007; Wang & Boyd, 2010; Di Cairano et al., 2013; Weiskircher et al., 2017) uses the philosophy of receding horizon and predicts the future outcome of actions in order to determine what action the agent should take in the next step. If the future-horizon to consider is sufficiently short and the dynamics is deterministic, the prediction can often be approximated well by linear dynamics,

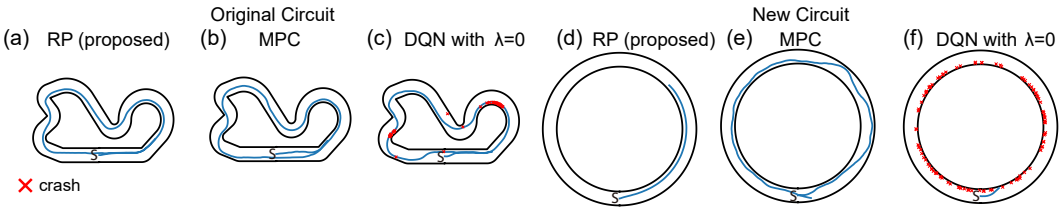


Figure 1: The trajectories produced by the the policy trained by our proposed method ((a) and (d)), 4-step MPC ((b), (e)), and the policy trained with penalized DQN ((c) and (f)). The trajectories on circular circuit were produced by the policies trained on the original circuit. S represents the initial position of the agent. The red marks represents the places at which the agent crashed into the wall.

which can be evaluated instantly. However, because MPC must finish its assessment of the future before taking every action, its performance is limited by the speed of the predictions. When we apply MPC to environments with multiple agents and stochastic dynamics, the computational load of prediction is especially heavy and it can be difficult to finish the prediction in time. MPC requires this computation for each time-step even if the current state is similar to the ones experienced in the past. Meanwhile, if the prediction is done for only a short horizon, MPC may suggest a move to a state leading to a catastrophe.

In an effort to reduce the difficulty in evaluating the safeness of policies, we propose a novel generative model-based approach that looks for a solution of a CMDP problem by decomposing the CMDP into a pair of MDPs: a reconnaissance MDP (R-MDP) and planning MDP (P-MDP). The purpose of R-MDP is to (1) *recon* the state space using the generative model and (2) train a baseline policy for the *threat function* , which is a Q-function analogue of D.

In R-MDP, we use generative model to selectively sample trajectories containing rare dangerous events, and learn the threat function for the baseline policy in the way of supervised learning. Once we obtain a good approximation of the threat function for the baseline policy, we can determine whether a given action is safe at each state or not by just evaluating the threat function. This process does not involve prediction, which can be computationally demanding. We will theoretically show that **we can increase the set of safe actions by improving the safeness of the baseline policy**. In P-MDP, we train the reward-seeking policy while using the threat function to make sure that unsafe actions are not chosen. We may say that P-MDP is a version of original MDP in which the agents are only allowed to select an action from the set of safe policies defined by the threat function. P-MDP can be solved with standard RL methods like DQN (Mnih et al., 2015). With our framework, the user is freed from the need of monitoring $E^{\pi}[D]$ throughout the whole training process.

We will also show that our approach enjoys the following useful properties: 1) If we can find a safe baseline policy from the R-MDP problem, the learning of P-MDP will always be safe. 2) So long that we define the danger with the same D function, we can re-use the threat function constructed for one CMDP problem to solve another CMDP problem with a different reward function and different constraint threshold. 3) When dealing with a problem with multiple sources of danger, we can use a basic rule of probability to upper-bound the threat functions by a sum of sub-threat functions, with each summand corresponding to different source of danger each. The property (2) allows us to train an agent that can safely navigate a circuit irrespective of the course-layout1(d). In this experiment, we represented the circuit's wall as a set of point obstacles, and computed the threat functions for the collision with each obstacle point. The property (3) allows us to find a good reward-seeking policy for a sophisticated task like *safely navigating through a crowd of randomly moving obstacles*. Although our method does not guarantee to find the optimal solution of the CMDP problem, there has not been any study to date that has succeeded in solving a CMDP in dynamical environments as high-dimensional as the ones discussed in this study.

2 Problem Formulation and Theoretical Results

We assume that the system in consideration is a discrete-time constrained Markov Decision Process (CMDP) with finite horizon, defined by a tuple (S, A, r, d, P, P_0), where S is the set of states, A is the set of actions, $P(s'|s, a)$ is the density of the state transition probability from s to s' when the

action is a, $r(s,a)$ is the reward obtained by action a at state s, $d(s,a)$ is the non-negative danger of taking action a at state s, and P_0 is the distribution of the initial state. We use $\pi(a|s)$ to denote the policy π's probability of taking an action a at a state s. Finally, let us use 1_B to represent the indicator function of an event B. Now we formally present the optimization problem (1).

$$\arg\max_{\pi} E_\pi \left[\sum_{t=0}^{T-1} \gamma^t r(s_t,a_t)\right], \text{ s.t.} E_\pi \left[\sum_{t=0}^{T-1} \beta^t d(s_t,a_t)\right] \leq c, \tag{2}$$

where $c \geq 0$ specifies the safety level, $\gamma, \beta \in [0,1)$ are the discount factors, and $E_\pi[\cdot]$ denotes the expectation with respect to π, P and P_0. We use plain E to denote the expectation with respect to P_0.

2.1 Properties of threat functions and secure policies

In our formulation, we use the following *threat function* and *danger function* as a danger-analogue of the action-value function and state-value function, respectively.

$$\mathscr{T}_t^\eta(s_t,a_t) = E_\eta \left[\sum_{k=t}^{T-1} \beta^{k-t} d(s_k,a_k) \mid s_t, a_t\right], \ \mathscr{D}_t^\eta(s_t) = E_\eta \left[\mathscr{T}_t^\eta(s_t,a_t)\right]. \tag{3}$$

We say that a policy η is *safe* if $E[\mathscr{D}_0^\eta(s_0)] \leq c$. Indeed, the set of safe policies is the set of feasible policies for the CMDP (1). Before we proceed further, we describe several key definitions and theorems that stem from the definition of the threat function. For now, let us consider a time-dependent safety threshold x_t defined at each time t, and let η be any policy. Let us also use $\boldsymbol{x}$ to denote $(x_0,\ldots,x_{T-1})$. Then the set of $(\eta,\boldsymbol{x})$-secure actions is the set of actions that are deemed *safe* by η for the safety threshold $\boldsymbol{x}$ in the sense of the following definition;

Definition 1 ($(\eta,\boldsymbol{x})$-secure actions and $(\eta,\boldsymbol{x})$-secure states)**.** Let $A^{\eta,\boldsymbol{x}}(s,t) = \left\{a; \mathscr{T}_t^\eta(s,a) \leq x_t\right\}$.

$$A^{\eta,\boldsymbol{x}}(s) = \bigcap_{t\in\{0,\cdots,T-1\}} A^{\eta,\boldsymbol{x}}(s,t), \ S^{\eta,\boldsymbol{x}} = \left\{s \in S; A^{\eta,\boldsymbol{x}}(s) \neq \emptyset\right\}. \tag{4}$$

This $(\eta,\boldsymbol{x})$-secure set of actions $A^{\eta,\boldsymbol{x}}(s)$ represents the agent's freedom in seeking the reward under the safety protocol created by the policy η. The set of secure actions for an arbitrary η could be empty for some states. $(\eta,\boldsymbol{x})$-secure states is defined as a set of states for which there is non-empty $(\eta,\boldsymbol{x})$-secure actions. If we use $\mathrm{supp}(p)$ to denote the support of a distribution p, we can use this definition to define a set of policies that is at least as safe as η. First, let us define the set of distributions,

$$\mathcal{F}^\eta(s) = \left\{p(\cdot); \int_a p(a)\mathscr{T}_t^\eta(s,a)da \leq E_\eta[\mathscr{T}_t^\eta(s,a)] \ \forall t\right\}.$$

Then the following set of policies are at least as safe as η.

Definition 2 (General $(\eta,\boldsymbol{x})$-secure policies)**.**

$$\Pi_G^{\eta,\boldsymbol{x}} = \{\pi; \text{ for } s \in S^{\eta,\boldsymbol{x}}, \ \mathrm{supp}(\pi(\cdot|s)) \subseteq A^{\eta,\boldsymbol{x}}(s), \text{ otherwise, } \pi(\cdot|s) \in \mathcal{F}^\eta(s)\}. \tag{5}$$

Now, we are ready to develop our theory for determining when a given policy is safe. The following theorem enables us to bound $\mathscr{D}_t^\pi(s_t)$ without evaluating the expectation with respect to π.

Theorem 1. For a given policy η and a sequence of safety thresholds $\boldsymbol{x} = (x_0,\ldots,x_{T-1})$, let π be a policy in $\Pi_G^{\eta,\boldsymbol{x}}$. Let us use $d_{TV}(p,q)$ to denote the total variation distance[1] between two distributions p and q. Then for all $t \in \{0,\ldots,T-1\}$

$$\mathscr{D}_t^\pi(s_t) \leq \mathscr{D}_t^\eta(s_t) + 2\sum_{k=t}^{T-1} \beta^{k-t} x_k E_\pi \left[z_k \mid s_t\right]. \tag{6}$$

where $z_t = 1_{s_t \in S^{\eta,\boldsymbol{x}}} d_{TV}(\pi(\cdot|s_t), \eta(\cdot|s_t))$ is a distance measure of the two polices.

The proof of this result uses practically same logic as the one used for Theorem 1 in (Achiam et al., 2017a). Please see the Appendix for the detail. In practical application, it is more convenient to set $x_t = x$ for all t. If we also bound z_t from above by 1, we obtain the following useful result.

[1]Total variation distance is defined as $d_{TV}(p(a),q(a)) = \frac{1}{2}\sum_a |p(a) - q(a)|$.

Corollary 2. If $E[\mathscr{D}_0^\eta(s_0)] \le c$, let $\boldsymbol{x} = (x_c^\eta, \ldots, x_c^\eta)$ with $x_c^\eta = \frac{1}{2}(c - E[\mathscr{D}_0^\eta(s_0)])\frac{1-\beta}{1-\beta^T}$. Then a policy π is safe if $\pi \in \Pi_G^{\eta,\boldsymbol{x}_c^\eta}$, (i.e., $E[\mathscr{D}_0^\pi(s_0)] \le c$).

The above corollary provides us a safety guarantee for π when η itself is safe. But in fact, if we look inside a smaller subset of $\Pi_G^{\eta,\boldsymbol{x}}$, we can guarantee the safety even when η itself may not be safe.

Definition 3 ($(\eta, \boldsymbol{x})$-secure policies)**.** Let greedy-$\eta(a|s) = 1_{a=\arg\min_{a'} \mathscr{T}_t^\eta(s,a')}$.

$$\Pi^{\eta,\boldsymbol{x}} = \left\{\pi; \text{ for } s \in S^{\eta,\boldsymbol{x}},\ \text{supp}(\pi(\cdot|s)) \subseteq A^{\eta,\boldsymbol{x}}(s),\ \text{ otherwise, } \pi(\cdot|s) = \text{greedy-}\eta(\cdot|s)\right\}. \quad (7)$$

The $(\eta, \boldsymbol{x})$-secure policies $\Pi^{\eta,\boldsymbol{x}}$ is indeed a subset of $\Pi_G^{\eta,\boldsymbol{x}}$ because greedy-η is just the one-step policy improvement from η. It turns out that we can construct a pool of absolutely safe policies *explicitly* from $\mathscr{T}_t^\eta(s,a)$ and c alone even when η itself is not necessarily safe.

Corollary 3. If $E[\mathscr{D}_0^{\text{greedy-}\eta}(s_0)] \le c$, by setting $\boldsymbol{x} = (x_{c,g}^\eta, \ldots, x_{c,g}^\eta)$ with $x_{c,g}^\eta = \frac{1}{2}(c - E[\mathscr{D}_0^{\text{greedy-}\eta}(s_0)])\frac{1-\beta}{1-\beta^T}$, any policy $\pi \in \Pi^{\eta,\boldsymbol{x}_{c,g}^\eta}$ is safe, i.e., $E[\mathscr{D}_0^\pi(s_0)] \le c$.

This result can be derived using a result similar to Eq.(6) (See Appendix for the detail). In the next section, we will use $\Pi^{\eta,\boldsymbol{x}_{c,g}^\eta}$ to construct a pool of safe policies from which to seek a good and safe reward-seeking policy. Now, several remarks are in order. First, if we set $\beta = 1$, then $x_c^\eta \to 0$ as $T \to \infty$. This is in agreement with the law of large numbers; that is, any accident with positive probability is bound to happen at some point. Also, note that we have $A^{\eta,\boldsymbol{x}}(s) \subseteq A^{\eta',\boldsymbol{x}}(s)$ whenever $\mathscr{T}_t^{\eta'}(s,a) \le \mathscr{T}_t^\eta(s,a)$ for any t. Thus, by finding the risk-minimizing η, we can maximize the pool of safe policies. Whenever we can, we shall therefore look not just for a safe η, but also for the threat minimizing policy. Finally and most importantly, note that the threshold expression in Corollary 3 is free of π. We can use this result to tackle the CMDP problem by solving two separate MDP problems. More particularly, in seeking a solution to the CMDP problem we can (i) first look for an η satisfying $E[\mathscr{D}_0^{\text{greedy-}\eta}(s_0)] \le c$, and then (ii) look for a safe reward maximizing policy π in Π^η. We will further articulate this procedure in the next section. Hereafter unless otherwise noted, we will use Π^η to denote $\Pi^{\eta,\boldsymbol{x}_{c,g}^\eta}$, and use S^η, $A^\eta(s)$ to denote $S^{\eta,\boldsymbol{x}_{c,g}^\eta}$, $A^{\eta,\boldsymbol{x}_{c,g}^\eta}(s)$.

3 Reconnaissance-MDP (R-MDP) and Planning-MDP (P-MDP)

In the previous section, we have shown that we can create a pool of safe policies using a baseline safe policy. Also, by training a policy to minimize the threat, we can find a safe policy that corresponds to a larger pool of secure policies, and we look within this pool to search for a possibly better reward seeking policy. This motivates us to solve two types of MDP problem: the one with the aim of minimizing the threat, and the one with the goal of maximizing the reward.

The purpose of the Reconnaissance MDP (R-MDP) is thus to *reconnoiter* the system prior to the reward maximization process and to look for the threat minimization safe policy η^* that solves

$$\eta^* = \arg\min_\eta E_\eta\left[\sum_{t=0}^{T-1} \beta^t d(s_t, a_t)\right]. \quad (8)$$

Indeed, solution η^* of argmin is not unique, up to freedom of the actions on the states unreachable by any optimal policy η^*. For our ensuing discussions, we will chose η^* to be a version whose policy on each unreachable state s^* is computed by (8) with initial state being s^*. If the problem is of infinite horizon, η^* computed from any initial state will be same because of the Markov property.

As a process, R-MDP is same as the original MDP except that we have a danger function in place of a reward function, and that the goal of the agent in the system is to minimize the danger. The following is true in general about R-MDP.

Corollary 4. If the set of feasible policies of the original CMDP is nonempty, the optimal R-MDP policy η^* is safe. Thus, every policy in Π^{η^*} is safe.

After we solve the R-MDP, the Planning MDP (P-MDP) searches within Π^{η^*} for a good reward-seeking policy π^*. The P-MDP is similar to the original MDP except that the agent is only allowed to

take actions from A^{η^*} when $s \in S^{\eta^*}$ and that it follows greedy-η^* at non-secure states $s \notin S^{\eta^*}$.

$$\pi^* = \arg\max_{\pi \in \Pi^{\eta^*}} E_\pi \left[\sum_{t=0}^{T-1} \gamma^t r(s_t, a_t) \right]. \tag{9}$$

In implementation, we do not explicitly construct Π^{η^*}. Instead, we evaluate $\mathscr{T}_t^{\eta^*}(s,a)$ for every considered state-action pair in P-MDP and make sure that all suggested reward-seeking actions are in A^{η^*}. Note that, if policy η^* is safe, every policy in Π^{η^*} is guaranteed to be safe. More particularly, in such case, any policy in the entire learning process of the P-MDP is safe. We would refer to the whole procedure of solving R-MDP and P-MDP as *Reconnaissance and Planning (RP) algorithm*. The following table summarizes the algorithm. Naturally, whether this algorthm works in application

Algorithm 1 RP algorithm

1: Obtain the baseline policy η^* by either solving R-MDP or selecting a heuristic policy
2: Estimate $\mathscr{T}_t^{\eta^*}(\cdot,\cdot)$
3: Solve the P-MDP (9) while referring the evaluation of $\mathscr{T}_t^{\eta^*}$ at every considered state-action pair so that all actions will be chosen from $A^{\eta^*,\boldsymbol{x}}$ (Eq. (7))

depends on how well we can evaluate the threat in R-MDP. However, in many applications, the set of dangerous events to be avoided can be extremely rare. These rare events make the empirical approximation $E_\pi[\cdot]$ difficult. We resolve this problem by evaluating $\mathscr{T}_t^{\eta}(s,a)$ with the generative model. With the generative model, we can freely explore the system from any arbitrary initial state, and evaluate $\mathscr{T}_t^{\eta}(s,a)$ for any (s,a) and η. To facilitate the learning of $\mathscr{T}_t^{\eta^*}(s,a)$, we use the generative model to preferentially sample the states with relatively high estimated $\mathscr{T}_t^{\eta^*}(s,a)$ more frequently. We will next introduce a technique to approximate the threat function.

3.1 Threat decomposition and approximate RP algorithm

We will present a useful bound on the threat function that can be used when the danger is be described in terms of multiple risky events. Suppose that there are N risky events $\{E_1, ..., E_N\}$ to consider, and that the goal is optimize the reward while avoiding *any* risky event. More formally, let us represent the indicator function of E_n by $d^n(s_t^{(o)}, s_t^{(n)}, a_t)$ where $s_t^{(n)}$ is the state of the system relevant to E_n and $s^{(o)}$ is the state of the system not relevant to any risky events. Let us also assume that the transition probabilities decompose into $p(s_{t+1}|s_t,a_t) = p(s_{t+1}^{(o)}|s_t^{(o)},a_t)\prod_{n=1}^N p(s_{t+1}^{(n)}|s_t^{(n)},s_t^{(o)},a_t)$. If d is the indicator function of $\cup_{n=1}^m E_n$, we can formalize this result as follows:

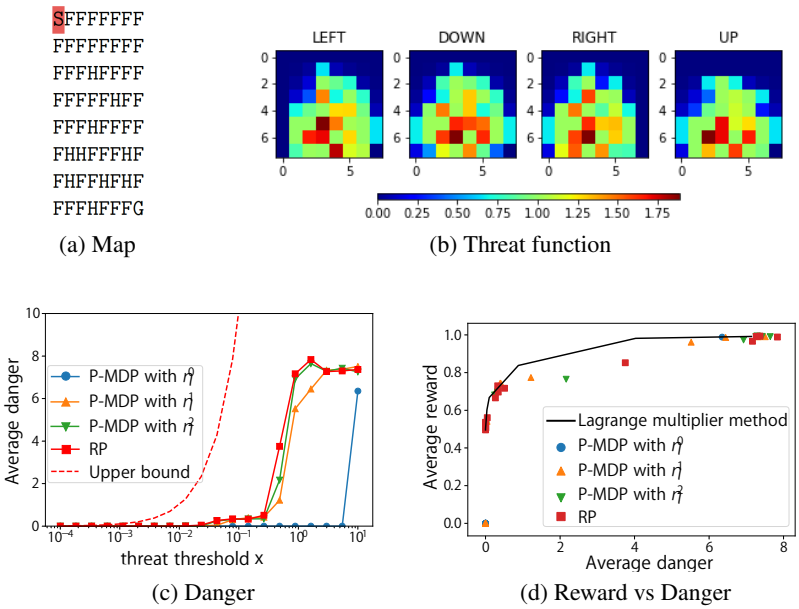
Theorem 5. Let η be a policy that takes action based solely on $s_t^{(o)}$, and let $\mathcal{T}^\eta$ be the threat function defined for the indicator function of $\cup_{n=1}^N E_n$. Then $\mathscr{T}_t^{\eta}(s_t,a_t) \leq \sum_{n=1}^N \mathscr{T}_t^{\eta,n}(s_t^{(o)}, s_t^{(n)}, a_t)$.

This result is especially useful in navigation-like tasks. For example, when E_n is the collision event with nth obstable, s_n will be the state of nth obstacle, and s_o will be the aggregate state of the system that are not related to any obstacles(state of the agent, etc). In such case, each $\mathscr{T}^{\eta,n}$ can be estimated using the simulator containing just the nth obstable object and the agent. Algorithm 2 is the algorithm that uses Theorem 5. For other variants of Theorem 5, please see the Appendix. Theorem 5 suggests

Algorithm 2 Approximate RP algorithm

1: Pick a heuristic policy $\eta^{(o)}$ which depends on $s^{(o)}$ only.
2: Estimate the threat functions $\mathscr{T}_t^{\eta^{(o)},n}(\cdot,\cdot)$ for all sub R-MDPs.
3: Solve the P-MDP (9) while referring to the evaluation of $\sum_n \mathscr{T}_t^{\eta^{(o),n}}$ at every considered state-action pair so that all chosen actions will satisfy $\sum_n \mathscr{T}_t^{\eta^{(o),n}} \leq x_t$.

that that threat function can be treated like risk potential. Wolf & Burdick (2008); Rasekhipour et al. (2016b); Ji et al. (2016). Risk-potential based methods also work by evaluating the risk of collision with each obstacle for each location in the environment, and by superimposing the results. However, most of these works define the risk potential rather heuristically.



(a) Map (b) Threat function

(c) Danger (d) Reward vs Danger

Figure 2: (a) Map of FrozenLake8x8-v0. Here the symbols 'S', 'G', 'H' and 'F' denote the position of start, goal, hole to be avoided, and frozen surface, respectively. (b) Heat map of the threat function of optimal R-MDP baseline policy. The warmer color represents the higher value of the threshold function at the corresponding position and action. The threshold tends to be higher around the hole. (c) Average danger suffered by PMDP policies vs safety threshold x. (d) Scatter plot of average rewards and average dangers obtained with different safety thresholds.

4 Experiment

We conducted a series of experiments to (i) analyze the nature of the threat function, (ii) the effect of the choice of the baseline policy η on the performance of P-MDP, and (iii) the sheer efficacy of the RP algorithm implemented with the threat decomposition approximation. We will also show that we can re-use the threat function computed for one CMDP to solve another CMDP problem with similar safety constraint. We will demonstrate the power of our method on high-dimensional problems.

4.1 Frozen Lake Environment

We considered an example of the frozen lake environment *FrozenLake8x8-v0* in OpenAI Gym where the agent navigates in the 8 by 8 grid world in Fig. 2 (a). The goal of the agent is to start from 'S' and navigate its way through the slippy environment to reach the goal 'G' while avoiding the holes 'H'. Each time on a frozen surface 'F', the agent can choose to move in one of the four directions, but the surface is slippery so the agent's actual movement may be different from the intended direction. We first construct the R-MDP as described in Sec 3. The R-MDP is a tabular MDP which can be solved by value iteration. The threat function of the optimal R-MDP policy is shown in Fig. 2 (b). The threat values are indeed higher when the agent is closer to the holes. From the threat function we can see that, there is a safe path from 'S' to 'G' following the right most column. But the tricky part is at positions (6, 8) and (7, 8). On these positions, the only action that can avoid the danger is 'RIGHT', which is not the best reward-seeking action ('DOWN').

The P-MDP is again a tabular MDP, and we can solve it by value iteration. In Fig. 2 (c) and (d), we compare the performance of P-MDP based on the optimal R-MDP policy η^* against those of the P-MDPs based on the sub-optimal R-MDP policies. Based on a completely random policy for η^0, we defined $\eta^1 =$ greedy-η^0 and $\eta^2 =$ greedy-η^1. In other words, both η_1, η_2 are the outcomes of applying policy iterations to the random policy. We also plot the reward and danger trade-off using the Lagrange multiplier method. We can see that, even though η^0 is completely random, the P-MDP constructed from η^0 generates safe policies when the threat threshold x is small. When the baseline

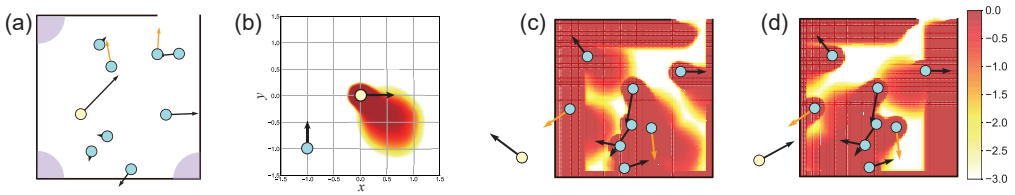


Figure 3: (a) Illustration of Jam task. The light blue circles are obstacles and the yellow circle is the agent. Three shaded corners are safe-zones. The arrow attached to each object shows its direction of movement. (b) Heat map of the trained threat function whose value at point (x, y) represent the threat when the obstacle (light blue) is placed at (x, y) with the same velocity. (c) and (d) are the heat maps of the sum of the threat functions of all moving obstacles. The heat value at (x, y) is the threat posed to the agent at (x, y) when it is moving in the direction indicated at the left bottom corner.

policy becomes better, the reward tends to become higher and the danger becomes smaller. At the same time, the difference becomes small after the second iteration. We see that, two policy iteration is enough to obtain near-optimal policy for this problem. The upper bound from Theorem 1 is shown as a dashed curve in Fig. 2 (c) which is in accordance with our theory.

4.2 High-Dimensional Environments

For high-dimensional environments, we consider three tasks on 2-D fields. See Fig. 1 and Fig. 3 for visualizations. For Point Gather, the agent's goal is to collect as many apples as possible while avoiding bombs. For Circuit, the agent's goal is to complete one lap around the circuit without crashing into a wall. The agent can control its movement by regulating its acceleration and steering. LiDAR sensors are used to compute the distance to obstacles. For Jam, the LiDAR-equipped agent's goal is to navigate its way out of a room from the exit located at the top right corner as quickly as possible without bumping into 8 randomly moving objects. The observation of each environment consists of 29, 364, 392 real values respectively and an agent needs to deal with stochasticity in Jam. For further details, please see Appendix F. Before explaining the experimental results, we would like to note that the total reward changes depending on how much we severely penalizes the agent for each crash. As a reference, we show the results when imposing heavier penalty on the crash in Appendix H.

As we discussed earlier, finding the optimal policy of the R-MDP is challenging in high-dimensional environments. However, since the constraint in these environments is to avoid collision with *any* obstacles, we can use the approximation method introduced in Sec 3.1 that uses a set of sub-R-MDPs containing one obstacle and one agent each. Thus, to guarantee the safety in P-MDP, we only learned a threat function of a heuristic baseline policy $\eta^{(o)}$ in a sub-R-MDP with one obstacle. For Circuit and Jam, we treated *wall* as a set of immobile obstacles so that we can construct the threat function of any shape. After about 10 minutes of data collection with the generative model, this threat function could be learned in less than half an hour without using GPUs. Please see the supplemental material for more detailed conditions and videos of the agent trained by each method. Fig. 3(c),(d) are the heat maps for the upper bound of the threat function computed in the way of Theorem 5. The color of each heat map at pixel z represents $\sum_n \mathscr{T}_0^{\eta,n}(s(z), a)$, where $s(z)$ represents the state at which the agent's current location is z and its velocity and direction is given by the picture located at the left corner of the heat map. We see that our threat function is playing a role similar to the artificial potential field (Ge & Cui, 2000; Cetin et al., 2011; Lam et al., 2010). Because our threat function is computed using all aspects of the agent's state (acceleration, velocity, location), we can provide more comprehensive measure of risk compared to other risk metrics such as TTC (Time To Collision) (Lee, 1976) used in smart automobiles that considers only 1D movement.

Fig. 4 plots the average reward and the crash rate against the training iteration for various methods: (i) classical MPC (exhaustive search), (ii) DQN with Lagrange penalty, and (iii) Constrained Policy Optimization (CPO) (Achiam et al., 2017a). DQN and CPO are model-free algorithms, while RP and MPC are generative model-based algorithms. (ii) is essentially the same as the Lagrangian-based methods like (Geibel & Wysotzki, 2005; Geibel, 2006). The results of DQN can be considered as a reference of the difficulty of solving CMDP with model-free RL methods. To compute the threat in

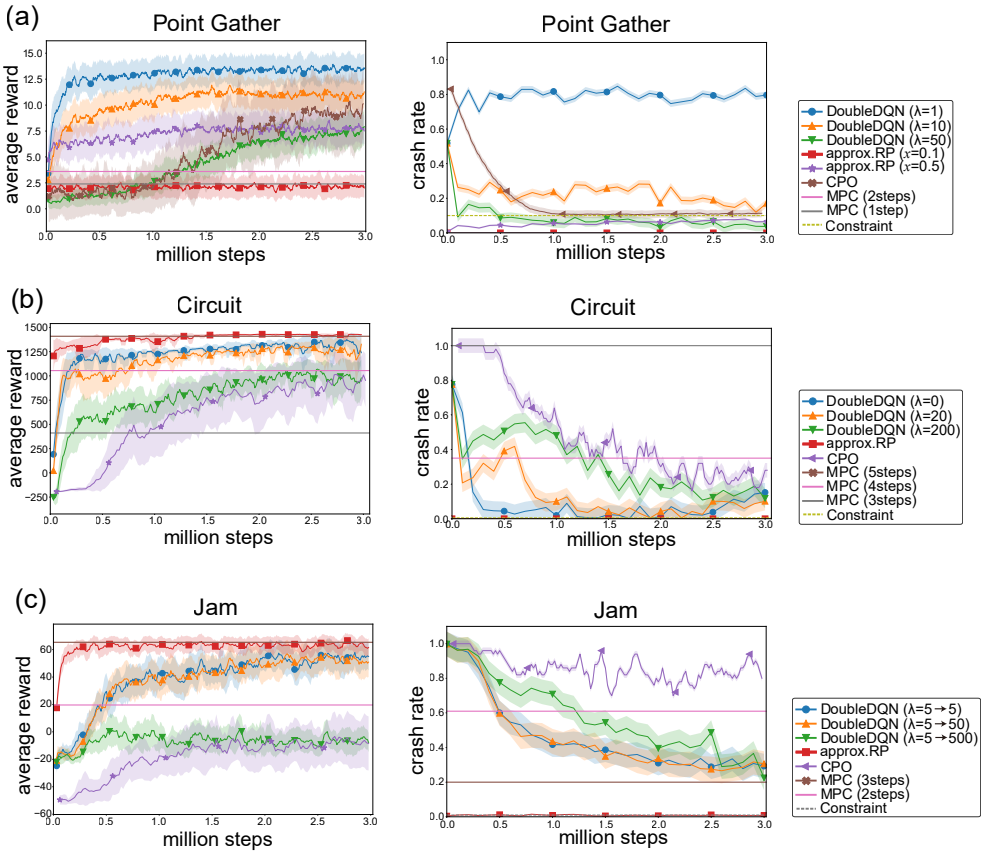


Figure 4: Comparison of multiple CMDP methods in terms of rewards (left panels) and crash rate (right panels) in diffferent environments (a) Point Gather, (b) Circuit and (c) Jam.

MPC, we took the same strategy of threat decomposition as our method G.4 in order to make the prediction efficient.

The average values and the 90% confidence intervals in the plot were computed over 10 seeds. The curve plotted for our approx. RP corresponds to the result obtained by a DQN-based P-MDP solution. The plot does not include the R-MDP phase. As we can see, our method achieves the highest reward at almost all phases of the training for both Circuit and Jam, while maintaining the lowest crash rate. Our method is safer than the 3-step MPC for both Jam and Circuit as well, a method with significantly higher runtime computational cost. For point-gather, RP with $x = 0.1$ performs worse in terms of reward than the penarlized DQN and CPO. Although 0.1 is a value that is slightly larger than the minimum safety threshold that guarantees the safety, our RP with $x = 0.1$ performs very conservatively in term of safety. This suggest that the pool of actions allowed for $x = 0.1$ is probably still too small. Fortunately, however, we can fine-tune the threshold to increase the pool of allowed actions for P-MDP without re-learning the threat function. Our RP with $x = 0.5$ performs competitively while satisfying the prescribed constraint. That we can fine-tune the threshold without retraining is a significant advantage of our method over Lagrange-type methods. Also, by the token of Corollary 3, that our policy is experimentally safe suggests that greedy-$\eta^{(o)}$ is also safe for the choice of the baseline policy $\eta^{(o)}$.

4.3 Recycling of the threat function for a CMDP in different environment

As we mentioned in introduction, we can reuse the threat function for different CMDP tasks with different goals if their safety constraints are defined similarly. In order to verify this powerful feature of our algorithm, we conducted two sets of experiments in which we apply a policy learned on one environment to the tasks on another environment. For the first set of experiments, we trained a safe

Environment	approx. RP	MPC (4steps)	DQN (λ=0)	DQN (λ=200)
Training env.	1439 (**0**)	1055 (0.35)	1432 (0.05)	933 (0.4)
Narrowed env.	377 (**0**)	959 (0.55)	-151 (1.0)	-145 (0.99)
Circle	130 (**0**)	351 (**0**)	-189 (1.0)	-171 (1.0)
Computation Time (s)	0.5	18.2	0.4	0.4

Table 1: Performance of trained policies on known and unknown Circuit environments. The values in the table are the obtained rewards, and inside the parenthesis are the probabilities of crashing. *Time* stands for the computation time consumed in making 100 action steps. Note that our method never crashes in this experiment. In this experiment, every crash is penalized by 200pts. See Appendix for the result with heavier crash penalty.

Environment	approx. RP	MPC (3steps)	DQN (λ=5)	DQN (λ=500)
N =3	78.2 (**0**)	77.5 (0.05)	77.2 (0.04)	4.4 (0.17)
N =8 (training env.)	69.1 (**0**)	65.3 (0.2)	47.1 (0.38)	-1.0 (0.24)
N = 15	33.0 (**0.02**)	36.6 (0.45)	16.5 (0.66)	-16.8 (0.51)
Computation Time (s)	1.2	285	0.4	0.4

Table 2: Performance of trained policies on and known and unknown Jam environments. Values in the table are reported in the same way as in Table 1. In this experiment, every crash is penalized by 50 pts. Note that we achieve the lowest crashing rate while achieving high score. Our RP stands out more if the crash is penalized more heavily (See Appendix)

policy for the Circuit task, and evaluated its performance on the environments that are different from the original circuit used in the training, (1) narrowed circuit with original shape, and (2) differently shaped circuit with same width. For the second set of experiments, we trained a safe policy for the Jam task, and tested its performance on other Jam tasks with different numbers of randomly moving obstacles. Figs. 1 and 2 show the results. For the modified Jam, we have no results for MPC with more than 3-step prediction since the exhaustive search cannot be completed within reasonable time-frame. The 4-step MPC requires 18.2secs per each 100 steps for Circuit, and the 3-step MPC requires 285secs per each 100 steps for the original Jam. We find that, even in varying environments, the policy obtained by our method can guarantee safety with high probability while seeking high reward. On the other hand, DQN in new environment is failing both in terms of reward and safety.

5 Conclusion

In this study we proposed a method that isolates the safety seeking procedure from reward-seeking procedure in solving CMDP. Although our method does not guarantee to find the optimal reward-seeking safe policy, it can perform significantly better than classical methods both in terms of safety and reward in high-dimensional dynamic environments like Jam. Our method is designed so that *more training* in R-MDP will always help increase the search space in P-MDP. Our treatment of the threat function not only allows us to solve similar problems without the need of re-learning, it also helps us obtain more comprehensive measure of danger at each state than conventional methods. Overall, we find that utilizing threat functions is a promising approach to safe RL and further research on framework may lead to new CMDP methods applicable to complex, real-world environments.

References

Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. *arXiv preprint arXiv:1705.10528*, 2017a.

Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization, 2017b. `https://github.com/jachiam/cpo`.

Eitan Altman. *Constrained Markov decision processes*, volume 7. CRC Press, 1999. ISBN 0849303826.

dexterousMarcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, and Alex Ray. Learning in-hand manipulation. *arXiv preprint arXiv:1808.00177*, 2018.

Christel Baier, Boudewijn Haverkort, Holger Hermanns, and J-P Katoen. Model-checking algorithms for continuous-time markov chains. *IEEE Transactions on software engineering*, 29(6):524–541, 2003. ISSN 0098-5589.

Maxime Bouton, Jesper Karlsson, Alireza Nakhaei, Kikuo Fujimura, Mykel J Kochenderfer, and Jana Tumova. Reinforcement learning with probabilistic guarantees for autonomous driving. *arXiv preprint arXiv:1904.07189*, 2019.

Omer Cetin, Sefer Kurnaz, Okyay Kaynak, and Hakan Temeltas. Potential field-based navigation task for autonomous flight control of unmanned aerial vehicles. *International Journal of Automation and Control*, 5(1):1–21, 2011. ISSN 1740-7516.

Yinlam Chow, Ofir Nachum, Edgar Duenez-Guzman, and Mohammad Ghavamzadeh. A lyapunov-based approach to safe reinforcement learning. In *Neural Information Processing Systems 2018*, 2018.

Yinlam Chow, Ofir Nachum, Aleksandra Faust, Mohammad Ghavamzadeh, and Edgar Duenez-Guzman. Lyapunov-based safe policy optimization for continuous control. *arXiv preprint arXiv:1901.10031*, 2019.

Stefano Di Cairano, Daniele Bernardini, Alberto Bemporad, and Ilya V Kolmanovsky. Stochastic mpc with learning for driver-predictive vehicle control and its application to hev energy management. *IEEE Transactions on Control Systems Technology*, 22(3):1018–1031, 2013. ISSN 1063-6536.

Jerry Ding, Eugene Li, Haomiao Huang, and Claire J Tomlin. Reachability-based synthesis of feedback policies for motion planning under bounded disturbances. In *2011 IEEE International Conference on Robotics and Automation*, pp. 2160–2165. IEEE. ISBN 1612843859.

Michael Everett, Yu Fan Chen, and Jonathan P. How. Motion planning among dynamic, decision-making agents with deep reinforcement learning. *Intelligent Robots and Systems (IROS), 2018 IEEE/RSJ International Conference on*, 2018.

Paolo Falcone, Francesco Borrelli, Jahan Asgari, H Eric Tseng, and Davor Hrovat. A model predictive control approach for combined braking and steering in autonomous vehicles. In *Mediterranean Conference on Control & Automation*, pp. 1–6. IEEE, 2007. ISBN 1424412811.

Shuzhi Sam Ge and Yan Juan Cui. New potential functions for mobile robot path planning. *IEEE Transactions on robotics and automation*, 16(5):615–620, 2000. ISSN 1042-296X.

Peter Geibel. Reinforcement learning for mdps with constraints. In *European Conference on Machine Learning*, pp. 646–653. Springer, 2006.

Peter Geibel and Fritz Wysotzki. Risk-sensitive reinforcement learning applied to control under constraints. *Journal of Artificial Intelligence Research*, 24:81–108, 2005.

Stephen James, Paul Wohlhart, Mrinal Kalakrishnan, Dmitry Kalashnikov, Alex Irpan, Julian Ibarz, Sergey Levine, Raia Hadsell, and Konstantinos Bousmalis. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. *arXiv preprint arXiv:1812.07252*, 2018.

Jie Ji, Amir Khajepour, Wael William Melek, and Yanjun Huang. Path planning and tracking for vehicle collision avoidance based on model predictive control with multiconstraints. *IEEE Transactions on Vehicular Technology*, 66(2):952–964, 2016. ISSN 0018-9545.

Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, and Vincent Vanhoucke. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018.

Chi-Pang Lam, Chen-Tun Chou, Kuo-Hung Chiang, and Li-Chen Fu. Human-centered robot navigation—towards a harmoniously human–robot coexisting environment. *IEEE Transactions on Robotics*, 27(1):99–112, 2010. ISSN 1552-3098.

David N Lee. A theory of visual control of braking based on information about time-to-collision. *Perception*, 5(4):437–459, 1976. doi: 10.1068/p050437.

Wei Liu and Marcelo Jr. Incremental sampling-based algorithm for risk-aware planning under motion uncertainty. In *IEEE International Conference on Robotics and Automation*, pp. 2051–2058, 2014.

Jan Marian Maciejowski. *Predictive control: with constraints*. Pearson education, 2002. ISBN 0201398230.

Megumi Miyashita, Shirou Maruyama, Yasuhiro Fujita, Mitsuru Kusumoto, Tobias Pfeiffer, Eiichi Matsumoto, Ryosuke Okuta, and Daisuke Okanohara. Toward onboard control system for mobile robots via deep reinforcement learning. In *Proceedings of workshop on machine learning systems (LearningSys) in the twenty-ninth annual conference on neural information processing systems (NIPS)*, 2018.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, and Georg Ostrovski. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. ISSN 0028-0836.

Santiago Paternain, Luiz Chamon, Miguel Calvo-Fullana, and Alejandro Ribeiro. Constrained reinforcement learning has zero duality gap. In *Advances in Neural Information Processing Systems*, pp. 7553–7563, 2019.

Yadollah Rasekhipour, Amir Khajepour, Shih-Ken Chen, and Bakhtiar Litkouhi. A potential field-based model predictive path-planning controller for autonomous road vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 18(5):1255–1267, 2016a. ISSN 1524-9050.

Yadollah Rasekhipour, Amir Khajepour, Shih-Ken Chen, and Bakhtiar Litkouhi. A potential field-based model predictive path-planning controller for autonomous road vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 18(5):1255–1267, 2016b. ISSN 1524-9050.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, and Marc Lanctot. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. ISSN 0028-0836.

Sean Summers, Maryam Kamgarpour, John Lygeros, and Claire Tomlin. A stochastic reach-avoid problem with random obstacles. In *Proceedings of the 14th international conference on Hybrid systems: computation and control*, pp. 251–260.

Ryo Takei, Haomiao Huang, Jerry Ding, and Claire J Tomlin. Time-optimal multi-stage motion planning with guaranteed collision avoidance via an open-loop game formulation. In *2012 IEEE International Conference on Robotics and Automation*, pp. 323–329. IEEE. ISBN 1467314056.

Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton. Chainer: a next-generation open source framework for deep learning. In *Proceedings of workshop on machine learning systems (LearningSys) in the twenty-ninth annual conference on neural information processing systems (NIPS)*, volume 5, pp. 1–6, 2015.

Eiji Uchibe and Kenji Doya. Constrained reinforcement learning from intrinsic and extrinsic rewards. In *Development and Learning, 2007. ICDL 2007. IEEE 6th International Conference on*, pp. 163–168. IEEE, 2007. ISBN 1424411157.

Erwin Walraven and Matthijs TJ Spaan. Column generation algorithms for constrained pomdps. *Journal of artificial intelligence research*, 62:489–533, 2018. ISSN 1076-9757.

Yang Wang and Stephen Boyd. Fast model predictive control using online optimization. *IEEE Transactions on control systems technology*, 18(2):267–278, 2010. ISSN 1063-6536.

Thomas Weiskircher, Qian Wang, and Beshah Ayalew. Predictive guidance and control framework for (semi-) autonomous vehicles in public traffic. *IEEE Transactions on control systems technology*, 25(6):2034–2046, 2017. ISSN 1063-6536.

Min Wen, Rüdiger Ehlers, and Ufuk Topcu. Correct-by-synthesis reinforcement learning with temporal logic constraints. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pp. 4983–4990. IEEE, 2015. ISBN 1479999946.

Michael T Wolf and Joel W Burdick. Artificial potential functions for highway driving with collision avoidance. In *2008 IEEE International Conference on Robotics and Automation*, pp. 3731–3736. IEEE, 2008. ISBN 1424416469.

Eiichi Yoshida, Claudia Esteves, Igor Belousov, Jean-Paul Laumond, Takeshi Sakaguchi, and Kazuhito Yokoi. Planning 3-d collision-free dynamic robotic motion through iterative reshaping. *IEEE Transactions on Robotics*, 24(5):1186–1198, 2008. ISSN 1552-3098.

APPENDIX

A PROOF OF THEOREM 1

Proof. Let's first prove the theorem by induction. At $t = T$, Eq. (6) holds because both sides are zero. Suppose Eq. (6) holds at $t + 1$. Then at time t, the danger value function of π at t satisfies

$$\begin{aligned}\mathscr{D}_t^\pi(s_t) =& E_\pi\left[d(s_t,a_t) + \beta\mathscr{D}_{t+1}^\pi(s_{t+1}) \mid s_t\right]\\ \leq& E_\pi\Big[d(s_t,a_t) + \beta\Big(\mathscr{D}_{t+1}^\eta(s_{t+1}) + \sum_{k=t+1}^{T-1}\beta^{k-t-1}x_k E_\pi[z_k|s_{t+1}]\Big) \mid s_t\Big]\\ =& E_\pi\Big[\mathscr{T}_t^\eta(s_t,a_t) \mid s_t\Big] + \sum_{k=t+1}^{T-1}\beta^{k-t}x_k E_\pi[z_k|s_t] \end{aligned} \tag{10}$$

The above inequality follows from the induction hypothesis at $t + 1$, and the last equality is from the definition of $\mathscr{T}_t^\eta$ and the property of conditional expectation.

Since $\pi \in \Pi_G^{\eta,\boldsymbol{x}}$, when $s_t \notin S^{\eta,\boldsymbol{x}}$ we know that $E_\pi\Big[\mathscr{T}_t^\eta(s_t,a_t) \mid s_t\Big] \leq E_\eta[\mathscr{T}_t^\eta(s_t,a_t) \mid s_t]$. Noting the fact above and $\mathscr{D}_t^\eta(s_t) = E_\eta[\mathscr{T}_t^\eta(s_t,a_t) \mid s_t]$, for the first term in Eq. (10), we have

$$\begin{aligned}& E_\pi\Big[\mathscr{T}_t^\eta(s_t,a_t) \mid s_t\Big] - \mathscr{D}_t^\eta(s_t)\\ \leq& 1_{s_t\in S^{\eta,\boldsymbol{x}}}\int_{a_t}(\pi(a_t|s_t) - \eta(a_t|s_t))\mathscr{T}_t^\eta(s_t,a_t)da_t\\ \leq& 1_{s_t\in S^{\eta,\boldsymbol{x}}}\int_{a_t}|\pi(a_t|s_t) - \eta(a_t|s_t)|\mathscr{T}_t^\eta(s_t,a_t)da_t\\ \leq& 1_{s_t\in S^{\eta,\boldsymbol{x}}}2TV(\pi(\cdot|s_t),\eta(\cdot|s_t))\max_{a\in\text{supp}(\pi(\cdot|s_t))}\mathscr{T}_t^\eta(s_t,a)\\ \leq& 2z_t x_t \end{aligned} \tag{11}$$

Here the first inequality holds because the term is negative when $s_t \notin S^{\eta,\boldsymbol{x}}$. The second inequality is true by taking the absolute value with fact that $\mathscr{T}_t^\eta(s_t,a_t) \geq 0$. The thirdinequality follows from the property of the total variation distance. The last inequality is true because of the construction of $\Pi_G^{\eta,\boldsymbol{x}}$.

Now we get

$$\begin{aligned}\mathscr{D}_t^\pi(s_t) \leq& \mathscr{D}_t^\eta(s_t) + 2z_t x_t + 2\sum_{k=t+1}^{T-1}\beta^{k-t}x_k E_\pi[z_k|s_t]\\ =& \mathscr{D}_t^\eta(s_t) + 2\sum_{k=t}^{T-1}\beta^{k-t}x_k E_\pi[z_k|s_t]\end{aligned} \tag{12}$$

which completes the induction step and the proof of Theorem 1.Notice that the logic used here is essentially same as the one used in the proof of Theorem 1 of (Achiam et al., 2017a). However, our result is slightly different in that we consider finite time-horizon and we utilize our assumption that $\mathscr{T}_t^\eta(s,a) \leq x_t$ for $(\eta, \mathbf{x})$-secure states.

□

B PROOF OF COROLLARY 3

Proof. First, we will derive a result similar to Theorem 1.

Theorem 2. For a given policy η and a sequence of safety thresholds $\boldsymbol{x} = (x_0, \ldots, x_{T-1})$, let π be a policy in $\Pi^{\eta,\boldsymbol{x}}$. Then for all $t \in \{0, \ldots, T-1\}$,

$$\begin{aligned}\mathscr{D}_t^\pi(s_t) \leq& \mathscr{D}_t^{\text{greedy-}\eta}(s_t) + 2\sum_{k=t}^{T-1} \beta^{k-t} x_k E_\pi\left[z_k^g \mid s_t\right], \\ \leq& \min_a \mathscr{T}_t^\eta(s_t,a) + 2\sum_{k=t}^{T-1} \beta^{k-t} x_k E_\pi\left[z_k^g \mid s_t\right]. \end{aligned} \tag{13}$$

where z_t^g is a distance measure between π and greedy-η given by $z_t^g = 1_{s_t \in S^{\eta,\boldsymbol{x}}} d_{TV}(\pi(\cdot|s_t), \text{greedy-}\eta(\cdot|s_t))$.

Let's prove the theorem by induction. At $t = T$, Eq. (13) holds because both sides are zero. Suppose Eq. (13) holds at $t+1$. Then at time t, the danger value function of π at t satisfies

$$\begin{aligned}\mathscr{D}_t^\pi(s_t) =& E_\pi\left[d(s_t,a_t) + \beta \mathscr{D}_{t+1}^\pi(s_{t+1}) \mid s_t\right] \\ \leq& E_\pi\Big[d(s_t,a_t) + \beta\Big(\mathscr{D}_{t+1}^{\text{greedy-}\eta}(s_{t+1}) + 2\sum_{k=t+1}^{T-1} \beta^{k-t-1} x_k E_\pi[z_k^g|s_{t+1}]\Big) \mid s_t\Big] \\ =& E_\pi\Big[\mathscr{T}_t^{\text{greedy-}\eta}(s_t,a_t) \mid s_t\Big] + 2\sum_{k=t+1}^{T-1} \beta^k x_k E_\pi[z_k^g|s_t] \end{aligned} \tag{14}$$

The above inequality follows from the induction hypothesis at $t+1$, and the last equality is from the definition of $\mathscr{T}_t^{\text{greedy-}\eta}$ and the property of conditional expectation.

Since $\pi \in \Pi^{\eta,\boldsymbol{x}}$, when $s_t \notin S^{\eta,\boldsymbol{x}}$ we know that $\pi(\cdot|s_t) = \text{greedy-}\eta(\cdot|s_t)$. Noting the fact above and $\mathscr{D}_t^{\text{greedy-}\eta}(s_t) = E_{\text{greedy-}\eta}[\mathscr{T}_t^{\text{greedy-}\eta}(s_t,a_t) \mid s_t]$, for the first term in Eq. (14), we have

$$\begin{aligned} & E_\pi\Big[\mathscr{T}_t^{\text{greedy-}\eta}(s_t,a_t) \mid s_t\Big] - \mathscr{D}_t^{\text{greedy-}\eta}(s_t) \\ =& 1_{s_t \in S^{\eta,\boldsymbol{x}}} \int_{a_t} (\pi(a_t|s_t) - \text{greedy-}\eta(a_t|s_t)) \mathscr{T}_t^{\text{greedy-}\eta}(s_t,a_t) da_t \\ \leq& 1_{s_t \in S^{\eta,\boldsymbol{x}}} \int_{a_t} |\pi(a_t|s_t) - \text{greedy-}\eta(a_t|s_t)| \mathscr{T}_t^{\text{greedy-}\eta}(s_t,a_t) da_t \\ \leq& 1_{s_t \in S^{\eta,\boldsymbol{x}}} 2TV(\pi(\cdot|s_t), \text{greedy-}\eta(\cdot|s_t)) \max_{a \in \text{supp}(\pi(\cdot|s_t))} \mathscr{T}_t^{\text{greedy-}\eta}(s_t,a) \\ \leq& 1_{s_t \in S^{\eta,\boldsymbol{x}}} 2TV(\pi(\cdot|s_t), \text{greedy-}\eta(\cdot|s_t)) \max_{a \in \text{supp}(\pi(\cdot|s_t))} \mathscr{T}_t^{\eta}(s_t,a) \\ \leq& 2 z_t^g x_t \end{aligned} \tag{15}$$

Here the first inequality is true by taking the absolute value with fact that $\mathscr{T}_t^{\text{greedy-}\eta}(s_t,a_t) \geq 0$. The second inequality follows from the property of the total variation distance. The third inequality holds because greedy-η is the policy after one step of policy improvement from η, hence $\mathscr{T}_t^{\text{greedy-}\eta}(s_t,a) \leq \mathscr{T}_t^\eta(s_t,a)$. The last inequality is true because of the construction of $\Pi^{\eta,\boldsymbol{x}}$.

Now we get

$$\begin{aligned}\mathscr{D}_t^{\pi}(s_t) \leq & \mathscr{D}_t^{\text{greedy-}\eta}(s_t) + 2z_t x_t + 2\sum_{k=t+1}^{T-1} \beta^{k-t} x_k E_{\pi}[z_k^g | s_t] \\ = & \mathscr{D}_t^{\text{greedy-}\eta}(s_t) + 2\sum_{k=t}^{T-1} \beta^{k-t} x_k E_{\pi}[z_k^g | s_t] \\ \leq & \min_a \mathscr{T}_t^{\eta}(s_t, a) + 2\sum_{k=t}^{T-1} \beta^{k-t} x_k E_{\pi}\left[z_k^g \mid s_t\right] \end{aligned} \tag{16}$$

which completes the induction step and the proof of Theorem 2.

Now using Theorem 2, Corollary 3 follows using the same arguments as Corollary 2.

□

C PROOF OF THEOREM 5

By the assumption about the transition probability, $E_{\eta}[d^n(s_k^{(o)}, s_k^n, a_k)|s_t, a_t] = E_{\eta}[d^n(s_k^{(o)}, s_k^n, a_k)|s_t^{(o)}, s_t^{(n)}, a_t]$. Using this property, the claim of Theorem 5 immediately follows by applying the union bound $d(s_t, a_t) \leq \sum_{n=1}^{N} d^n(s_t^o, s_t^n, a_t)$.

D ANOTHER INTERPRETATION FOR P-MDP

Since we restrict attention to policies in Π^{η}, the P-MDP can be viewed as an MDP defined of the smaller state space S^{η}. Namely, we constructed the tuple $(S^{\eta}, A^{\eta}, r_P^{\eta}, P_P^{\eta}, P_0)$, whose components are defined as follows. The function $r_P^{\eta}(\cdot|s)$ is the restriction of the reward r to $A^{\eta}(s)$ for all $s \in S^{\eta}$. P_P^{η} is a transition probability function derived from the original state transition probability P such that, for all $s_1, s_2 \in S^{\eta}$, $P_P^{\eta}(s_2|s_1, a) = P(s_2|s_1, a) + P((s_1, a) \xrightarrow{(S^{\eta})^c} s_2)$ where $(s_1, a) \xrightarrow{(S^{\eta})^c} s_2$ is the set of all trajectories from s_1 to s_2 that (1) take a detour to $(S^{\eta})^c$ at least once after taking the action a at s_1, (2) take the action $a^*(s') = \arg\min_{a \in A} \mathscr{T}^{\eta}(s', a)$ for all $s' \in (S^{\eta})^c$, and (3) lead to s_2 without visiting any other states in S^{η}.

E VALUE ITERATION AND Q-LEARNING FOR P-MDP

When the transition probabilities are available, we can solve the P-MDP using value iteration. The value iteration for P-MDP can be described by the following algorithm.

Algorithm 3 Value Iteration for P-MDP

Input: threat function $\mathscr{T}^{\eta}(s, a)$, transition probabilities $P(s'|s, a)$
Initialize $V(s) \leftarrow 0$
while before convergence **do**
  for each s, a **do**
    $Q(s, a) \leftarrow \sum_{s'} P(s'|s, a)(r(s, a) + \gamma V(s'))$
  end for
  for each $s \in S^{\eta}$ **do**
    $V(s) \leftarrow \max_{a \in A^{\eta}(s)} Q(s, a)$
  end for
  for each $s \notin S^{\eta}$ **do**
    $a^{\text{greedy-}\eta} \leftarrow \arg\min_a \mathscr{T}^{\eta}(s, a)$

    $V(s) \leftarrow Q(s, a^{\text{greedy-}\eta})$
  end for
end while

When the transition probabilities are not available, we can solve the P-MDP using Q-learning or its Deep RL versions.

Algorithm 4 Q-learning for P-MDP

Input: threat function $\mathscr{T}^\eta(s,a)$, learning rate α and a behavior policy π
Initialize $Q(s,a)$
while before convergence **do**
  sample a transition s, a, s' by π
  if $s' \in S^\eta$ **then**
    $Q_{\text{target}}(s,a) \leftarrow r(s,a) + \gamma \max_{a' \in A^\eta(s')} Q(s',a')$
  else
    $a^{\text{greedy-}\eta} \leftarrow \arg\min_{a'} \mathscr{T}^\eta(s',a')$
    $Q_{\text{target}}(s,a) \leftarrow r(s,a) + \gamma Q(s', a^{\text{greedy-}\eta})$
  end if
  $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha Q_{\text{target}}(s,a)$
end while

F ENVIRONMENTS

CIRCUIT

In this task, the agent's goal is to complete one lap around the circuit without crashing into the wall. Each state in the system was set to be the tuple of (1) location, (2) velocity, (3) the direction of the movement, and (4) the raw LiDAR input for each one degree. Thus, the state observation has 364 real values. The set of actions allowed for the agent was $\{$0.15rad to left, 0.05rad to left, stay course, 0.05 rad to right, 0.15 rad to right $\} \times \{$ 0.02 unit acceleration, no acceleration, 0.02 unit deceleration $\}$ (15 options). At all time, the speed of the agent was truncated at $\{-0.04, 0.1\}$. We reward the agent for the distance of travel along the course in the right direction, which accumulates to 1250pts for one lap. We also give negative rewards for the stopping and collision at the time step. We penalize the agent 1pt for stopping, and penalized the agent by 200pts for colliding. We set the length of the episode to 200 steps, which is the approximate number of steps required to make one lap. In the computation of each summand in the threat decomposition, we augmented the generated dataset by flipping the state with respect to $x - axis$(left-right). The constraint for CMDP was set to 1e-3.

JAM

In this task, the agent's goal is to navigate its way out of a room from the exit located at the top left corner without bumping into 8 randomly moving objects. We set three circular safety zones centered at each corner except for the top left corner, i.e., exit. Any moving obstacle entered into the safety zone disappear. Without the safety zone, the task seems to be too difficult, i.e., there is a situation that the agent cannot avoid the collision even if the agent tries his best. We set the safety zone to ease the problem hoping the probability that the agent can solve the task when employing the optimal policy becomes reasonably high. The field was 3×3 square and the radius of the safety zone located at three corners were set to 0.5. The radius of the agent and moving obstacles were set to 0.1. We reward the agent for its distance from the exit, and controlled its value so that the accumulated reward at the goal will be around 85. The agent is given 10 points when it reaches the goal, penalized -0.05 points for stopping the advance, and given 50pts penalty for each collision. Similar to the setting as in Circuit, the agent is allowed to change direction and acceleration of the movement simultaneously at each time point. The observation of agent has the information of relative location, relative velocity, and relative direction of 8 nearest cars in addition to the observation of Circuit. It is composed of totally 396 real values. The set of actions allowed for the agent is $\{$0.30 rad to left, 0.10 rad to left, stay course, 0.10 rad to right, 0.30 rad to right $\} \times \{$ 0.02 unit acceleration, no acceleration, 0.02 unit deceleration $\}$. At all time, the speed of the agent is truncated at $\{-0.1, 0.1\}$. Each obstacle in the environment is allowed to take a random action from $\{$0.15rad to left, 0.05rad to left, stay course, 0.05 rad to right, 0.15 rad to right $\} \times \{$ 0.02 unit acceleration, no acceleration, 0.02 unit deceleration $\}$. The speed of the environment is truncated at $\{0, 0.06\}$. We set the length of each episode to 100

steps. In the computation of each summand in the threat decomposition, we augmented the generated dataset by flipping the state with respect to $x - axis$(left-right). The constraint for CMDP was set to 0.01.

POINT GATHER

In this task, the goal is to collect as many green apples as possible while avoiding red bombs. There are 2 apples and 8 bombs in the field. We used the exact same task setting as the one used in the original paper, and the agent is rewarded 10pts when the agent collected apple. The point mass agent receives 29-dimensional state and can take two-dimensional continuous actions. For the implementation of DQN, we discretize each action variable into 9 values. The state variable takes real values, which include position, velocity, direction, distance to the bomb, and etc. The action variables determine the direction and the velocity of the agent. As is done in the experiment in (Achiam et al., 2017a), we set the length of each episode to 15 steps. At every step, our selected baseline policy randomly chooses a direction from $[-14.33, 14.33]$degree range with uniform distribution, and move forward with probability 0.9 and move backward with probability 0.1. After deciding the direction of the move, the baseline policy prompts the agent to move at at the speed chosen uniformly from $[0, 1]$unit range. For the approximation of threat function, we sampled $1e6$ trajectories by simualtors, and we did not bother to the compute average value for each fixed initial state. The constraint for CMDP was set to 0.1.

G MODEL ARCHITECTURE AND OPTIMIZATION DETAILS

G.1 RP-ALGORITHM

We implement our method on Chainer (Tokui et al., 2015). The code is available in supplementary material.

LEARNING OF THREAT FUNCTION IN R-MDP

For the Circuit and Jam tasks, the agent must avoid collisions with both moving obstacles and the wall. For these environments, it is computationally difficult to obtain the best η. Thus, we follow the approximate RP algorithm by training a threat function for a baseline policy $\eta^{(o)}$. We used $\eta^{(o)}$that (1) decides the direction of the movement by staying course with probability 0.6, turning right with probability 0.2 and turning left with probability 0.2, and (2) decides its speed by accelerating with probability 0.2 and decelerating with probability 0.2. We chose not to train η specifically for this experiment, because η to be used in the decomposition must not depend on the state of all but one obstacle, and we could not expect to improve its safety much by training it. The threat function for the collision with the immobile point is used to avoid collision with the wall, which can be considered as a set of immobile points. We thus trained two threat functions in total: one corresponding to the collision with an immobile point, and one corresponding to the collision with a randomly moving obstacle. Threat function for $\eta^{(o)}$ can be trained by using generative models starting at various initial points to collect samples and by conducting supervised learning.

For the threat function for the collision with immobile object, we used a neural network with three fully connected layers (100-50-15). For the threat function for the collision with moving obstacles, we used a network with four fully connected layers (500-500-500-15). For the training dataset, we sampled 100,000 initial state and simulated 10,000 paths of length 5 from each initial state. We train the network with Adam. The parameter settings for the training of the threat function of immobile point and of mobile obstacle are ($\alpha = 1e-2, \epsilon = 1e-2$, batchsize = 512, number of epochs = 20), and ($\alpha = 1e-3, \epsilon = 1e-2$, batchsize = 512, number of epochs = 25), respectively. We used relu as the activation function for the approximation of all threat functions.

For the Point Gather task, we again use the approximate RP algorithm. The threat function is estimated by a three-layer (100-100-81) fully connected neural network.

SOLVING P-MDP

For the Planning MDP, we use DQN. The network used for the Q-function in P-MDP looks like $\Phi(\ell, s)$, where ℓ is the result of applying lider outputs to 1D conv network, s is the combination of location, velocity and angle, and Φ is a three-layered fully connected network (50-20-15) with tanh activation functions.

The convolutional layer has 3 output channels with kernel size = 6 and stride = 1, followed by max pooling with kernel size = 8. In Point Gather, we use fully connected neural network with three layers (50-50-81). We trained the network with Adam ($\alpha = 1e-3, \epsilon = 1e-2$). We linearly decayed ε for ε-greedy from 1 to 0.05 over 3M iterations. We used one GPU, with which it took no longer than about 24 hours to complete 3M iterations.

G.2 DQN WITH LAGRANGE COEFFICIENT

As for the network, we used the same DQN architecture as the one we used in P-MDP. We used tanh for the activation function. For Lagrange coefficient, we test with three Lagrange coefficients for each task, $\{0, 20, 200\}$ for Circuit, $\{5, 50, 500\}$ for Jam, $\{1, 10, 50\}$ for Point Gather, respectively. For the Jam task, the initial Lagrange coefficients are all set to 5 and gradually incremented to the final values $\{1, 10, 50\}$ as done in (Miyashita et al., 2018). This heuristic pushes the agent to learn the goal-oriented policy first, and then learn the collision avoidance.

G.3 CONSTRAINED POLICY OPTIMIZATION

As for CPO, we use the same implementation publicized on Github (Achiam et al., 2017b), i.e., the policy is a Gaussian policy implemented by a three layer MLP with hidden units 64 and 32 for all tasks. We used tanh for the activation function.

G.4 MODEL PREDICTIVE CONTROL

We tested MPC with receding horizon $\in \{3, 4, 5\}$ for Circuit, $\in \{2, 3\}$ for Jam and $\in \{1, 2\}$ for Point Gather. At every step, selected the action with the highest reward among those that were deemed safe by the prediction. To select the next safe action, we used tree-search on the sequence of actions, and prematurely pruned branches that were sure to violate the safety condition. Also, for the JAM experiment, we used the same strategy as our approximate-RP. More particularly, we allowed MPC to make predictions in a set of environments consisting of one pair of obstacle and ego-object each, and evaluated the safety by summing the danger values computed from all environments. This was done in order to reduce the computation time of prediction (which already is large), because the number of combinatorial cases explodes as we increase the number of obstacle objects.

H PERFORMANCE EVALUATIONS WITH LARGER COLLISION PENALTY ON CIRCUIT AND JAM

Here we show the score computed with a larger collision penalty on Circuit and Jam tasks to stress the fact that the performance will vary arbitrarily depending on how much we put emphasis on the constraints. In below, the collision penalty is increased to be 1500pts and 500pts instead of the ones used in Tables1 and 2 where the collision penalties are 200pts and 50pts, respectively.

Environment	approx. RP	MPC (4steps)	DQN (λ=0)	DQN (λ=200)
Training env.	1439 (**0**)	600 (0.35)	1367 (0.05)	413 (0.4)
Narrowed env.	377 (**0**)	244 (0.55)	-1451 (1.0)	-1432 (0.99)
Circle	130 (**0**)	351 (**0**)	-1489 (1.0)	-1471 (1.0)
Computation Time (s)	0.5	18.2	0.4	0.4

Table 3: Performance of trained policies on unknown Circuit environments. The values in the table are the obtained rewards, and inside the parenthesis are the probabilities of crashing. *Time* stands for the computation time consumed in making 100 action steps. Note that our method never crashes in this experiment. In this experiment, every crash is penalized by 1500pts.

Environment	approx. RP	MPC (3steps)	DQN (λ=5)	DQN (λ=500)
N =3	78.2 (**0**)	55 (0.05)	59.2 (0.04)	-72.1 (0.17)
N =8 (training env.)	69.1 (**0**)	-24.7 (0.2)	-123.9 (0.38)	-109 (0.24)
N = 15	24.0 (**0.02**)	-165.9 (0.45)	-280.5 (0.66)	-246.3 (0.51)
Computation Time (s)	1.2	285	0.4	0.4

Table 4: Performance of trained policies on and unknown Jam environments. Values in the table are reported in the same way as in Table 3. In this experiment, every crash is penalized by 500 pts. Note that we achieve the lowest crashing rate while achieving high score.

I ADDITIONAL RELATED WORK

As mentioned in the introduction, one classic strategy for safe maximization of the reward is MPC, which uses a simulator/generator to directly predict the outcome of candidate actions. There are many applications and variations of this approach (Maciejowski, 2002; Yoshida et al., 2008; Wang & Boyd, 2010; Di Cairano et al., 2013; Liu & Jr, 2014; Weiskircher et al., 2017; Rasekhipour et al., 2016b). One straight forward approach for constrained Markov Decision process is to formulate the objective with Lagrange multiplier. This approach has been used widely in practice, including (Geibel & Wysotzki, 2005; Geibel, 2006; Uchibe & Doya, 2007; Everett et al., 2018; Walraven & Spaan, 2018; Paternain et al., 2019) and CPO(Achiam et al., 2017a). CPO is a method that gradually improves the safe policy by making a local search for a better safe policy in the neighborhood of the current safe policy. By nature, at each update, CPO has to determine whether a suggested member in the neighborhood of current safe policy satisfies the safety constraint. In implementation, this is again done by the evaluation of Lagrange coefficients. Accurate selection of *safe* policies in the neighborhood is especially difficult when the danger to be avoided is "rare" and "catastrophic", because we would need massive number of samples to verify whether a given policy is safe or not. This is a problem common to all approaches using the Lagrange multipliers, because the optimization of Lagrange coefficient requires accurate evaluation of the expected danger. Moreover, because each update is incremental, we have to repeat this process multiple times (usually, several dozens of times for Point Circle, and several thousands of times for Ant Gather and Humanoid Circle). Lyapunov based approach (Chow et al., 2018; 2019) are also similar in nature. At each step of the algorithm, Lyapunov based approach uses safety-margin function ϵ—a state-dependent measure of how bold the agent can be while remaining safe– to construct a set of safe policies in the neighborhood of the baseline policy. For the accurate computation of the margin, one must use the transition probability to solve a linear programming problem in the space whose dimension equals the number of states in the system. This computation requires $O(|X||A|^2(|A|+2))$ computation time. Especially when dealing with system with rare dangerous events, the set of policies found this way may still contain unsafe policies. Reachability analysis is a theory that provides a safety guarantee. However, they built their theory on more restrictive setting than the one we consider here (Takei et al.; Ding et al.; Summers et al.; Rasekhipour et al., 2016a). Model-checking is another approach to guarantee the safety. Once the constraints are represented in the form of temporal logic constraints or computation-tree logic (Baier et al., 2003; Wen et al., 2015; Bouton et al., 2019), we can ensure the safety by using model checking systems. However, it is sometimes difficult to express the constraints in such a structured form. Also, even when we can represent the constraints in the structured form, the computational cost can be heavy. When the state-action space is large, the computation time required for the model-checking system can become prohibitively heavy.