Efficient and Safe Exploration in Deterministic Markov Decision Processes with Unknown Transition Models

Erdem Bıyık[*1], Jonathan Margoliash[*2,3], Shahrouz Ryan Alimo[2], and Dorsa Sadigh[1,4]

Abstract— We propose a safe exploration algorithm for deterministic Markov Decision Processes with unknown transition models. Our algorithm guarantees safety by leveraging Lipschitz-continuity to ensure that no unsafe states are visited during exploration. Unlike many other existing techniques, the provided safety guarantee is deterministic. Our algorithm is optimized to reduce the number of actions needed for exploring the safe space. We demonstrate the performance of our algorithm in comparison with baseline methods in simulation on navigation tasks.

I. Introduction

Guaranteeing safety is a vital issue for many modern robotics systems, such as unmanned aerial vehicles (UAVs), autonomous cars, or domestic robots [1], [2], [3]. One approach is to attempt to specify all potential scenarios a robot may encounter a priori. However, this is usually impractical due to the fact that such solutions are either computationally expensive to compute or that robots today need to deal with uncertain and diverse environments. Hence, we need to design algorithms for robots that can *safely* and *autonomously* learn about the uncertain environment they live in, which can potentially address both problems [4], [5].

Reinforcement learning algorithms autonomously perform exploration, and they have shown promising results in many fields of artificial intelligence. Therefore, it is natural to explore the implications of reinforcement learning in robotics. Unlike most applications in artificial intelligence, where unsafe outcomes from learning can occur in simulation, in robotics, we would need to avoid unsafe scenarios at all costs. Therefore, *safe learning*, which is the process of applying a learning algorithm such as reinforcement learning while still satisfying a set of safety specifications, has attracted great interest in recent years. Safety usually has two main interpretations: one is related to stochasticity of the environment, where the goal is to guarantee staying in a given performance bound as is commonly studied in robust control [6], [7], [8], [9], [10], [11], [12]. The second interpretation is due to the system falling in an undesirable physical state, which is a common interpretation used in

robotics [13], [14], [15], [16], [17]. In this paper, we focus on safe learning and exploration as the avoidance of 'unsafe' - physically undesirable - states. We refer to [18] for a survey on safe reinforcement learning.

To perform exploration in safety-critical systems, prior knowledge of the task is often incorporated into the exploration process. In [19] and [20], the authors proposed a method to safely explore a deterministic Markov Decision Process (MDP) using Gaussian processes. In their work, they assumed the transition model is known and that there exists a predefined safety function. Both of these assumptions can be quite restrictive when the system is going to operate in unknown environments. In our work, we plan to address both of these challenges by considering unknown transition models, and no access to a predefined safety function. Similarly, other work has considered reachability analysis as well as Gaussian processes to perform safe reinforcement learning [17], [21], and used a safety metric to improve their algorithm. However, it is not trivial to derive an appropriate safety metric in many robotics tasks. Other techniques utilize teacher demonstrations to avoid unsafe states [13], [14], [22]. However, teaching demonstrations are usually difficult to capture as operating robots with high degrees of freedom can be challenging, especially if the system dynamics are unknown due to the existing uncertainty in the environment.

In our work, we propose an algorithm to safely and autonomously explore a deterministic MDP whose transition function is *unknown*. We take a natural definition of safety, similar to [23]: *If we can recover from a state s, i.e. we can move from it to a state that is known to be safe, then the state s is also safe.* Instead of relying on Gaussian processes or other estimation procedures, our exploration algorithm directly leverages the underlying continuity assumptions, and so guarantees safety deterministically. We demonstrate our algorithm in simulation on two different navigation tasks.

II. Problem Definition

Our goal in this project is to design an algorithm for a robot to safely and efficiently explore uncertain parts of the environment. We want an algorithm that deterministically ensure safety and expand the size of the known safe state set. The theoretical work in this section will build towards formalizing this goal in equation (7).

A. Introductory Assumptions

First, we begin by formalizing our interactions with the environment.

* First two authors contributed equally and are listed in alphabetical order. Emails: ebiyik@stanford.edu, jmargoli@eng.ucsd.edu, sralimo@jpl.nasa.gov, dorsa@cs.stanford.edu

[1] Erdem Bıyık and Dorsa Sadigh are with Electrical Engineering, Stanford University, CA, 94305, United States.

[2] Jonathan Margoliash and Shahrouz Ryan Alimo are with Jet Propulsion Laboratory, California Institute of Technology, CA, 91109, United States.

[3] Jonathan Margoliash is with UC San Diego Jacobs School of Engineering, CA, 92093, United States.

[4] Dorsa Sadigh is with Computer Science, Stanford University, CA, 94305, United States.

Assumption 1: We model the dynamical system living in an unknown environment as a deterministic MDP [24]. Such an MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, f(s,a) \rangle$ with a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, and an unknown deterministic transition model $f : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$. Let $s_0 \in \mathcal{S}$ be the initial state of the MDP.

For example, for a quadrotor, the states in $\mathcal{S}$ might be the quadrotor's pitch, yaw and roll, the quadrotor's angular and linear velocities, and its height from the ground, all concatenated together to form a vector in $\mathbb{R}^{k_1}$. The actions in $\mathcal{A}$ could be the fixed rotation speeds of the rotors (a vector in $\mathbb{R}^{k_2}$), and the transition function f would apply those speeds to the rotors over a fixed interval of duration dt.

The algorithm we will develop is applicable to finite state and action spaces. When $\mathcal{S}$ or $\mathcal{A}$ are fundamentally continuous, our algorithm can be applied to finite, fine-grained discretizations of those space. We show in Section IV how to handle this discretization.

We now make a few definitions that will help us reason about our knowledge of the environment and our ability to take a sequence of actions to efficiently explore the environment.

Definition 1: We denote knowledge the actor has about the transition function f as a set $\mathcal{T} \subset \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ such that the following implication relation holds:

$$(s, a, s') \in \mathcal{T} \implies f(s, a) = s' \tag{1}$$

Definition 2: Let A, with appropriate superscripts when necessary, denote a sequence of actions; A_i the ith action in the list and $|A|$ the cardinality. We overload the transition function f such that $f(s, A) = s'$ if and only if taking the actions $A_1, A_2, \ldots, A_{|A|}$ sequentially from state s yields the final state s'. Lastly, $\mathbb{A}$ denotes the set of all possible ordered sequences of actions A.

Without further assumptions about f, it is not possible to perform safe exploration from s_0, because we cannot take any action from the initial state with limited knowledge of the action's safety. Therefore, we assume we are given an initial safe set $S_0 \subseteq \mathcal{S}$ such that the initial state of the system is in this set, i.e. $s_0 \in S_0$. Furthermore, we assume as in [19] that for any $s, s' \in S_0$, we are given a list of actions A such that $f(s, A) = s'$.

Assumption 2: Formally, we assume we are given an initial safe set $S_0 \subseteq \mathcal{S}$ such that $s_0 \in S_0$, an initial knowledge set $\mathcal{T}_0$, and restate the above assumption as:

$$\forall s, s' \in S_0, \exists k \in \mathbb{Z}_{\geq 0}, \exists a_1, \ldots, a_k \in \mathcal{A}, \exists s_1, \ldots, s_{k-1} \in \mathcal{S}$$
$$(s, a_1, s_1), (s_1, a_2, s_2), \ldots, (s_{k-1}, a_k, s') \in \mathcal{T}_0$$

We further make a set of assumptions on Lipschitz continuity of the transition function to enable safe exploration.

Assumption 3:
- $f(s, a)$ is L_s-Lipschitz continuous over the states with some distance metric $d^s : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$:

$$d^s(f(s, a), f(s', a)) \leq L_s d^s(s, s') \tag{2}$$

- Similarly, $f(s, a)$ is L_a-Lipschitz continuous over the actions with d^s and the additional distance metric d^a :

$\mathcal{A} \times \mathcal{A} \to \mathbb{R}$:

$$d^s(f(s, a), f(s, a')) \leq L_a d^a(a, a') \tag{3}$$

Practically, the Euclidean distance is often used for both d^s and d^a.

Note that these requirements are mild and naturally satisfied in most domains: If we take the same action from two similar states, we will end up in similar states; and if we take similar actions from the same state, we will again end up in similar states. For this algorithm, we assume that L_s and L_a have been estimated via some prior methodology, and we note that larger than optimal values of these constants, while leading to less efficient algorithms, still satisfy Assumption 3.

With this setup, we can now define our notion of safety.

Definition 3: We define $s \in \mathcal{S}$ *to be safe with respect to* S_0 and knowledge set $\mathcal{T}$ if there exists a recovery algorithm that can use the information in $\mathcal{T}$ and the Lipschitz assumptions to confidently produce actions which will transition the MDP from s into S_0 after a finite number of steps. We define that action a *is safe at state s* if all possible outcomes of $f(s, a)$, with respect to $\mathcal{T}$ and Lipschitz assumptions, are safe. When calling a set safe, the choice of S_0 should be clear from context.

For instance, in our quadrotor example, S_0 might include various hovering states, and thus we consider any state to be safe if we can return to a hovering position from that state. We note that our notion of safety is similar to the safety definition of [23].

This definition of safety, while theoretically satisfying, is not computable in its current form, as we have not described what it means to "use information to confidently produce actions". With a bit more work, we will do so in Definition 7 below.

We now state some important observations about our definition of safety.

Remark 1: Suppose S is a safe set with respect to some knowledge set $\mathcal{T}$. Then:

$$S \subseteq \bar{S} := \{ s \in \mathcal{S} | \exists A \in \mathbb{A} \text{ s.t. } f(s, A) \in S_0 \} \tag{4}$$

However, this upper bound on the safe set cannot normally be calculated, as f is unknown.

Remark 2: Suppose S is safe with respect to some $\mathcal{T}$, and that $\mathcal{T} \subseteq \mathcal{T}'$. Then S is safe with respect to $\mathcal{T}'$.

B. Computing the Safe Set

We now utilize the knowledge set $\mathcal{T}$ to determine which states and actions are safe.

Definition 4: In order to handle unknown states, we define an uncertain transition function f_u, parameterized by knowledge $\mathcal{T}$, that maps each state-action pair to all of its possible outcomes:

$$f_u(s, a; \mathcal{T}) := \bigcap_{(s', a', s'') \in \mathcal{T}} \phi(s'', L_s d^s(s', s) + L_a d^a(a', a))$$

where $\phi(s, x) := \{ s' \in \mathcal{S} | d^s(s, s') \leq x \}$ denotes the hypersphere centered at s with radius x over the distance metric d^s. A visualization of f_u-function is shown in Fig. 1.
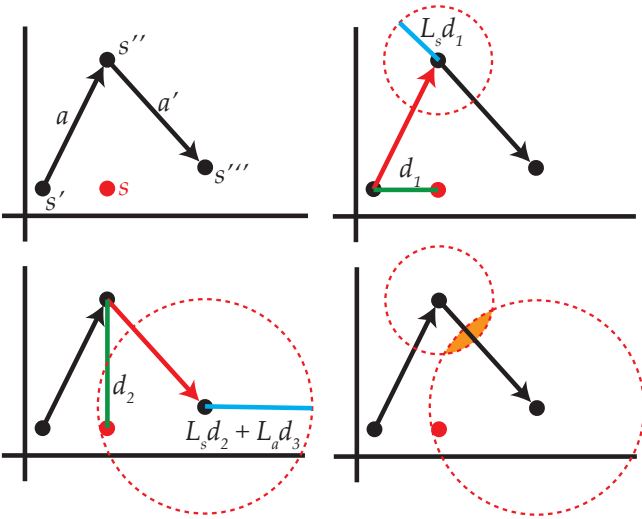
Fig. 1. A visualization of Lipschitz assumption and f_u-function has been shown. **Top left.** We are given two state transitions: $f(s', a) = s''$ and $f(s'', a') = s'''$. We want to know the possible outcomes when we take action a from state s. **Top right.** Considering the first transition, the outcome of $f(s, a)$ must be within a circle of radius $L_s d_1$ from s'', where $d_1 = d_s(s', s)$. **Bottom left.** Similarly, when we consider the second transition, we note that the outcome must be within a circle of radius $L_s d_2 + L_a d_3$ from s''', where $d_2 = d_s(s'', s)$ and $d_3 = d_a(a, a')$. **Bottom right.** The states that satisfy both of those conditions lie on the intersection, which is shown as the shaded area. It is exactly equal to $f_u(s, a; \mathcal{T})$ where $\mathcal{T} = \{(s', a, s''), (s'', a', s''')\}$.

Definition 5: If S is a safe set with respect to knowledge $\mathcal{T}$, then we define the expansion function as:

$$E_{\mathcal{T}}(S) := S \cup \{s \in \mathcal{S} | \exists a \in \mathcal{A} \text{ s.t. } f_u(s, a; \mathcal{T}) \subseteq S\} \quad (5)$$

We also let $E_{\mathcal{T}}^{(k)}(S) = E_{\mathcal{T}}(E_{\mathcal{T}}^{(k-1)}(S))$ with $E_{\mathcal{T}}^{(1)}(S) = E_{\mathcal{T}}(S)$. Moreover, we denote $\bar{E}_{\mathcal{T}}(S) = \lim_{k \to \infty} E_{\mathcal{T}}^{(k)}(S)$ to be the fixed point of the expansion function.

Using these definitions, we make the following crucial observation that is similar to Eq. (4):

Theorem 1: If S is safe with respect to $\mathcal{T}$, then so is $E_{\mathcal{T}}(S)$.

Proof: Let $s \in E_{\mathcal{T}}(S)$. Then, at least one of the following is true:

- $s \in S$
- $\exists a \in A$ such that $f_u(s, a; \mathcal{T}) \subseteq S$

If $s \in S$, then since S is safe with respect to $\mathcal{T}$, so is s. Otherwise, since there exists an a such that $f_u(s, a; \mathcal{T}) \subseteq S$, even if $f(s, a)$ is still unknown, we do know $f(s, a) \in S$, which is safe. From there we can use the recovery algorithm provided by Definition 3 for S to return to S_0. ∎

Definition 6: If we take action a_i from the state s_i at some time step $i \geq 0$, then we recursively define our knowledge after taking step i to be $\mathcal{T}_{i+1} = \mathcal{T}_i \cup \{(s_i, a_i, f(s_i, a_i))\}$.

Definition 7: For timestep $i \geq 0$, recursively define the safe set after step i to be

$$S_{i+1} = \bar{E}_{\mathcal{T}_i}(S_i) \quad (6)$$

Using Theorem 1, we see that this definition is justified, i.e. S_{i+1} is safe with respect to $\mathcal{T}_i$ for all i. We now have a computable set which we can pair with our more theoretical definition of safety, Definition 3.

Remark 3: At step i, a state s is safe if and only if $\exists a \in \mathcal{A}$ such that the action a is safe at s.

We are now ready to state our goal:

Problem 1: Our goal is to maximize the rate of safe exploration over the number of actions taken, given an initial safe set S_0 and state $s_0 \in S_0$. Formally,

$$\max_{a_1, a_2, \ldots, a_m} \frac{|S_m|}{m}$$
$$\text{subject to } s_{k+1} = f(s_k, a_{k+1}) \in S_k \subseteq \bar{S} \subseteq \mathcal{S},$$
$$a_{k+1} \in \mathcal{A}, \text{ for } k = 0, 1, \ldots, m-1 \quad (7)$$

where m is the total number of actions taken, and S_i is the set of states deterministically known to be safe by the algorithm after i actions, and $\bar{S}$ is as defined in Eq. (4).

III. ALGORITHM

In order to present our algorithm that efficiently expands the safe set, we first introduce some notations and functions.

A. Preliminaries

Definition 8: We define the *path-knowledge* function $q : \mathcal{S} \times \mathcal{S} \to \{0, 1\}$ parametrized with $\mathcal{T}$ that carries the transition triplets:

$$q(s, s'; \mathcal{T}) = \begin{cases} 1 & \text{if } \exists A : (s, A_1, f(s, A_1)), \\ & (f(s, A_1), A_2, f(f(s, A_1), A_2)), \cdots \in \mathcal{T} \\ & \wedge f(s, A) = s' \\ 0 & \text{otherwise} \end{cases}$$

That is, $q(s, s'; \mathcal{T}) = 1$ if and only if there exist triplets in $\mathcal{T}$ that let us move from s to s', and $q(s, s'; \mathcal{T}) = 0$ otherwise.

While performing safe exploration, it is both desirable and useful to learn the transition function. While $\mathcal{T}$ implicitly performs this, it is useful to denote it as a function.

Definition 9: We therefore denote the transitions that have been learnt with certainty (without ambiguity) as function $f_c : \mathcal{S} \times \mathcal{A} \to \mathcal{S} \cup \{\gamma\}$:

$$f_c(s, a; \mathcal{T}) := \begin{cases} s' & \text{if } (s, a, s') \in \mathcal{T} \\ \gamma & \text{otherwise} \end{cases}$$

where γ is a placeholder that represents "an unknown state". Since the MDP is deterministic, f_c is a properly defined function, i.e.

$$(s, a, s') \wedge (s, a, s'') \in \mathcal{T}_i \implies s' = s''$$

We also overload the f_c function as $\mathcal{S} \times \mathbb{A} \to \mathcal{S} \cup \{\gamma\}$ similar to f.

Definition 10: We define the *path-planning* function $g : \mathcal{S} \times \mathcal{S} \to \mathbb{A}$:

$$g(s, s'; \mathcal{T}) := \begin{cases} \arg\min_{A: f_c(s, A; \mathcal{T}) = s'} |A| & \text{if } q(s, s'; \mathcal{T}) = 1 \\ \emptyset & \text{otherwise} \end{cases}$$

That is, $g(s, s'; \mathcal{T})$ gives the smallest list of actions that moves from s to s' if such a transition is known in $\mathcal{T}$. In the case that there exist several such sequences, we assume $g(s, s'; \mathcal{T})$ gives any one of them.

Remark 4: Due to the assumption that we know how to move from one state to another inside the initial safe set, we have the following two relations for $\forall i$:

$$s, s' \in S_0 \implies q(s, s'; \mathcal{T}_i) = 1$$
$$s, s' \in S_0, s \neq s' \implies g(s, s'; \mathcal{T}_i) \neq \emptyset$$

B. Efficient Exploration

In order to efficiently expand the safe set, we must optimize for the list of actions that will lead to the largest safe set expansion with minimum number of actions. We will do this in two steps:

1) We first define a measure that corresponds to the possible safe set expansion from taking a new action.
2) We then greedily optimize to take actions which are considered efficient under that measure.

The most straightforward approach for defining a measure that corresponds to safe set expansion is to measure the amount of safe set expansion for each possible outcome of an action, and compute an expected value over all such possibilities. We call this the safe set expansion measure.

However, as we will practically demonstrate in our results, this approach can be highly suboptimal especially for continuous dynamics. Instead, we develop a second measure that quantifies the uncertainty reduction on the outcomes of all state-action pairs by taking an action. This prioritizes exploration towards the safe set boundary in addition to exploring actions which will expand the boundary.

Definition 11: We define the uncertainty reduction measure:

$$\Delta(s, a; \mathcal{T}_i) :=$$
$$\sum_{s'' \in U} p(s''; U) \sum_{\{s', a'\}} |f_u(s', a'; \mathcal{T}_i) \setminus f_u(s', a'; \mathcal{T}_i \cup \{s, a, s''\})|$$

where the inner summation is over all state-action pairs, $U = f_u(s, a; \mathcal{T}_i)$, and $p(s''; U)$ is the modeled probability that action a will move from state s to s''.

While the underlying MDP is deterministic, for some actions we only know that they will end up in a specific set of points, thus it makes sense to model our uncertainty as a probability distribution over that set. While different probability models can be employed for $p(s''; U)$, we are going to use uniform distribution over U for simplicity.

To show that our measure is well defined, we have the following theorem.

Theorem 2: The uncertainty does not increase when some more actions are taken. In fact, the updated uncertainty is a subset of the previous one. Formally, for all $s \in \mathcal{S}, a \in \mathcal{A}$, we have $\mathcal{T}_i \subseteq \mathcal{T}_j \implies f_u(s, a; \mathcal{T}_i) \supseteq f_u(s, a; \mathcal{T}_j)$ and thus $\Delta(s, a; \mathcal{T}_i) \geq 0$.

Proof: $f_u(s, a; \mathcal{T}_j) = f_u(s, a; \mathcal{T}_i) \cap f_u(s, a; \mathcal{T}_j \setminus \mathcal{T}_i) \subseteq f_u(s, a; \mathcal{T}_i)$ for all s and a. ∎

By Theorem 2, we can write:

$$\Delta(s, a; \mathcal{T}_i) =$$
$$\sum_{s'' \in U} p(s''; U) \sum_{\{s', a'\}} |f_u(s', a'; \mathcal{T}_i)| - |f_u(s', a'; \mathcal{T}_i \cup \{s, a, s''\})|$$

Our goal is now to take steps to maximize the reduction in uncertainty. We realize that it is computationally difficult to maximize this reduction over sequences of multiple uncertain actions, so instead we will greedily optimize for the best immediate reduction in uncertainty. However, we still cannot directly maximize $\Delta(s, a; \mathcal{T}_i)$ over all s and a, because at any step i we are at a fixed state s_i and getting to state s may require many intermediate actions. Thus we must optimize over paths of actions $A \in \mathbb{A}$ starting at s_i. A second realization is that though it may be more efficient in some circumstances to try to get near a state s via paths containing uncertain actions, it is difficult to optimize over such paths, thus we only consider paths to s which consist of certain actions.

The following theorem will be useful for our algorithm:

Theorem 3: Taking actions that are already in the collection of knowledge, $\mathcal{T}_i$, does not lead to any safe set expansion.

Proof: We first note that the update rule depends only on the uncertainty function f_u, which depends on s, a and $\mathcal{T}_i$. Suppose $(s, a, s') \in \mathcal{T}_i$. When we take the action a from state s, we do not add a new element to $\mathcal{T}_i$. Since $\mathcal{T}_i$ does not change, there will be no change in $f_u(s, a; \mathcal{T}_i)$ for any s and a, so we will not be able to expand the safe set. ∎

Due to Theorem 3, if we take a path of certain actions from s_i to s and then take an uncertain action at s, we know that the entire safe set expansion will come from the last uncertain step. Thus, we wish to perform the following optimization:

$$\max_{A \in \mathbb{A}, a \in \mathcal{A}} \quad \frac{1}{|A| + 1} \Delta(s, a; \mathcal{T}_i)$$
$$\text{subject to} \quad s = f_c(s_i, A; \mathcal{T}_i),$$
$$f_u(s, a; \mathcal{T}_i) \subseteq S_{i+1} \tag{8}$$

Again due to Theorem 3, if two different action lists lead to the same state from state s_i, then the one with smaller cardinality is better off in the optimization. This means, we can just optimize over the shortest paths between state s_i and the optimization variable state s. The optimization can then be reformulated as follows:

$$\max_{s \in \mathcal{S}, a \in \mathcal{A}} \quad \frac{1}{|g(s_i, s; \mathcal{T}_i)| + 1} \Delta(s, a; \mathcal{T}_i)$$
$$\text{subject to} \quad q(s_i, s; \mathcal{T}_i) = 1,$$
$$f_u(s, a; \mathcal{T}_i) \subseteq S_{i+1} \tag{9}$$

This optimization is over finite variables for finite discrete MDPs.

C. Overall Algorithm

We first present the pseudocodes for the algorithm blocks that we so far formalized. Algorithm 1 is the pseudocode for computing $\bar{E}_{\mathcal{T}_i}(S_i)$, which corresponds to the procedure that we use for expanding the safe set at iteration i.

In order to perform the optimization for efficient exploration, we compute expected total uncertainty reduction as in Algorithm 2. Then, Algorithm 3 benefits from that procedure to optimize for uncertainty reduction. Note that

Algorithm 1 Safe Set Expansion
1: **procedure** EXPANDSAFESET($S_i, \mathcal{T}_i, f_u$)
2:     $S_{i+1} \leftarrow S_i$
3:     **while** S_{i+1} not converged **do**
4:         $S_{i+1} \leftarrow S_{i+1} \cup \{s | f_u(s,a;\mathcal{T}_i) \subseteq S_{i+1} \text{ for } \exists a \in \mathcal{A}\}$
5:     **return** S_{i+1}

---

in Algorithm 2, we use the scaling factor of $\frac{1}{|U|}$ in line 7, since we are using a uniform distribution over the possible outcomes of an action.

Algorithm 2 Expected Uncertainty Reduction Computation
1: **procedure** EXPECTEDREDUCTION(f_u, s, a, L_s, L_a)
2:     $U \leftarrow f_u(s,a;\mathcal{T}_i)$
3:     $v \leftarrow 0$
4:     **for** $s'' \in U$ **do**
5:         $f_u'(s',a';\mathcal{T}_i) \leftarrow f_u(s',a';\mathcal{T}_i) \cap \phi(s'', L_s d^s(s,s') + L_a d^a(a,a'))$ ($\forall s' \in \mathcal{S}, \forall a' \in \mathcal{A}$)
6:         $\Delta(s',a') \leftarrow |f_u(s',a';\mathcal{T}_i)| - |f_u'(s',a';\mathcal{T}_i)|$ ($\forall s' \in \mathcal{S}, \forall a' \in \mathcal{A}$)
7:         $v \leftarrow v + \frac{1}{|U|} \sum_{s' \in \mathcal{S}, a' \in \mathcal{A}} \Delta(s',a')$
8:     **return** v

Algorithm 3 Optimization for Uncertainty Reduction
1: **procedure** OPTIMIZEGREEDILY($S_{i+1}, s_i, \mathcal{T}_i, f_u, L_s, L_a$)
2:     $V \leftarrow 0$
3:     **for** $s \in S_{i+1}$ **do**
4:         $G \leftarrow g(s_i, s; \mathcal{T}_i)$ ▷ Shortest-path algorithm
5:         **if** $G = \emptyset$ and $s_i \neq s$ **then**
6:             **continue**
7:         **for** $a \in \mathcal{A}$ **do**
8:             $v \leftarrow$ EXPECTEDREDUCTION(f_u, s, a, L_s, L_a)
9:             $v \leftarrow v/(|G| + 1)$
10:           **if** $v > V$ **then**
11:               $V \leftarrow v$
12:               $G^*, a^* \leftarrow G, a$
13:     **return** G^*, a^*

Lastly, we note that it is possible to have state-action pairs that are not in the current knowledge set, but have only one possible outcome. Adding these pairs to the knowledge in each iteration can possibly increase efficiency. We present this procedure in Algorithm 4. Note we check if $|U| = 1$ for discrete MDPs. For continuous MDPs, where we can sample state and action spaces as we will describe in Section IV, we check if U is a singleton.

We present the complete algorithm as a pseudocode in Algorithm 5.

IV. SIMULATIONS AND RESULTS

We call the algorithm developed above **Safe Exploration Optimized For Uncertainty Reduction**. We developed the following alternative methods as baselines to compare our algorithm against:

Algorithm 4 Knowledge Expansion (Discrete MDPs)
1: **procedure** EXPANDKNOWLEDGE($\mathcal{T}_i, f_u$)
2:     **for** $s \in \mathcal{S}$ **do**
3:         **for** $a \in \mathcal{A}$ **do**
4:             $U \leftarrow f_u(s,a;\mathcal{T}_i)$ ▷ U_1 is the first element
5:             **if** $|U| = 1$ **then**
6:                $\mathcal{T}_i \leftarrow \mathcal{T}_i \cup \{(s,a,U_1)\}$
7:     **return** $\mathcal{T}_i$

Algorithm 5 Efficient and Safe Exploration (Discrete MDPs)
Require: n ▷ Iteration count
Require: L_s, L_a ▷ Lipschitz-continuity constants
Require: $S_0 \subseteq \mathcal{S}$ ▷ Initial safe set
Require: $\mathcal{T}_0 \subset \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ ▷ Initial knowledge
Require: $s_0 \in S_0$ ▷ Initial state
1: Compute $f_u(s,a;\mathcal{T}_0)$ for $\forall s \in \mathcal{S}, \forall a \in \mathcal{A}$
2: **for** $i \leftarrow 0$ to $n-1$ **do**
3:     $\mathcal{T}_i \leftarrow$ EXPANDKNOWLEDGE($\mathcal{T}_i, f_u$)
4:     $S_{i+1} \leftarrow$ EXPANDSAFESET($S_i, \mathcal{T}_i, f_u$)
5:     $G^*, a^* \leftarrow$ OPTIMIZEGREEDILY($S_{i+1}, s_i, \mathcal{T}_i, f_u, L_s, L_a$)
6:     $s^* \leftarrow f(s_i, G^*)$ ▷ Take the certain actions
7:     $s_{i+1} \leftarrow f(s^*, a^*)$ ▷ Take the uncertain action
8:     $\mathcal{T}_{i+1} \leftarrow \mathcal{T}_i \cup (s^*, a^*, s_{i+1})$
9:     $f_u(s,a;\mathcal{T}_{i+1}) \leftarrow f_u(s,a;\mathcal{T}_i) \cap \phi(s_{i+1}, L_s d^s(s,s_{i+1}) + L_a d^a(a,a^*))$ ($\forall s \in \mathcal{S}, \forall a \in \mathcal{A}$)

Random Exploration. In this method, we perform the safe set expansion as in our algorithm. However, each action taken is chosen randomly from the set of all possible actions, and is not necessarily safe.

Safe Exploration with No Optimization. In this method, we again perform the safe set expansion as in our algorithm. However, each action taken is chosen randomly from the set of actions that are classified as safe at the current state.

Safe Exploration Optimized for Safe Set Expansion. This is similar to Safe Exploration Optimized for Uncertainty Reduction. However, instead of optimizing for the uncertainty reduction measure, we optimize for the safe set expansion measure described earlier. If the maximum expected safe set expansion amount is zero, we take the safe action that will, expectedly, push the system to its closest safety boundary at that time, so that it can possibly expand the safe set later.

We simulated two different environments with continuous state and action spaces, described below, to analyze the performance of our algorithm. In each environment, we used Euclidean distance for both d^s and d^a. We used *breadth-first search (BFS)* for the g function.

For both environments, we began by uniformly sampling the state and action spaces. We used only those original samples when calculating the Δ-function, and not any new states we might have encountered since starting the simulation, so that the optimization would not be biased towards already visited states.

To quantitatively assess the performance of our algorithm in comparison with the baselines, we defined and used the

following metrics:

- **Safe Set Size**. We plot the number of actions vs. $|S_i|$ to evaluate the safe set expansion efficiency.
- **Total Uncertainty.** We plot the number of actions vs. $\sum_{s\in\mathcal{S},a\in\mathcal{A}}|f_u(s,a;\mathcal{T}_i)|$ to analyze how fast the total uncertainty decreases. For consistency among the iterations, we sum only over states in the original sampling, i.e. we do not consider other states encountered since starting the simulation.

A. Muddy Jumper

We simulated a simple system with the transition model:

$$f(s,a) = s + a(1 - \psi(s))$$

where $a \in \mathcal{A} = [-C, C]$, $s \in \mathcal{S} = \mathbb{R}$, and $\psi(s)$ is the dampening factor. We simulated it as:

$$\psi(s) = \begin{cases} 0 & \text{if } |s| < A \\ \frac{|s|-A}{B-A} & \text{if } A \leq |s| < B \\ 1 & \text{otherwise} \end{cases}$$

which is plotted in Fig. 2. This environment was inspired by the idea of a robot jumping on muddy ground. When the dampening factor is 1, the robot is not able to move anymore, so those states are unsafe.
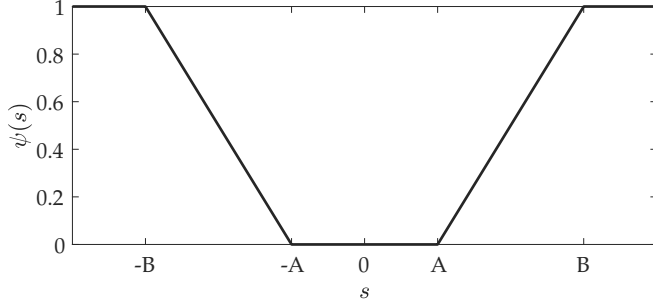

Fig. 2. Dampening factor for the Muddy Jumper environment. The regions that are inside $[-A, A]$ are not muddy, so the robot can move without any dampening. The outer regions have increasing amount of mud, which causes the robot's movements to be dampened. The robot becomes completely immobile outside $(-B, B)$.

With these, we can take $L_a = 1$ and $L_s = \frac{C+B-A}{B-A}$. For our simulations, we used $A = 3$, $B = 9$ and $C = 12$. We sample $\mathcal{S}$ as $\{-10, -9.8, \ldots, 9.8, 10\}$ and $\mathcal{A}$ as $\{-12, -11.8, \ldots, 11.8, 12\}$. Hence, the largest safe set is the interval between -8.8 and 8.8. We set $S_0 = [-3, 3]$ and $s_0 = 0$. We present the results of our algorithm in Fig. 3.

It can be seen that random exploration leads to fast uncertainty reduction, because the allowed actions include very large jumps. However more importantly, it terminates upon reaching unsafe states after only a few actions. Other exploration techniques tackle this problem by taking only safe actions. However, our algorithm outperforms the baselines in terms of efficiency. It is better than the safe exploration with no optimization as it optimizes the uncertainty reduction per action. It can be seen that optimizing for safe set expansion yields good results initially; however becomes highly suboptimal afterwards. In this case, the superiority of our algorithm can be due to that reducing overall uncertainty leads to larger
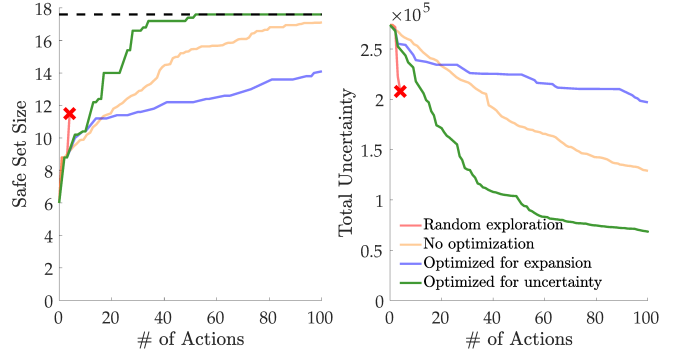

Fig. 3. Results of Muddy Jumper environment with our algorithm and the three other baselines. For safe exploration with no optimization, 10 runs have been averaged. The crosses at the end of random exploration lines indicate the system reached to an unsafe state. **Left.** Number of actions vs. safe set size. Dashed line shows the size of largest possible safe set. **Right.** Number of actions vs. total uncertainty.

safe set expansion in future iterations, whereas a greedy optimization for the expansion cannot achieve this. As a side note, it is also interesting to see that the safe exploration with no optimization outperformed the safe exploration optimized for safe set expansion in later iterations. This might be because optimizing for expansion may often get stuck at states that are not amongst the original state samples, so the optimization becomes only over the immediate next action.

B. Hilly Jumper

We simulate another environment with the following transition model:

$$f(s,a) = s + a - h'(s)$$

where $a \in \mathcal{A} = [-C, C]$, $s \in \mathcal{S} = \mathbb{R}$, and $h(s)$ is an environment-dependent function. We simulated it as:

$$h(s) = \begin{cases} -\frac{(s+A)^4}{4B^4} + \frac{(s+A)^2}{2B^2} & \text{if } s < -A \\ 0 & \text{if } -A \leq s < A \\ -\frac{(s-A)^4}{4B^4} + \frac{(s-A)^2}{2B^2} & \text{otherwise} \end{cases}$$

Then, we have

$$h'(s) = \begin{cases} -\frac{(s+A)^3}{B^4} + \frac{s+A}{B^2} & \text{if } s < -A \\ 0 & \text{if } -A \leq s < A \\ -\frac{(s-A)^3}{B^4} + \frac{s-A}{B^2} & \text{otherwise} \end{cases}$$

This environment was inspired by a robot jumping on hilly ground where $h(s)$ is the elevation function. Both the elevation function and its derivative are plotted in Fig. 4. When $|s| > A + B$ and $C < |h'(s)|$, the robot is on a very steep terrain, so it cannot return to the safe set, which is around 0-state.

With these, we can take $L_a = 1$. While this problem does not have a single Lipschitz-continuity constant over all states, as the slope increases without bounds in each direction, it locally satisfies Lipschitz-continuity around the central region. For our simulations, we used $A = 1.2$, $B = 4$, $C = 0.3$ and $L_s = 1.4$. We sample $\mathcal{S}$ as $\{-6.9, -6.8, \ldots, 6.8, 6.9\}$ and $\mathcal{A}$ as $\{-0.3, -0.2, \ldots, 0.2, 0.3\}$. Hence, the largest safe set is
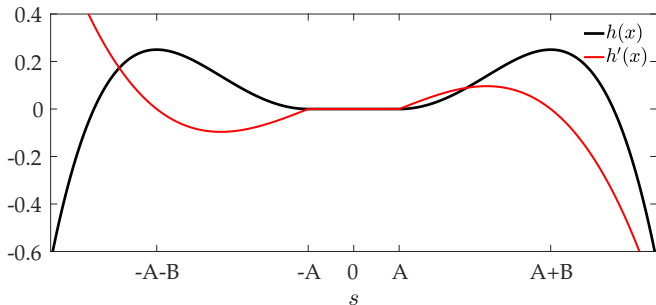
Fig. 4. Elevation function and its derivative are plotted for the Hilly Jumper environment. The robot has a constant maximum power, which prevents it from returning to the central region when it reaches points which are too steep.
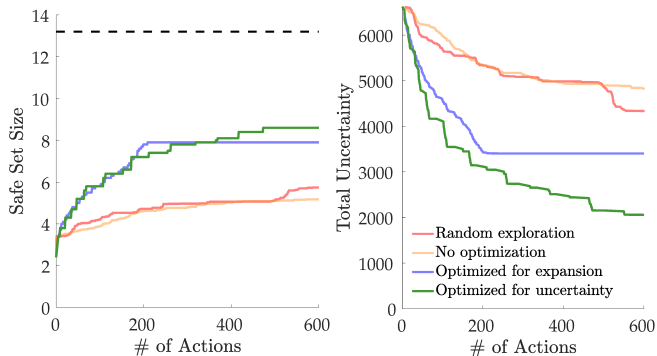

Fig. 5. Results of Hilly Jumper environment with our algorithm and the three other baselines. For both random exploration and safe exploration with no optimization methods, 10 runs have been averaged. **Left.** Number of actions vs. safe set size. Dashed line shows the size of largest possible safe set. **Right.** Number of actions vs. total uncertainty.

the interval between -6.6 and 6.6. We set $S_0 = [-1.2, 1.2]$ and $s_0 = 0$. We present the results of our algorithm in Fig. 5.

Unlike the Muddy Jumper environment, random exploration does not lead to very fast uncertainty reduction, because the action set contains only very short jumps. Due to the same reason, it does not crash —it is unlikely for the robot to leave the safe set with random small jumps. It is even harder due to the fact that the regions $A < |s| < A+B$ have a slope towards the central region. Safe exploration with no optimization performs even worse than random exploration, because it enforces safety constraint.

Similar to Muddy Jumper experiments, the baseline which optimizes over safe set expansion initially gives good performance; and later becomes suboptimal. In this case, the reason is the following: After the algorithm expands the safe set up to -6.2 on the negative side, it cannot expand further due to the sampling of state space. It also could not expand the positive side further from 1.7, because the algorithm gets stuck near the negative limit. This is because when the system is at a new state near -6.2, all actions have some level of uncertainty, so the optimization is only over the immediate actions. And when the system is near the limit, it does not leave that region; because if it finds some possibility of safe set expansion, it explores; if it cannot find a possible expansion, it still goes toward the boundary. In fact, we ran this algorithm up to 2000 actions, and observed

that the system was always in states $[6.10, 6.20]$ after 220th action. This baseline would require the following additional mechanisms to perform well: 1) to spot when to be confident that there is no possibility of expansion, and 2) to make sure the system moves to unexplored state regions when that confidence is obtained.

On the other hand, our algorithm, safe exploration optimized for uncertainty reduction, outperforms all the baseline methods in terms of efficiency. It can be noted that none of the algorithms reaches the maximum expandable safe set within 600 actions. While the explored safe set might be expanded further with more iterations, it is also limited due to the sampling of state space —denser sampling could increase this limit; however it causes a significant computational burden.

V. DISCUSSION

For safe exploration tasks, computational cost is a problem in general. Our algorithm has polynomial complexity in the number of states and actions. While the use of Gaussian processes enables faster computation, directly using Lipschitz-continuity makes the algorithm computationally heavier, though we do note that our algorithm is parallelizable. For reference, we initially sample 101 states, 121 possible actions in Muddy Jumper; and 139 states, 7 possible actions in Hilly Jumper. However, the number of states increases during algorithm execution as the system visits states that are not amongst the initial samples. Additionally, it can be a concern for low-memory systems that our framework requires the storage of $\mathcal{T}_i$ that grows linearly with the number of uncertain actions taken.

Both our example environments had 1-dimensional state spaces. We note that in higher dimensional problems, the number of states necessarily grows exponentially in the dimension of state space. In particular, the number of states on the boundary of the safe set at any step is likewise exponential in the dimension of the state space. Since our algorithm does not extrapolate from data in order to produce its safety guarantees, it by design must explore this exponentially sized safe set boundary.

For some specific applications, our algorithm's requirement that how to move from one state to another is known inside the initial safe set can be too restrictive. In such cases, our algorithm can be readily applied provided that there exist some uncertain but safe actions for each state in the initial safe set. While this may hurt efficiency, it will enable the use of our algorithm in broader configurations.

In this formulation, our algorithm is limited to deterministic environments. Further research could generalize it to stochastic MDPs and to systems with disturbances. Similarly, our framework requires prior knowledge of the Lipschitz continuity parameters L_s and L_a. In settings where it is impractical to provide estimates of these parameters prior to running this algorithm, this algorithm could be modified to learn them online. However, either of these generalization would come at the expense of losing the algorithm's deterministic guarantees.

As long as Lipschitz-continuity assumptions can be made, our algorithm can be applied to both linear and nonlinear systems, as well as to systems where safe state set boundaries are very complex. We have demonstrated our algorithm on two simulated environments, and we are planning to design real robotics experiments to showcase our algorithm.

Lastly, in each iteration of our algorithm, we currently take only actions we are certain about before taking a final, uncertain, action that we learn from (see the first constraint in (9)). This algorithm could potentially be improved by having it optimize over and learn from paths that include several uncertain actions in sequence rather than just one.

VI. Conclusion

In this paper, we presented an algorithm to safely explore safety-critical deterministic MDPs that is efficient in terms of the number of actions it takes. Unlike some previous works, our algorithm does not require the transition function to be known a priori, except for some little prior knowledge.

Future work will demonstrate our algorithm's use in practice. In addition, future work can be done to further improve the efficiency of our algorithm by allowing it to plan along sequences of multiple uncertain actions. We are also planning to relax the determinism requirement on the MDP and apply our algorithm to stochastic environments. And lastly, exploration is needed into combining this Lipschitz grounded approach with model-based approaches to handle higher dimensional state and action spaces.

Finally, we will study different methods for transferring from a source (e.g., simulation) domain to a target (e.g., real-world) domain. In order for a robotic system to adapt to a new domain, the system must often explore the parameters of the new environment, but must also do so safely. In the future work, we will leverage our work on safe exploration in MDPs and the Delaunay-based optimization [25] to address this problem.

References

[1] D. Sadigh and A. Kapoor, "Safe control under uncertainty with probabilistic signal temporal logic," in *Proceedings of Robotics: Science and Systems (RSS)*, June 2016.

[2] D. Dey, D. Sadigh, and A. Kapoor, "Fast safe mission plans for autonomous vehicles," Proceedings of Robotics: Science and Systems Workshop, Tech. Rep., June 2016.

[3] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient smt solver for verifying deep neural networks," in *International Conference on Computer Aided Verification*. Springer, 2017, pp. 97–117.

[4] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, 2009.

[5] J. Kober and J. Peters, "Reinforcement learning in robotics: A survey," in *Reinforcement Learning*. Springer, 2012, pp. 579–610.

[6] S. P. Coraluppi and S. I. Marcus, "Risk-sensitive and minimax control of discrete-time, finite-state markov decision processes," *Automatica*, vol. 35, no. 2, pp. 301–309, 1999.

[7] M. Heger, "Consideration of risk in reinforcement learning," in *Machine Learning Proceedings 1994*. Elsevier, 1994, pp. 105–111.

[8] M. Sato, H. Kimura, and S. Kobayashi, "Td algorithm for the variance of return and mean-variance reinforcement learning," *Transactions of the Japanese Society for Artificial Intelligence*, vol. 16, no. 3, pp. 353–362, 2001.

[9] V. S. Borkar, "Q-learning for risk-sensitive control," *Mathematics of operations research*, vol. 27, no. 2, pp. 294–311, 2002.

[10] C. Gaskett, "Reinforcement learning under circumstances beyond its control," *Proceedings of the International Conference on Computational Intelligence for Modelling Control and Automation (CIMCA2003)*, February 2003.

[11] P. Geibel and F. Wysotzki, "Risk-sensitive reinforcement learning applied to control under constraints," *Journal of Artificial Intelligence Research*, vol. 24, pp. 81–108, 2005.

[12] A. Aswani, H. Gonzalez, S. S. Sastry, and C. Tomlin, "Provably safe and robust learning-based model predictive control," *Automatica*, vol. 49, no. 5, pp. 1216–1226, 2013.

[13] P. Abbeel and A. Y. Ng, "Exploration and apprenticeship learning in reinforcement learning," in *Proceedings of the 22nd international conference on Machine learning*. ACM, 2005, pp. 1–8.

[14] P. Abbeel, A. Coates, and A. Y. Ng, "Autonomous helicopter aerobatics through apprenticeship learning," *The International Journal of Robotics Research*, vol. 29, no. 13, pp. 1608–1639, 2010.

[15] F. Berkenkamp, A. Krause, and A. P. Schoellig, "Bayesian optimization with safety constraints: safe and automatic parameter tuning in robotics," *arXiv preprint arXiv:1602.04450*, 2016.

[16] A. D. Ames, X. Xu, J. W. Grizzle, and P. Tabuada, "Control barrier function based quadratic programs for safety critical systems," *IEEE Transactions on Automatic Control*, vol. 62, no. 8, pp. 3861–3876, 2017.

[17] A. K. Akametalu, S. Kaynama, J. F. Fisac, M. N. Zeilinger, J. H. Gillula, and C. J. Tomlin, "Reachability-based safe learning with gaussian processes." in *53rd IEEE Conference on Decision and Control (CDC)*. Citeseer, 2014, pp. 1424–1431.

[18] J. Garcıa and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.

[19] M. Turchetta, F. Berkenkamp, and A. Krause, "Safe exploration in finite markov decision processes with gaussian processes," in *Advances in Neural Information Processing Systems*, 2016, pp. 4312–4320.

[20] 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 (AAAI)*, 2018.

[21] J. H. Gillula and C. J. Tomlin, "Guaranteed safe online learning of a bounded system," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, 2011, pp. 2979–2984.

[22] J. Garcia and F. Fernández, "Safe exploration of state and action spaces in reinforcement learning," *Journal of Artificial Intelligence Research*, vol. 45, pp. 515–564, 2012.

[23] T. M. Moldovan and P. Abbeel, "Safe exploration in markov decision processes," *arXiv preprint arXiv:1205.4810*, 2012.

[24] R. S. Sutton, A. G. Barto, *et al.*, *Reinforcement learning: An introduction*. MIT press, 1998.

[25] S. R. Alimo, P. Beyhaghi, and T. R. Bewley, "Optimization combining derivative-free global exploration with derivative-based local refinement," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE, 2017, pp. 2531–2538.