Certifiable Algorithms for Reinforcement Learning: Safety-Critical and Game-Theoretic Perspectives

Liyuan Zheng

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2021

Reading Committee:

Professor Lillian J. Ratliff, Chair

Professor Baosen Zhang

Professor Kevin Jamieson

Program Authorized to Offer Degree:
Electrical & Computer Engineering

University of Washington

Abstract

Certifiable Algorithms for Reinforcement Learning: Safety-Critical and Game-Theoretic Perspectives

Liyuan Zheng

Chair of the Supervisory Committee:
Professor Lillian J. Ratliff
Electrical & Computer Engineering

Reinforcement learning has seen significant advances over the last decade in simulated or controlled dynamic systems. These successes have lead to interests in deploying learning algorithms in more complex environments such as safety-critical and multi-agent settings. However, in those environments, some important certifications of existing reinforcement learning algorithms including safety constraints satisfaction and convergence guarantees are lacking. Thus, certifiable reinforcement learning algorithms are desired and we introduce our proposed algorithms in this thesis.

First, we tackle the problem on finding reinforcement learning policies for control systems with pre-defined state and action constraints. We propose a new approach, termed Vertex Networks, with guarantees on safety during both the exploration and execution stages, by incorporating the safety constraints into the policy network architecture. Leveraging the geometric property that all points within a convex set can be represented as the convex combination of its vertices, the proposed algorithm first learns the convex combination weights and then uses these weights along with the pre-calculated vertices to output an action. The output action is guaranteed to be safe by construction and numerical examples illustrate that the algorithm outperforms other baseline methods.

Second, we address safe reinforcement learning problem with known transition kernel and

unknown constraints. We leverage Constrained Markov Decision Process, which is a class of stochastic decision problems in which the decision maker must select a policy that satisfies auxiliary cost constraints. We present an algorithm `C-UCRL` and show that it achieves sub-linear regret ($O(T^{\frac{3}{4}}\sqrt{\log(T/\delta)})$) with respect to the reward while satisfying the constraints even while learning with probability $1-\delta$. As an extension to unknown transition kernel setting, we present a lower bound on constraint violation, which proves the inevitability of constraint violation of algorithms in constrained Markov decision process.

Third, we present our work on game-theoretic reinforcement learning, where we formulate actor-critic algorithms as a Stackelberg game. We adopt the game-theoretic viewpoint and model the actor and critic interaction as a two-player general-sum game with a leader-follower structure known as a Stackelberg game. We propose a meta-framework for Stackelberg actor-critic algorithms where the leader player follows the total derivative of its objective instead of the usual individual gradient. From a theoretical standpoint, we develop a policy gradient theorem for the refined update and provide a local convergence guarantee for the Stackelberg actor-critic algorithms to a local Stackelberg equilibrium. From an empirical standpoint, we demonstrate via simple examples that the learning dynamics we study mitigate cycling and accelerate convergence compared to the usual gradient dynamics given cost structures induced by actor-critic formulations.

Finally, we study two-player competitive reinforcement learning problem. In order to let one player take advantage from the competition and benefit from learning with a learning opponent, we adopt the hierarchical Stackelberg game formulation and proposed the novel Stackelberg MADDPG algorithm. We also design and open-source new competitive reinforcement learning benchmark tasks and demonstrate the performance and behavior of our algorithm on them.

The contributions of this thesis are steps towards certifiable reinforcement learning algorithms from safety and game-theoretic perspectives under complex environments.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I spent several wonderful years at the University of Washington and I would like to acknowledge all of the individuals and their contributions in making this thesis possible. I want to express my extreme gratitude to Prof. Lillian Ratliff for her advising on my journey to pursue my Ph.D. degree. I was fortunate to be one of her first students. Her support and open-mindedness provided me with ample flexibility to explore diverse areas of research and identify my own interest. Her advice extends well beyond research as she sets a role model on how to be a great researcher.

I would like to thank the members of my general and final exam committee, Prof. Baosen Zhang, Prof. Kevin Jamieson, and Prof. Behcet Acikmese. The guidance they provided and the discussions we had helped to clarify the problems I focused on and to shape this thesis. Specifically, Baosen has been giving advice to me on several projects throughout my whole Ph.D.. No words are sufficient to express my appreciation for the guidance I received from him.

I have had a wonderful group of collaborators over my graduate study. At the early stage of my Ph.D., I was fortunate to be mentored by Shreyas Sekar and Daniel Calderone, who have provided me with endless advice and have helped me navigate through graduate school to become an independent researcher. Tanner Fiez has been an amazing labmate and friend. His hard-working and enthusiasm for knowledge inspire me a lot and constantly drive me to become better. Yuanyuan Shi has been an incredible collaborator from all aspects. Her innovative thinking and fast learning ability are inspiring. More importantly, she is the one I can discuss anything with, including naive questions and ideas. It has been great to work with Boling Yang, as we have bonded on many things from our research interests

to badminton. The collaboration with Benjamin Chasnov and Zane Alumbaugh has been super positive and productive. Outside of the University of Washington, I have had a great experience collaborating with Prof. Samuel Coogan and his student Esther Ling from the Georgia Institute of Technology.

I would like to acknowledge all the professors and mentors I have taken classes and communicated with, which shaped my understanding in many aspects. I am also grateful to all of the wonderful people I have known and interacted with, including Yize Chen, Yue Sun, Yihan Jiang, Chase Dowling, Mitas Ray, Sarah Li, Kun Su, Jimin Kim, Bowen Xue, Pan Li, Hao Wang, Baicen Xiao, Ling Zhang, Yang Zheng, Xiulong Liu, Yaxuan Zhou, Chen Gong, Shan Lin, Jingchao Liu, Peiyun Zhai, Chen Gang, Yue Wang, Ban Wang, Yu Xiang, Shifeng Zhu, Maolong Tang, Bosong Sun, and many others.

Special thanks to Mengyi Wang and Hupi for their companion of my Ph.D. journey, especially during the COVID-19 pandemic. Last but most importantly, I would like to express my gratitude to my mom, Wenjuan Chen, my dad, Xiang Zheng, my grandma Shufang Yu, for their unconditional love and support.

DEDICATION

To everyone loves and believes in me.♡

Chapter 1

INTRODUCTION

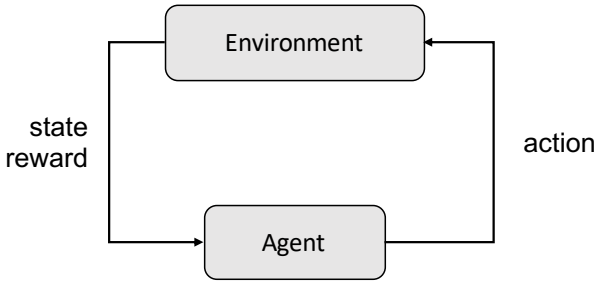


Figure 1.1: Reinforcement learning framework.

1.1 Motivation

Reinforcement learning is a paradigm to learn optimal feedback control strategies by directly interacting with a dynamic system. Over the last couple of years, reinforcement learning algorithms have yielded impressive results on a variety of applications including playing video games like Atari with super-human performance from image observations [1], surpassing the best human players at board games like Go [2]. Besides the impressive and eye-catching advances in games, the data-driven nature of reinforcement learning methods also allows it to be applied directly in the physical world applications, where high-precision simulation or large-scale parallel training is feasible, such as robot locomotion and manipulation [3, 4], and autonomous vehicles [5].

Researchers in this area are certainly confident about the prospective development of reinforcement learning. Recently, Silver et al. hypothesize that intelligence, and its associated abilities, can be understood as subserving the maximization of reward [6]. They suggest that

agents that learn through trial-and-error experience to maximize reward could learn behavior that exhibits most if not all of these abilities, and therefore that powerful reinforcement learning agents could constitute a solution to artificial general intelligence. This statement seems to argue that, reinforcement learning through reward feedback should be the ultimate paradigm to achieve artificial general intelligence. However, by the time of this thesis, we haven't achieved artificial general intelligence or its associated abilities. As stated by Michael I. Jordan, the revolution hasn't happened yet [7].

In fact, at the current stage, there are a lot of limitations and shortcomings in existing reinforcement learning algorithms. The algorithms suffer from high variance. [8] shows that it is possible to get different learning results by averaging different runs with the same hyperparameters, but different random seeds. The algorithms are known to have worse sample complexity comparing to other machine learning mechanisms such as supervised learning [9]. Most algorithms do not have safety constraints satisfaction guarantee when applied to safety-critical environments [10]. And when multiple agents are involved, reinforcement learning algorithms often overlook the interaction between agents and thus have no convergence guarantee [11].

That is, even though reinforcement learning showed significant empirical performance in some applications and domain experts are optimistic about its prospects, there is still a great potential for improvement. In complex environments such as safety-critical and multiple agents settings, reinforcement learning has not shown equivalent successes comparing to well-structured environments, and certifiable reinforcement learning algorithms are desired. This motivates the researches in this thesis.

In general, the goal of reinforcement learning is to learn a policy mapping states to actions, which seeks to maximize the summation of a numerical reward signal. The policy is not learning which actions to take from a batch of labeled data, but instead must discover which actions yield the most reward by trying them. It typically learns to make sequential decisions by interacting with the environment with full freedom, gradually improving its performance at the task as learning progresses. Figure 1.1 shows an abstract framework of

the interaction between reinforcement learning agent and the environment. This trial-and-error nature of reinforcement learning makes it often suffer from high variance and lack of stability and safety when deployed on real-world systems.

Unlike in simulation, in the physical world there are often additional safety constraints, or specifications that lead to constraints, on the learning problem. As a result, any algorithm that is deployed on real-world systems needs to ensure the safety of itself and the environment that it interacts with. For instance, a robot arm should avoid behaviors that could cause it to damage itself or the objects around it, and autonomous vehicles must avoid crashing into others while navigating [10]. In real-world applications such as the above, constraints are an integral part of the problem description, and maintaining constraint satisfaction during learning is critical. That is, they are hard constraints.

There are two main sources of safety constraints in reinforcement learning problems. For the first one, the safety criterion is known and the constrained region is pre-defined. The learner must be restricted within the safety region while exploring the environment during the learning procedure. For the second one, the constraints are unknown a priori. They are received by the learner as cost signals just like the reward, and the constraints must be learned by interacting with the environment. In this case, maintaining constraints satisfaction during learning is very hard and it requires a deliberate tradeoff between exploration and exploitation. In this setting, safe reinforcement learning requires the learner to act conservatively in the beginning. Once the algorithm safely gathers data on the task and learns about the costs associated with constraints, it can start to exploit for higher performance.

On the other hand, reinforcement learning algorithms sometimes involve multiple learners. As a matter of fact, not only in reinforcement learning but also in general machine learning, a number of problems consist of a hybrid of several learners. The learners are rarely acting in isolation within the environment, instead, they are typically in the presence of multiple autonomous agents who may be passing information to each other and optimizing their objectives either competitively or cooperatively. One famous example is the generative adversarial networks [12], and other examples in reinforcement learning include multi-agent

reinforcement learning [13] and actor-critic algorithms [14].

In some reinforcement learning applications, such as actor-critic and competitive multi-agent environment settings, there exist natural hierarchical orders between agents, and game-theoretic reinforcement learning algorithms explicitly considering such orders are desired. The problems involving multiply learner interactions have been formulated as games, where learning algorithms have been developed from a game-theoretic perspective to achieve equilibrium points of the games. Stackelberg game, which is a special kind of game formulation, characterizes the interaction between a leader and a follower. The leader in the game is distinguished by the ability to act before the follower. As a result of this structure, the leader optimizes accounting for how the follower responds, while the follower selects a best response to the action of the leader. The typical equilibrium concept studied in this class of games is known as a Stackelberg equilibrium. The importance of the order of play in optimization problems present in machine learning applications such as generative adversarial networks has spurred the development of local refinements of the Stackelberg equilibrium notion [15, 16] along with the design and analysis of iterative algorithms seeking to compute local Stackelberg equilibrium in nonconvex-nonconcave games [15, 16, 17, 18]. Reinforcement learning algorithms formulated in Stackelberg game are desirable methods that account for the order of play in multi-agent environments.

1.2 Contributions

In this thesis, we propose several certifiable reinforcement learning algorithms from safety and game-theoretic perspectives. We develop safe reinforcement learning algorithms for both pre-defined constraints and unknown constraints settings and novel actor-critic and multi-agent algorithms by formulating the learning with actor and critic or agents as a Stackelberg game.

In Chapter 2, we present our work on finding reinforcement learning policies for control systems with pre-defined state and action constraints. Previous safe reinforcement learning methods seeking to ensure constraint satisfaction, or safety, have focused on adding a projec-

tion step to the policy during learning. Yet, this approach requires solving an optimization problem at every policy execution step, which can lead to significant computational costs and has no safety guarantee with the projection step removed after training. To tackle this problem, this work proposes a new approach, termed Vertex Networks, with guarantees on safety during both the exploration and execution stages, by incorporating the safety constraints into the policy network architecture. Leveraging the geometric property that all points within a convex set can be represented as the convex combination of its vertices, the proposed algorithm first learns the convex combination weights and then uses these weights along with the pre-calculated vertices to output an action. The output action is guaranteed to be safe by construction. Numerical examples illustrate that the proposed VN algorithm outperforms projection-based reinforcement learning methods. This work is published in [19].

In Chapter 3, we present our work on safe reinforcement learning with known transition kernel and unknown constraints. In this work, we leverage Constrained Markov Decision Process, which is a class of stochastic decision problems in which the decision maker must select a policy that satisfies auxiliary cost constraints. This work extends traditional upper confidence reinforcement learning for settings in which the reward function and the constraints, described by cost functions, are unknown a priori but the transition kernel is known. Such a setting is well-motivated by a number of applications including exploration of unknown, potentially unsafe, environments. We present an algorithm `C-UCRL` and show that it achieves sub-linear regret ($O(T^{\frac{3}{4}}\sqrt{\log(T/\delta)})$) with respect to the reward while satisfying the constraints even while learning with probability $1-\delta$. Illustrative examples are provided. As an extension to unknown transition kernel setting, we present a lower bound on constraint violation, which proves the inevitability of constraint violation of algorithms in constrained Markov decision process. This work is published in [20] and proof details and additional experiment examples can be found in a longer version [21].

In Chapter 4, we present our work on game-theoretic reinforcement learning, where we formulate actor-critic algorithms as a Stackelberg game. The hierarchical interaction between the actor and critic in actor-critic based reinforcement learning algorithms naturally

lends itself to a game-theoretic interpretation. We adopt this viewpoint and model the actor and critic interaction as a two-player general-sum game with a leader-follower structure known as a Stackelberg game. Given this abstraction, we propose a meta-framework for Stackelberg actor-critic algorithms where the leader player follows the total derivative of its objective instead of the usual individual gradient. From a theoretical standpoint, we develop a policy gradient theorem for the refined update and provide a local convergence guarantee for the Stackelberg actor-critic algorithms to a local Stackelberg equilibrium. From an empirical standpoint, we demonstrate via simple examples that the learning dynamics we study mitigate cycling and accelerate convergence compared to the usual gradient dynamics given cost structures induced by actor-critic formulations. Finally, extensive experiments on OpenAI gym environments show that Stackelberg actor-critic algorithms always perform at least as well and often significantly outperform the standard actor-critic algorithm counterparts. This work is partially published in [22] and more results can be found in a longer version [23].

In Chapter 5, we study two-player competitive reinforcement learning problems. In order to let one player take advantage from the competition and benefit from learning with a learning opponent, we adopt the hierarchical Stackelberg game formulation and proposed the novel Stackelberg MADDPG algorithm. We also design and open-source new competitive reinforcement learning benchmark tasks and demonstrate the performance and behavior of our algorithm on them. Empirically, we show that the leader in the Stackelberg MADDPG algorithm learns a better policy and gains advantages in the competitive game.

Chapter 6 concludes the thesis with a discussion of the future directions of the works in this thesis and the general safety and game-theoretic reinforcement learning topics. When reading the thesis, each of the chapters can be read independently of one another, although Chapter 5 is highly related to Chaper 4. Each chapter focuses on a novel certifiable reinforcement learning algorithm designed for a specific problem formulation and application. Each chapter is associated with concise introduction and motivation sections to be self-contained.

Part I

SAFE REINFORCEMENT LEARNING

Chapter 2

SAFE REINFORCEMENT LEARNING VIA VERTEX NETWORKS

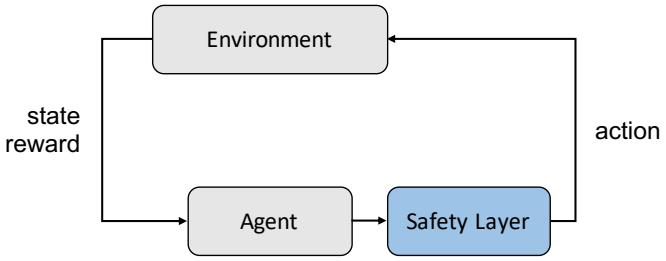


Figure 2.1: Framework of safe reinforcement learning with "safety layer".

2.1 Introduction

Over the last couple of years, reinforcement learning algorithms have yielded impressive results on a variety of applications. These successes include playing video games with super-human performance [1], robot locomotion and manipulation [3, 4], autonomous vehicles [5], and many benchmark continuous control tasks [24]. In reinforcement learning, an agent learns to make sequential decisions by interacting with the environment, gradually improving its performance at the task as learning progresses.

Policy optimization algorithms [3, 25] for reinforcement learning assume that agents are free to explore any behavior during learning, as long as it leads to performance improvement. However, in many real-world applications, there is often additional safety constraints, or specifications that lead to constraints, on the learning problem. For instance, a robot arm should avoid behaviors that could cause it to damage itself or the objects around it,

and autonomous vehicles must avoid crashing into others while navigating [10]. In fact, constraints are integral to many real-world problems, and maintaining constraint satisfaction during learning is critical. In addition, these are often hard constraints that must be satisfied at all times. Therefore, in this work, our goal is to maintain constraint satisfaction at each step throughout the whole learning process. This problem is sometimes called the *safe reinforcement learning* problem [26]. In particular, we define safety as staying within some pre-specified safety sets for both states and actions.

In existing safe reinforcement learning literature, a "safety layer" is often leveraged to maintain constraint satisfaction during training under different problem settings [27, 28, 29]. Figure 2.1 provides a framework of those approach in reinforcement learning settings. In these approaches, action of the policy at each time step is not directly executed, but sent to the safety layer which projects it into the required safety region. The policy can explore the action space freely to maximize performance, while the projection maintains constraint satisfaction. However, these approaches either involves solving a computationally expensive optimization problem at each time step [27, 29], or has strict assumptions about the constraint violation [28]. As a matter of fact, if real-time optimization is allowed by the application, then it is often more advantageous to solve a model predictive control (MPC) problem than to ask for a policy learned by reinforcement learning. Moreover, if the projection is not differentiable, these methods decouple the action with the policy, which may result in an unstable policy with the projection step removed after training.

To alleviate the above limitation of projection-based approaches, we propose Vertex Networks (VNs), where we encode the safety constraints into the policy via neural network architecture design. In VNs, we design a novel safety layer, where instead of solving a projection optimization problem, we compute the vertices of the safety region at each time step and design the action to be the convex combination of those vertices, allowing policy optimization algorithms to explore only inside the safe region during training. Therefore we are able to integrate the constraints into the learning process by design and leverage standard policy optimization training techniques.

2.1.1 Contribution

The contributions of this work can be briefly summarized as follows: (1) To the best of our knowledge, this is the first attempt to encode safety constraints into policies by explicit network design. (2) In simulation on benchmark inverted pendulum and hovercraft control examples, the proposed method achieves better performance as well as constraint satisfaction compared with the projection-based method.

2.1.2 Related work

Safe reinforcement learning or learning-based control with safety certification methods have gained significant attention in recent years. A common technique is to employ a "safety layer" or "safety framework" [30, 31, 27, 32, 28, 29, 33, 34], which consists of a projection-based supervisory element between the agent and the system. If the action generated by the agent would cause the system to leave the safe region, the safety layer intervenes and project to the closest action where the system would stay in the safety region. The techniques proposed in [30, 31] are based on a differential game formulation that results in solving a min-max optimal control problem, which can provide the largest possible safe set, but offers very limited scalability. MPC-like schemes are used in [27, 32], which are computationally expensive if executed online. A closed-form solution can be obtained by locally linearizing the learned model [28]. As a result, this safety layer is differentiable and can be applied to any policy optimization algorithm. However, this approach assumes that only one of the half-space constraints to be violated, which is unlikely to always hold during exploration. These projection-based methods have also been leveraged in the settings where the constraints are defined by control barrier functions [33, 34]. Special policies can be designed for tailored applications, e.g., power system frequency regulation [35, 36]. Finally, most similar to our setting, [29] consider polytope constraints and propose projected DDPG algorithm to maintain safety during learning. We compare to this method as a baseline in our experiments.

2.2 Problem Formulation

Consider a discrete time affine control system in which the system evolves according to

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t) + H(\mathbf{x}_t)\mathbf{u}_t + \mathbf{w}_t, \tag{2.1}$$

where $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{u} \in \mathbb{R}^m$, and f and H are known functions of appropriate dimensions. The unknown disturbance $\mathbf{w}_t$ is assumed to be in a compact set $\mathcal{W}$, where $\mathcal{W} = \{\mathbf{w} | \underline{\mathbf{w}} \leq \mathbf{w} \leq \overline{\mathbf{w}}\}$ is a convex polytope. Our goal is to maximize the cumulative reward over time horizon T, subject to safety constraints on $\mathbf{x}$ and actuator constraints on $\mathbf{u}$:

$$\max_{\mathbf{u}} \sum_{t=1}^{T} R(\mathbf{x}_t, \mathbf{u}_t) \tag{2.2a}$$

$$\text{s.t. } \mathbf{x}_{t+1} = f(\mathbf{x}_t) + H(\mathbf{x}_t)\mathbf{u}_t + \mathbf{w}_t \tag{2.2b}$$

$$\mathbf{x}_t \in \mathcal{X} \tag{2.2c}$$

$$\mathbf{u}_t \in \mathcal{U}, \tag{2.2d}$$

where $\mathcal{X}$ and $\mathcal{U}$ are *convex polytopes*. A convex polytope can be defined as an intersection of linear inequalities (half-space representation) or equivalently as a convex combination of a finite number of points (convex-hull representation) [37]. This type of constraints are widely used in theory and practice—for example, see [38] and the references within. In this work, we assume that the problem defined in (2.2) is feasible for all possible disturbances in $\mathcal{W}$.

The goal of safe reinforcement learning is to find an optimal feedback controller $\mathbf{u}_t = \pi_\theta(\mathbf{x}_t)$, that maximizes the overall system reward (2.2a) while satisfies the safety constraints (2.2c) and the actuator constraints (2.2d). Solving (2.2) is a difficult task, even for linear systems with only the actuator constraints, except for a class of systems where analytic solutions can be found [39]. Therefore, reinforcement learning (and its different variants) have been proposed to search for a feedback controller.

Numerous learning approaches have been adopted to solve the problem when the constraints (2.2c) and (2.2d) are not present. However, there are considerably less successful applications of reinforcement learning to problems with hard constraints. In contrast to the

popular projection-based approaches (see Section 2.1.2), we propose a novel vertex policy network that encodes the *geometry* of the constraints into the network architecture, which makes a differentiable policy that can be trained by any policy optimization algorithm in an end-to-end way. We will discuss the proposed vertex policy network framework in detail in the next section.

2.3 Vertex Policy Network

The key idea of our proposed Vertex Network (VN) is based on the geometry property of a convex polytope. Given a bounded convex polytope $\mathcal{P}$, it is always possible to find a finite number of *vertices* such that the convex hull is $\mathcal{P}$. In addition, there is no smaller set of points whose convex hull forms $\mathcal{P}$ [37]. Then, the next proposition follows directly.

Proposition 2.1. *Let $\mathcal{P}$ be a convex polytope with vertices $P_1, \ldots, P_N$. For every point $\mathbf{p} \in \mathcal{P}$, there exists $\lambda_1, \ldots, \lambda_N$, such that*

$$\mathbf{p} = \lambda_1 P_1 + \cdots + \lambda_N P_N,$$

where $\lambda_i \geq 0, \forall i$ and $\lambda_1 + \cdots + \lambda_N = 1$.

The preceding proposition implies that we can search for the set of weights λ_i's instead of directly finding a point inside polytope.

Proposition 2.1 can be applied to find a feedback control policy. Since both the constraint sets $\mathcal{X}$, $\mathcal{U}$ and $\mathcal{W}$ are convex polytopes, the feasible control action at each time step must also live in a convex polytope. If its vertices are known, it suffices for the policy to learn the weights λ_i's. The benefit of having the weights as the output is threefold. Firstly, it is much easier to normalize a set of real numbers to be all positive and to sum to unity (the probability simplex) than to project into an arbitrary polytope. In this work, we use a softmax layer for this purpose. Secondly, this approach allows us to fully explore the interior of the feasible space, where projections could be biased towards the boundary of the set. Thirdly, we are able to use standard policy optimization training techniques by plugging in this policy network design.

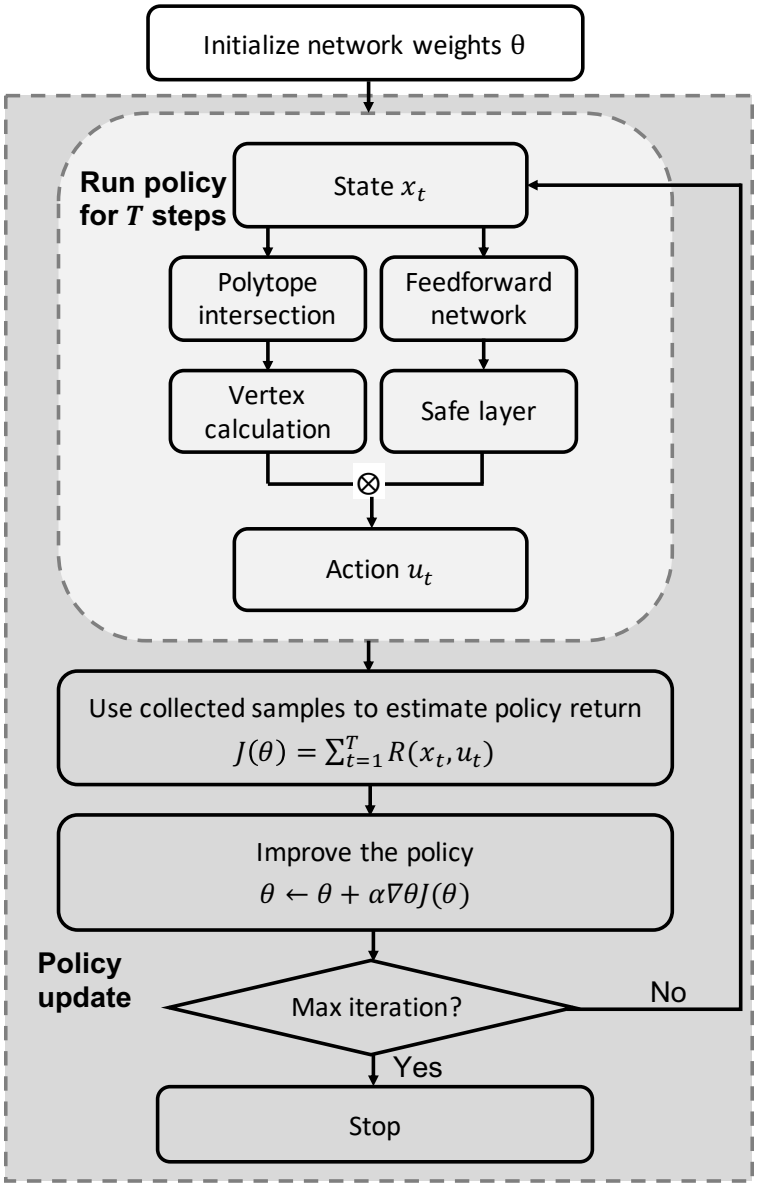


Figure 2.2: Flowchart of policy optimization with the proposed VN framework.

For general policy optimization algorithm, the parameterized policy neural network π_θ is updated by

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta), \tag{2.3}$$

where $J(\theta)$ is the expected return of the current policy and is defined by

$$J(\theta) = \mathbb{E}\left[\sum_{t=1}^{T} R(\mathbf{x}_t, \mathbf{u}_t)\right]. \tag{2.4}$$

$J(\theta)$ is often approximated by $\frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} R(\mathbf{x}_{i,t}, \mathbf{u}_{i,t})$, where N are the number of sampled trajectories generated by running the current policy $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ and T is the trajectory length. The overall algorithm procedure with the proposed VN framework is provided in Figure 2.2.

Below, we discuss the two major components of VN in detail: (1) the safety region and vertex calculation, and (2) the neural network architecture design for the safety layer.

2.3.1 Evolution of the Action Constraint Set

We require that at each time step the states of the system stay in the set $\mathcal{X}$ subject to all possible disturbances in $\mathcal{W}$, and the control actions at constrained to be in the set $\mathcal{U}$. As stated earlier, we assume $\mathcal{X}$, $\mathcal{U}$ and $\mathcal{W}$ to be convex polytopes. The main algorithmic challenge comes from the need to repeatedly intersect translated versions of these polytopes. To be concrete, suppose we are given $\mathbf{x}_t$. Then for the next step, we require that $\mathbf{x}_{t+1} \in \mathcal{X}$. This translates into an affine constraint on $\mathbf{u}_t$, since the control action mush satisfy

$$H(\mathbf{x}_t)\mathbf{u}_t \in \mathcal{X} \ominus \mathcal{W} - f(\mathbf{x}_t). \tag{2.5}$$

where $\ominus$ represents Pontryagin set difference, and $\mathcal{X} \ominus \mathcal{W} = \{\mathbf{x}|\mathbf{x} + \mathcal{W} \subseteq \mathcal{X}\}$ is a polytope. Since $\mathbf{x}_t$ is known, $H(\mathbf{x}_t)$ is a constant matrix in the above equation, and the constraint on $\mathbf{u}_t$ imposed by (2.5) is again polytopic. We denote this polytope as $\mathcal{S}_t$. The set to which $\mathbf{u}_t$ must belong is the intersection of $\mathcal{S}_t$ and the actuator constraints:

$$\mathcal{U}_t = \mathcal{S}_t \cap \mathcal{U}. \tag{2.6}$$

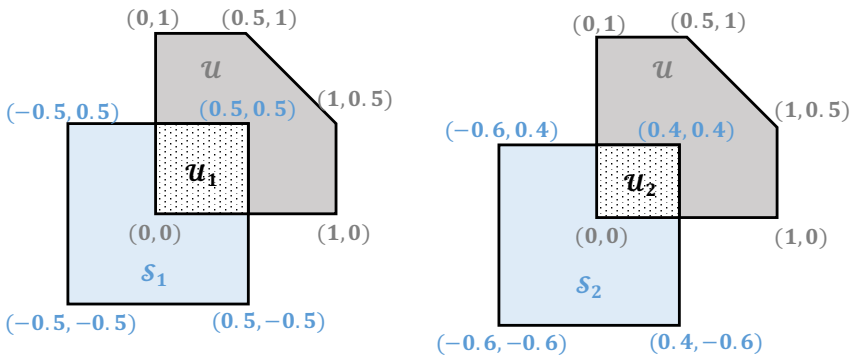


Figure 2.3: Evolution of action constraint set of Example 2.1. The left plot visualizes the safety set at time $t = 1$, and the right plot shows the safety set at time $t = 2$.

After identifying the vertices of $\mathcal{U}_t$, the algorithm in Figure 2.2 can be used to find the optimal feedback policies.

Remark 2.1. *Note that the safety requirement here is "one step ahead". One could use the methods in [40, 41] to compute a tighter "infinity steps ahead" invariant set $\mathcal{S}_t$ for linear systems. See [42] for more discussions on controlled invariant set. However, this is computationally more complicated and out of the scope of this work.*

In general, it is fairly straightforward to find the convex hull or the half-space representations of $\mathcal{S}_t$, since it just requires a linear transformation of $\mathcal{X} \ominus \mathcal{W}$. However, the intersection step in (2.6) and extracting the vertices are non-trivial. Below, we walk through a simple example to illustrate the steps and then discuss how to overcome the computational challenges in Section 2.3.2.

Example 2.1 (Intersection Step)**.** *Consider the following two–dimensional linear system:*

$$\mathbf{x}_{t+1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \mathbf{x}_t + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \mathbf{u}_t.$$

Suppose the action safety set $\mathcal{U}$ is a convex polytope defined by: $0 \leq u_1 \leq 1$, $0 \leq u_2 \leq 1$ and $u_1 + u_2 \leq 1.5$. The state safety set $\mathcal{X}$ is a square defined by $0 \leq x_1 \leq 1, 0 \leq$

$x_2 \leq 1$ and the initial state is $\mathbf{x}_1 = \begin{bmatrix} 0.5 & 0.5 \end{bmatrix}^T$. By simple calculation, $\mathcal{S}_1 = \{-0.5 \leq u_1 \leq 0.5, -0.5 \leq u_2 \leq 0.5\}$ and $\mathcal{U}_1$ is the box bounded by $\{(0,0),(0,0.5),(0.5,0),(0.5,0.5)\}$. Figure 2.3 (left) visualizes the intersection operations. Now suppose a feasible action $\mathbf{u}_1 = [0.1\ 0.1]^\top$ is chosen and the system evolves. Then, $\mathcal{S}_2 = \{-0.6 \leq u_1 \leq 0.4, -0.6 \leq u_2 \leq 0.4\}$. Performing the intersection of $\mathcal{S}_2$ and $\mathcal{U}$, we get that $\mathcal{U}_2$ is a rectangle defined by the vertices $\{(0,0),(0,0.4),(0.4,0),(0.4,0.4)\}$ as depicted in Figure 2.3 (right).

2.3.2 Intersection of Polytopes

It should be noted that finding the vertices of an intersection of polytopes is not easy [43]. If the polytopes are in half-space representation, then their intersection can be found by simply stacking the inequalities. However, finding the vertices of the resulting polytope can be computationally expensive. Similarly, directly intersecting two polytopes based on their convex-hull representation is also intractable in general.

Luckily, in many applications, we are not intersecting two generic polytopes at each step. Rather, as illustrated in Example 2.1, they are two polytopes with fix shapes, where the position of $\mathcal{U}$ is fixed as well and only the position of $\mathcal{S}_t$ depends on $\mathbf{x}_t$. It turns out that for many systems (see Section 2.4), we can find the resulting vertices by hand-designed rules. In addition, there are heuristics that work well for low-dimensional systems [44]. Applying the proposed VN technique to high-dimensional systems is the main future direction for this work.

For the case that $\mathcal{S}_t$ and $\mathcal{U}$ do not overlap, we pick the point in $\mathcal{U}$ that closest to $\mathcal{S}_t$ to be the vertex. By design, the output of the VN is the action within set $\mathcal{U}$, meanwhile transiting to the state closest to the safe state set $\mathcal{X}$. One could also choose to terminate the current training episode and reset the environment.

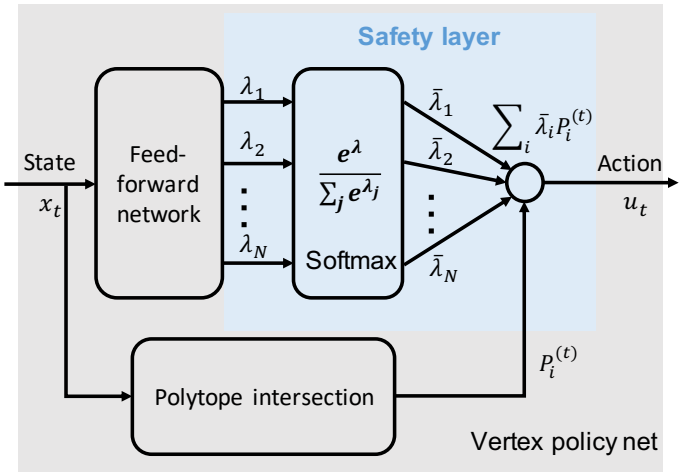


Figure 2.4: Illustration of the proposed safety layer architecture. The output of the policy network is modified to predict the weights $\lambda_i, i \in [1, N]$. These weights are normalized to $\bar{\lambda}_i, i \in [1, N]$ that satisfies $\bar{\lambda}_i \geq 0$ and $\sum_{i=1}^{N} \bar{\lambda}_i = 1$, via the softmax activation function. The action output is calculated as $\sum_{i=1}^{N} \bar{\lambda}_i P_i^{(t)}$, where $P_i^{(t)}$ are the safety polytope vertices.

2.3.3 Safety Layer

Once we obtain $\mathcal{U}_t$, the next step is to encode the geometry information into the policy network such that the generated action stays in $\mathcal{U}_t$. According to Proposition 2.1, it suffices for the policy network to generate the weights (or coefficients) of that convex combination.

Suppose that $\mathcal{U}_t$ can have at most N vertices, labeled $P_1^{(t)}, \ldots, P_N^{(t)}$. In the policy network architecture design, we add an intermediate safety layer that first generates N nodes $\lambda_1, \ldots, \lambda_N$. The value of these nodes, however, are not positive nor do they sum to 1. Therefore, a softmax unit is included as the activation function in order to guarantee the non-negativity and the summation constraints. In particular, we define $\bar{\lambda}_i = e^{\lambda_i}/(\sum_{j=1}^{N} e^{\lambda_j})$, the weights of a convex combination. The final output layer (action u_t) is defined as the multiplication of these normalized weights $\bar{\lambda}_i$ and the corresponding vertex values,

$$\mathbf{u}_t = \sum_{i=1}^{N} \bar{\lambda}_i P_i^{(t)}. \tag{2.7}$$

An illustration diagram is provided in Figure 2.4.

Note that the exact number of vertices of the polytope intersection may vary from state to state. In our design, we simply set N to be the largest possible number, since the convex combination with repeated vertices still captures all points in the polytope. Also, our method relies on an offline hand-designed rule to compute the vertices. Such hand-designed rule is a mapping from state to a sequence of vertices, which is "standardized" as the vertices are in a fixed order.

2.4 Experiments

In this section, we present and analyze the performance of the proposed VN experimentally. We first describe the baseline algorithms and then demonstrate the performance comparisons in two benchmark control tasks: (i) inverted pendulum and (ii) hovercraft tracking. The code for our experiments is available at `https://github.com/LeoZhengZLY/vertex-net`.

2.4.1 General Experimental Details

Baseline Learners and Comparators. In our simulations, we compare against two baseline algorithms. (1) We compare against DDPG algorithm [3], which is a state-of-the-art continuous control reinforcement learning algorithm, to optimize the controller policy. To add safety constraints to the vanilla DDPG algorithm, a natural approach is to artificially shape the reward such that the agent will learn to avoid undesired areas. This can be done by setting a low or negative reward to the unsafe states and actions (termed as penalty method in [26]). In our experiments, we include such a soft penalty in the reward function and train a standard policy network with DDPG algorithm as the first baseline.[1] (2) We also compare against the projected DDPG algorithm in [29] as the second baseline algorithm (refer to as PDDPG). This approach introduces a projection-based supervisory element between the agent and the system, which projects the unsafe actions into the safety set at each time step

[1]The reward function is consistent among all methods, meaning that our method also includes such soft penalty.

during training. According to [29], after the training, the supervisor is removed and the agent acts as a conventional reinforcement learning-based controller.

Implementation Details. We use the following hyperparameters for all experiments. For DDPG and PDDPG, we use a three-layer feed-forward neural network, with 256 nodes in each hidden layer. For our approach, we use the proposed VN to represent the policy and leverage DDPG to optimize it directly (refer to as VN-DDPG). It has two feed-forward layers (with 256 nodes in each hidden layer) and a final safety layer as described in Section 2.3.3.

2.4.2 Pendulum

Experiment Setup. For the inverted pendulum problem, we use the OpenAI gym environment (`pendulum-v0`), with the following specifications: mass $m = 1$, length $l = 1$. The system state is two-dimension that include angle θ and angular velocity ω of the pendulum, and the control variable is the applied torque u. We set the safe region to be $\theta \in [-1, 1]$ (radius) and torque limits $u \in \mathcal{U} = [-15, 15]$. The reward function is defined as $r = -(\theta^2 + 0.1\omega^2 + 0.001u^2)$, with the goal of learning an optimal feedback controller. With a discretization step size of $\Delta = 0.05$, the following are the discretized system dynamics:

$$\theta_{t+1} = \theta_t + \omega_t \Delta + \frac{3g}{2l} \sin(\theta_t) \Delta^2 + \frac{3}{ml^2} u \Delta^2 + d_t \tag{2.8a}$$

$$\omega_{t+1} = \omega_t + \frac{3g}{2l} \sin(\theta_t) \Delta + \frac{3}{ml^2} u \Delta \tag{2.8b}$$

where d_t is the disturbance random variable uniformly drawn from $[-0.05, 0.05]$.

Technical Details. To keep the next state in the safe region $\theta_{t+1} \in [-1, 1]$, we can compute the corresponding upper and lower bound of u to represent set $\mathcal{S}_t$ by (2.8a). Therefore, the vertices of VN can be found by intersecting $\mathcal{S}_t$ and $\mathcal{U}$. Under the case where $\mathcal{S}_t$ and $\mathcal{U}$ have no overlap, we pick -15 as the vertices if the upper bound of $\mathcal{S}_t$ is less than -15. Otherwise, we pick 15 as the vertices. For comparison, the output of normal policy network is constrained

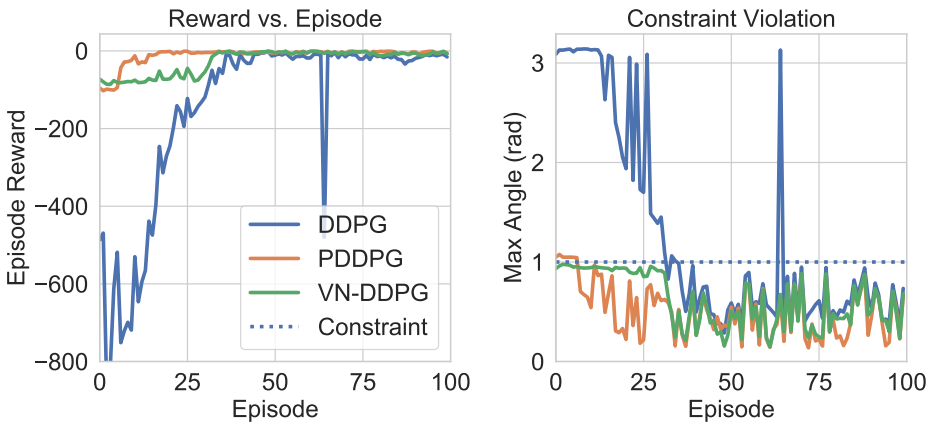


Figure 2.5: Comparison of accumulated reward and constraint violation (max angle) for the pendulum problem.

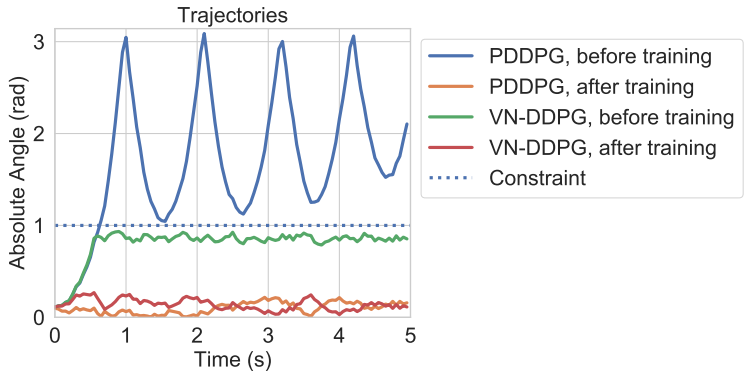


Figure 2.6: Representative trajectories (angle vs. time) generated by the policy before training and after training.

in $[-15, 15]$ by using `tanh` activation function in the final layer. The initial state of each episode is randomly sampled in the safe state region $[-1, 1]$.

Results. In Figure 2.5, we show a comparison of the accumulated reward and the max angle of each episode in training of DDPG, PDDPG, and VN-DDPG. We observe that both PDDPG and VN-DDPG maintain safety throughout the training process, and as a result,

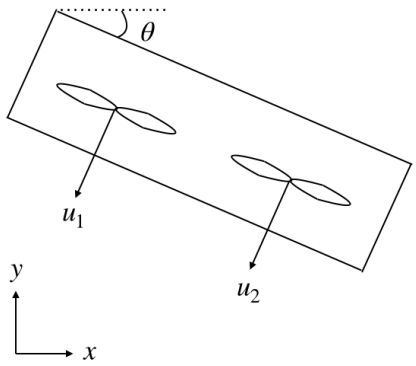


Figure 2.7: Hovercraft example. u_1 and u_2 denote starboard and port fan forces. θ, x, y are the tilt angle and the coordinate position.

it achieves higher reward in the early stage. It is also interesting to observe that the DDPG also becomes "safe" after training, since the reward function itself drives θ to be small. This suggests if we can train the DDPG with soft penalty offline, it might be able to obey the safety constraint for some control systems. However, the plot on max angle shows that even a well-trained policy may occasionally violate constraints if these hard constraints are not explicitly taken into account.

Figure 2.6 shows the pendulum angle trajectories of representative episodes before training versus after training generated by PDDPG and VN-DDPG. For VN-DDPG before training, the pendulum angle is maintained near the edge of the safe region.[2] However, the PDDPG fails to maintain safety with projection supervisor removed. Once the training is finished, both of the learned controller are safe. These suggest that PDDPG could be unsafe if the training has not converged but VN-DDPG maintains safety regardless of the neural network initialization.

2.4.3 Hovercraft

Experiment Setup. Consider the task of controlling a hovercraft that tracks a target position illustrated in Figure 2.7. The system dynamics are defined as follows [45]:

$$x_{t+1} = x_t + v_{x,t}\Delta + \frac{1}{2m}\sin\theta_t(u_1+u_2)\Delta^2 \tag{2.9a}$$

$$v_{x,t+1} = v_{x,t} + \frac{1}{m}\sin\theta_t(u_1+u_2)\Delta \tag{2.9b}$$

$$y_{t+1} = y_t + v_{y,t}\Delta + \frac{1}{2m}(\cos\theta_t(u_1+u_2)-g)\Delta^2 \tag{2.9c}$$

$$v_{y,t+1} = v_{y,t} + \frac{1}{m}(\cos\theta_t(u_1+u_2)-g)\Delta \tag{2.9d}$$

$$\theta_{t+1} = \theta_t + v_{\theta,t}\Delta + \frac{1}{2l}(u_1-u_2)\Delta^2 \tag{2.9e}$$

$$v_{\theta,t+1} = v_{\theta,t} + \frac{1}{l}(u_1-u_2)\Delta \tag{2.9f}$$

where $m = l = 1, g = 10$. The initial state of the hovercraft is set at position $(0, 0)$ and the target position is $(5, 5)$. To keep the tilt angle of the hovercraft in a safety region, we set the safe state region to be $\theta \in [-\overline{\theta}, \overline{\theta}]$. To better investigate the effect of the constraint, we did two experiments where the tilt angle upper bounds are set to be $\overline{\theta} = 0.01$ and $\overline{\theta} = 0.25$ radians, respectively. Considering the force exerted on two fans are coupled and subject to a total energy budget, the actuator constraint set is defined as $\mathcal{U} = \{u_1, u_2 | u_1 \geq 0, u_2 \geq 0, u_1 + u_2 \leq 20\}$.

Technical Details. For all the learners, we define the reward function $r = -(x - x_0)^2 - (y - y_0)^2 - \theta^2 - 0.1(v_x^2 + v_y^2 + v_\theta^2) - 0.001(u_1^2 + u_2^2)$ where (x_0, y_0) denotes the target position. The learner is incentivized to track the target position while keeping the tile angle and force small for the safety constraint. In addition, for DDPG and PDDPG, the output of policy network is truncated to be within the actuator constraint set $\mathcal{U}$. For the proposed VN-DDPG approach, Figure 2.8 shows how to use at most five vertices to represent the intersection of safe state region and the actuator constraint region. The gray area is the actuator constraint

[2]Since we initialize the neural network near to zero, the policy before training is simply the average of all vertices.

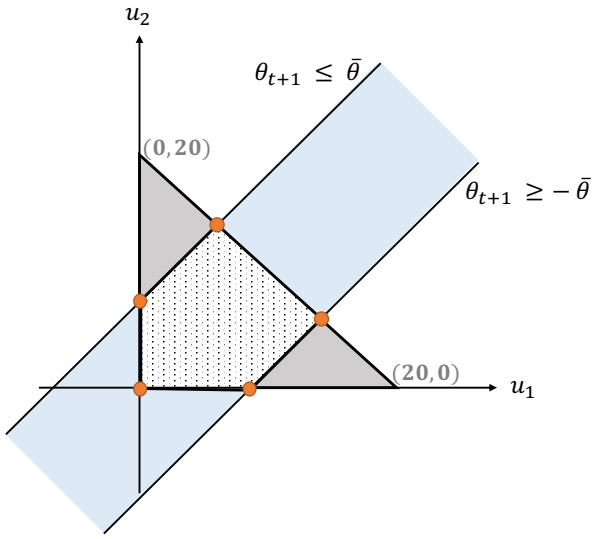


Figure 2.8: Illustration of polytope intersection of hovercraft examples.

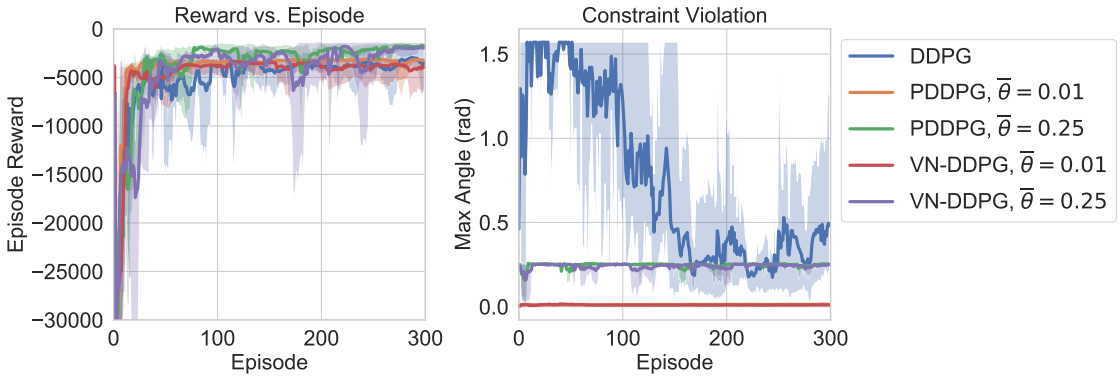


Figure 2.9: Comparison of accumulated reward and constraint violation (max tile angle) from Hovercraft control with different constraint upper bound.

set $\mathcal{U}$, the blue area are the constraint on u_1, u_2 imposed by $\theta_{t+1} \in [-\overline{\theta}, \overline{\theta}]$ and (2.9e). The orange vertices can be computed in closed-form by intersecting boundary equations.

Results. Figure 2.9 compares the accumulated reward and max tilt angle of each episode in the training of DDPG, PDDPG, and VN-DDPG. We observe that even trained DDPG could

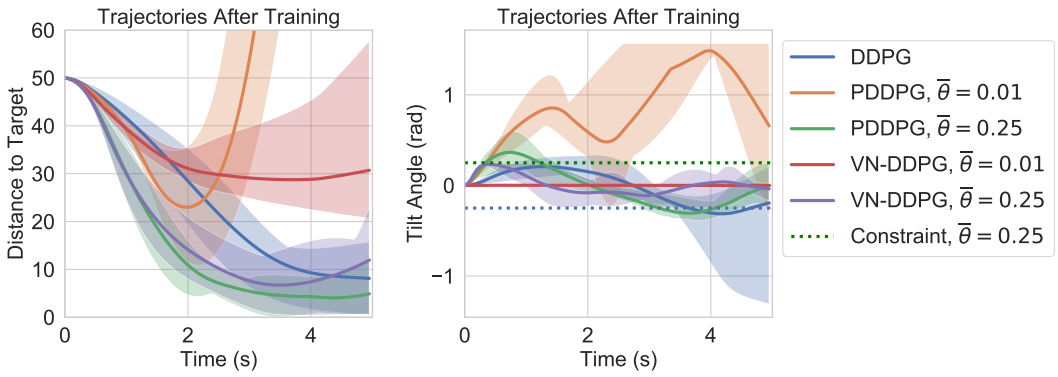


Figure 2.10: Trajectories (distance to target $||(x, y) - (x_0, y_0)||^2$ and tilt angles θ) of trained policies with different tilt angle upper limit.

still constantly violate the constraint. Both PDDPG and VN-DDPG are able to maintain constraint satisfaction during training and we can observe that when $\overline{\theta} = 0.01$, they converges to a suboptimal controller with lower episode reward since the constraint is too restrict and contradicts with the task (tracking the target position). Figure 2.10 visualizes the trajectories of trained policies. In both choices of the tilt angle upper limit $\overline{\theta}$, the constraint is never violated in the whole trajectory executing learned policies of VN-DDPG. When $\overline{\theta} = 0.01$, the hovercraft has a strict constraint on its tilt angle and fails to track the target position but with $\overline{\theta} = 0.25$, it is able to reach the target. On the contrary, running the learned DDPG and PDDPG policies will have tilt angle violation even with the soft penalty added in the reward. Note that the trained policy of PDDPG with $\overline{\theta} = 0.01$ has poor performance as it completely fails to track the target and violates the constraint. This is due to the fact that the constraint during training is too restrict and the projected actions deviate a lot from the original unconstrained policy. As a matter of fact, the decoupling of the action and the policy results in an unstable policy with the projection step removed after training.

2.5 Conclusion

In this work we design a novel policy network architecture called Vertex Network (VN), which is motivated by the problem of training an reinforcement learning algorithm with hard state and action constraints. Leveraging the geometric property that a convex polytope can be equivalently represented as the convex hull of a finite set of vertices, the output of VN satisfies the safety constraints by design. Empirically, we show that VN yields significantly better safety performance compared with a normal policy network with a constraint violation penalty or with a projection-based supervisor in two benchmark control systems.

Chapter 3

CONSTRAINED UPPER CONFIDENCE REINFORCEMENT LEARNING

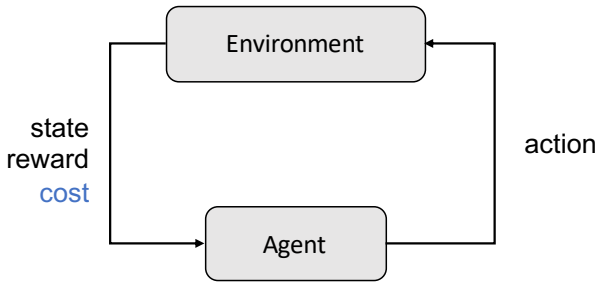


Figure 3.1: Framework of constrained reinforcement learning with cost feedback.

3.1 Introduction

Markov Decision Processes (MDPs) have been successfully utilized to model sequential decision-making problems in stochastic environments. In the typical approach to learning a policy, the decision-maker trades off between exploration and exploitation, gradually improving their performance at the task as learning progresses. Reinforcement learning, a standard paradigm of learning in MDPs, has shown exceptional success in a variety of domains such as video games [1], robotics [3, 4], recommender systems [46], and autonomous vehicles [5]. Yet, in many of these real-world applications there is additional constraints, or specifications that lead to constraints, on the learning problem.

For instance, a recommender system should avoid presenting offending items to users and autonomous vehicles must avoid crashing into others while navigating [26]. Building algorithms that respect safety constraints not only during normal operation, but also during

the initial learning period, is a question of particular interest [47]. This problem is known as the *safe exploration problem* [48, 10]. In the standard MDP framework, an approach for baseline performance is risk-sensitive reinforcement learning [49, 26], where the optimization criterion is transformed in order to reflect a subjective measure balancing the return and the risk.

On the other hand, in a safety-critical environment, it is more reasonable to separate the return and the risk criterion, and enforce constraint satisfaction in the learning procedure. A standard formulation for an environment with safety constraints is the constrained MDPs (CMDPs) [50]. A decision-maker facing a CMDP aims to maximize the total reward while satisfying the constraints on costs in expectation over the whole trajectory. Figure 3.1 provides a framework of reinforcement learning in CMDP.

In recent literature, policy gradient-based reinforcement learning algorithms have been proposed as a means to learn a policy for a CMDP. The following are two constrained policy search algorithms with state-of-the-art performance guarantees: Lagrangian-based actor-critic algorithm [51, 52, 53, 54, 55] and Constrained Policy Optimization (CPO) [56, 57]. However, for these policy gradient-based methods, safety is only approximately guaranteed *after* a sufficient learning period. The fundamental issue is that without a model, safety must be learned via trial and error, which means it may be violated during initial learning interactions.

Model-based approaches have utilized Gaussian processes to model the state safety values or the dynamic uncertainties [58, 59, 60, 33] or utilized Lyapunov-based methods [61] to guarantee safety during learning. Although these methods guarantee constraint satisfaction during learning, an arguably valuable analysis of the regret is lacking.

In unconstrained settings when the reward and transition kernel are unknown, upper confidence based reinforcement learning algorithms have been proposed—namely, `UCRL2` [62]—with sub-linear regret. The key idea is to build confidence intervals on the reward and transition kernel and iteratively solve for policies using value iteration.

In this work, we are not only interested in learning the optimal policy that satisfies the

constraints via interacting with the stochastic environment, but also in ensuring performance guarantees on the learning algorithm during learning. With some practical scenarios in mind, we make the assumption that the rewards and constraint costs are unknown. For instance, consider a robot navigation task; here may have an approximate model the dynamics of the robot (known with some uncertainty) and the reward and constraints which model the value of exploring the environment as unknown—e.g., constraints can be abstracted as costs which seek to limit the frequency of visiting a potentially hazardous states [63].

Motivated by upper confidence reinforcement learning [62], we introduce the constrained upper confidence reinforcement learning (`C-UCRL`) algorithm which combines elements of the classical `UCRL2` algorithm with robust linear programming. We define our goals as follows: (1) maintain constraint satisfaction throughout the learning process with high probability, and (2) achieve sub-linear regret comparing the rewards collected by the algorithm during learning with the reward of an optimal stochastic policy.

3.1.1 Contributions.

The contributions can be summarized as follows. Building on `UCRL2`, we introduce the `C-UCRL` algorithm (Algorithm 1). We show that `C-UCRL` is guaranteed to satisfy constraints during learning with probability at least $1-\delta$ (Theorem 3.1) and achieves $O(T^{\frac{3}{4}}\sqrt{\log(T/\delta)})$ reward regret (Theorem 3.2). Of independent interest, we note that when the state space is trivial, the setting we consider subsumes stochastic multi-armed bandits with per-round budget constraints, where the optimal policy is a randomized policy across arms.

3.2 Related Work

Recently, several policy gradient-based reinforcement learning algorithms have been proposed for learning policies for CMDPs. In particular, there are two noteable constrained policy search algorithms which enjoy state-of-the-art performance: a Lagrangian-based algorithm [51, 52] and Constrained Policy Optimization (CPO) [56]. The Lagrangian-based algorithm formulates the CMDP problem as a minimax problem and uses primal-dual gradi-

ent optimization to find the saddle point solution. While this procedure will asymptotically converge to the saddle point solution, in general there is no guarantee on policies being safe during the learning procedure. On the other hand, CPO—a method that derives from an extension of trust-region policy optimization (TRPO)—guarantees monotonic performance improvements on the expected reward and a guarantee on constraint satisfaction throughout training. While this algorithm is safe during learning, analyzing its convergence is challenging and the regret analysis with respect to reward is lacking.

As an alternative to policy gradient reinforcement learning algorithms, linear programming based algorithms have been proposed. In [63], CMDPs with known reward, constraints, transition kernel but uncertain initial state distribution are considered. Linear programming based algorithms are proposed to solve for safe policies in this setting. In our setting, however, the reward and constraints are stochastic and considered unknown a priori, which the stochastic transition kernel is known.

Most similar to our approach is `UCRL2`; in particular, our approach can be viewed as an extension of `UCRL2` [62], in some sense, by incorporating constraints; the one difference is that we assume the transition kernel is known while the classical `UCRL2` algorithm does not. We leave extending our setting to unknown transition kernels to future work. As alluded to in the introduction, in `UCRL2`, the reward and transition kernel are approximated and the policy is obtained by value iteration based methods in a "optimism in the face of uncertainty" fashion. Further, the performance of `UCRL2` is analyzed by bounding the regret with respect to the optimal *deterministic* policy. CMDPs, however, in general do not admit deterministic policies. In `C-UCRL`, the reward and constraints are approximated and the policy is obtained by solving a robust linear program. Performance is assessed by computing the *reward* regret with respect to the optimal randomized policy.

Finally, our work is related to the multi-armed bandit problem with constraints. Previous works, e.g., have considered the multi-armed bandit problem with an auxiliary cost in addition to the traditonal reward [64, 65]. The "game" (between the player and the environment) ends when the sum of current costs associated with the played arms exceeds the remaining

budget, which is fixed and known to the player. The typical approach is to construct upper confidence bounds for the reward-to-cost ratio and then utilize them in upper confidence bound-based algorithms. On the other hand, in our approach, we use upper confidence bounds for both reward and cost, and solve a linear program to obtain the policy policy. In related work, fairness constraints are incorporated into a multi-armed bandit setting; in particular, arms that are perceived to have less value/reward should never favored over better performing alternatives, despite a learning algorithm's uncertainty over the true payoffs [66]. In such settings, the algorithm is forced to pick arms uniformly until the player has enough confidence of the performance of arms. Connecting to this body of work, our problem reduces to a constrained multi-armed bandit problem when there is a single state. The main difference between our setting and that of the majority existing multi-armed bandit literature with constraints is that the optimal policy and policies obtainable by our algorithm can be a randomized or stochastic policy as opposed to a deterministic "best arm" policy.

3.3 Problem Formulation

An MDP is a tuple $(\mathcal{S}, \mathcal{A}, P, r)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{A} \to [0, 1]$ is the transition kernel such that $P(s'|s, a)$ is the probability of transitioning to state s' given that the previous state was s and the agent took action a in s, and $r : \mathcal{S} \times \mathcal{A} \to [0, 1]$ is the reward function. A stationary policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ is a map from states to a probability distribution over actions, with $\pi(a|s)$ denoting the probability of selecting action a in state s. We consider the setting in which the transition kernel $P(s'|s, a)$ is known to the agent, but the reward and costs are stochastic and unknown. In the example of a rover exploring the surface of Mars, the agent (rover) is aware of the transition probability of next state based on its action, but the *safety quality* of each state is unknown. Let $S = |\mathcal{S}|$ and $A = |\mathcal{A}|$ where $|\cdot|$ is the cardinality of its argument. We use the notation $[\cdot] = \{1, \ldots, \cdot\}$ for index sets.

3.3.1 Constrained Markov Decision Processes

A CMDP is an MDP augmented with 'cost' constraints that restrict the set of allowable policies for that MDP. For a given CMDP, we consider the performance measure to be the *infinite horizon average reward* which is given by

$$J(\pi) = \lim_{T\to\infty} \mathbb{E}_{\tau\sim\pi}\big[\tfrac{1}{T}\sum_{t=0}^{T-1} r(s_t, a_t)\big] \tag{3.1}$$

where τ denotes a trajectory $\tau = (s_0, a_0, s_1, \dots)$, and $\tau \sim \pi$ is shorthand for indicating that the distribution over trajectories depends on π: $s_0 \sim p(s_0), a_t \sim \pi(\cdot|s_t), s_{t+1} \sim P(\cdot|s_t, a_t)$. Similarly, define the *average constraint costs* by

$$C_i(\pi) = \lim_{T\to\infty} \mathbb{E}_{\tau\sim\pi}\big[\tfrac{1}{T}\sum_{t=0}^{T-1} c_i(s_t, a_t)\big]. \tag{3.2}$$

where $\{c_1, \dots, c_m\}$ with $c_i : \mathcal{S} \times \mathcal{A} \to [0, 1]$ are the cost constraints. The CMDP is then defined by

$$\max_\pi \{J(\pi)|\ C_i(\pi) \le d_i, \quad \forall\ i \in [m]\} \tag{3.3}$$

where $\{d_1, \dots, d_m\}$ are upper bounds on the average constraint costs. Note that without loss of generality both the reward and costs are random variables with a distribution supported on $[0, 1]$.

Denote the mean of reward and cost constraint functions as $\bar{r}(s, a) = \mathbb{E}[r(s, a)]$, $\bar{c}_i(s, a) = \mathbb{E}[c_i(s, a)]$ where the expectation is taken with respect to the distribution of the reward and cost function of that state-action pair (s, a). If the transition kernel $P(s'|s, a)$, the mean of the reward function $\bar{r}(s, a)$, and mean cost functions $\bar{c}_i(s, a)$ are all given, them we can solve the CMDP by solving the following linear program [50]:

$$\max_y \quad \textstyle\sum_{s,a} \bar{r}(s, a)y(s, a) \tag{3.4a}$$

$$\text{s.t.} \quad \textstyle\sum_{a'} y(s', a') = \sum_{s,a} P(s'|s, a)y(s, a) \tag{3.4b}$$

$$\textstyle\sum_{s,a} y(s, a) = 1, \quad y(s, a) \ge 0 \tag{3.4c}$$

$$\textstyle\sum_{s,a} \bar{c}_i(s, a)y(s, a) \le d_i,\ i \in [m] \tag{3.4d}$$

To simplify notation, we write the above linear program in matrix form as follows:

$$\max_y\{\ \bar{r}^\top y \mid I_o y = Py,\ \mathbf{1}^\top y = 1,\ y \geq 0,\ \bar{c}^\top y \leq d\} \tag{3.5}$$

where $\bar{r} \in \mathbb{R}^{SA}, y \in \mathbb{R}^{SA}$, $\bar{c} \in \mathbb{R}^{SA \times m}, d \in \mathbb{R}^m$, $P \in \mathbb{R}^{S \times SA}$, and $I_o \in \mathbb{R}^{S \times SA}$ is a sparse matrix built by placing S row blocks of length A in a block diagonal fashion, where each row block consists of all ones. Here, $y \in \mathbb{R}^{S \times A}$ represents the steady-state occupation measure [50] defined by

$$y(s,a) = \lim_{T\to\infty} \mathbb{E}_{\tau\sim\pi}\big[\tfrac{1}{T}\sum_{t=0}^{T-1} \mathbf{1}\{s_t = s, a_t = a\}\big]. \tag{3.6}$$

With $\bar{y}$ the solution of this linear program, the optimal stationary policy is

$$\bar{\pi}(a|s) = \bar{y}(s,a)/(\textstyle\sum_{a\in\mathcal{A}} \bar{y}(s,a)). \tag{3.7}$$

Remark 3.1. *It is worth noting that unlike in tabular MDPs without constraints, where the optimal policy is always deterministic, the optimal policy in CMDPs could be stochastic [67]. It is, in fact, trivial to solve the CMDP if the optimal policy in CMDPs is deterministic because that means the constraints are not active.*

3.4 Constrained Upper Confidence Reinforcement Learning Algorithm

Since the reward and constraint cost functions are unknown, motivated by `UCRL2`, we introduce `C-UCRL` (Algorithm 1). In general, the `C-UCRL` algorithm follows a principle of "optimism in the face of reward uncertainty; pessimism in the face of cost uncertainty." That is, it defines confidence intervals for the reward and cost of each state-action pair given the observations so far, and solves for the optimistic policy that satisfies the constraints. More specifically, in `C-UCRL`, given the current confidence interval estimates, we use a robust linear program [68] formulation to find a policy using the confidence intervals as determined at the current iteration.

In particular, in episode k, we start by executing the baseline policy π_0 for a constant h number of iterations[1]. It is common to assume a initial safe baseline policy [56] and assume

[1]The heuristic for choosing h is based on the mixing time of the Markov chain induced by π_0 given the known transition kernel for the CMDP.

Algorithm 1: Constrained UCRL (`C-UCRL`) algorithm

Input: safety parameter $\delta \in (0,1)$, baseline policy $\pi_0(a|s)$, episode length h.

Initialization: set $t = 1$, observe the initial state s_1

for episodes $k = 1, 2, \ldots, K$ **do**

$t_k = t$; `// initialize start time of episode` k

while $t \leq t_k + h$; `// Execute baseline policy` h `times for exploration` **do**

Draw action $a_t \sim \pi_0(\cdot|s_t)$

Observe reward r_t, costs $c_{i,t}$, and the next state s_{t+1}

$t \leftarrow t + 1$

end

$N_k(s,a) = \sum_{t'=1}^{t} \mathbf{1}(s_{t'} = a, a_{t'} = a),\ \forall (s,a) \in \mathcal{S} \times \mathcal{A}$; `// set the state-action count`

$R_k(s,a) = \sum_{t'=1}^{t} r_{t'} \mathbf{1}(s_{t'} = a, a_{t'} = a)$; `// compute cumulative reward`

$C_{i,k}(s,a) = \sum_{t'=1}^{t} c_{i,t'} \mathbf{1}(s_{t'} = a, a_{t'} = a)$; `// compute the cumulative costs`

$\hat{r}_k(s,a) = \frac{R_k(s,a)}{\max\{1, N_k(s,a)\}}, \quad \hat{c}_{i,k}(s,a) = \frac{C_{i,k}(s,a)}{\max\{1, N_k(s,a)\}}$; `// compute estimates`

$\tilde{y}_k \leftarrow$ arg max of (RLP) using $\tilde{r}_k(s,a)$ and $\tilde{c}_{i,k}(s,a)$ in (3.8) and (3.9), resp.

$\tilde{\pi}_k \leftarrow \tilde{y}_k(s,a) / (\sum_{a \in \mathcal{A}} \tilde{y}_k(s,a))$; `// recover policy`

while $t \leq t_k + kh$; `// Execute` $\tilde{\pi}_k$ `policy` $(k-1)h$ `times` **do**

Draw action $a_t \sim \tilde{\pi}_k(\cdot|s_t)$

Observe reward r_t, costs $c_{i,t}$, and the next state s_{t+1}

$t \leftarrow t + 1$

end

end

under such policy, the Markov chain resulting from the CMDP is irreducible and aperiodic [69]. This baseline policy could, e.g., be obtained by some prior information about which states are safe to start the conservative exploration[2]. After executing π_0, we define estimates of the reward and costs by

$$\hat{r}_k(s,a) = \frac{R_k(s,a)}{\max\{1, N_k(s,a)\}}$$

and

$$\hat{c}_{i,k}(s,a) = \frac{C_{i,k}(s,a)}{\max\{1, N_k(s,a)\}},$$

respectively, where $N_k(s,a)$, $R_k(s,a)$, and $C_{i,k}(s,a)$ are the state-action count, and cumulative reward and costs, respectively, as defined in Algorithm 1. The visitation frequency random variable $N_k(s,a)$ is defined to be the sum of indicators of whether or not the state-action pair (s,a) was visited in each iteration over all episodes. The corresponding reward $R_k(s,a)$ and constraint costs $C_{i,k}(s,a)$ are defined similarly.

Using these estimates, we define

$$\tilde{r}_k(s,a) = \min\left\{\hat{r}_k(s,a) + \left(\frac{\log(SA(m+1)\pi^2 t_k^3/3\delta)}{2\max\{1, N_k(s,a)\}}\right)^{1/2}, 1\right\} \tag{3.8}$$

and

$$\tilde{c}_{i,k}(s,a) = \min\left\{\hat{c}_{i,k}(s,a) + \left(\frac{\log(SA(m+1)\pi^2 t_k^3/3\delta)}{2\max\{1, N_k(s,a)\}}\right)^{1/2}, 1\right\}, \tag{3.9}$$

where

$$\left(\frac{\log(SA(m+1)\pi^2 t_k^3/3\delta)}{2\max\{1, N_k(s,a)\}}\right)^{1/2}$$

defines the confidence interval as we show in Section 3.5. We then use (3.8) and (3.9) to define the following robust linear program:

$$\max_y\{\ \tilde{r}_k^\top y \mid I_o y = P y,\ \mathbf{1}^\top y = 1,\ y \geq 0,\ \tilde{c}_k^\top y \leq d\}. \tag{RLP}$$

A few comments here on guaranteeing that the feasible set is non-trivial are warranted. Our analysis results are predicated on π_0 and h being chosen such that in each episode the robust

[2]Choosing π_0 is an important component of `C-UCRL`. In Section 3.6, we provide some intuitive choices for the simple examples we present, while we leave further development on how to select π_0, either heuristically or theoretically, to future work.

linear program we solve has at least one feasible solution. The duration h is chosen based on the mixing time of the induced Markov chain under the baseline policy with the goal of ensuring with high probability that the feasible set is not empty; for instance, 'sufficient' exploration will guarantee that $\tilde{c}_1^\top y \leq d$ for some $y \in \{I_o y = Py,\ \mathbf{1}^\top y = 1, y \geq 0\}$. It is possible that in the first episode, even after h iterations of executing the baseline policy, that there is no y such that $\tilde{c}_1^\top y \leq d$. A heuristic we use in practice is to run the baseline policy π_0 for as many iterations as it takes for $y_0 \in \{I_o y = Py,\ \mathbf{1}^\top y = 1,\ y \geq 0,\ \tilde{c}_1^\top y \leq d\}$. Then, we are guaranteed that in all future episodes, y_0 is always in the feasible set of (RLP). We leave further exploration of theoretically guaranteeing that the (RLP) has a non-trivial feasible set in the first episode to future work.

Returning to the description of the algorithm, in episode k, the solution $\tilde{y}_k$ to the robust linear program is then used to construct the policy $\tilde{\pi}_k$ via (3.7). This policy is executed for a linearly increasing number of iterations $(k-1)h$ where k is the episode index and h is the fixed duration used for executing the baseline policy. To summarize, for each episode of `C-UCRL`, we execute the baseline policy for h steps, estimate the reward and costs, and then execute $\tilde{\pi}_k$ for a linearly increasing (in the number of epochs) number of steps $(k-1)h$, making kh the total duration of episode k.

3.5 Theoretical Results

In this section, we summarize our analysis results. We first show that `C-UCRL` has guarantees on constraint satisfaction during learning. Then, we provide regret analysis with respect to the reward, showing that the regret is sub-linear.

3.5.1 Constraint/Safety Guarantees

To capture constraint satisfaction, we leverage the notion of δ-safety.

Definition 3.1 (δ-safe)**.** *An algorithm is δ-safe if, with probability at least $1-\delta$, for all time steps t, the policy executed by the algorithm satisfies $C_i(\pi_t) \leq d_i$, $\forall i \in [m]$.*

Following [62], we define the set of *plausible CMDPs* by the confidence intervals for the reward and each of the constraint costs. In particular, at episode k, let $\mathcal{M}_k$ be the set of plausible CMDPs with states and actions as in the underlying true CMDP M, define by all such CMDPs satisfying the following:

$$|\hat{r}_k(s,a) - \bar{r}(s,a)| \leq \left(\frac{\log(SA(m+1)\pi^2 t_k^3/3\delta)}{2\max\{1, N_k(s,a)\}}\right)^{1/2}, \tag{3.10}$$

$$|\hat{c}_{i,k}(s,a) - \bar{c}_i(s,a)| \leq \left(\frac{\log(SA(m+1)\pi^2 t_k^3/3\delta)}{2\max\{1, N_k(s,a)\}}\right)^{1/2}, \ i \in [m] \tag{3.11}$$

for all state-action pairs $(s,a) \in \mathcal{S} \times \mathcal{A}$. Let $\mathcal{M}$ be the set of plausible for all episodes k.

Lemma 3.1. *For any fixed $k \geq 1$, the probability that the true CMDP M is not contained in the set of plausible CMDPs $\mathcal{M}_k$ at episode k is at most $6\delta/(\pi^2 t_k^2)$. Furthermore, with probability at least $1-\delta$, for every state-action pair (s,a), cost c_i and episode k,* `C-UCRL` *satisfies the following:*

$$|\hat{r}_k(s,a) - \bar{r}(s,a)| \leq \left(\frac{\log(SA(m+1)\pi^2 t_k^3/3\delta)}{2\max\{1, N_k(s,a)\}}\right)^{1/2},$$

$$|\hat{c}_{i,k}(s,a) - \bar{c}_i(s,a)| \leq \left(\frac{\log(SA(m+1)\pi^2 t_k^3/3\delta)}{2\max\{1, N_k(s,a)\}}\right)^{1/2}$$

Hence, the probability that the true CMDP M is not in the set of all plausible CMDPs for any episode k is at most δ—that is, $\Pr\{M \notin \mathcal{M}\} \leq \delta$.

Proof. Consider any fixed state-action pair (s,a) and its visitation frequency $N_k(s,a)$ up to episode k. If the state-action pair (s,a) has not been visited, then (3.10) and (3.11) trivially hold since $N_k(s,a) = 0$ by definition and the right-hand sides of (3.10) and (3.11) are greater than one when $N_k(s,a) = 0$.

On the other hand, if $N_k(s,a)$ is not zero, meaning the state-action pair has been visited, then since for each (s,a) pair, the reward and constraint costs are all supported on $[0,1]$ and independent identically distributed (iid) real-valued random variables, we can apply Hoeffding's inequality to get a bound on the deviation between the true mean $\bar{r}(s,a)$ (respectively, $\bar{c}_i(s,a)$) and the empirical mean $\hat{r}_k(s,a)$ (respectively, $\hat{c}_{i,k}(s,a)$) given n iid samples of the state-action pair (s,a):

$$\Pr\left\{|\hat{r}_k(s,a) - \bar{r}(s,a)| \geq \epsilon\right\} \leq 2\exp(-2n\epsilon^2) \tag{3.12}$$

Consider

$$\epsilon = \left(\tfrac{1}{2n}\log\left(\tfrac{SA(m+1)\pi^2 t_k^3}{3\delta}\right)\right)^{1/2},$$

then

$$\Pr\left\{|\hat{r}_k(s,a) - \bar{r}(s,a)| \geq \left(\tfrac{1}{2n}\log\left(\tfrac{SA(m+1)\pi^2 t_k^3}{3\delta}\right)\right)^{1/2}\right\} \leq 2\exp\left(-2n\tfrac{1}{2n}\log\left(\tfrac{SA(m+1)\pi^2 t_k^3}{3\delta}\right)\right)$$
$$= \tfrac{6\delta}{SA(m+1)\pi^2 t_k^3}$$

Similarly, for each state-action pair (s,a) and constraint cost indexed by i,

$$\Pr\left\{|\hat{c}_{i,k}(s,a) - \bar{c}_i(s,a)| \geq \left(\tfrac{1}{2n}\log\left(\tfrac{SA(m+1)\pi^2 t_k^3}{3\delta}\right)\right)^{1/2}\right\} \leq \tfrac{6\delta}{SA(m+1)\pi^2 t_k^3}. \tag{3.13}$$

Noting that from the above argument, the confidence intervals hold with probability one when (s,a) has not be visited, taking a union bound over all possible values of $n \in \{1,\ldots,t_k\}$ gives

$$\Pr\left\{\cup_{n=1}^{t_k}\left\{|\hat{r}_k(s,a) - \bar{r}(s,a)| \geq \left(\tfrac{\log(SA(m+1)\pi^2 t_k^3/3\delta)}{2\max\{1,N_k(s,a)\}}\right)^{1/2}\right\}\right\} \leq \textstyle\sum_{n=1}^{t_k} \tfrac{6\delta}{SA(m+1)\pi^2 t_k^3} = \tfrac{6\delta}{SA(m+1)\pi^2 t_k^2}$$

and

$$\Pr\left\{\cup_{n=1}^{t_k}\left\{|\hat{c}_{i,k}(s,a) - \bar{c}_i(s,a)| \geq \left(\tfrac{\log(SA(m+1)\pi^2 t_k^3/3\delta)}{2\max\{1,N_k(s,a)\}}\right)^{1/2}\right\}\right\} \leq \textstyle\sum_{n=1}^{t_k} \tfrac{6\delta}{SA(m+1)\pi^2 t_k^3} = \tfrac{m6\delta}{SA(m+1)\pi^2 t_k^2}$$

where we have now written $N_k(s,a)$ for the number of visits in (s,a) up to episode k. This proves (3.10) and (3.11).

Now, further union bounding over all state-action pairs (s,a) gives

$$\Pr\left\{\cup_{n=1}^{t_k}\cup_{s,a}\left\{|\hat{r}_k(s,a) - \bar{r}(s,a)| \geq \epsilon_r(n)\right\}\right\} \leq \textstyle\sum_{n=1}^{t_k}\sum_{s,a} \tfrac{6\delta}{SA(m+1)\pi^2 t_k^3} = \tfrac{6\delta}{(m+1)\pi^2 t_k^2} \tag{3.14}$$

for the reward. Analogously, taking a further union bound over all state-action pairs (s,a) and all constraint costs $i \in [m]$, gives

$$\Pr\left\{\cup_{n=1}^{t_k}\cup_{s,a,i}\left\{|\hat{c}_{i,k}(s,a) - \bar{c}_i(s,a)| \geq \epsilon_r(n)\right\}\right\} \leq \textstyle\sum_{n=1}^{t_k}\sum_{s,a,i} \tfrac{6\delta}{SA(m+1)\pi^2 t_k^3} = \tfrac{m6\delta}{(m+1)\pi^2 t_k^2} \tag{3.15}$$

for the constraint costs. Summing (3.14) and (3.15), we get the first claim of the lemma—i.e.,

$$\Pr\{M \notin \mathcal{M}_k\} \leq \tfrac{6\delta}{\pi^2 t_k^2}$$

Now, since $\sum_{\ell=1}^{\infty} \frac{1}{\ell^2} = \frac{\pi^2}{6}$, if in (3.14) and (3.15), we additionally union bounded over all episodes $k \in \{1, \ldots, \infty\}$, we get that

$$\Pr\left\{ \cup_{t_k=1}^{\infty} \cup_{n=1}^{t_k} \cup_{s,a} \left\{ |\hat{r}_k(s,a) - \bar{r}(s,a)| \geq \epsilon_r(n) \right\} \right\} \leq \sum_{t_k=1}^{\infty} \sum_{n=1}^{t_k} \sum_{s,a} \frac{6\delta}{SA(m+1)\pi^2 t_k^3} = \frac{\delta}{m+1}$$

and

$$\Pr\left\{ \cup_{t_k=1}^{\infty} \cup_{n=1}^{t_k} \cup_{s,a,i} \left\{ |\hat{c}_{i,k}(s,a) - \bar{c}_i(s,a)| \geq \epsilon_r(n) \right\} \right\} \leq \sum_{t_k=1}^{\infty} \sum_{n=1}^{t_k} \sum_{s,a,i} \frac{6\delta}{SA(m+1)\pi^2 t_k^3} = \frac{m\delta}{m+1}$$

so that

$$\Pr\{M \notin \mathcal{M}\} \leq \delta$$

which proves the final statement in the lemma. □

Given that, for each episode, we can bound the gaps between the estimated reward (respectively, costs) and the mean reward (respectively, mean costs), with probability $1 - \delta$, we can provide an assurance on C-UCRL being δ-safe.

Theorem 3.1. C-UCRL *is δ-safe.*

Proof. According to Lemma 3.1, with probability at least $1 - \delta$, $\bar{c}_i(s,a) \leq \tilde{c}_{i,k}(s,a)$. The occupation measure $\tilde{y}_k$ obtained at each episode via (RLP) satisfies $\sum_{s,a} \tilde{c}_{i,k}(s,a)\tilde{y}_k(s,a) \leq d_i$. Hence, $C_i(\tilde{\pi}_k) = \sum_{s,a} \bar{c}_i(s,a)\tilde{y}_k(s,a) \leq d_i$ with probability $1 - \delta$. □

3.5.2 Regret Analysis of C-UCRL

Given that we have shown that C-UCRL is δ-safe, we now analyze the reward regret. In episode k of C-UCRL, we execute a baseline policy π_0 for h times and policy $\tilde{\pi}_k$ for $(k-1)h$ times. The pseudo-regret of episode k is given by

$$\Delta_k = h[J(\bar{\pi}) - J(\pi_0)] + (k-1)h[J(\bar{\pi}) - J(\tilde{\pi}_k)] = h\bar{r}^\top(\bar{y} - y_0) + (k-1)h\bar{r}^\top(\bar{y} - \tilde{y}_k).$$

We first upper bound the per-step pseudo-regret of executing policy $\tilde{\pi}_k$, $\bar{r}^\top(\bar{y} - \tilde{y}_k)$, where the first term is the expected average reward under the optimal policy $\bar{\pi}$ and the second term is the sub-optimal expected average reward under policy $\tilde{\pi}_k$.

Using the confidence bounds in Lemma 3.1, define

$$\epsilon_r(s,a) = \left(\frac{\log(SA(m+1)\pi^2 t_k^3/3\delta)}{2\max\{1,N_k(s,a)\}}\right)^{1/2}, \tag{3.16}$$

and $\epsilon_c(s,a) = \epsilon_r(s,a)$ for each state-action pair and let ϵ_r and ϵ_c denote the vectors containing the values across all state-action pairs[3]. Define the following two linear programs:

$$\max_y\{r^\top y | Ay = 0, \mathbf{1}^\top y = 1, y \geq 0, c^\top y \leq d\} \tag{3.17}$$

$$\max_y\{(r+\epsilon_r)^\top y | Ay = 0, \mathbf{1}^\top y = 1, y \geq 0, (c+\epsilon_c)^\top y \leq d\}. \tag{3.18}$$

where $0 \leq r \leq \mathbf{1}$, $0 \leq c \leq \mathbf{1}$, $\epsilon_r \geq 0$, and $\epsilon_c \geq 0$ hold element wise.

Lemma 3.2. *Assuming the domains of* (3.17) *and* (3.18) *are not empty, let* y_1 *and* y_2 *be solutions for each of the problems, respectively. If, for some constant* $\alpha > 0$ *and* $\beta > 0$*, there exist* $y_0 \in \{y | Ay = 0, \mathbf{1}^\top y = 1, y \geq 0, (c+\epsilon_c)^\top y \leq d\}$ *such that* $r^\top(y_1 - y_0) = \alpha > 0$ *and* $c^\top(y_1 - y_0) = \beta > 0$*, then* $r^\top(y_1 - y_2) \leq \frac{2\alpha}{\beta}\|\epsilon_c\|_1 + \|\epsilon_r\|_1$.

Proof. Let

$$y_3 = \arg\max_y\{r^\top y | Ay = 0, \mathbf{1}^\top y = 1, y \geq 0, (c+\epsilon_c)^\top y \leq d\}.$$

We first find the upper bound of $r^\top(y_1 - y_3)$ where we note that y_3 and y_1 are the solutions of same linear program over different domains. Since the domain of y_3 is smaller than y_1, we know that $r^\top(y_1 - y_3) \geq 0$. First, consider the trivial case that y_1 satisfies $(c+\epsilon_c)^\top y_1 \leq d$. In this case, $y_1 = y_3$ and $r^\top(y_1 - y_3) = 0$. Now we only consider the case such that $(c+\epsilon_c)^\top y_1 > d$. Note that $(c+\epsilon_c)^\top y_0 \leq d$. Hence, there exists a $\gamma \in [0,1)$ such that $y_4 = y_0 + \gamma(y_1 - y_0)$ and $(c+\epsilon_c)^\top y_4 = d$—i.e., $\gamma = (d - (c+\epsilon_c)^\top y_0)/((c+\epsilon_c)^\top(y_1 - y_0))$. Further, we have

$$y_1 - y_4 = y_1 - y_0 - \gamma(y_1 - y_0) = (1-\gamma)(y_1 - y_0),$$

[3]We note that it is possible to define separate confidence bounds for the reward and constraint costs, however, for simplicity of the statement and proof of Lemma 3.1, we define them to be the same.

so that $c^\top(y_1 - y_4) = (1-\gamma)\beta > 0$ and

$$c^\top(y_1 - y_4) = (c+\epsilon_c)^\top(y_1 - y_4) - \epsilon_c^\top(y_1 - y_4) \tag{3.19}$$
$$= c^\top y_1 + \epsilon_c^\top y_1 - d - \epsilon_c^\top(y_1 - y_4) \tag{3.20}$$
$$\leq d - d + \epsilon_c^\top y_1 - \epsilon_c^\top(y_1 - y_4) \tag{3.21}$$
$$\leq \|\epsilon_c\|_1\|y_1\|_\infty + \|\epsilon_c\|_1\|y_1 - y_4\|_\infty \tag{3.22}$$
$$= 2\|\epsilon_c\|_1 \tag{3.23}$$

Combining this bound with

$$\frac{r^\top(y_1 - y_4)}{c^\top(y_1 - y_4)} = \frac{r^\top(y_1 - y_0)}{c^\top(y_1 - y_0)} = \frac{\alpha}{\beta}, \tag{3.24}$$

we have that

$$0 < r^\top(y_1 - y_4) \leq 2\tfrac{\alpha}{\beta}\|\epsilon_c\|_1.$$

Since the domain for each of these problems is convex, we know that

$$y_4 \in \{y | Ay = 0, \mathbf{1}^\top y = 1, y \geq 0, (c+\epsilon_c)^\top y \leq d\}.$$

Due to optimality, $r^\top y_3 \geq r^\top y_4$ so that

$$r^\top(y_1 - y_3) \leq 2\tfrac{\alpha}{\beta}\|\epsilon_c\|_1.$$

We leverage the bounud on $r^\top(y_1 - y_3)$ to obtain a bound on $r^\top(y_3 - y_2)$. Note that y_3 and y_2 are the solutions of two linear programs with different objectives but the same domain. According to optimality of the solutions, we know that $r^\top y_3 \geq r^\top y_2$ and $(r+\epsilon_r)^\top y_2 \geq (r+\epsilon_r)^\top y_3$. Combining these facts, we have that

$$0 \leq r^\top(y_3 - y_2) \leq \epsilon_r^\top(y_2 - y_3) \leq \|\epsilon_r\|_1\|y_2 - y_3\|_\infty \leq \|\epsilon_r\|_1 \tag{3.25}$$

Now, combining the bounds on $r^\top(y_3 - y_2)$ and $r^\top(y_1 - y_3)$, we have that

$$r^\top(y_1 - y_2) = r^\top(y_1 - y_3) + r^\top(y_3 - y_2) \leq \tfrac{2\alpha}{\beta}\|\epsilon_c\|_1 + \|\epsilon_r\|_1. \tag{3.26}$$

□

We can use the preceding lemma to get a bound on the pseudo-regret.

Proposition 3.1. *Denote* $\mathcal{Y} = \{y|(I_o - P)y = 0, \mathbf{1}^\top y = 1, y \geq 0\}$. *If there exists* $y_0 \in \mathcal{Y}$ *such that* $\bar{r}^\top(\bar{y} - y_0) \geq \alpha > 0, \bar{c}_i^\top(\bar{y} - y_0) \geq \beta > 0$, *then with probability at least* $1 - \delta$,

$$\bar{r}^\top(\bar{y} - \tilde{y}_k) \leq 2(\tfrac{2\alpha m}{\beta} + 1) \textstyle\sum_{s,a} \left(\frac{\log(SA(m+1)\pi^2 t_k^3/3\delta)}{2\max\{1, N_k(s,a)\}}\right)^{1/2}. \tag{3.27}$$

Proof. By definition

$$\bar{y} = \arg\max_y \{\bar{r}^\top y | (I_o - P)y = 0, \mathbf{1}^\top y = 1, y \geq 0, \bar{c}_i^\top y \leq d_i, i \in [m]\}$$

and

$$\tilde{y}_k = \arg\max_y \{\tilde{r}_k^\top y | (I_o - P)y = 0, \mathbf{1}^\top y = 1, y \geq 0, \tilde{c}_{i,k}^\top y \leq d_i, i \in [m]\}.$$

Define a sequence of subproblems by adding the confidence value to one additional constraint at a time as follows:

$$\begin{aligned}
y^{(1)} &= \arg\max_y \{\bar{r}^\top y | (I_o - P)y = 0, \mathbf{1}^\top y = 1, y \geq 0, \tilde{c}_{1,k}^\top y \leq d_1, \bar{c}_i^\top y \leq d_i, i \in \{2, \ldots, m\}\} \\
y^{(2)} &= \arg\max_y \{\bar{r}^\top y | (I_o - P)y = 0, \mathbf{1}^\top y = 1, y \geq 0, \tilde{c}_{1,k}^\top y \leq d_1, \tilde{c}_{2,k}^\top y \leq d_2, \bar{c}_i^\top y \leq d_i, i \in \{3, \ldots, m\}\} \\
&\vdots \\
y^{(m)} &= \arg\max_y \{\bar{r}^\top y | (I_o - P)y = 0, \mathbf{1}^\top y = 1, y \geq 0, \tilde{c}_{i,k}^\top y \leq d_i, i \in \{1, \ldots, m\}\}
\end{aligned}$$

Using the same proof technique as for that of Lemma 3.2, we obtain the bounds for each of the subproblems

$$\bar{r}^\top(\bar{y} - y^{(1)}), \bar{r}^\top(y^{(1)} - y^{(2)}), \ldots, \bar{r}^\top(y^{(m-1)} - y^{(m)}), \bar{r}^\top(y^{(m)} - \tilde{y}_k).$$

Combining each of the bounds and the fact that

$$|\tilde{r}_k(s,a) - \bar{r}(s,a)| \leq 2\sqrt{\tfrac{\log(SA(m+1)\pi^2 t_k^3/3\delta)}{2\max\{1, N_k(s,a)\}}},$$

and

$$|\tilde{c}_{i,k}(s,a) - \bar{c}_i(s,a)| \leq 2\sqrt{\tfrac{\log(SA(m+1)\pi^2 t_k^3/3\delta)}{2\max\{1, N_k(s,a)\}}},$$

we have that

$$\bar{r}^\top(\bar{y}-\tilde{y}_k)=\bar{r}^\top(\bar{y}-y^{(1)})+\cdots+\bar{r}^\top(y^{(m)}-\tilde{y}_k)\leq m\tfrac{2\alpha}{\beta}\|\tilde{c}_k-\bar{c}\|_1+\|\tilde{r}_k-\bar{r}\|_1$$
$$\leq 2(\tfrac{2\alpha m}{\beta}+1)\textstyle\sum_{s,a}\sqrt{\frac{\log(SA(m+1)\pi^2t_k^3/3\delta)}{2\max\{1,N_k(s,a)\}}}$$

which completes the proof. □

Note that according to Proposition 3.1, with probability at least $1-\delta$, the per-step pseudo-regret of executing policy $\tilde{\pi}_k$ depends on the confidence intervals of reward and costs of all state-action pairs. This is intuitive since in order for the policy $\tilde{\pi}_k$ to be close to the optimal policy $\bar{\pi}$, we need to have good approximations of the reward and costs for all state-action pairs. To ensure this, we need to constantly explore the CMDP so that $N_k(s,a)$ is not 'too small' for any state-action pair. Since the Markov chain resulting from the baseline policy is irreducible and aperiodic, the steady state occupation measure $y_0(s,a)$ corresponding to the baseline policy $\pi_0(a|s)$ has the property that $y_0(s,a)>0, \forall s,a$. Due to this universal exploration demand, we execute the baseline policy π_0 for a constant number of times in each linear increasing episode in the `C-UCRL` algorithm.

To have a upper bound on the regret derived in Proposition 3.1, we need to have a lower bounds on $N_k(s,a)$. Given our assumptions on the baseline policy as discussed above, define $\rho>0$ such that $y_0(s,a)\geq\rho>0$ for all state-action pairs $(s,a)\in\mathcal{S}\times\mathcal{A}$. The following lemma gives a lower bound on the number of times each state-action pair is visited in episode k.

Lemma 3.3. *Given a fixed total number of episodes K, with probability at least $1-\delta$, for every state-action pair (s,a) and episode $k\in[K]$,*

$$N_k(s,a)\geq(k-1)\rho h-(k-1)\big(72\xi\rho h\log\big(\tfrac{\varphi\cdot SAK}{\delta}\big)\big)^{1/2} \tag{3.28}$$

where ξ the mixing time of the Markov chain induced by policy π_0, $\rho>0$ is such that $y_0(s,a)\geq\rho>0$ for all state-action pairs $(s,a)\in\mathcal{S}\times\mathcal{A}$, and $\varphi=\sum_{s,a}\frac{y'(s,a)^2}{y_0(s,a)}$, where y' is the initial state action distribution and y_o is the steady state action distribution under the baseline policy.

Proof. Consider the exploration phase (when the baseline policy π_0 is executed) of the k-th episode in Algorithm 1. For a given episode ℓ and for a fixed state-action pair (s,a), let $X_{\ell,1},\ldots,X_{\ell,h}$ be the indicator variables of whether state-action pair (s,a) has be selected at each step within the episode ℓ. Let $Y_\ell = \sum_{i=1}^{h} X_{\ell,i}$ and thus $\mathbb{E}[Y_\ell] = y_0(s,a)h$. Applying the Chernoff-Hoeffding bound in [70, Theorem 3], gives

$$\Pr\{\mathbb{E}[Y_\ell] - Y_\ell \geq \epsilon y_0(s,a)h\} \leq \varphi \cdot \exp\left(-\tfrac{\epsilon^2 y_0(s,a)h}{72\xi}\right). \tag{3.29}$$

Setting

$$\epsilon = \sqrt{\tfrac{72\xi}{y_0(s,a)h}\log(\tfrac{\varphi SAK}{\delta})}, \tag{3.30}$$

the above bound becomes

$$\Pr\left\{Y_\ell \leq y_0(s,a)h - \sqrt{72\xi y_0(s,a)h\log(\tfrac{\varphi SAK}{\delta})}\right\} \leq \tfrac{\delta}{SAK} \tag{3.31}$$

Using the assumption that $y_0(s,a) \geq \rho > 0, \forall (s,a)$, the union bound over all state-action pairs (s,a) and episodes $k \in [K]$ is given by

$$\Pr\left\{\bigcup\nolimits_{(s,a),\ell}\left\{Y_\ell \leq \rho h - \sqrt{72\xi\rho h\log(\tfrac{\varphi SAK}{\delta})}\right\}\right\} \leq \sum_{\ell=1}^{K}\sum_{s,a}\tfrac{\delta}{SAK} = \delta \tag{3.32}$$

Now, we note that

$$\left\{\sum_{\ell=1}^{k-1} Y_\ell \leq (k-1)\left(\rho h - \sqrt{72\xi\rho h\log(\tfrac{\varphi SAK}{\delta})}\right)\right\} \subset \bigcup_{\ell=1}^{k-1}\left\{Y_\ell \leq \rho h - \sqrt{72\xi\rho h\log(\tfrac{\varphi SAK}{\delta})}\right\}$$

and $N_k(s,a) \geq \sum_{\ell=1}^{k-1} Y_\ell$ since in each episode $\tilde{\pi}_\ell$ is executed $h\ell$ times after the baseline policy so that $N_\ell(s,a)$ may be larger. Hence,

$$N_k(s,a) \geq (k-1)\rho h - (k-1)\sqrt{72\xi\rho h\log(\tfrac{\varphi SAK}{\delta})} \tag{3.33}$$

holds with probability at least $1-\delta$. □

Combining Proposition 3.1 and Lemma 3.3 and summing over K episodes, we obtain the total regret bound for `C-UCRL`.

Theorem 3.2. *Suppose that* $\delta \leq \varphi SAK \exp(-\frac{\rho h}{288\xi})$. *Under the assumptions of Proposition 3.1, with probability at least* $1-\delta$, `C-UCRL` *has total pseudo-regret* $\Delta(T) = O(T^{\frac{3}{4}}\sqrt{\log(T/\delta)})$.

Proof. According to Proposition 3.1, the total regret of K episodes is

$$\begin{aligned}\sum_{k=1}^{K}\Delta_k &= \sum_{k=1}^{K} h\bar{r}^\top(\bar{y}-y_0) + (k-1)h\bar{r}^\top(\bar{y}-\tilde{y}_k)\\ &= hK\bar{r}^\top(\bar{y}-y_0) + h\sum_{k=2}^{K}(k-1)\bar{r}^\top(\bar{y}-\tilde{y}_k)\\ &\leq 2hK + 2(\tfrac{2\alpha m}{\beta}+1)h\sum_{k=2}^{K}(k-1)\sum_{s,a}\sqrt{\tfrac{\log(SA(m+1)\pi^2 t_k^3/3\delta)}{2\max\{1,N_k(s,a)\}}}\end{aligned}$$

Let

$$\zeta = \rho h - \left(72\xi\rho h\log(\tfrac{\varphi SAK}{\delta})\right)^{1/2}.$$

Since $\delta \leq \varphi SAK\exp(-\frac{\rho h}{288\xi})$, we have that

$$\left(72\xi\rho h\log(\frac{\varphi SAK}{\delta})\right)^{1/2} \geq \frac{1}{2}\rho h$$

so that $\zeta \leq \frac{1}{2}\rho h$.

Combining this with Lemma 3.3, we have that

$$\begin{aligned}\Delta(T) = \sum_{k=1}^{K}\Delta_k &\leq 2hK + 2(\tfrac{2\alpha m}{\beta}+1)h\sum_{k=2}^{K}(k-1)\sum_{s,a}\left(\tfrac{\log(SA(m+1)\pi^2 t_k^3/3\delta)}{2\max\{1,N_k(s,a)\}}\right)^{1/2}\\ &\leq 2hK + 2(\tfrac{2\alpha m}{\beta}+1)hSA\sum_{k=2}^{K}(k-1)\left(\tfrac{\log(SA(m+1)\pi^2 t_k^3/3\delta)}{2(k-1)\zeta}\right)^{1/2}\\ &\leq 2hK + 2(\tfrac{2\alpha m}{\beta}+1)hSA\sqrt{\log(SA(m+1)\pi^2 T^3/3\delta)}\sum_{k=1}^{K-1}\sqrt{\tfrac{k}{2\zeta}}\\ &= 2hK + 2(\tfrac{2\alpha m}{\beta}+1)hSA\sqrt{\tfrac{\log(SA(m+1)\pi^2 T^3/3\delta)}{2\zeta}}\sum_{k=1}^{K-1}\sqrt{k}\\ &\leq 2hK + 2(\tfrac{2\alpha m}{\beta}+1)hSA\sqrt{\tfrac{\log(SA(m+1)\pi^2 T^3/3\delta)}{\rho h}}\sum_{k=1}^{K-1}\sqrt{k}\\ &\leq 2hK + 2(\tfrac{2\alpha m}{\beta}+1)hSA\sqrt{\tfrac{\log(SA(m+1)\pi^2 T^3/3\delta)}{\rho h}}(K-1)\left(\tfrac{K}{2}\right)^{1/2}\\ &= O(K) + O(K\sqrt{K\log(T/\delta)})\\ &\leq O(T^{\frac{3}{4}}\sqrt{\log(T/\delta)})\end{aligned}$$

where the second to last inequality follows from Jensen's inequality and the final step follows from $T = \sum_{k=1}^{K} kh = \frac{K(K-1)}{2}h$ so that $K < (2T/h)^{1/2}$.

□

Remark 3.2. *Adding constants related to the dimension of the CMDP, we have the regret bound*

$$\Delta(T) \leq O(mSAT^{\frac{3}{4}}\sqrt{\log(mSAT/\delta)}). \tag{3.34}$$

3.5.3 Specializing to the Constrained Multi-Armed Bandit Setting

Constrained Multi-Armed Bandits (CMABs) can be viewed as a special case of CMDPs, where there is only one state, $S = 1$ and the transition kernel is trivially staying in that state with all actions. The policy in a CMAB is a probabilistic distribution over actions/arms $y(a)$ and the goal is to solve the following linear program:

$$\max_{y}\{\bar{r}^\top y \mid \mathbf{1}^\top y = 1, y \geq 0, \bar{c}^\top y \leq d\}. \tag{3.35}$$

Similarly, the per-step pseudo-regret is defined as $\bar{r}^\top(\bar{y} - \tilde{y}_k)$ where $\bar{y}$ is the optimal randomized policy and $\tilde{y}_k$ is the policy execute in episode k of `C-UCRL`. Running `C-UCRL` with $S = 1$, the following corollaries hold.

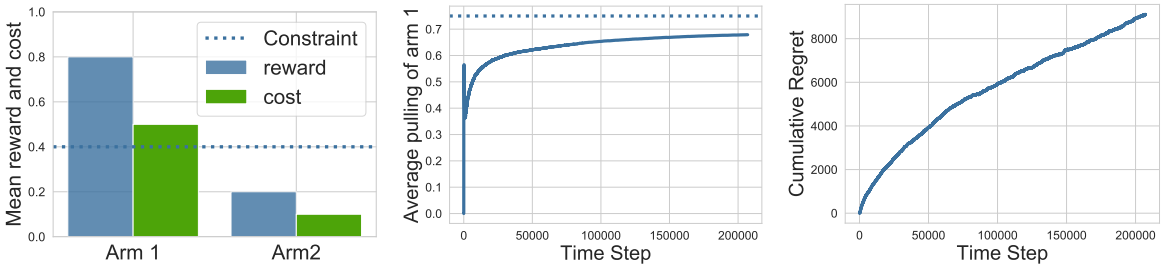
Corollary 3.1. *In CMABs,* `C-UCRL` *is δ-safe.*

Corollary 3.2. *In CMABs, Under the assumptions of Proposition 3.1, with probability at least $1 - \delta$,* `C-UCRL` *has total pseudo-regret $\Delta(T) = O(T^{\frac{3}{4}}\sqrt{\log(T/\delta)})$.*

The proofs of the above two corollaries follow directly from the corresponding results in the preceding section.

3.6 Experiments

The goal of this section is to explore a few illustrative examples which highlight different features of our approach.



(a) (b) (c)

Figure 3.2: Two armed bandit with per-round budget constraint: (a) mean reward and cost of each arm as well as the per-round constraint; (b) average number of times arm one is pulled; (c) the cumulative regret of `C-UCRL`.

3.6.1 Two Armed Bandit with per Round Budget Constraints

We first consider a simple two arms bandit example. As stated before, the CMDP reduces to a constrained multi-armed bandit problem when $|\mathcal{S}| = 1$. The reward and cost of each arm are unknown and stochastic. In our simulation, the reward and cost is draw from a binomial distribution, with the mean shown in Figure 3.2(a). Even though arm one has a better reward, we cannot pull arm one all the time since the constraint is set to be less than the mean cost of arm one. The optimal policy is to pull arm one with probability 0.75 and arm two with probability 0.25. The baseline policy we use to start exploration is pulling the two arms uniformly at random. Figure 3.2(b) and 3.2(c) show the average number of times arm one is pulled and the cumulative regret of `C-UCRL`, respectively. The average pull count of arm one never exceeds 0.75.

3.6.2 Three State CMDP

To demonstrate the performance of `C-UCRL`, we consider a simple three state CMDP. As show in Figure 3.3(a), the CMDP we consider has three states and two actions. An agent can take either a *risky* exploratory action in which the navigate to another state or they can take the *safe* action and remain in the current state. There is no reward or cost for

Algorithm 2: risk-sensitive UCRL2 (`RS-UCRL2`) algorithm

Input: safety parameter $\delta \in (0,1)$, baseline policy $\pi_0(a|s)$, episode length h, risk sensitive parameter λ.

Initialization: set $t = 1$, observe the initial state s_1

for episodes $k = 1, 2, \ldots, K$ **do**

$t_k = t$; `// initialize start time of episode` k

while $t \leq t_k + h$; `// Execute baseline policy` h `times for exploration` **do**

Draw action $a_t \sim \pi_0(\cdot|s_t)$

Observe reward r_t, costs $c_{i,t}$, and the next state s_{t+1}

$t \leftarrow t + 1$

end

$N_k(s,a) = \sum_{t'=1}^{t} \mathbf{1}(s_{t'} = a, a_{t'} = a),\ \forall (s,a) \in \mathcal{S} \times \mathcal{A}$; `// set the state-action count`

$R_k(s,a) = \sum_{t'=1}^{t} (r_{t'} - \lambda^\top c_{t'}) \mathbf{1}(s_{t'} = a, a_{t'} = a)$; `// cumulative reward cost trade-off`

$\hat{r}_k(s,a) = \frac{R_k(s,a)}{\max\{1, N_k(s,a)\}}, \quad \tilde{r}_k(s,a) = \hat{r}_k(s,a) + \sqrt{\frac{7 \log(2SAt_k/\delta)}{2 \max\{1, N_k(s,a)\}}}$; `// compute estimates`

$\tilde{y}_k \leftarrow \arg\max\{\tilde{r}_k^\top y | I_o y = Py, \mathbf{1}^\top y = 1, y \geq 0\}$

$\tilde{\pi}_k \leftarrow \tilde{y}_k(s,a) / (\sum_{a \in \mathcal{A}} \tilde{y}_k(s,a))$; `// recover policy`

while $t \leq t_k + kh$; `// Execute` $\tilde{\pi}_k$ `policy` $(k-1)h$ `times` **do**

Draw action $a_t \sim \tilde{\pi}_k(\cdot|s_t)$

Observe reward r_t, costs $c_{i,t}$, and the next state s_{t+1}

$t \leftarrow t + 1$

end

end

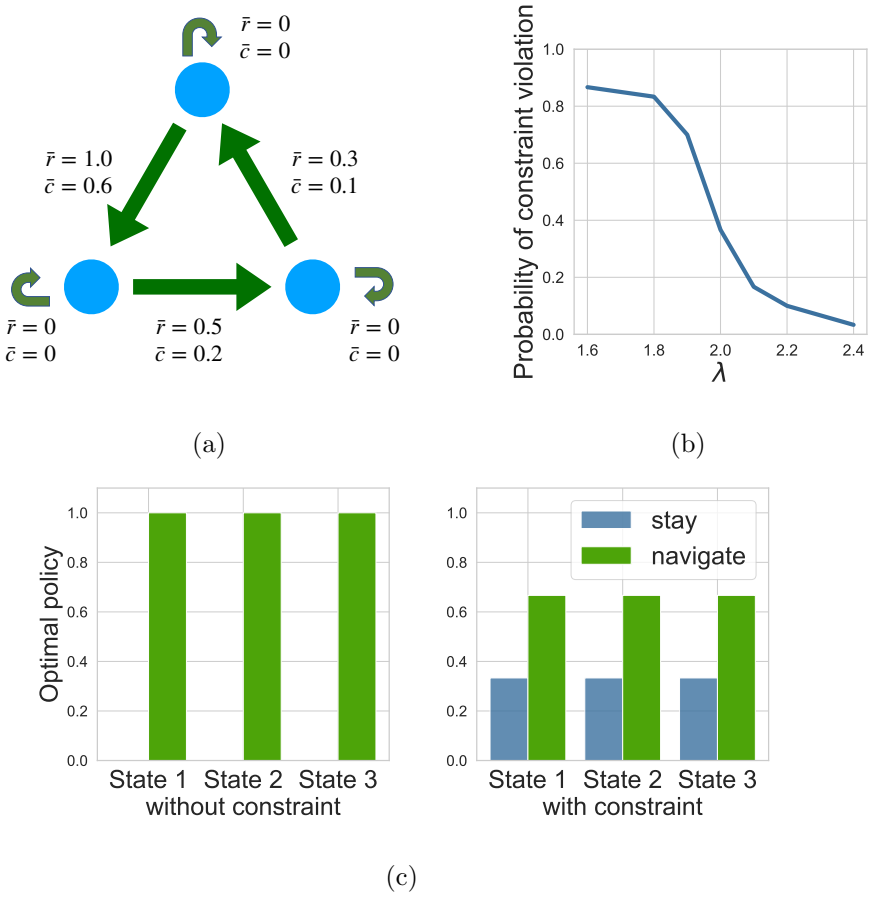
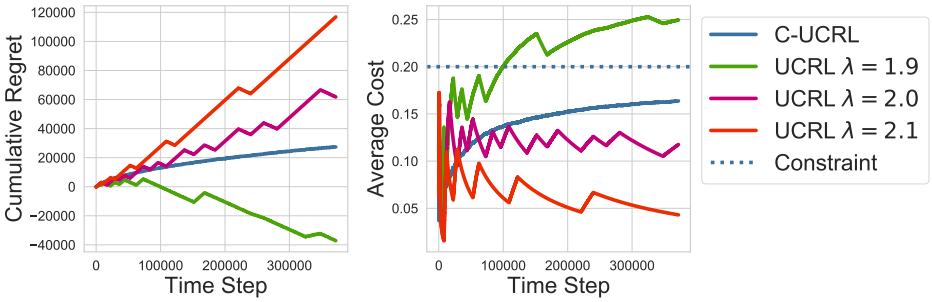
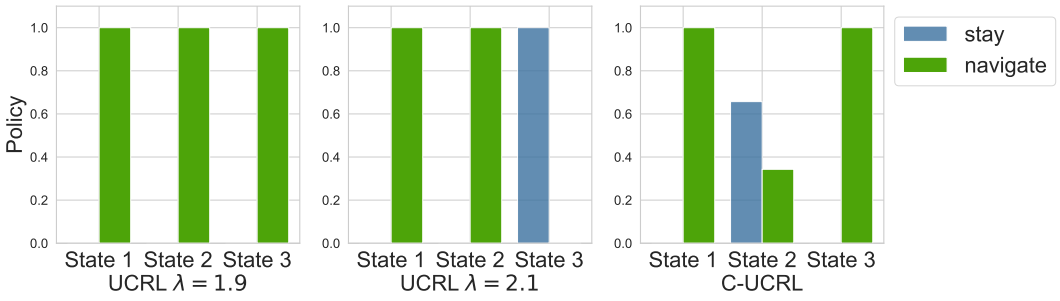


Figure 3.3: Simple CMDP. (a) CMDP structure; (b) probability of constraint violation in 30 training episodes by risk-sensitive `UCRL2` (`RS-UCRL2`); (c) optimal policy computed with the true mean reward and mean cost, with and without the constraint on cost, $d = 0.2$.

staying in the current state but there will be a stochastic reward and cost if the agent navigates. In the simulation, the reward and cost of each state-action pair are each draw from a binomial distribution, with the means defined in the labels on edges in Figure 3.3(a). Obviously, without this constraint, the optimal policy is to navigate in each of the states. In this problem, we consider the constraint that in expectation, the average cost should be less than 0.2. This constraint prevents the agents from continuously navigating between the three states. In particular, as shown in Figure 3.3(c), the constrained optimal policy



(a)



(b)

Figure 3.4: `C-UCRL` vs. `RS-UCRL2`: (a) Cumulative regret and average cost for `C-UCRL` and risk sensitive `UCRL2`; (b) Policy learned by `C-UCRL` and `RS-UCRL2`.

is a randomized policy that has positive probability on the safe action in each state. The relatively conservative baseline policy we use in `C-UCRL` for exploration is staying in the current state with probability 0.8 and navigate to the next state with probability 0.2.

We compare our approach with the `UCRL2` algorithm. However, `UCRL2` does not allow for constraints or multiple reward/cost criteria. Hence, we leverage the idea of risk sensitive reinforcement learning [49, 47], where we treat a linear combination of reward and cost—i.e., $r - \lambda c$—as the reward for the `UCRL2` algorithm (Algorithm 2). The hyperparameter λ represents the trade off between the reward and cost, the combination of which represents the reward in the classical implementation of `UCRL2`; we refer to risk-sensitive `UCRL2` by `RS-UCRL2`. Figure 3.3(b) shows the constraint violation probability in 30 training episodes by `RS-UCRL2`

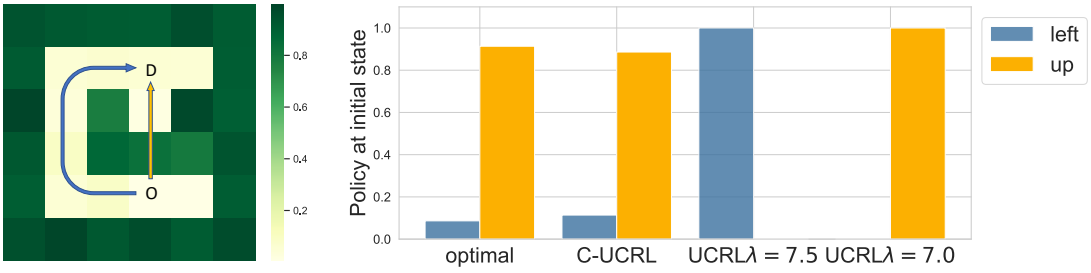
algorithm with different λ. Figure 3.4(a) shows the cumulative regret and average cost of the `C-UCRL` and `RS-UCRL2` algorithms. As we can see, when the cost value is underestimated ($\lambda = 1.9$), applying `RS-UCRL2` directly leads to a 'good' reward (i.e., the regret is negative as it gets more reward than the optimal randomized policy), yet the constraints are violated. On the other hand, when the costs are overestimated ($\lambda = 2.1$), `RS-UCRL2` is too conservative about the cost and, thus, receives high regret. We can observe that `C-UCRL` does not violate the constraint during learning though in this experiment, δ is set to be 0.1, meaning that with probability at least 0.9, the constraint will not be violated in all episodes.

The fundamental problem with `RS-UCRL2` is that with only one criterion, the policy it learns will always be a deterministic policy, while in this CMDP, the optimal policy is randomized. Figure 3.4(b) shows the policy learned by `C-UCRL` and `RS-UCRL2`. When $\lambda = 1.9$, `RS-UCRL2` learn the optimal policy as there is no constraint, which leads to constraint violation. When $\lambda = 2.1$, the policy learned by `RS-UCRL2` is to stay in one state forever. On the contrary, the policy learned by `C-UCRL` algorithm converges to the optimal randomized policy.

3.6.3 Grid World with Safety Constraints

Motivated by the goal of ensuring safety in reinforcement learning safety, we validate our algorithms using a 2D grid-world exploration problem [47, 2.24]. This example also represents a crude abstraction of rovers exploring the surface of Mars as described in [60].

Figure 3.5(a) shows the CMDP structure. The green color in each state represents the mean cost of that state, and the darker the color, the higher the cost is. In the Mars exploration problem, those darker states are the states with large slope that the agents want to avoid. The constraint we enforce is the upper bound of the per-step probability of step into those state with large slope—i.e., the more risky or potentially unsafe states to explore. The agent starts from the origin state 'O' and receives reward 1 if it reaches the destination state 'D' after which it returns to the origin. In the simulation, the cost of each state is draw from a binomial distribution, with the mean shown in the figure. At each time step, the



(a) (b)

Figure 3.5: Grid World with Safety Constraints. (a) Grid world structure: the states with darker green color have larger mean cost, and 'O' and 'D' are the origin and destination states, respectively; (b) Policy learned by different algorithms: the blue column represents the probability of going 'West' (choose blue route) and orange column represents the probability of going 'North' (choose orange route).

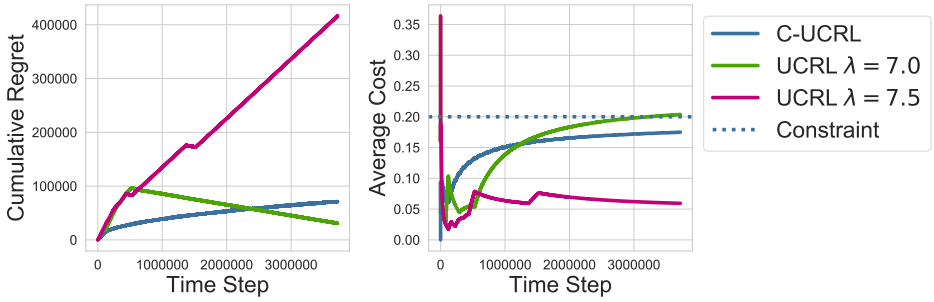
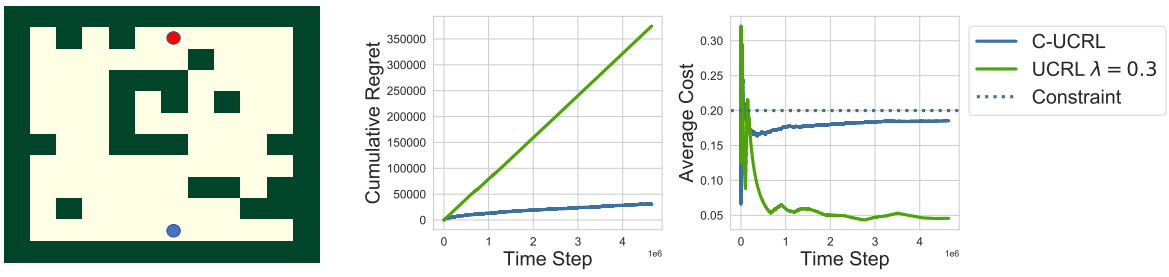


Figure 3.6: Cumulative regret and average cost of `C-UCRL` and `RS-UCRL2`.

agent can take action to move into any of its four neighboring states. Due to the stochastic environment, transitions are stochastic (i.e., even if the agent's action is to go "North", the environment can send the vehicle with a small probability to "East").

Without safety constraints, the optimal policy is obviously to always choose the orange route in Figure 3.5(a). However, with constraints, as we can see in Figure 3.5(b), the optimal policy is a randomized policy that use both blue and orange routes with some probabilities. The relatively conservative baseline policy we use in `C-UCRL` for exploration is choose both routes uniformly at random. Figure 3.6 show the cumulative regret and average cost of the



(a) (b)

Figure 3.7: Grid World with Safety Constraints. (a) CMDP grid world structure: the states with green color have mean cost equals to 1 and others have no cost; the blue state is the origin state and the red state is the destination state. (b) Cumulative regret and average reward of `C-UCRL` and `RS-UCRL2`.

`C-UCRL` and `RS-UCRL2` algorithm and Figure 3.5(b) shows the policy learned by them. As we can see, `RS-UCRL2` either learns to only choose orange or blue route respectively, causing either constraint violation or large reward regret, while `C-UCRL` converges to the optimal policy.

Figure 3.7(a) shows the structure of another larger scale safety grid world example. The green states in the figure have mean cost 1 and the others have zero cost. The blue state is the origin state and the red state is the destination state, which has reward 1. Figure 3.7(b) shows the cumulative regret and average cost of the `C-UCRL` algorithm and `RS-UCRL2` algorithm. The `RS-UCRL2` algorithm is able to learn a policy that does not violate the constraint if we choose a conservative λ, however, with much larger reward regret as compared to `C-UCRL`.

3.7 Extension: A Lower Bound on Constraint Violation with Unknown Transition

So far we have studied learning algorithm in CMDPs with known transition kernel and we proposed a `C-UCRL` algorithm, which is able to achieve $O(T^{\frac{3}{4}}\sqrt{\log(T/\delta)})$ reward regret and zero constraints violation with high probability. One natural question to ask is, what if the transition kernel is unknown? Are we still able to design an algorithm to maintain constraint

satisfaction throughout the learning process?

The short answer is, there is no such algorithm. In fact, we can show that by establishing a lower bound on constraint violation when the transition kernel is unknown. Before we show that, let's look at some background about lower bound on reward regret in traditional MDPs.

In a MDP setting, the goal of reinforcement learning is to find a policy to maximize the expected average reward of a MDP $\mathcal{M}$

$$J(\mathcal{M}, \pi) = \lim_{T\to\infty} \mathbb{E}_{\tau\sim\pi}\left[\frac{1}{T}\sum_{t=0}^{T-1} r(s_t, a_t)\right]. \tag{3.36}$$

The reward regret is defined as

$$\Delta(\mathcal{M}, T) = TJ(\mathcal{M}, \bar{\pi}) - \sum_{t=0}^{T-1} r(s_t, a_t), \tag{3.37}$$

where $\bar{\pi}$ is the optimal policy that maximizes the average reward.

According to the lower bound on reward regret in [62, Theorem 5], for any algorithm, there is a MDP $\mathcal{M}$ with S states, A actions, and diameter D[4], such that for any initial state the expected regret after T steps is

$$\mathbb{E}[\Delta(\mathcal{M}, T)] = \Omega(\sqrt{DSAT}). \tag{3.38}$$

Moreover, this lower bound result is proved to be unimprovable according to [71, Conjecture 1].

Now we establish the lower bound on constraint violation in CMDP. Following the common metric in literature [72], the regret of constraint violation in a CMDP $\mathcal{M}_c$ is defined as

$$\Delta_c(\mathcal{M}_c, T) = \max_{i\in[m]}[\sum_{t=0}^{T-1} c_i(s_t, a_t) - Td_i]. \tag{3.39}$$

This measure is a relatively a weak one since $c_i(s_t, a_t) - d_i$ could be negative when the policy is not violating the constraints, which reduces the overall constraint violation regret.

[4]This diameter is closely related to the mixing time we consider in other work. For a comparison of the diameter to other mixing time parameters, we refer to [62].

[72] introduces another measure

$$\Delta_c^+(\mathcal{M}_c, T) = \max_{i\in[m]} \sum_{t=0}^{T-1} [c_i(s_t, a_t) - d_i]_+, \tag{3.40}$$

which is stronger since it only count the true (positive) constraint violation.

With the constraint violation regret defined, we can now establish the corresponding lower bound of it.

Theorem 3.3. *For any algorithm, there is a CMDP $\mathcal{M}_c$ with S states, A actions, and diameter D, such that for any initial state, the expected constraint violation regret of after T steps is*

$$\mathbb{E}[\Delta_c(\mathcal{M}_c, T)] = \Omega(\sqrt{DSAT}).$$

Now we provide the proof sketch on how to establish the CMDP $\mathcal{M}_c$. For simplicity, we only consider the case where there is one constraint, $m = 1$. Following the assumption in our work, the reward and cost are normalized to be in $[0, 1]$. In the CMDP $\mathcal{M}_c$, we define the cost feedback of each state action pair to be $c(s, a) = 1 - r(s, a)$, where r is the reward in MDP $\mathcal{M}$. And the constraint threshold d is defined to be $1 - J(\mathcal{M}, \bar{\pi})$. In this setting, the cost $c(s, a)$ and threshold d are in the range $[0, 1]$ and they are the negation of the immediate reward and optimal average reward in $\mathcal{M}$, added with a constant 1. Any algorithm that learns to maximize the average reward in $\mathcal{M}$ can be applied directly in $\mathcal{M}_c$ to minimize the average cost. That being said, if there is a algorithm achieves a constraint violation regret in $\mathcal{M}_c$ better than $\Omega(\sqrt{DSAT})$, such algorithm can be used directly in $\mathcal{M}$ to achieve the same reward regret. This contradicts with [62, Theorem 5].

The result can also be extended to the stronger regret measure since $\Delta_c^+(\mathcal{M}_c, T) \geq \Delta_c(\mathcal{M}_c, T)$.

3.8 Conclusion

In this work we formulate the problem of safe reinforcement learning when the transition kernel is known but the reward and constraint costs are unknown a priori as a CMDP

and propose a `C-UCRL` algorithm to learn the optimal policy. Theoretically, we show that `C-UCRL` is guaranteed to satisfy the constraints during learning with probability at least $1-\delta$ and achieves $O(T^{\frac{3}{4}}\sqrt{\log(T/\delta)})$ reward regret. Empirically, we provide examples which demonstrate two key properties relative to comparable algorithms: (1) `C-UCRL` is able to learn the optimal policy which in general is a randomized policy as opposed to a deterministic policy, and (2) `C-UCRL` has high-probability guarantees on remaining safe while learning.

Part II

GAME-THEORETIC REINFORCEMENT LEARNING

Chapter 4

STACKELBERG ACTOR-CRITIC

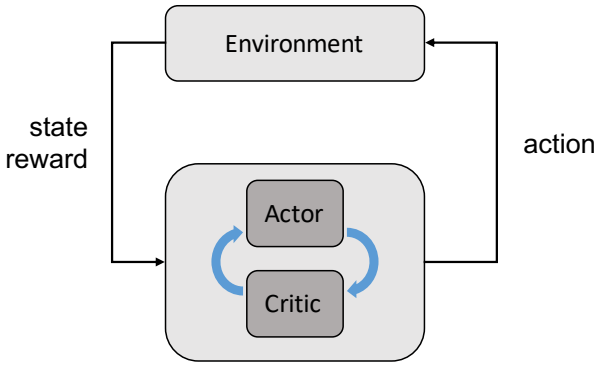


Figure 4.1: Framework of reinforcement learning algorithm with actor critic interaction.

4.1 Introduction

The algorithmic techniques for reinforcement learning can be classified into policy-based, value-based, and actor-critic methods [73]. Policy-based methods directly optimize a parameterized policy to maximize the expected return, while value-based methods estimate the expected return and then infer an optimal policy from the value-function by selecting the maximizing actions. Actor-critic methods bridge policy-based and value-based methods by learning the parameterized policy (actor) and the value-function (critic) together. In particular, actor-critic methods learn a critic that approximates the expected return of the actor while concurrently learning an actor to optimize the expected return based on the critic's estimation.

In this work, we adopt a game-theoretic perspective of actor-critic reinforcement learning

algorithms. Figure 4.1 provides a framework of reinforcement learning with actor critic interaction. To provide some relevant background from game theory, recall that Stackelberg games are a class of games that describe interactions between a leader and a follower [74]. In a Stackelberg game, the leader is distinguished by the ability to act before the follower. As a result of this structure, the leader optimizes its objective accounting for the anticipated response of the follower, while the follower selects a best response to the leader's action to optimize its own objective. The interaction between the actor and critic in reinforcement learning has an intrinsic hierarchical structure reminiscent of a Stackelberg game. Indeed, the actor aims to be at an optimum knowing that the critic responds near-optimally to the selected parameters, while the critic seeks to be at an optimum given the actor parameters or vice versa between the actor and critic. This observation forms the basis of our work which contributes a novel game-theoretic modeling framework along with theoretical and empirical results.

Modeling Contributions. We explicitly cast the interaction between the actor and critic as a two-player general-sum Stackelberg game toward solving reinforcement learning problems. Notably, this perspective deviates from the majority of work on actor-critic reinforcement learning algorithms which implicitly neglect the interaction structure by independently optimizing the actor and critic objectives using individual gradient dynamics. In order to solve the game iteratively in a manner that reflects the interaction structure, we study learning dynamics in which the player deemed the leader updates its parameters using the total derivative of its objective defined using the implicit function theorem and the player deemed the follower updates using the typical individual gradient dynamics. We refer to this gradient-based learning method as the Stackelberg gradient dynamics. The designations of leader and follower between the actor and critic can result in distinct game-theoretic outcomes and we explore both choices and explain how the proper roles depends on the respective objective functions.

Theoretical Contributions. The Stackelberg gradient dynamics were previously studied in general nonconvex games and enjoy a number of theoretical guarantees [15]. In this

work we tailor the analysis of this learning dynamic to the reinforcement learning problem. To do this, we begin by developing a policy gradient theorem for the total derivative update (Theorem 4.1). Then, building off of this result, we develop a meta-framework of Stackelberg actor-critic algorithms. Specifically, this framework adapts the standard actor-critic, deep deterministic policy gradient, and soft-actor critic algorithms to be optimized using the Stackelberg gradient dynamics in place of the usual individual gradient dynamics. For the class of Stackelberg actor-critic algorithms this meta-framework admits, we prove a local convergence guarantee (Theorem 4.2) to a local Stackelberg equilibrium defined by gradient-based sufficient conditions.

Experimental Contributions. From an empirical standpoint, we begin by pointing out in Section 4.3 that the objective functions in actor-critic algorithms commonly exhibit a type of hidden structure in terms of the parameters. Given this observation, we develop simple, yet illustrative examples comparing the behavior of Stackelberg actor-critic algorithms with standard actor-critic algorithms. In particular, we observe that the Stackelberg dynamics mitigate cycling in the parameter space and accelerate convergence. We discover from extensive experiments on OpenAI gym environments that similar observations carry over to complex problems and that our Stackelberg actor-critic algorithms always perform at least as well and often significantly outperform the standard actor-critic algorithm counterparts.

4.2 Related Work

Game-theoretic frameworks have been studied extensively in reinforcement learning but mostly in multi-agent setting [75]. In multi-agent reinforcement learning, the decentralized learning scheme is mostly adopted in practice [13], where agents typically behave independently and optimize their own objective with no explicit information exchange. A shortcoming of this method is that agents fail to consider the learning process of other agents and simply treat them as a static component of the environment [11]. To resolve this, several works design learning algorithms that explicitly account for the learning behavior of other agents [76, 77, 78], which is shown to improve learning stability and induce cooperation. In

contrast, [79] studies a competitive policy optimization method for multi-agent reinforcement learning which performs recursive reasoning about the behavior of opponents to exploit them in two-player zero-sum games.

The past research taking a game-theoretic viewpoint of single-agent reinforcement learning is limited despite the fact that there is often implicitly multiple players in reinforcement learning algorithms. [80] proposes a framework that casts model-based reinforcement learning as a two-player general-sum Stackelberg game between a policy player and a model player. However, they only consider optimizing the objective of each player using the typical individual gradient dynamics with timescale separation as an approximation to Stackelberg gradient dynamics. Concurrent to this work, [81] analyzes the Stackelberg gradient dynamics with timescale separation for bilevel optimization with application to reinforcement learning. For reinforcement learning, they give a convergence guarantee for an actor-critic algorithm under assumptions such as exact linear function approximation which result in the total derivative being equivalent to the individual gradient. We provide a complimentary study by developing a general framework for Stackelberg actor-critic algorithms that we analyze without such assumptions and also extensively evaluate empirically on reinforcement learning tasks.

Single-agent reinforcement learning algorithms with second-order information trace back to natural policy gradient methods [82] and the natural actor-critic algorithm [83, 69]. Since then, such techniques have been proposed for both policy-based and actor-critic methods [25, 84, 85, 86]. However, the gradient information in the past works do not account for the interaction between the actor and critic as in this work.

4.3 Motivation & Preliminaries

In this section, we begin by presenting background on Stackelberg games and the relevant equilibrium concept. Then, to motivate and illustrate the utility of Stackelberg-based actor-critic algorithms, we highlight a key hidden structure that exists in actor-critic objective formulations and explore the behavior of Stackelberg gradient dynamics in comparison to individual gradient dynamics given this design. Finally, we provide the necessary mathematical

background and formalism for actor-critic reinforcement learning algorithms.

4.3.1 Game-Theoretic Preliminaries

A Stackelberg game is a game between two agents where one agent is deemed the leader and the other the follower. Each agent has an objective they want to optimize that depends on not only their own actions but also on the actions of the other agent. Specifically, the leader optimizes its objective under the assumption that the follower will play a best response. Let $f_1(x_1, x_2)$ and $f_2(x_1, x_2)$ be the objective functions that the leader and follower want to minimize, respectively, where $x_1 \in X_1 \subseteq \mathbb{R}^{d_1}$ and $x_2 \in X_2 \subseteq \mathbb{R}^{d_2}$ are their decision variables or strategies and $x = (x_1, x_2) \in X_1 \times X_2$ is their joint strategy. The leader and follower aim to solve the following problems:

$$\min_{x_1 \in X_1} \{ f_1(x_1, x_2) \big|\ x_2 \in \arg\min_{y \in X_2} f_2(x_1, y) \}, \tag{L}$$

$$\min_{x_2 \in X_2} f_2(x_1, x_2). \tag{F}$$

Since the leader assumes the follower chooses a best response $x_2^*(x_1) = \arg\min_y f_2(x_1, y)$,[1] the follower's decision variables are implicitly a function of the leader's. In deriving sufficient conditions for the optimization problem in (L), the leader utilizes this information by the total derivative of its cost function which is given by

$$\nabla f_1(x_1, x_2^*(x_1)) = \nabla_1 f_1(x) + (\nabla x_2^*(x_1))^\top \nabla_2 f_1(x).$$

where $\nabla x_2^*(x_1) = -(\nabla_2^2 f_2(x))^{-1} \nabla_{21} f_2(x)$. [2]

Hence, a point $x = (x_1, x_2)$ is a local solution to (L) if $\nabla f_1(x_1, x_2^*(x_1)) = 0$ and $\nabla^2 f_1(x_1, x_2^*(x_1)) > 0$. For the follower's problem, sufficient conditions for optimality are $\nabla_2 f_2(x_1, x_2) = 0$ and $\nabla_2^2 f_2(x_1, x_2) > 0$. This gives rise to the following equilibrium concept which characterizes sufficient conditions for a local Stackelberg equilibrium.

---

[1] Under sufficient regularity conditions on the follower's optimization problem, the best response map is a singleton. This is a generic condition in games [87, 15].

[2] The partial derivative of $f(x_1, x_2)$ with respect to the x_i is denoted by $\nabla_i f(x_1, x_2)$ and the total derivative of $f(x_1, h(x_1))$ for some function h, is denoted ∇f where $\nabla f(x_1, h(x_1) = \nabla_1 f(x_1, h(x_1)) + (\nabla h(x_1))^\top \nabla_2 f(x_1, h(x_1))$.

Definition 4.1 (Differential Stackelberg Equilibrium, [15]). *The joint strategy* $x^* = (x_1^*, x_2^*) \in X_1 \times X_2$ *is a differential Stackelberg equilibrium if* $\nabla f_1(x^*) = 0$, $\nabla_2 f_2(x^*) = 0$, $\nabla^2 f_1(x^*) > 0$, *and* $\nabla_2^2 f_2(x^*) > 0$.

The Stackelberg learning dynamics derive from the first-order gradient-based sufficient conditions and are given by

$$x_{1,k+1} = x_{1,k} - \alpha_1 \nabla f_1(x_{1,k}, x_{2,k})$$
$$x_{2,k+1} = x_{2,k} - \alpha_2 \nabla_2 f_2(x_{1,k}, x_{2,k})$$

where α_i, $i = 1, 2$ are the learning rates for the leader and follower, respectively.

4.3.2 Motivating Examples

In the next section we present several common actor-critic formulations including the "vanilla" actor-critic, deep deterministic policy gradient, and soft actor-critic. A common theme among them is that the actor and critic objectives exhibit a simple hidden structure in the parameters. In particular, the actor objective typically has a hidden linear structure in terms of the parameters θ which is abstractly of the form $Q_w(\theta) = w^\top \mu(\theta)$. Analogously, the critic objective usually has a hidden quadratic structure in the parameters w which is abstractly of the form or $(R(\theta) - Q_w(\theta))^2$. The terminology of hidden structure in this context refers to the fact that the specified structure appears when the functions transforming the parameters are removed.[3] Interestingly, similar observations have been made regarding generative adversarial network formulations and exploited to gain insights into gradient learning dynamics for optimizing them [88, 89].

Based on this observation, we investigate simple, yet illustrative reinforcement learning problems with the aforementioned structure and compare and contrast the behavior of the Stackelberg gradient dynamics with the usual individual gradient dynamics. As we demon-

[3]The actor and critic functions could be approximated by neural nets in practice but we consider the simplest linear case, which captures the hidden structure and gives insights for general cases.

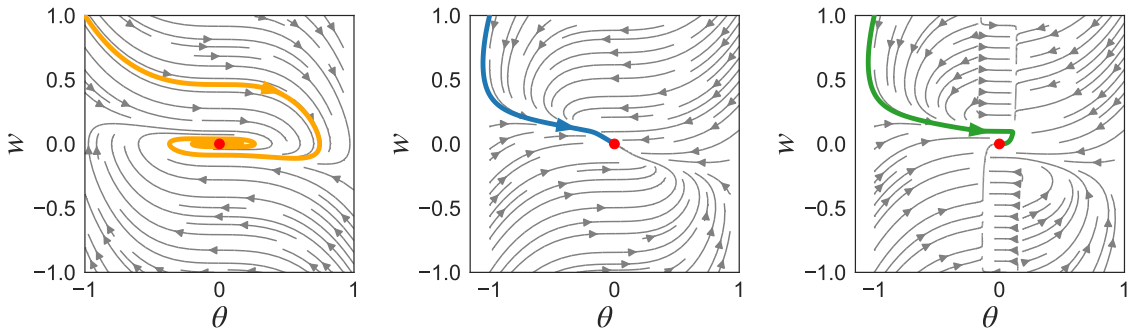
strate later in Section 4.5, the insights we uncover from this study generally carry over to complex reinforcement learning problems.

Example. Consider a single step Markov decision process where the reward function is given by $R(\theta) = -\frac{1}{5}\theta^2$ and $\theta \in [-1, 1]$ is the decision variable of actor. Suppose that the critic is designed using the most basic linear function approximation $Q_w(\theta) = w\theta$ with $w \in [-1, 1]$. The actor seeks to find the action that maximizes the value indicated by the critic and the critic approximates the rewards of actions generated by the actor. Thus, the actor has objective $J(\theta, w) = Q_w(\theta) = w\theta$ and the critic has objective $L(\theta, w) = \mathbb{E}_{\theta\sim\rho}[(R(\theta) - Q_w(\theta))^2]$. For simplicity, we assume the critic only minimizes the mean square error of the sample action generated by current actor θ. The critic objective is then $L(\theta, w) = (R(\theta) - Q_w(\theta))^2 = (w \cdot \theta + \frac{1}{5}\theta^2)^2$.

Actor-Critic & Deep Deterministic Policy Gradient. The structure of this example closely mirrors the hidden structure of both the "vanilla" actor-critic and deep deterministic policy gradient formulations as described in the next section. The typical way to optimize the objectives is by performing individual gradient dynamics (gradient descent-ascent) on the actor and critic parameters. Figure 4.2 shows the vector fields and trajectories of each of the updates: individual gradient play[4], Stackelberg gradient play, and regularized Stackelberg gradient play.

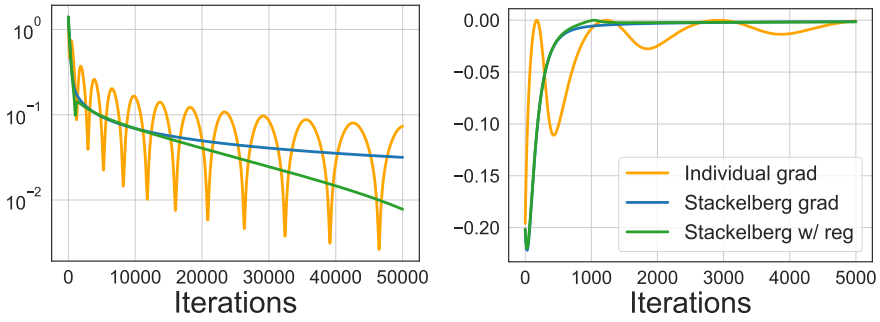
Figure 4.2(a) shows the gradient vector field and the parameter trajectories under the individual gradient dynamics. We observe that although the trajectory eventually converges to the equilibrium point $(\theta^*, w^*) = (0, 0)$, it cycles significantly. Such cycling behavior may be an indication of reduced reliability along the learning path and is often exacerbated by noise. Generally speaking, it is more desirable to observe smooth, monotonic changes in performance as compared to cycling behavior or noisy fluctuations around a observable

[4]In the learning in games literature, this is also often referred to as simultaneous gradient play or simultaneous gradient descent-ascent.



(a) Individual gradient (b) Stackelberg gradient (c) Regularized Stackelberg gradient

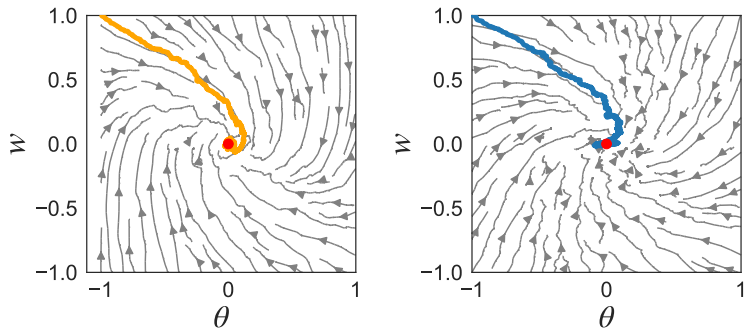
Figure 4.2: Vector fields and trajectories of the individual gradient, Stackelberg gradient and regularized Stackelberg gradient updates. The Stackelberg updates eliminate cycling by changing the shape of the vector field.



(a) Error (b) Return

Figure 4.3: (a) Convergence error $\|w - w^*\|^2 + \|\theta - \theta^*\|^2$ where $(\theta^*, w^*) = (0, 0)$ is the equilibrium. (b) The return $R(\theta)$ of the actor. The Stackelberg update eliminates cycling and hence, converges more directly to the equilibrium as can be seen in (a), whereas the individual gradient update oscillates significantly. Regularization helps to speed up convergence.

trend. The reason for this is that when we go to deploy such algorithms in the real world, it can be extremely costly to have the algorithm perform in oscillatory or even unpredictable ways. This is in particular true when, as is often the case, there are unmodeled exogenous inputs or environmental factors.



(a) SAC (b) STSAC-AL

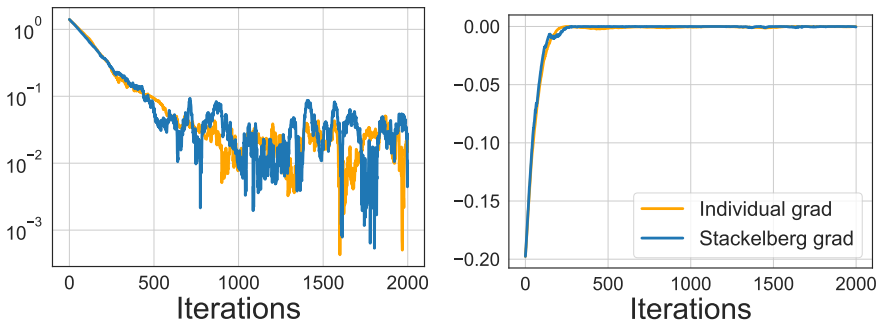
Figure 4.4: Vector fields and trajectories of the SAC and STSAC-AL updates.

On the other hand, Stackelberg gradient converges more directly to the equilibrium point $(\theta^*, w^*) = (0, 0)$. Figure 4.2(b) shows the vector field and parameter trajectories under the Stackelberg gradient dynamics, the details of which will be introduced in Section 4.4 and Figure 4.2(c) shows the trajectories of the regularized Stackelberg gradient introduced in Section 4.4.5. We observe that the cycling behavior is completely eliminated as a result of the consideration given to the interaction structure.

Figure 4.3(a) shows the error to equilibrium $\|w - w^*\|^2 + \|\theta - \theta^*\|^2$ for the individual gradient dynamics and the Stackelberg gradient dynamics Figure 4.3(b) shows the return $R(\theta)$ of each of the updates. We can observe that the cycling is mitigated and convergence accelerated by optimizing using the Stackelberg gradient, which leads to more stable returns along the learning.

Soft Actor-Critic. The soft actor-critic algorithm also exhibits a similar structure, but with entropic regularization included in the actor objective. In Figures 4.4 and 4.5, we show the result of adding entropy regularization to the actor's objective using the SAC algorithm.

We show the vector fields along with the parameter trajectories for the individual gradient dynamics and the Stackelberg gradient dynamics in Figure 4.4(a) and Figure 4.4(b), respectively. Since SAC involves sampling from an stochastic policy, we plot the empirical



(a) Error of `SAC` and `STSAC-AL` (b) Return of `SAC` and `STSAC-AL`

Figure 4.5: (a) Error for each algorithm, `SAC` and `STSAC-AL`, $\|w - w^*\|^2 + \|\theta - \theta^*\|^2$ where $(\theta^*, w^*) = (0, 0)$ is the equilibrium. (b) Return of the actor $R(\theta)$.

mean gradient vector fields in Figure 4.4(a) and Figure 4.4(b), where the gradients for update are estimated by samples. Given the entropic regularization, both learning algorithms behave similarly, and both gradient updates converge much faster and the gap between them are less significant (Figure 4.5(a) and 4.5(b)). This perhaps indicates that the individual gradient dynamics are more well-suited to optimize this form of objectives and highlights the importance of considering how game dynamics perform on types of hidden structures when optimizing actor-critic algorithms in reinforcement learning.

Importantly, regardless of the objective function structure, the Stackelberg gradient dynamics tend to converge rather directly to the equilibrium and for some hidden structures they significantly mitigate oscillations and stabilize training. It is well-known that this is a desirable property of the reinforcement learning algorithms owing to the implications for both evaluation and real-world applications [90]. Together, this motivating section suggests that introducing the Stackelberg dynamics as a "meta-algorithm" on existing actor-critic methods is likely to lead to more favorable convergence properties. We demonstrate this empirically in Section 4.5.

4.3.3 Actor-Critic Algorithms

We consider discrete-time Markov decision processes (MDPs) with continuous state space $\mathcal{S}$ and continuous action space $\mathcal{A}$. We denote the state and action at time step t by s_t and a_t, respectively. The initial state s_0 is determined by the initial state density $s_0 \sim \rho(s)$. At time step t, the agent in state s_t takes an action a_t according to a policy $a_t \sim \pi(\cdot|s_t)$ and obtains a reward $r_t = r(s_t, a_t)$. The agent then transitions to state s_{t+1} determined by the transition function $s_{t+1} \sim P(s'|s_t, a_t)$. A trajectory $\tau = (s_0, a_0, \ldots, s_T, a_T)$ gives the cumulative rewards or return defined as $R(\tau) = \sum_{t=0}^{T} \gamma^t r(s_t, a_t)$, where the discount factor $0 < \gamma \leq 1$ assigns weights to rewards received at different time steps. The expected return of π after executing a_t in state s_t can be expressed by the Q function

$$Q^\pi(s_t, a_t) = \mathbb{E}_{\tau\sim\pi}\big[\sum_{t'=t}^{T} \gamma^{t'-t} r(s_{t'}, a_{t'})|s_t, a_t\big].$$

Correspondingly, the expected return of π in state s_t can be expressed by the value function V defined as

$$V^\pi(s_t) = \mathbb{E}_{\tau\sim\pi}\big[\sum_{t'=t}^{T} \gamma^{t'-t} r(s_{t'}, a_{t'})|s_t\big].$$

The goal of reinforcement learning is to find an optimal policy that maximizes the expected return which is given by

$$\begin{aligned} J(\pi) &= \mathbb{E}_{\tau\sim\pi}\big[\sum_{t=0}^{T} \gamma^t r(s_t, a_t)\big] = \int_\tau p(\tau|\pi) R(\tau) \mathrm{d}\tau \\ &= \mathbb{E}_{s\sim\rho, a\sim\pi(\cdot|s)}\big[Q^\pi(s, a)\big], \end{aligned}$$

where $p(\tau|\pi) = \rho(s_0) \prod_{t=0}^{T} \pi(a_t|s_t) P(s_{t+1}|s_t, a_t)$.

The policy-based approach [91] parameterizes the policy π by the parameter θ and finds the optimal parameter choice θ^* by maximizing the expected return

$$J(\theta) = \mathbb{E}_{s\sim\rho, a\sim\pi_\theta(\cdot|s)}\big[Q^\pi(s, a)\big]. \tag{4.1}$$

This optimization problem can be solved by gradient ascent. By the policy gradient theorem [92],

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\rho, a\sim\pi_\theta(\cdot|s)}\left[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s, a)\right],$$

where ∇_θ denotes the derivative with respect to θ. A common method to approximate $Q^\pi(s,a)$ in the policy gradient is by sampling trajectories and averaging returns, which is known as REINFORCE [91].

"Vanilla" Actor-Critic (`AC`). The actor-critic method [14, 93] relies on a critic function $Q_w(s,a)$ parameterized by w to approximate $Q^\pi(s,a)$. By replacing $Q_w(s,a)$ with $Q^\pi(s,a)$ in (4.1), the actor which is parameterized by θ has the objective

$$J(\theta, w) = \mathbb{E}_{s\sim\rho, a\sim\pi_\theta(\cdot|s)}\big[Q_w(s,a)\big]. \tag{4.2}$$

The objective is optimized using gradient ascent where

$$\nabla_\theta J(\theta, w) = \mathbb{E}_{s\sim\rho, a\sim\pi_\theta(\cdot|s)}[\nabla_\theta \log \pi_\theta(a|s) Q_w(s,a)]. \tag{4.3}$$

The critic which is parameterized by w has the objective to minimize the mean square error between the Q-functions

$$L(\theta, w) = \mathbb{E}_{s\sim\rho, a\sim\pi_\theta(\cdot|s)}[(Q_w(s,a) - Q^\pi(s,a))^2], \tag{4.4}$$

where the function $Q^\pi(s,a)$ is approximated by Monte Carlo estimation or bootstrapping [73].

The actor-critic method optimizes the objectives with individual gradient dynamics [83, 94] which gives rise to the updates

$$\theta \leftarrow \theta + \alpha_\theta \nabla_\theta J(\theta, w), \tag{4.5}$$

$$w \leftarrow w - \alpha_w \nabla_w L(\theta, w), \tag{4.6}$$

where α_θ and α_w are the learning rates of actor and critic. Clearly, even in this basic actor-critic method, the actor and critic are coupled since J and L depend on both θ and w, which naturally lends to a game-theoretic interpretation.

Deep Deterministic Policy Gradient (`DDPG`). The `DDPG` algorithm [3] is an off-policy method with subtly different objective functions for the actor and critic. In particular, the formulation has a deterministic actor $\mu_\theta(s) : \mathcal{S} \to \mathcal{A}$ with the objective

$$J(\theta, w) = \mathbb{E}_{\xi\sim\mathcal{D}}\left[Q_w(s, \mu_\theta(s))\right]. \tag{4.7}$$

The critic objective is the mean square Bellman error

$$L(\theta, w) = \mathop{\mathbb{E}}_{\xi\sim\mathcal{D}}[(Q_w(s,a) - (r + \gamma Q_0(s', \mu_\theta(s'))))^2], \tag{4.8}$$

where $\xi = (s, a, r, s')$, $\mathcal{D}$ is a replay buffer, and Q_0 is a target Q network.[5]

Soft Actor-Critic (`SAC`). The `SAC` algorithm [95] exploits the double Q-learning trick [96] and employs entropic regularization to encourage exploration. The actor's objective $J(\theta, w)$ is

$$\mathbb{E}_{\xi\sim\mathcal{D}}\big[\min_{i=1,2} Q_{w_i}(s, a_\theta(s)) - \eta \log(\pi_\theta(a_\theta(s)|s))\big], \tag{4.9}$$

where $a_\theta(s)$ is a sample from $\pi_\theta(\cdot|s)$ and η is entropy regularization coefficient. The parameter of the critic is the union of both Q networks parameters $w = \{w_1, w_2\}$ and the critic objective is defined correspondingly by

$$L(\theta, w) = \mathbb{E}_{\xi\sim\mathcal{D}}\big[\textstyle\sum_{i=1,2} \left(Q_{w_i}(s,a) - y(r,s')\right)^2\big], \tag{4.10}$$

where

$$y(r,s') = r + \gamma(\min_{i=1,2} Q_{0,i}(s', a_\theta(s')) - \eta \log(\pi_\theta(a_\theta(s')|s'))).$$

The target networks in `DDPG` and `SAC` are updated by taking the Polyak average of the network parameters over the course of training, and the actor and critic networks are updated by individual gradient dynamics identical to (4.5)–(4.6).

4.4 Stackelberg Framework

In this section, we begin by formulating the actor-critic interaction as two-player general-sum Stackelberg game and introduce a Stackelberg framework for actor-critic algorithms, under which we develop novel Stackelberg versions of existing algorithms: Stackelberg actor-critic (`STAC`), Stackelberg deep deterministic policy gradient (`STDDPG`), and Stackelberg soft actor-critic (`STSAC`). Following this, we give a local convergence guarantee for the algorithms to

[5] In the `DDPG` algorithm, the next-state actions used in the target network come from the target policy instead of the current policy. To be consistent with `SAC`, we use the current policy.

Algorithm 3: Stackelberg Actor-Critic Framework

Input: actor-critic algorithm `ALG`, player designations, and learning rate sequences $\alpha_{\theta,k}, \alpha_{w,k}$.

if actor is leader, update actor and critic in `ALG` with:

$$\theta_{k+1} = \theta_k + \alpha_{\theta,k} \nabla J(\theta_k, w_k) \tag{4.11}$$

$$w_{k+1} = w_k - \alpha_{w,k} \nabla_w L(\theta_k, w_k) \tag{4.12}$$

if critic is leader, update actor and critic in `ALG` with:

$$\theta_{k+1} = \theta_k + \alpha_{\theta,k} \nabla_\theta J(\theta_k, w_k) \tag{4.13}$$

$$w_{k+1} = w_k - \alpha_{w,k} \nabla L(\theta_k, w_k) \tag{4.14}$$

a local Stackelberg equilibrium. Finally, a regularization method for practical usage of the algorithms is discussed.

4.4.1 Meta-Algorithm

Given an actor-critic formulation, in particular, the objectives of the actor and critic defined by $J(\theta, w)$ and $L(\theta, w)$, we can interpret the problem as a two-player general-sum Stackelberg game. If we view the actor as the leader and the critic as a follower, then the players aim to solve the following optimization problems, respectively:

$$\max_\theta \{ J(\theta, w^*(\theta)) \big|\ w^*(\theta) = \arg\min_{w'} L(\theta, w') \} \tag{AL}$$

$$\min_w L(\theta, w). \tag{CF}$$

On the other hand, if we view the critic as the leader and the actor as the follower, then the players aim to solve the following optimization problems, respectively:

$$\min_w \{ L(\theta^*(w), w) \big|\ \theta^*(w) = \arg\max_{\theta'} J(\theta', w) \} \tag{CL}$$

$$\max_\theta J(\theta, w). \tag{AF}$$

As described in Section 4.3.1, we propose to optimize the objectives using a learning algorithm that accounts for the structure of the problems. Specifically, since the leader assumes the follower selects a best response, it is natural to optimize the leader objective by following the total derivative given that the follower's decision is implicitly a function of the leader's. The meta-framework we adopt for Stackelberg refinements of actor-critic methods is in Algorithm 3. The distinction compared to the usual actor-critic methods is that in the updates we replace the individual gradient for the leader by the implicitly defined total derivative which accounts for the interaction structure whereas the rest of the actor-critic method remains identical.

The dynamics with the actor as the leader are given by (4.11)–(4.12) where the actor's total derivative $J(\theta, w)$ is

$$\nabla_\theta J(\theta, w) - \nabla^\top_{w\theta} L(\theta, w)(\nabla^2_w L(\theta, w))^{-1} \nabla_w J(\theta, w). \tag{4.15}$$

When the critic is the leader the dynamics are given by (4.13)–(4.14) where the critic's total derivative $\nabla L(\theta, w)$ is

$$\nabla_w L(\theta, w) - \nabla^\top_{\theta w} J(\theta, w)(\nabla^2_\theta J(\theta, w))^{-1} \nabla_\theta L(\theta, w). \tag{4.16}$$

We now consider instantiations of this framework and explain how the total derivative can be obtained from sampling along with natural choices of leader and follower.

4.4.2 Stackelberg "Vanilla" Actor-Critic

We start by instantiating the Stackelberg meta-algorithm for the "vanilla" actor-critic (`AC`) algorithm for which the actor and critic objectives are given in (4.2) and (4.4), respectively.[6] In this on-policy formulation, the critic assists the actor in learning the optimal policy by approximating the value function of the current policy. To give an accurate approximation,

[6]We only demonstrate the "vanilla" actor-critic algorithm and its Stackelberg version here and in our experiments, but the framework could be generalized to more on-policy actor-critic algorithms (e.g., A2C, A3C, [94]).

the critic aims to be selecting a best response $w^*(\theta) = \arg\min_{w'} L(\theta, w')$. Thus, the actor naturally plays the role of leader and the critic the follower.

However, estimating the total derivative $\nabla J(\theta, w)$ as defined in (4.15) is not straightforward and we analyze each component individually. The individual gradient $\nabla_\theta J(\theta, w)$ can be computed by policy gradient theorem as given in (4.3). Moreover, $\nabla_w J(\theta, w) = \mathbb{E}_{s\sim\rho, a\sim\pi_\theta(\cdot|s)}[\nabla_w Q_w(s, a)]$, which follows by direct computation, and similarly

$$\begin{aligned}\nabla_w^2 L(\theta, w) = \mathbb{E}_{s\sim\rho, a\sim\pi_\theta(\cdot|s)} \big[&2\nabla_w Q_w(s, a)\nabla_w^\top Q_w(s, a) \\ &+2(Q_w(s, a) - Q^\pi(s, a))\nabla_w^2 Q_w(s, a)\big] .\end{aligned}$$

To compute $\nabla_{w\theta} L(\theta, w)$ in (4.15), we begin by obtaining $\nabla_\theta L(\theta, w)$ with the following policy gradient theorem.

Theorem 4.1. *Given an MDP and actor-critic parameters (θ, w), the gradient of $L(\theta, w)$ with respect to θ is given by*

$$\begin{aligned}\nabla_\theta L(\theta, w) = \mathbb{E}_{\tau\sim\pi_\theta}[&\nabla_\theta \log \pi_\theta(a_0|s_0) \\ &(Q_w(s_0, a_0) - Q^\pi(s_0, a_0))^2 + \textstyle\sum_{t=1}^T \gamma^t \nabla_\theta \log \pi_\theta(a_t|s_t) \\ &(Q^\pi(s_0, a_0) - Q_w(s_0, a_0))Q^\pi(s_t, a_t)].\end{aligned}$$

Proof. The derivative is computed as follows:

$$\begin{aligned}\nabla_\theta L(\theta, w) &= \nabla_\theta \int_{s_0} \rho(s_0) \int_{a_0} \pi_\theta(a_0|s_0) \left(Q_w(s_0, a_0) - Q^\pi(s_0, a_0)\right)^2 \mathrm{d}a_0 \mathrm{d}s_0 \\ &= \int_{s_0} \rho(s_0) \int_{a_0} \nabla_\theta \pi_\theta(a_0|s_0) \left(Q_w(s_0, a_0) - Q^\pi(s_0, a_0)\right)^2 \mathrm{d}a_0 \mathrm{d}s_0 \\ &\quad + \int_{s_0} \rho(s_0) \int_{a_0} \pi_\theta(a_0|s_0) \nabla_\theta \left(Q_w(s_0, a_0) - Q^\pi(s_0, a_0)\right)^2 \mathrm{d}a_0 \mathrm{d}s_0 \\ &= \int_{s_0} \rho(s_0) \int_{a_0} \pi_\theta(a_0|s_0) \nabla_\theta \log \pi_\theta(a_0|s_0) \left(Q_w(s_0, a_0) - Q^\pi(s_0, a_0)\right)^2 \mathrm{d}a_0 \mathrm{d}s_0 \\ &\quad + 2 \int_{s_0} \rho(s_0) \int_{a_0} \pi_\theta(a_0|s_0) \left(Q^\pi(s_0, a_0) - Q_w(s_0, a_0)\right) \nabla_\theta Q^\pi(s_0, a_0) \mathrm{d}a_0 \mathrm{d}s_0.\end{aligned}$$

From here, it remains to compute $\nabla_\theta Q^\pi(s_0, a_0)$. To do so, recall that $Q^\pi(s_t, a_t)$ and $V^\pi(s_t)$ are given by

$$Q^\pi(s_t, a_t) = \mathbb{E}_{\tau\sim\pi}\big[\sum_{t'=t}^{T} \gamma^{t'-t} r(s_{t'}, a_{t'})|s_t, a_t\big] = r(s_t, a_t) + \gamma \int_{s'} P(s'|s_t, a_t) V^\pi(s') \mathrm{d}s',$$

and

$$V^\pi(s_t) = \mathbb{E}_{\tau\sim\pi}\big[\sum_{t'=t}^{T} \gamma^{t'-t} r(s_{t'}, a_{t'})|s_t\big] = \int_a \pi_\theta(a|s_t) Q^\pi(s_t, a) \mathrm{d}a.$$

Hence, $\nabla_\theta Q^\pi(s_0, a_0)$ is computed as follows:

$$\begin{aligned}
&\nabla_\theta Q^\pi(s_0, a_0) = \gamma \int_{s_1} P(s_1|s_0, a_0) \nabla_\theta V^\pi(s_1) \mathrm{d}s_1 \\
&= \gamma \int_{s_1} P(s_1|s_0, a_0) \int_{a_1} \left(\nabla_\theta \pi_\theta(a_1|s_1) Q^\pi(s_1, a_1) + \pi_\theta(a_1|s_1) \nabla_\theta Q^\pi(s_1, a_1)\right) \mathrm{d}a_1 \mathrm{d}s_1 \\
&= \gamma \int_{s_1} P(s_1|s_0, a_0) \int_{a_1} \pi_\theta(a_1|s_1) \nabla_\theta \log \pi_\theta(a_1|s_1) Q^\pi(s_1, a_1) \mathrm{d}a_1 \mathrm{d}s_1 \\
&\quad + \gamma^2 \int_{s_1} P(s_1|s_0, a_0) \int_{a_1} \pi_\theta(a_1|s_1) \int_{s_2} P(s_2|s_1, a_1) \nabla_\theta V^\pi(s_2) \mathrm{d}s_2 \mathrm{d}a_1 \mathrm{d}s_1 \\
&= \gamma \int_{s_1} P(s_1|s_0, a_0) \int_{a_1} \pi_\theta(a_1|s_1) \nabla_\theta \log \pi_\theta(a_1|s_1) Q^\pi(s_1, a_1) \mathrm{d}a_1 \mathrm{d}s_1 \\
&\quad + \gamma^2 \int_{s_1} P(s_1|s_0, a_0) \int_{a_1} \pi_\theta(a_1|s_1) \int_{s_2} P(s_2|s_1, a_1) \int_{a_2} \pi_\theta(a_2|s_2) \nabla_\theta \log \pi_\theta(a_2|s_2) Q^\pi(s_2, a_2) \mathrm{d}a_2 \mathrm{d}s_2 \mathrm{d}a_1 \mathrm{d}s_1 \\
&\quad + \gamma^3 \int_{s_1} P(s_1|s_0, a_0) \int_{a_1} \pi_\theta(a_1|s_1) \int_{s_2} P(s_2|s_1, a_1) \int_{a_2} \pi_\theta(a_2|s_2) \int_{s_3} P(s_3|s_2, a_2) \nabla_\theta V^\pi(s_3) \mathrm{d}s_3 \mathrm{d}a_2 \mathrm{d}s_2 \mathrm{d}a_1 \mathrm{d}s_1 \\
&= \gamma \int_\tau p(\tau_{1:1}|\theta) \nabla_\theta \log \pi_\theta(a_1|s_1) Q^\pi(s_1, a_1) \mathrm{d}\tau_{1:1} \\
&\quad + \gamma^2 \int_\tau p(\tau_{1:2}|\theta) \nabla_\theta \log \pi_\theta(a_2|s_2) Q^\pi(s_2, a_2) \mathrm{d}\tau_{1:2} \\
&\quad + \dots \\
&= \int_\tau \sum_{t=1}^{T} \gamma^t p(\tau_{1:t}|\theta) \nabla_\theta \log \pi_\theta(a_t|s_t) Q^\pi(s_t, a_t) \mathrm{d}\tau. \qquad (4.17)
\end{aligned}$$

where the last equality is obtained by unrolling and marginalization for the entire length of the trajectory.

Thus, coming back to the computation of $\nabla_\theta L(\theta, w)$, we have that

$$\begin{aligned}
\nabla_\theta L(\theta, w) &= \int_{s_0} \rho(s_0) \int_{a_0} \pi_\theta(a_0|s_0) \nabla_\theta \log \pi_\theta(a_0|s_0) \left(Q_w(s_0, a_0) - Q^\pi(s_0, a_0)\right)^2 \mathrm{d}a_0 \mathrm{d}s_0 \\
&\quad + 2 \int_{s_0} \rho(s_0) \int_{a_0} \pi_\theta(a_0|s_0) \left(Q^\pi(s_0, a_0) - Q_w(s_0, a_0)\right) \nabla_\theta Q^\pi(s_0, a_0) \mathrm{d}a_0 \mathrm{d}s_0 \\
&= \int_\tau p(\tau_0|\theta) \nabla_\theta \log \pi_\theta(a_0|s_0) \left(Q_w(s_0, a_0) - Q^\pi(s_0, a_0)\right)^2 \\
&\quad + 2 \sum_{t=1}^{T} \gamma^t p(\tau_{0:t}|\theta) \nabla_\theta \log \pi_\theta(a_t|s_t) \left(Q^\pi(s_0, a_0) - Q_w(s_0, a_0)\right) Q^\pi(s_t, a_t) \mathrm{d}\tau \\
&= \mathbb{E}_{\tau \sim \pi_\theta} \Bigg[\nabla_\theta \log \pi_\theta(a_0|s_0) \left(Q_w(s_0, a_0) - Q^\pi(s_0, a_0)\right)^2 \\
&\qquad\qquad + \sum_{t=1}^{T} \gamma^t \nabla_\theta \log \pi_\theta(a_t|s_t) \left(Q^\pi(s_0, a_0) - Q_w(s_0, a_0)\right) Q^\pi(s_t, a_t) \Bigg]
\end{aligned}$$

which completes the proof. □

Theorem 4.1 allows us to compute $\nabla_{\theta w} L(\theta, w)$ directly by $\nabla_w(\nabla_\theta L(\theta, w))$ since the distribution of $\nabla_\theta L(\theta, w)$ does not depend on w and ∇_w can be moved into the expectation.

The critic in `AC` is often designed to approximate the state value function $V^\pi(s)$ which has computational advantages, and the policy gradient can be computed by advantage estimation [97]. In this formulation, $J(\theta, w) = \mathbb{E}_{\tau \sim \pi_\theta}\big[r(s_0, a_0) + V_w(s_1)\big]$ and $L(\theta, w) = \mathbb{E}_{s \sim \rho}[(V_w(s) - V^\pi(s))^2]$. Then $\nabla_\theta L(\theta, w)$ can be computed by the next proposition.

Proposition 4.1. *Given an MDP and actor-critic parameters (θ, w), if the critic has the objective function $L(\theta, w) = \mathbb{E}_{s \sim \rho}[(V_w(s) - V^\pi(s))^2]$, then $\nabla_\theta L(\theta, w)$ is given by*

$$\mathop{\mathbb{E}}_{\tau \sim \pi_\theta}[2 \sum_{t=0}^{T} \gamma^t \nabla_\theta \log \pi_\theta(a_t|s_t)(V^\pi(s_0) - V_w(s_0)) Q^\pi(s_t, a_t)].$$

Proof. The critic's objective is given by $L(\theta, w) = \mathbb{E}_{s \sim \rho}\left[\left(V_w(s) - V^\pi(s)\right)^2\right]$. Hence, taking the derivative with respect to θ, we have that

$$\begin{aligned}
\nabla_\theta L(\theta, w) &= \int_{s_0} \rho(s_0) \nabla_\theta (V_w(s_0) - V^\pi(s_0))^2 \mathrm{d}s_0 \\
&= 2 \int_{s_0} \rho(s_0)(V^\pi(s_0) - V_w(s_0)) \nabla_\theta V^\pi(s_0) \mathrm{d}s_0. \qquad (4.18)
\end{aligned}$$

Now we compute $\nabla_\theta V^\pi(s_0)$ in (4.18). Use the result of (4.17), we have

$$\begin{aligned}\nabla_\theta V^\pi(s_0) &= \int_{a_0} \nabla_\theta \pi_\theta(a_0|s_0) Q^\pi(s_0,a_0) + \pi_\theta(a_0|s_0)\nabla_\theta Q^\pi(s_0,a_0)\mathrm{d}a_0 \\ &= \int_\tau \pi_\theta(a_0|s_0)\left(\nabla_\theta \log \pi_\theta(a_0|s_0) Q^\pi(s_0,a_0) + \sum_{t=1}^{T} \gamma^t p(\tau_{1:t}|\theta)\nabla_\theta \log \pi_\theta(a_t|s_t) Q^\pi(s_t,a_t)\right)\mathrm{d}\tau.\end{aligned} \tag{4.19}$$

Substituting (4.19) into (4.18), we have that

$$\begin{aligned}\nabla_\theta L(\theta,w) &= 2\int_\tau \sum_{t=0}^{T} \gamma^t p(\tau_{0:t}|\theta)\nabla_\theta \log \pi_\theta(a_t|s_t)\left(V^\pi(s_0) - V_w(s_0)\right) Q^\pi(s_t,a_t)\mathrm{d}\tau \\ &= \mathbb{E}_{\tau\sim\pi_\theta}\left[2\sum_{t=0}^{T}\gamma^t \nabla_\theta \log \pi_\theta(a_t|s_t)\left(V^\pi(s_0) - V_w(s_0)\right) Q^\pi(s_t,a_t)\right]\end{aligned}$$

which completes the proof. □

Given these derivations, terms in (4.15) can be estimated by sampled trajectories, and `STAC` updates using (4.11)–(4.12).

4.4.3 Stackelberg DDPG and SAC

In comparison to on-policy methods where the critic is designed to evaluate the actor using sampled trajectories generated by the current policy, in off-policy methods the critic minimizes the Bellman error using samples from a replay buffer. Thus, the leader and follower designation between the actor and critic in off-policy methods is not as clear. To this end, we propose variants of `STDDPG` and `STSAC` where the leader and follower order can be switched. Given the actor as the leader (`AL`), the algorithms are similar to policy-based methods, where the critic plays an approximate best response to evaluate the current actor. On the other hand, given the critic as the leader (`CL`), the actor plays an approximate best response to the critic value, resulting in behavior closely resembling that of the value-based methods.

As shown in (4.7)–(4.8) for `DDPG` and (4.9)–(4.10) for `SAC`, the objective functions of off-policy methods are defined in expectation over an arbitrary distribution from a replay

buffer instead of the distribution induced by the current policy. Thus, each terms in the total derivatives updates in (4.15) and (4.16) can be computed directly and estimated by samples. Then, `STDDPG` and `STSAC` update using (4.11)–(4.12) or (4.13)–(4.14) depending on the choices of leader and follower.

4.4.4 Convergence Guarantee

Consider, without loss of generality, the actor is designated as the leader and the critic the follower. Then, the actor and critic updates with the Stackelberg gradient dynamics and learning rates sequences $\{\alpha_{\theta,k}\}, \{\alpha_{w,k}\}$ are of the form

$$\theta_{k+1} = \theta_k + \alpha_{\theta,k}(\nabla J(\theta, w) + \epsilon_{\theta,k+1}), \tag{4.20}$$

$$w_{k+1} = w_k - \alpha_{w,k}(\nabla_w L(\theta, w) + \epsilon_{w,k+1}), \tag{4.21}$$

where $\{\epsilon_{\theta,k+1}\}, \{\epsilon_{w,k+1}\}$ are stochastic processes. The results in this section assume the following.

Assumption 4.1. *The maps $\nabla J : \mathbb{R}^m \to \mathbb{R}^{m_\theta}$, $\nabla_w L : \mathbb{R}^m \to \mathbb{R}^{m_w}$ are Lipschitz, and $\|\nabla J\| < \infty$. The learning rate sequences are such that $\alpha_{\theta,k} = o(\alpha_{w,k})$ and $\sum_k \alpha_{i,k} = \infty$, $\sum_k \alpha_{i,k}^2 < \infty$ for $i \in \mathcal{I} = \{\theta, w\}$. The noise processes $\{\epsilon_{i,k}\}$ are zero mean, martingale difference sequences: given the filtration $\mathcal{F}_k = \sigma(\theta_s, w_s, \epsilon_{\theta,s}, \epsilon_{w,s},\ s \le k)$, $\{\epsilon_{i,k}\}_{i\in\mathcal{I}}$ are conditionally independent, $\mathbb{E}[\epsilon_{i,k+1}|\ \mathcal{F}_k] = 0$ a.s., and $\mathbb{E}[\|\epsilon_{i,k+1}\||\ \mathcal{F}_k] \le c_i(1 + \|(\theta_k, w_k)\|)$ a.s. for some constants $c_i \ge 0$ and $i \in \mathcal{I}$.*

The following result gives a local convergence guarantee to a local Stackelberg equilibrium under the assumptions. For this result, recall that for a continuous-time dynamical system of the form $\dot{z} = -g(z)$, a stationary point z^* of the system is said to be locally asymptotically stable or simply stable if the spectrum of the Jacobian denoted by $-Dg(z)$ is in the open left half plane.

Theorem 4.2. *Consider an MDP and actor-critic parameters (θ, w). Given a locally asymptotically stable differential Stackleberg equilibrium (θ^*, w^*) of the continuous-time limiting*

system $(\dot{\theta}, \dot{w}) = (\nabla J(\theta, w), -\nabla_w L(\theta, w))$*, under Assumption 4.1 there exists a neighborhood* U *for which the iterates* (θ_k, w_k) *of the discrete-time system in* (4.20)–(4.21) *converge asymptotically almost surely to* (θ^*, w^*) *for* $(\theta_0, w_0) \in U$*.*

Now we provide a proof sketch. Without loss of generality, the actor plays the role of the leader. Consider a differential Stackelberg equilibrium of the game (θ^*, w^*) which is locally asymptotically stable[7] for the continuous time dynamical system

$$\begin{bmatrix} \dot{\theta} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} \nabla J(\theta, w) \\ -\nabla_w L(\theta, w)) \end{bmatrix}$$

where the total derivative of actor in the Stackelberg gradient is given by

$$\nabla J(\theta, w) = \nabla_\theta J(\theta, w) - \nabla_{w\theta}^\top L(\theta, w)(\nabla_w^2 L(\theta, w))^{-1} \nabla_w J(\theta, w).$$

and the individual gradient for the critic is $\nabla_w L(\theta, w)$. The actor and critic employ the discrete time updates given in Algorithm 3 where the actor is the leader. Since the actor and critic have unbiased estimates of their gradients and the learning rates are chosen as stated in Section 4.4.4, then the result of the theorem follows from Theorem 7 in [15]. That is, from an initial point $(\theta_0, w_0) \in U$, the Stackelberg gradient dynamics converge asymptotically to $(\theta^*, w^*) \in U$ almost surely.

Indeed, the result holds by the following reasoning. Under the assumptions on the noise processes and stepsize sequences, we treat the updates in Algorithm 3 as a stochastic approximation process (θ_k, w_k). Then, we define asymptotic pseudo-trajectories—i.e., linear interpolations between iterates (θ_k, w_k) and (θ_{k+1}, w_{k+1}). Since (θ^*, w^*) is locally asymptotically stable, there exists a neighborhood of (θ^*, w^*) and a local Lyapunov function on that neighborhood. This Lyapunov function can be used to show that the continuous time flow also starting from iterates (θ_k, w_k) and the asymptotic pseudo-trajectories are contracting onto one another asymptotically, for any sequence of iterates starting at $(\theta_0, w_0) \in U$. Hence, the iterates (θ_k, w_k), in turn, converge asymptotically to (θ^*, w^*) almost surely.

[7] That is, the local linearization of the above dynamics around the point (θ^*, w^*) are in the open left-half complex plane.

Comments on designing gradient estimators. Methods such as REINFORCE (or Monte Carlo method) provide an unbiased estimator of the follower's individual gradient. Obtaining an unbiased estimate of the total derivative for the leader, on the other hand, is a bit more nuanced. This is because there are multiple gradients being multiplied by one another in the expectation. However, as a heuristic, one way to approximate it is using the expected value of each of the terms that shows up in the total derivative.

Depending on the actor-critic algorithm and objective functions, following either Theorem 4.1 (Proposition 4.1) or direct derivatives, each term in the total derivative can be computed as an expectation over a distribution of state and action (generated by current policy in `AC` and any arbitrary policy in `DDPG` and `SAC`). Take `DDPG` as an example where $J(\theta, w) = \mathbb{E}_{\xi\sim\mathcal{D}}\left[Q_w(s, \mu_\theta(s))\right]$, and $L(\theta, w) = \mathbb{E}_{\xi\sim\mathcal{D}}\left[\left(Q_w(s,a) - (r + \gamma Q_0(s', \mu_\theta(s')))\right)^2\right]$. The second term in total derivative appears to be a multiplication of several expectations:

$$
\begin{aligned}
\nabla J(\theta, w) &= \nabla_\theta J(\theta, w) - \nabla^\top_{w\theta} L(\theta, w)(\nabla^2_w L(\theta, w))^{-1}\nabla_w J(\theta, w) \\
&= \mathbb{E}_{\xi\sim\mathcal{D}}\left[\nabla_\theta Q_w(s, \mu_\theta(s))\right] - \mathbb{E}_{\xi\sim\mathcal{D}}\Bigg[\nabla_{w\theta}\left(\left(Q_w(s,a) - (r + \gamma Q_0(s', \mu_\theta(s')))\right)^2\right)^\top \\
&\quad \left(\nabla^2_w\left(\left(Q_w(s,a) - (r + \gamma Q_0(s', \mu_\theta(s')))\right)^2\right)\right)^{-1}\nabla_w Q_w(s, \mu_\theta(s))\Bigg] \\
&\approx \mathbb{E}_{\xi\sim\mathcal{D}}\left[\nabla_\theta Q_w(s, \mu_\theta(s))\right] - \mathbb{E}_{\xi\sim\mathcal{D}}\left[\nabla_{w\theta}\left(\left(Q_w(s,a) - (r + \gamma Q_0(s', \mu_\theta(s')))\right)^2\right)\right]^\top \\
&\quad \left(\mathbb{E}_{\xi\sim\mathcal{D}}\left[\nabla^2_w\left(\left(Q_w(s,a) - (r + \gamma Q_0(s', \mu_\theta(s')))\right)^2\right)\right]\right)^{-1}\mathbb{E}_{\xi\sim\mathcal{D}}\left[\nabla_w Q_w(s, \mu_\theta(s))\right].
\end{aligned}
$$

For this approximation, we can obtain an unbiased estimate by resetting the simulator as described in [92, Chapter 11] to estimate each term in the product of expectations. As a result, this is a reasonable heuristic in practice for an approximation to the total derivative. Our policy gradient theorems also provide us a way to derive the estimates of each of these individual terms. Obtaining unbiased estimates as an active area of research (see, e.g., [81, 98]). Moreover, from both a theoretical and practical perspective, understanding how the batch size affects the estimate of follower Hessian and the total derivative remains open.

This result is effectively giving the guarantee that the discrete-time dynamics locally converge to a stable, game theoretically meaningful equilibrium of the continuous-time system

using stochastic approximation methods given suitable learning rate sequences and unbiased gradient estimates [99].

4.4.5 Implicit Map Regularization

The total derivative in the Stackelberg gradient dynamics requires computing the inverse of follower Hessian $\nabla_2^2 f_2(x)$. Since critic networks in practical reinforcement learning problems may be highly non-convex, $(\nabla_2^2 f_2(x))^{-1}$ can be ill-conditioned. Thus, instead of computing this term directly in the Stackelberg actor-critic algorithms, we compute a regularized variant of the form $(\nabla_2^2 f_2(x) + \lambda I)^{-1}$. This regularization method can be interpreted as the leader viewing the follower as optimizing a regularized cost $f_2(x) + \frac{\lambda}{2}\|x_2\|^2$, while the follower actually optimizes $f_2(x)$. Interestingly, the regularization parameter λ can serve to interpolate between the Stackelberg and individual gradient updates for the leader as we now formalize.

Proposition 4.2. *Consider a Stackelberg game where the leader updates using the regularized total derivative* $\nabla^\lambda f_1(x) = \nabla_1 f_1(x) - \nabla_{21}^\top f_2(x)(\nabla_2^2 f_2(x) + \lambda I)^{-1}\nabla_2 f_1(x)$. *As* $\lambda \to 0$ *then* $\nabla^\lambda f_1(x) \to \nabla f_1(x)$ *and when* $\lambda \to \infty$ *then* $\nabla^\lambda f_1(x) \to \nabla_1 f_1(x)$.

4.5 Experiments

We now show the results of extensive experiments comparing the Stackelberg actor-critic algorithms with the comparable actor-critic algorithms. We find that the actor-critic algorithms with the Stackelberg gradient dynamics always perform at least as well and often significantly outperform the standard gradient dynamics. Moreover, we provide game-theoretic interpretations of the results.

We run experiments on the OpenAI gym platform [100] with the Mujoco Physics simulator [101]. The performance of each algorithm is evaluated by the average episode return versus the number of time steps (state transitions after taking an action according to the policy). For a fair comparison, the hyper-parameters for the actor and critic including the neural

network architectures are set equal when comparing the Stackelberg actor-critic algorithms with the stand normal actor-critic algorithms.

Implementation Details This section includes complete details about our experiments. Our implementation is developed based on public resource Spinning Up[8] and our source code is available at `https://github.com/LeoZhengZLY/stackelberg-actor-critic-algos`.

We follow the default neural network architecture used in Spinning Up. Particularly, the `AC` and `STAC` use networks of size (64, 32) with `tanh` units for both the policy and the value function. The `DDPG`, `STDDPG`, `SAC`, and `STSAC` use networks of size (256, 256) with `relu` units. The `AC` and `STAC` collected 4000 steps of agent-environment interaction per batch and use vanilla gradient descent optimizer and the `DDPG`, `STDDPG`, `SAC`, and `STSAC` use Adam optimizer with mini-batches of size 100 at each gradient descent step.

The policy gradient terms for `AC` and `STAC` are estimated by generalized average estimator (GAE) [97] and critics are updated by Monte Carlo method [73]. In discrete control task (`CartPole`), we set the Hessian regularization hyper-parameter $\lambda = 0$, and in continuous control tasks (others), we set the regularization hyper-parameter $\lambda = 500$.

The performances for `AC` and `STAC` are measured as the average trajectory return across the batch collected at each epoch. Performances for `DDPG`, `STDDPG`, `SAC`, and `STSAC` are measured once every $10,000$ steps by running the deterministic policy (or, in the case of `SAC`, the mean policy) without action noise for ten trajectories, and reporting the average return over those test trajectories.

In our Stackelberg framework, the learning rule for the leader involves computing an inverse-Hessian-vector product for the $\nabla_2^2 f_2(x_1, x_2)$ inverse term and Jacobian-vector product for the $\nabla_{12} f_2(x_1, x_2)$ terms. The second term can be computed directly by `autograd.grad` in `torch`. For the inverse-Hessian-vector term, we implement the conjugate gradient method using `autograd.grad` iteratively. This enable us to compute and estimate the total derivative on GPU directly and perform Stackelberg gradient update. In all experiments, the

[8]Developed by Josh Achiam in 2018: `https://spinningup.openai.com/en/latest/`

Algorithm 4: Stackelberg Actor-Critic Framework with Unrolling Follower Update and Regularization

Input: actor-critic algorithm `ALG`, player designations, follower unrolling steps m, regularization hyperparameter λ, and learning rate sequences $\alpha_{1,k}, \alpha_{2,k}$.

for $k = 0, 1, 2 \dots$ **do**

if actor is leader, **then** update actor and critic in `ALG` with

$$\theta_{k+1} = \theta_k + \alpha_{1,k}(\nabla_\theta J(\theta_k, w_{k,0}) - (\nabla^\top_{w\theta} L \circ (\nabla^2_w L + \lambda I)^{-1} \circ \nabla_w J)(\theta_k, w_{k,0}))$$

$$w_{k,l+1} = w_{k,l} - \alpha_{2,k} \nabla_w L(\theta_k, w_{k,l}), \quad l \in [0, m-1]$$

$$w_{k+1,0} = w_{k,m}$$

if critic is leader, **then** update actor and critic in `ALG` with

$$w_{k+1} = w_k - \alpha_{1,k}(\nabla_w L(\theta_{k,0}, w_k) - (\nabla^\top_{\theta w} J \circ (\nabla^2_\theta J + \lambda I)^{-1} \circ \nabla_\theta L)(\theta_{k,0}, w_k))$$

$$\theta_{k,l+1} = \theta_{k,l} + \alpha_{2,k} \nabla_\theta J(\theta_{k,l}, w_k), \quad l \in [0, m-1]$$

$$\theta_{k+1,0} = \theta_{k,m}$$

end

Stackelberg versions of actor-critic algorithms roughly take twice the time to train.

In Algorithm 4, we provide a more detailed version of our Stackelberg actor-critic algorithm framework when multiple follower unrolling steps and implicit map regularization are involved.

Performance. Figures 4.6(a)–4.6(d) show the performance of `STAC` and `AC` on several tasks. We also experiment with the common heuristic of "unrolling" the critic m steps between actor steps. For each task, `STAC` with multiple critic unrolling steps performs the best. This is due to the fact when the critic is closer to the best response, then the real response of the critic is closer to what is anticipated by the Stackelberg gradient for the actor. Interestingly, in CartPole, `STAC` with $m = 1$ performs even better than `AC` with $m = 80$.

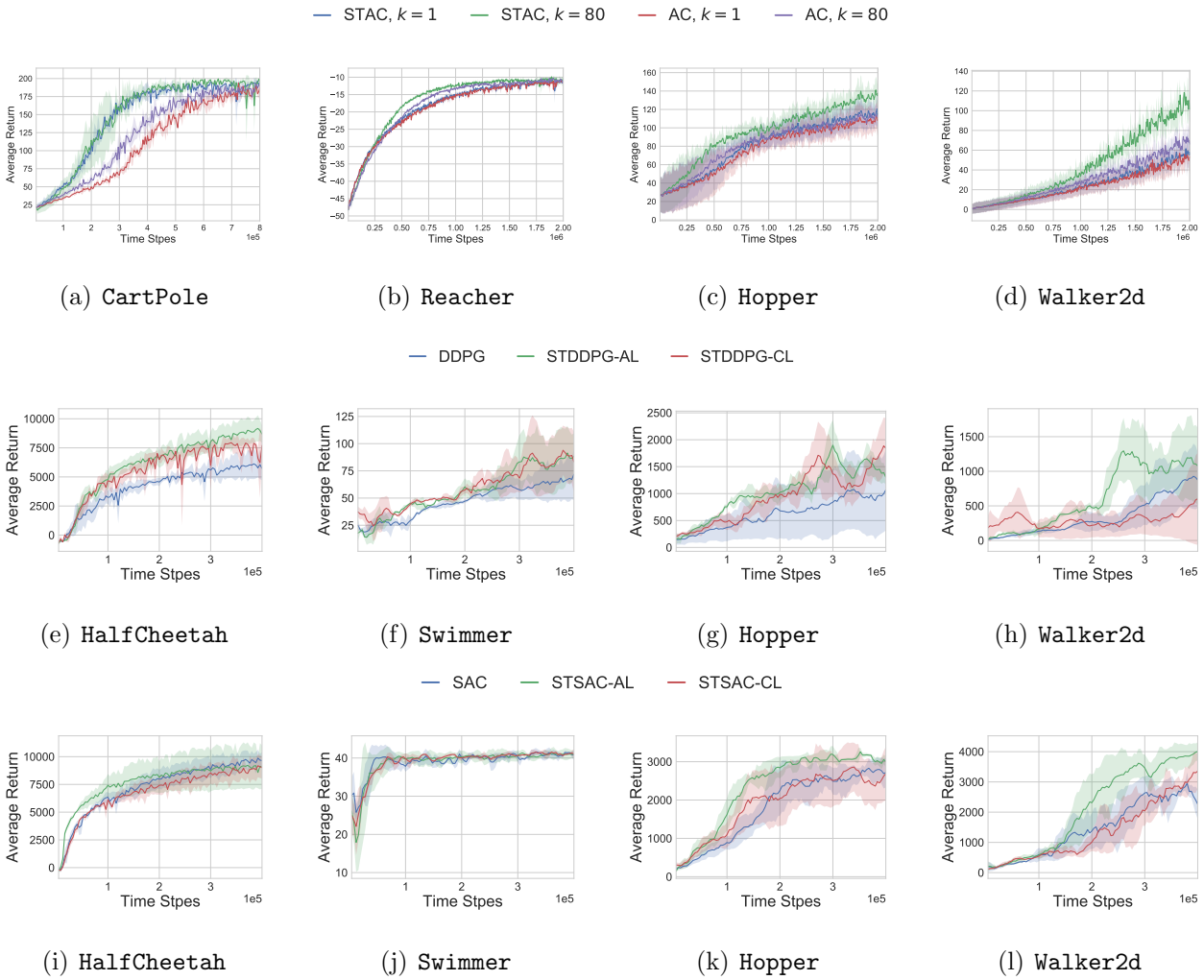


Figure 4.6: Comparison of `AC`, `DDPG`, `SAC` with their Stackelberg versions on OpenAI gym environments. Note in (a)–(d) the Stackelberg versions are red/purple and in (e)-(l) they are green/red.

Figures 4.6(e)–4.6(h) show the performance of `STDDPG-AL` and `STDDPG-CL` in comparison to `DDPG`. We observe that on each task, `STDDPG-AL` outperforms `DDPG` by a clear margin, whereas `STDDPG-CL` has overall better performance than `DDPG` except on `Walker2d`. Figures 4.6(i)–4.6(l) show the performance of `STSAC-AL` and `STSAC-CL` in comparison to `SAC`. For this formulation, the advantage afforded by the Stackelberg gradient is not as apparent.

In all experiments, when the actor is the leader, the Stackelberg versions either outperform

or are comparable to the existing actor-critic algorithms, offering compelling evidence that the Stackelberg framework has an empirical advantage in many tasks and settings. We now provide game-theoretic interpretations of the experimental results and connect back to the examples and observations from Section 4.3.2.

Game-Theoretic Interpretations. `SAC` is considered the state-of-the-art model-free reinforcement learning algorithm and we observe it significantly outperforms `DDPG` (e.g., on `Hopper` and `Walker2d`). The common interpretation of its advantage is that `SAC` encourages exploration by penalizing low entropy policies. Here we provide another viewpoint.

From a game-theoretic perspective, the objective functions of `AC` and `DDPG` take on hidden linear and hidden quadratic structures for the actor and critic. This structure can result in cyclic behavior for individual gradient dynamics as shown in Section 4.3.2. `SAC` constructs a more well-conditioned game structure by regularizing the actor objective, which leads to the learning dynamics converging more directly to the equilibrium as seen in Section 4.3.2. This also explains why we observe improved performance with `STAC` and `STDDPG-AL` compared to `AC` and `DDPG`, but the performance gap between `STSAC-AL` and `SAC` is not as significant.

Comparing `AL` with `CL`, the actor as the leader always outperforms the critic as the leader in our experiments. As described in Section 4.3.2, the critic objective is typically a quadratic mean square error objective which results in a hidden quadratic structure whereas the actor's objective typically is in the form of a hidden linear due to parameterization of the Q network and policy. As a result, the critic cost structure is more well-suited for computing an approximate local best response since it is more likely to be well-conditioned. Thus, the critic being the follower is a more natural hierarchical structure of the game. Unrolling the critic for multiple steps to approximate this structure and has been shown to perform well empirically [25]. Algorithm 4 shows a similar heuristic can be employed for the Stackelebrg framework.

4.6 Conclusion

In this work, we revisit the standard actor-critic algorithms from a game-theoretic perspective to capture the hierarchical interaction structure and introduce a Stackelberg framework for actor-critic algorithms. In this framework, we introduce novel Stackelberg versions of existing actor-critic algorithms. In experiments on a number of environments, we show that the Stackelberg actor-critic algorithms always outperform the existing counterparts when the actor plays the leader.

Chapter 5

STACKELBERG COMPETITIVE REINFORCEMENT LEARNING

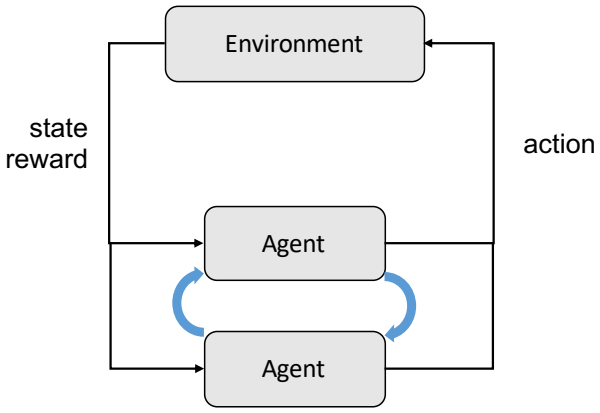


Figure 5.1: Framework of multi-agent reinforcement learning algorithm with player interaction.

5.1 Introduction

Beyond single-agent paradigm, a wide variety of reinforcement learning applications involve the participation of more than one single agent/player, including robotic teams, distributed control, resource management, cyber-physical systems, etc. [102, 103]. These applications are modeled systematically as multi-agent reinforcement learning problems. Specifically, multi-agent reinforcement learning addresses the sequential decision-making problem of multiple autonomous agents that interact with others in a common environment, each of which aims to optimize its own long-term return [13].

Among the multi-agent reinforcement learning problems, the competitive setting is a

critical part, where there are usually two agents and the return of agents sum up to zero. These problems are formulated as zero-sum two-player games in the framework of competitive Markov decision process [104].

In both single-agent and multi-agent reinforcement learning problems, policy gradient based methods are prominent algorithms that directly optimize for policies, which enjoy simplicity in implementation and deployment [73]. In two-player zero-sum competitive MDP, the policy gradient based methods approximate the gradient of the objective of each player by samples and perform gradient descent ascent on the decision spaces. However, it has been shown that in zero-sum games, simultaneous gradient descent ascent could converge to spurious locally asymptotically stable points that lack game-theoretic meaning [105]. Moreover, in some game structures such as bilinear games, divergence behavior is observed [79].

Recently, novel algorithms have been proposed to break the individual gradient paradigm and exploit the game-theoretic nature of two-player games. Figure 5.1 provides a abstract framework of those algorithms. Instead of assuming the other player as stationary, these algorithms account for the interaction of learning agents and update each agent by conjecturing other player's learning dynamic. For example, LOLA proposed in [77] leverages a second-level reasoning (a player thinks that the other player thinks its strategy stays constant) and CoPG [79] uses a infinite-level reasoning update rule to optimize each agent. These algorithms show positive results on a set of games with better convergence property. However, one limitation of those approaches is that they treat two players with equal priority, and specifically, in [77], two players need to have exactly the same decision spaces.

In fact, in a wide variety of applications, the performance of one player is favored over the other. For example, the vanilla formulation of generative adversarial networks [12] is a zero-sum game with a generator and a discriminator. In most of the applications of generative adversarial networks, the generator plays a more important role and the discriminator is only utilized as a "teacher" to guide the training of the generator. Similarly, in the competitive reinforcement learning setting, the second player could be the potential perturbation or be a learning opponent to guide the training of the primal player. In competitive human-robot

interaction, the second player could be a human expert trying to compete and teach the robot. In those cases, we would like the primal agent to benefit from learning from competing with another learning agent.

From a game-theoretic perspective, the Stackelberg game framework fits this motivation in this setting. According to [74, Chapter 4], in the two-player game with unique follower best responses, the payoff of the leader in Stackelberg equilibrium is better than Nash. Leveraging the fact that the leader benefits from the hierarchical order in Stackelberg games, in this work, we formulate the competitive reinforcement learning as a Stackelberg game. We extend the current state-of-the-art multi-agent reinforcement learning algorithm MADDPG [106] to its Stackelberg version, termed ST-MADDPG, adopting the total derivative Stackelberg learning update rule. Moreover, we also design and open-source new competitive reinforcement learning benchmark tasks and demonstrate the performance and behavior of our algorithm on them. Empirically, we show that the leader in Stackelberg MADDPG algorithm learns a better policy and gains advantages in the competitive game.

5.2 Background

5.2.1 Competitive Markov Game

In this work, we consider a two-player zero-sum fully observable competitive Markov game (also called competitive MDP). A competitive Markov game is a tuple of $(\mathcal{S}, \mathcal{A}^1, \mathcal{A}^2, P, r)$, where $\mathcal{S}$ is the state space, $s \in \mathcal{S}$ is a state, for player $i \in \{1, 2\}$, A^i is the player i's action space with $a^i \in \mathcal{A}^i$. $P : \mathcal{S} \times \mathcal{A}^1 \times \mathcal{A}^2 \to \mathcal{S}$ is the transition kernel such that $P(s'|s, a^1, a^2)$ is the probability of transitioning to state s' given that the previous state was s and the agents took action a^1, a^2 simultaneously in s. $r : \mathcal{S} \times \mathcal{A}^1 \times \mathcal{A}^2 \to [0, 1]$ is the reward function of player 1 and by the zero-sum nature of the competitive setting, player 2 receives the negation of r as its own reward feedback. Each agent uses a stochastic policy π^i_θ, parameterized by θ^i.

A trajectory $\tau = (s_0, a^1_0, a^2_0, \ldots, s_T, a^1_T, a^2_T)$ gives the cumulative rewards or return defined as $R(\tau) = \sum_{t=0}^{T} \gamma^t r(s_t, a^1_t, a^2_t)$, where the discount factor $0 < \gamma \leq 1$ assigns weights to rewards

received at different time steps. The expected return of $\pi = \{\pi^1, \pi^2\}$ after executing a_t^1, a_t^2 in state s_t can be expressed by the Q function

$$Q^\pi(s_t, a_t^1, a_t^2) = \mathbb{E}_{\tau\sim\pi}\big[\sum_{t'=t}^{T} \gamma^{t'-t} r(s_{t'}, a_{t'}^1, a_{t'}^2) | s_t, a_t^1, a_t^2\big]. \tag{5.1}$$

Correspondingly, the expected return of π in state s_t can be expressed by the value function V defined as

$$V^\pi(s_t) = \mathbb{E}_{\tau\sim\pi}\big[\sum_{t'=t}^{T} \gamma^{t'-t} r(s_{t'}, a_{t'}^1, a_{t'}^2) | s_t\big]. \tag{5.2}$$

The game objective is defined as the expected return given by

$$\begin{aligned} J(\pi) &= \mathbb{E}_{\tau\sim\pi}\big[\sum_{t=0}^{T} \gamma^t r(s_t, a_t^1, a_t^2)\big] = \int_\tau p(\tau|\pi) R(\tau) \mathrm{d}\tau \\ &= \mathbb{E}_{s\sim\rho, a^1\sim\pi^1(\cdot|s), a^2\sim\pi^2(\cdot|s)}\big[Q^\pi(s, a^1, a^2)\big], \end{aligned} \tag{5.3}$$

where $p(\tau|\pi) = \rho(s_0)\prod_{t=0}^{T} \pi^1(a_t^1|s_t)\pi^2(a_t^2|s_t)P(s_{t+1}|s_t, a_t^1, a_t^2)$ and $\rho(s_0)$ is the initial state distribution.

In competitive Markov games, player 1 aims to find a policy maximizing the game objective, while player 2 aims to minimize it. They solve for $\max_{\theta^1} J(\pi^1, \pi^2)$ and $\min_{\theta^2} J(\pi^1, \pi^2)$ respectively.

5.2.2 MADDPG

It has been shown that naive policy gradient methods perform poorly in simple multi-agent settings experiments [106]. Thus, more advanced multi-agent reinforcement learning algorithms have been proposed and MADDPG [106] is one of the state-of-the-art. The idea of MADDPG is to adopt the framework of centralized training with decentralized execution. Specifically, they use a centralized critic network Q_w to approximate the Q^π function, and update the policy network π_θ^i of each agent using the global critic.

Consider the deterministic policy setting, each player has policy μ_θ^i, with parameter θ^i. The game objective is

$$J(\theta^1, \theta^2) = \mathbb{E}_{\xi\sim\mathcal{D}}\left[Q_w(s, \mu_\theta^1(s), \mu_\theta^2(s))\right]. \tag{5.4}$$

where $\xi = (s, a^1, a^2, r, s')$, $\mathcal{D}$ is a replay buffer.

The policy gradient of each player can be written as

$$\nabla_{\theta^1} J(\theta^1, \theta^2) = \mathbb{E}_{\xi \sim \mathcal{D}} \left[\nabla_{\theta^1} \mu_\theta^1(s) \nabla_{a^1} Q_w(s, a^1, a^2)|_{a^1 = \mu_\theta^1(s)} \right], \tag{5.5}$$

and

$$\nabla_{\theta^2} J(\theta^1, \theta^2) = \mathbb{E}_{\xi \sim \mathcal{D}} \left[\nabla_{\theta^2} \mu_\theta^2(s) \nabla_{a^2} Q_w(s, a^1, a^2)|_{a^2 = \mu_\theta^2(s)} \right]. \tag{5.6}$$

The critic objective is defined as the mean square Bellman error

$$L(w) = \mathbb{E}_{\xi \sim \mathcal{D}}[\big(Q_w(s, a^1, a^2) - (r + \gamma Q_{w'}(s', \mu_{\theta'}^1(s'), \mu_{\theta'}^1(s'))\big)^2], \tag{5.7}$$

where $Q_{w'}$ and $\mu_{\theta'}^1, \mu_{\theta'}^2$ are target networks obtained by polyak averaging the Q_w and $\mu_\theta^1, \mu_\theta^2$ network parameters over the course of training.

In MADDPG, the centralized critic is updated by gradient descent and the two agent's policy are update by simultaneous gradient descent and ascent

$$\theta^1 \leftarrow \theta^1 + \alpha^1 \nabla_{\theta^1} J(\theta^1, \theta^2), \tag{5.8}$$

$$\theta^2 \leftarrow \theta^2 - \alpha^2 \nabla_{\theta^2} J(\theta^1, \theta^2). \tag{5.9}$$

5.3 Stackelberg MADDPG Algorithm

In this section, we propose the novel Stackelberg MADDPG (ST-MADDPG) algorithm. The original MADDPG algorithm updates two agent's policies simultaneously in a decentralized way. The implicit assumption made behind this algorithm is that the other player is viewed as a part of the stationary environment when updating one agent. As presented in [79], this simultaneous gradient descent ascent updates could lead to poor convergence property in practice. Similar to the ideas proposed in [77, 79], we would like the learning algorithm to be able to account for the interaction of agents' learning dynamics.

Beyond the interaction consideration, we also want the algorithm to have a hierarchical structure that favors one agent's performance over the other. As illustrated in Section 5.1,

Algorithm 5: ST-MADDPG algorithm

for episodes $k = 1, 2, \ldots, K$ **do**

receive initial state s_0;

for $t = 1, 2, \ldots, T$ **do**

for each agent i, select action $a^i = \mu^i_\theta(s)$ according to the corrent policy;

execute actions (a^1, a^2) and observe reward r and new state s';

store (s, a^1, a^2, r, s') in replay buffer $\mathcal{D}$;

$s \leftarrow s'$;

sample a random minibatch of N transitions $(s_i, a^1_i, a^2_i, r_i, s'_i)$ from $\mathcal{D}$;

set $y_i = r_i + Q_{w'}(s_i, \mu^1_{\theta'}(s_i), \mu^1_{\theta'}(s_i))$;

update the critic by minimizing the loss:

$$L(w) = \frac{1}{N}\sum_{i=1}^{N}[\big(Q_w(s_i, a^1_i, a^2_i) - y_i\big)^2]$$

update the leader policy using the total gradient computed by (5.12):

$$\theta^1 \leftarrow \theta^1 + \alpha^1 \nabla J(\theta^1, \theta^2)$$

update the follower policy using the policy gradient:

$$\theta^2 \leftarrow \theta^2 - \alpha^2 \nabla_{\theta^2} J(\theta^1, \theta^2)$$

update the target networks:

$$w' \leftarrow \tau w + (1 - \tau) w'$$

$$\theta^{i'} \leftarrow \tau \theta^i + (1 - \tau)\theta^{i'}$$

end

end

in a wide variety of applications, we would like the primal agent to benefit from learning from competing with the other learning agent.

Motivated by these requirements, we propose the ST-MADDPG algorithm, detailed in Algorithm 5. We leverage a Stackelberg game formulation in the agents' policy update rules, with one player set to be leader and the other follower. For the detailed definition of Stackelberg game, Stackelberg equilibrium, and total derivative update rule, we refer to Section 4.3.1.

Specifically, in ST-MADDPG algorithm, if we set player 1 to be the leader, the update rules of both player's policy are

$$\theta^1 \leftarrow \theta^1 + \alpha^1 \nabla J(\theta^1, \theta^2), \tag{5.10}$$

$$\theta^2 \leftarrow \theta^2 - \alpha^2 \nabla_{\theta^2} J(\theta^1, \theta^2), \tag{5.11}$$

where $\nabla J(\theta^1, \theta^2)$ is the total derivative

$$\nabla J(\theta^1, \theta^2) = \nabla_{\theta^1} J(\theta^1, \theta^2) - \nabla_{\theta^1 \theta^2} J(\theta^1, \theta^2)(\nabla^2_{\theta^2} J(\theta^1, \theta^2))^{-1} \nabla_{\theta^2} J(\theta^1, \theta^2). \tag{5.12}$$

The second order terms of the total derivative in (5.12) can be computed by applying chain rule directly

$$\nabla_{\theta^1 \theta^2} J(\theta^1, \theta^2) = \mathbb{E}_{\xi \sim \mathcal{D}} \left[\nabla_{\theta^1} \mu^1_\theta(s) \nabla_{a^1 a^2} Q_w(s, a^1, a^2) (\nabla_{\theta^2} \mu^2_\theta(s))^T |_{a^1 = \mu^1_\theta(s), a^2 = \mu^2_\theta(s)} \right], \tag{5.13}$$

and

$$\nabla^2_{\theta^2} J(\theta^1, \theta^2) = \mathbb{E}_{\xi \sim \mathcal{D}} \left[\nabla^2_{\theta^2} \mu^2_\theta(s) \nabla_{a^2} Q_w(s, a^1, a^2) |_{a^2 = \mu^2_\theta(s)} \right]. \tag{5.14}$$

In ST-MADDPG, to obtain an estimator of the total derivative $\nabla J(\theta^1, \theta^2)$, each part of (5.12) are computed by samples from replay buffer respectively.

5.4 Experiments

5.4.1 Benchmark Environments

Unlike the popular OpenAI gym in single-agent reinforcement learning settings, there are limited commonly used benchmark environments for multi-agent reinforcement learning tasks,

especially for two-agent competitive settings. To provide a benchmark for our purpose and future research in multi-agent reinforcement learning, we create several environments in the open-sourced repository `https://github.com/bchasnov/masuite`.

CartPole. The control of the `CartPole` system is widely used as a benchmark problem for testing the efficiency of reinforcement learning algorithms. In the `CartPole` system, a pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The system is controlled by applying a force within $[-1, 1]$ to the cart. The pendulum starts upright, and the goal is to prevent it from falling over. A reward of $+1$ is provided for every timestep that the pole remains upright within the threshold angle (30 degrees here). The episode ends when the pole is more than the threshold angle from vertical, or the cart moves more than 2.4 units from the center.

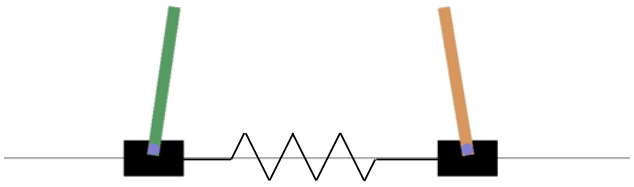


Figure 5.2: CartPole2P environment.

CartPole2P. We create a novel two-player competitive environment called `CartPole2P`. It is basically two coupled `CartPole` agents and the illustrating figure of this environment can be found in Figure 5.2. To add the competitive feature to the task, we make two specific designs.

1. The two single `CartPole` are connected by a spring with an initial unforced length. When the distance of two `CartPole` is longer or shorter than the unforced length, the same magnitude of force will be applied on both `CartPole` with opposite direction.

2. The reward received by two agents are their original reward (+1 if upright, 0 if ended), subtract the original reward of the other. When both poles are upright, they each get 0 rewards. Once one agent ends, that agent will keep receiving reward -1 and the surviving agent will get reward +1 for every timestep until the end of the system. The `CartPole2P` episode ends when both agents end. In this reward setting, the `CartPole2P` environment is a zero-sum competitive Markov game. The goal of each agent is to prevent its own pole from falling over, while seeking to push or pull the other agent through the spring to break the balance of the opponent.

Note that in the current version of `CartPole2P`, we are not accounting for the collision of two agents, meaning that they can "pass through" each other. Adding the collision could potentially make the environment more complicated and we leave that as a future experiment.

5.4.2 Results

Now we show the experiment results of comparing the ST-MADDPG with MADDPG. We run both algorithms on the `CartPole2P` and observe the behaviors of the learned policies. We train both algorithms with 4 random seeds, and the trained policies are run with 2000 maximal episode steps. The episode return of agent 1 and the episode length are then observed. The value of the episode return means how many timesteps agent 1 is able to survive after agent 2 falls and the negative value represents that agent 1 ends before agent 2. If the episode length is 2000 and the return is 0, it is a tie game and both agents are able to survive until the end.

Table 5.1 and 5.2 show the statistics of 100 episodes running the trained policy with each random seed. In Table 5.1, in most cases, the ability of agent 1 and agent 2 are balanced. For seeds 1 and 4, agent 1 wins and losses a part of games in the total 100 episodes. For seed

MADDPG	seed 1	seed2	seed3	seed 4
average episode return	-399	0	42	65.7
std episode return	647	0	13.4	62.4
max episode return	308	0	50	140
min episode return	-1880	0	0	-56
average episode length	666	2000	570	495

Table 5.1: Statistics about the trajectory running the policy learned by MADDPG.

ST-MADDPG	seed 1	seed2	seed3	seed 4
average episode return	1700	-35.9	0	47.1
std episode return	186	33.9	0	4.45
max episode return	1890	7	0	54
min episode return	1300	-87	0	38
average episode length	2000	107	2000	164

Table 5.2: Statistics about the trajectory running the policy learned by ST-MADDPG.

2, both agents always survive until the end of each episode and for seed 3, agent 1 always survives longer with a small amount of steps. In Table 5.2, agent 1 (the leader) tends to have learned a better policy than the other. For seeds 1 and 4, agent 1 wins all 100 games. Specifically, for seed 1, agent 1 always survives until the end of the episode and it wins at least 1300 timesteps longer than agent 2. We also observe that ST-MADDPG can also result in a balanced local equilibrium as shown in the case with seeds 2 and 3.

Figure 5.3 and 5.4 shows the frames of a representative trajectory when testing the policy learned from MADDPG and ST-MADDPG respectively. Note that in these frames the spring is hidden. We can observe that with the policy learned from MADDPG, agents are competing with the other intensely by pushing and pulling the other agent. While they

are able to keep their own pole upright, they fail to break the balance of the other agent and win the game. Meanwhile, in the trajectory of ST-MADDPG, the leader manages to learn a policy to pull the follower out of the frame and survive longer.

5.5 Conclusion

In this work, we study two-player competitive reinforcement learning problems. In order to let one player take advantage from the competition and benefit from learning with a learning opponent, we adopt the hierarchical Stackelberg game formulation and proposed the novel Stackelberg MADDPG algorithm. We also design and open-source new competitive reinforcement learning benchmark tasks and demonstrate the performance and behavior of our algorithm on them. Empirically, we show that the leader in the Stackelberg MADDPG algorithm learns a better policy and gains advantages in the competitive game.

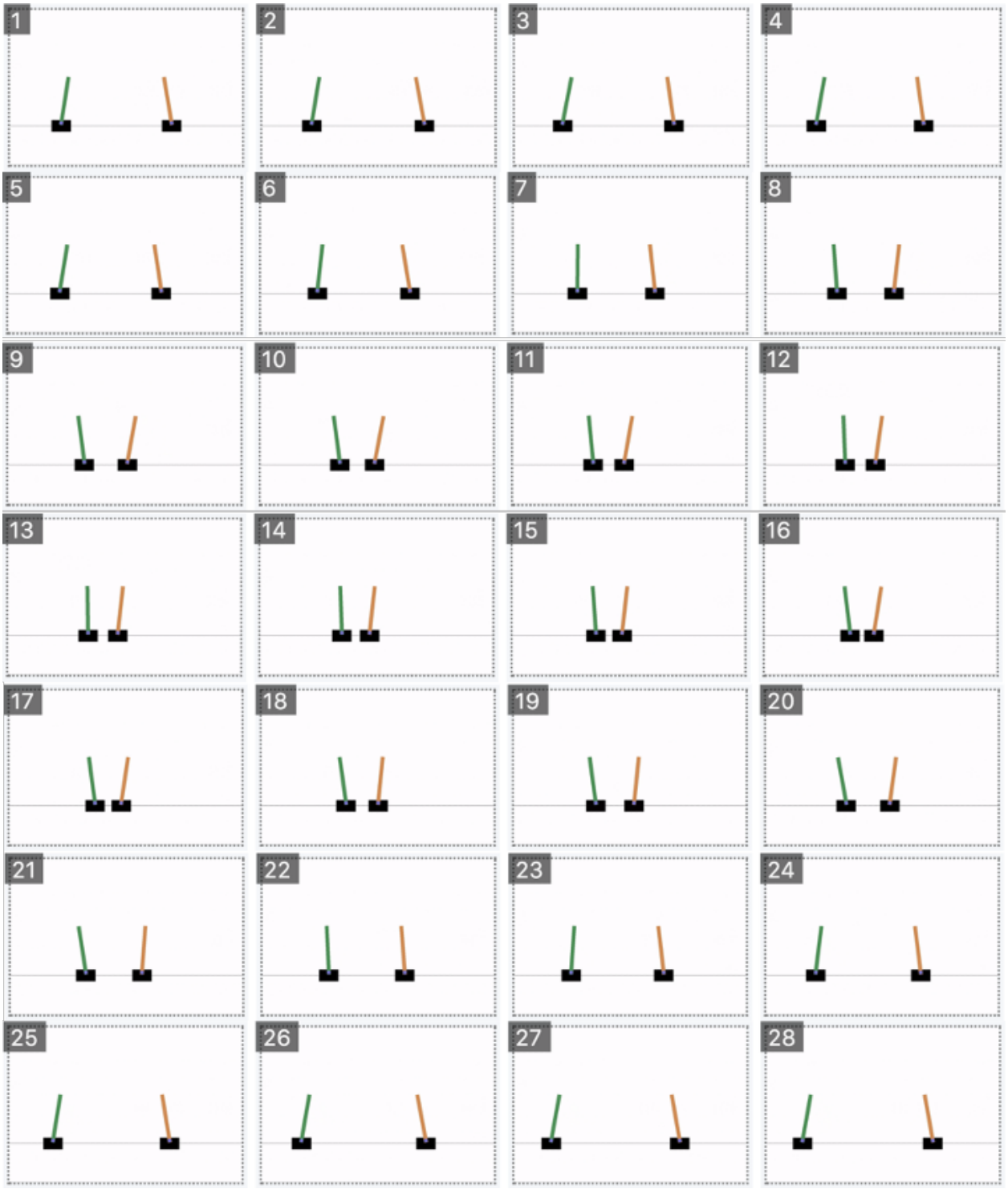


Figure 5.3: Frames of the trajectory running the policy learned by MADDPG.

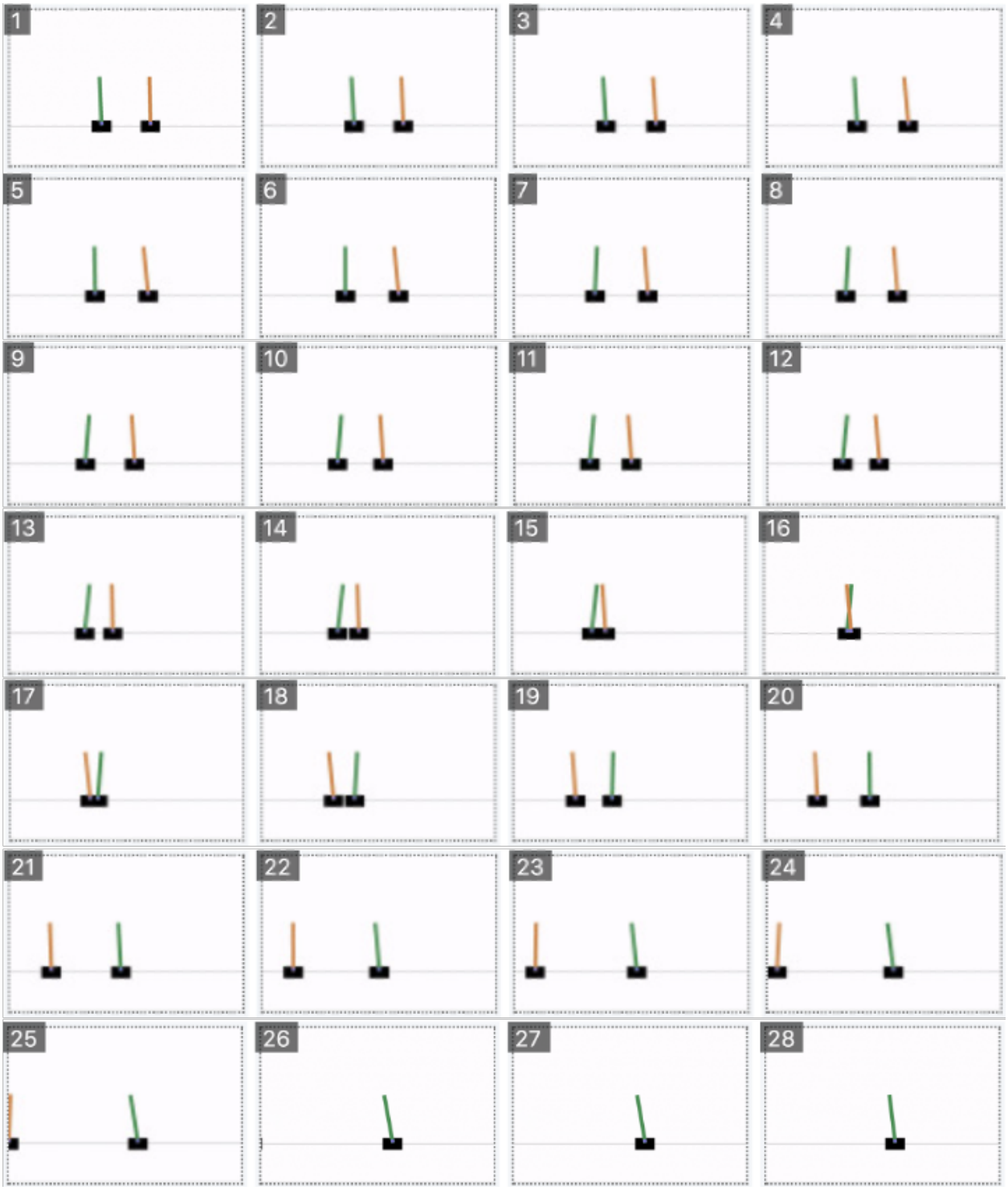


Figure 5.4: Frames of the trajectory running the policy learned by ST-MADDPG. In this example the green `CartPole` is the leader and the orange one is the follower.

Chapter 6

CONCLUSION AND FUTURE DIRECTIONS

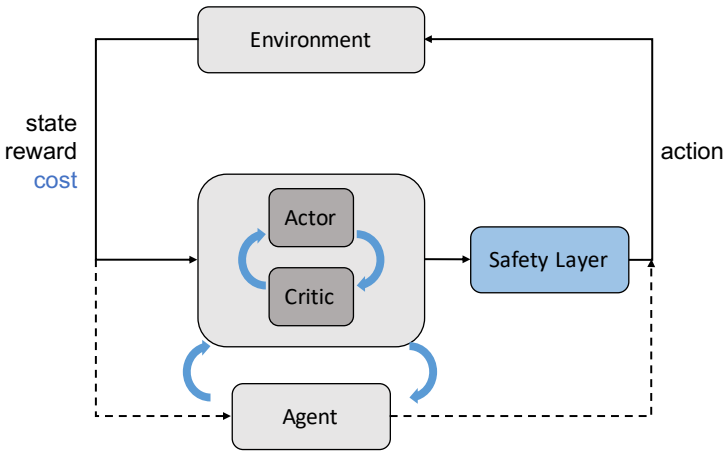


Figure 6.1: Reinforcement learning framework summarizing the algorithms in this thesis.

In this thesis, we address reinforcement learning problems from safety and game-theoretic perspectives and propose several novel certifiable reinforcement learning algorithms. In each chapter, we tackle one specific problem formulation and application, which represents one sub-problem in the whole reinforcement learning area. Figure 6.1 shows the framework that groups and summarizes all the algorithms in this thesis together. The algorithms we propose contribute to the steps toward certifiable reinforcement learning from safety and game-theoretic perspectives under general complex environments. These approaches obtain safety guarantee or convergence property both theoretically and empirically.

In addition to the conclusions given in each chapter, a number of generalizations and discussions on future directions can be drawn from this thesis.

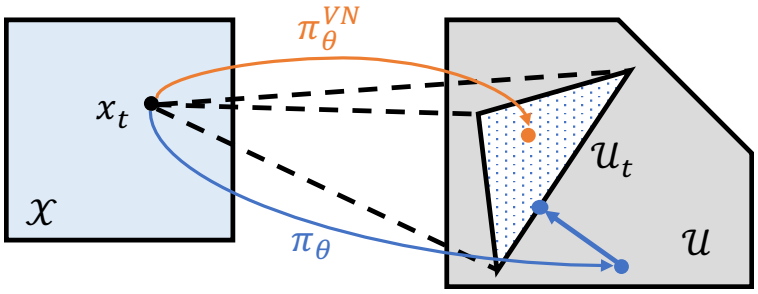


Figure 6.2: Comparison of projection-based method to Vertex Network.

6.1 Safe Reinforcement Learning

Safe Reinforcement Learning with Hard Constraints. In Chapter 2, we design a novel policy network architecture called Vertex Network, which is motivated by the problem of training an reinforcement learning algorithm with hard state and action constraints. Leveraging the geometric property that a convex polytope can be equivalently represented as the convex hull of a finite set of vertices, the output of Vertex Network satisfies the safety constraints by design.

Figure 6.2 illustrates the idea of our proposed Vertex Networks, as well as the widely used projection-based methods that we discussed in Chapter 2. The projection-based methods leverage an extra optimization step to maintain safety, which decouples the policy into two steps and cost extra computations (blue arrows in the figure). On the other hand, our method is an attempt to integrate the safety constraints into neural networks architecture by designing an end-to-end policy that maps states to safe action subset directly (orange arrow in the figure).

We should point out that we are not claiming that our proposed Vertex Networks approach is a complete replacement for the projection-based methods. In fact, it provides another option in the safe reinforcement learning method toolbox so that we can make tradeoff based on specific tasks. There are indeed limitations in our current proposed method:

1. Our method only works for the affine-control system with polytope constraints in both state and action spaces, and it is hard to compute vertices of polytope intersection in high dimensional space.

2. Our method only considers one step safety ahead, which might lead to states where there is no safe action to execute.

Recently, some approaches are proposed in the literature, which could resolve one of the limitations. We now briefly introduce those methods.

For high dimension control tasks, especially when the state safety and actuator constraints are not symmetrical, it could be hard to find hand-designed rules to compute the vertices of their intersection polytope. If such hand-designed rules are not accessible, solving a polytope intersection problem online could be as hard or even more complicated than solving a projection quadratic programming. One idea to deal with high dimension control tasks or none polytope constraints is to still leverage an extra projection step but integrate that into the end-to-end policy neural network training.

[107, 108] proposed the idea to integrate an optimization step as a differentiable layer in a neural network. The main contribution of the papers is the methodology to compute the gradient of an optimization problem so that it could be plugged in the backpropagation. They then extend the idea into applications such as MPC solver [109], robust policy network training [110], and stable dynamic model learning [111].

However, even by leveraging the techniques in these papers to integrate the projection step into a neural network layer, the feed-forward step of the policy network still requires solving an optimization problem online. As stated in Chapter 2, if real-time optimization is allowed by the application, then it is often more advantageous to solve a MPC problem than to ask for a policy learned by reinforcement learning.

The other drawback of our work is that the safety is only "one step ahead". That is, we only require the action to result in a safe state in the next step, which may generate a trajectory that goes to an unsafe region essentially. In the last step of that trajectory, no

feasible action is safe and the intersection of two polytopes is an empty set. Computing the control invariant set is one solution but it is computationally expensive and will lose our advantage of using Vertex Networks. In this work, we rely on the negative reward received during reinforcement learning training to drive the system away from the undesired regions. Thus, our approach can be viewed as a combination of the penalty method [26] and Vertex Networks.

One promising approach to maintain the safety of the whole future trajectory is through control barrier functions, which gain a lot of attention recently [112, 113, 33, 34, 114]. The idea is to design a barrier function, which provides a specification of action such that once the it is satisfied, there is always a feasible trajectory that stays in the safe region. The shortcoming of this approach is that the design of the control barrier function relies heavily on the heuristic about the dynamic system of each specific task.

Currently there is no best solution to the reinforcement learning with hard constraints problem as the algorithm varies from each application. The general method for this problem is still an ongoing research direction.

Safe Reinforcement Learning with Cost Feedback. In Chapter 3 we formulate the problem of safe reinforcement learning when the transition kernel is known but the reward and constraint costs are unknown a priori as a CMDP and propose a `C-UCRL` algorithm to learn the optimal policy. Theoretically, we show that `C-UCRL` is guaranteed to satisfy the constraints during learning with probability at least $1-\delta$ and achieves $O(T^{\frac{3}{4}}\sqrt{\log(T/\delta)})$ reward regret.

This work appears to be the first to study a provably efficient learning algorithm on CMDPs [115]. After this work, there are several other works about this topic including regret analysis [72, 116, 117] and sample complexity analysis [118, 119]. The algorithms proposed in [72, 116, 117] consider the setting that the transition kernel is no longer assumed to be known and achieve $O(\sqrt{T})$ regret on both reward and constraint violation under slightly different settings: [72] consider stochastic reward and costs; [117] consider adversarially

chosen reward and costs; [116] consider linear CMDP setting.

An interesting fact is that, due to the transition kernel being unknown, it seems impossible to guarantee constraint satisfaction during learning (zero regret on constraint violation). We provide a formal proof of that by showing a lower bound on constraint violation in CMDP with unknown transition kernel in Section 3.7. Together with the results in [72, 116, 117], we can draw the conclusions that, with full knowledge about the transition kernel of a CMDP, `C-UCRL` can maintain constraint satisfaction; with no prior information, the constraint violation is inevitable for any algorithm.

One interesting question to ask is, is there any middle ground in between? How much prior information about the transition kernel is sufficient to have zero constraint violation? Recent work in [120] addresses this question by considering the setting where a concise abstract model of the safety aspects is given using the Factored CMDP framework. They propose an algorithm to achieve zero constraint violation using a small subset of features describe the dynamics relevant for the safety constraints. However, they fail to provide reward regret analysis, which could be another interesting future direction.

6.2 Game-Theoretic Reinforcement Learning

In Chapter 4 and 5, present our works on game-theoretic reinforcement learning, where we formulate actor-critic algorithms and two-player competitive MDP as a Stackelberg game. We adopt the game-theoretic viewpoint in order to account for the hierarchical order of play between learning agents. In those Stackelberg methods, the leader player follows the total derivative of its objective instead of the usual individual gradient. Our methods improve the convergence property in actor-critic algorithms and demonstrate new behavior patterns in competitive two-agents setting on our benchmark experiments.

Our works are among the first to take a game-theoretic view of reinforcement learning. Concurrently in the literature, game-theoretic reinforcement learning algorithms have been proposed to tackle the model-based reinforcement learning [80], actor-critic [81], and multi-agent reinforcement learning [76, 77, 78]. However, our works are the only ones that formulate

the problems as Stackelberg games and explicitly leverage the second-order Stackelberg total derivative for the leader update rule. There are several potential future directions to extend our work and we discuss them here.

In the Stackelberg game framework, the leader needs to estimate the follower's best response implicit mapping by computing the inverse of the follower's Hessian matrix. In our algorithm, the inverse of Hessian is computed by the conjugate gradient method, without explicitly computing the Hessian matrix itself. Similar to any stochastic gradient-based method, the inverse Hessian matrix is estimated by samples. However, unlike first-order methods, we have very limited heuristics on the sample complexity of the Hessian matrix. How many samples are enough to give an accurate Hessian estimation is an interesting open question. On the other hand, exploring methods other than conjugate gradient is another future direction. For instance, the method introduced in [121] is claimed to be more effective.

In Section 4.4.5, we introduce the implicit map regularization in the Stackelberg gradient, and we use this regularization in both Stackelberg actor-critic algorithm in Chapter 4 and Stackelberg MADDPG algorithm in Chapter 5. However, the hyperparameter tuning of this regularization is currently based on a heuristic. One future direction is to build theoretical foundations for selecting the regularization or designing a reducing sequence mechanism of such hyperparameter.

Another future direction is to study the game structure where both players leverage the leader's total derivative gradient update. It would be interesting to observe more behavior patterns especially in the two-player competitive game setting and compare the learning dynamics and the policies learned from the leader-follower game and leader-leader game.

BIBLIOGRAPHY

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[3] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning.," in *International Conference on Learning Representation (ICLR)*, 2016.

[4] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[5] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani, "Deep reinforcement learning framework for autonomous driving," *Electronic Imaging*, vol. 2017, no. 19, pp. 70–76, 2017.

[6] D. Silver, S. Singh, D. Precup, and R. S. Sutton, "Reward is enough," *Artificial Intelligence*, p. 103535, 2021.

[7] M. I. Jordan, "Artificial intelligence—the revolution hasn't happened yet," *Harvard Data Science Review*, vol. 1, no. 1, 2019.

[8] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.

[9] S. M. Kakade, *On the sample complexity of reinforcement learning.* University of London, University College London (United Kingdom), 2003.

[10] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete problems in ai safety," *arXiv preprint arXiv:1606.06565*, 2016.

[11] P. Hernandez-Leal, M. Kaisers, T. Baarslag, and E. M. de Cote, "A survey of learning in multiagent environments: Dealing with non-stationarity," *Autonomous Agents and Multi-Agent Systems*, 2017.

[12] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, pp. 2672–2680, 2014.

[13] K. Zhang, Z. Yang, and T. Başar, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," *arXiv preprint arXiv:1911.10635*, 2019.

[14] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in neural information processing systems*, pp. 1008–1014, 2000.

[15] T. Fiez, B. Chasnov, and L. J. Ratliff, "Implicit learning dynamics in stackelberg games: Equilibria characterization, convergence analysis, and empirical study," in *International Conference on Machine Learning*, 2020.

[16] C. Jin, P. Netrapalli, and M. I. Jordan, "What is local optimality in nonconvex-nonconcave minimax optimization?," in *International Conference on Machine Learning*, 2020.

[17] Y. Wang, G. Zhang, and J. Ba, "On solving minimax optimization locally: A follow-the-ridge approach," in *International Conference on Learning Representations*, 2019.

[18] T. Fiez and L. Ratliff, "Gradient descent-ascent provably converges to strict local minmax equilibria with a finite timescale separation," *arXiv preprint arXiv:2009.14820*, 2020.

[19] L. Zheng, Y. Shi, L. J. Ratliff, and B. Zhang, "Safe reinforcement learning of control-affine systems with vertex networks," in *Learning for Dynamics and Control*, pp. 336–347, PMLR, 2021.

[20] L. Zheng and L. Ratliff, "Constrained upper confidence reinforcement learning," in *Proceedings of the 2nd Conference on Learning for Dynamics and Control*, vol. 120 of *Proceedings of Machine Learning Research*, pp. 620–629, PMLR, 2020.

[21] L. Zheng and L. J. Ratliff, "Constrained upper confidence reinforcement learning," *arXiv preprint arXiv:2001.09377*, 2020.

[22] L. Zheng, T. Fiez, Z. Alumbaugh, B. Chasnov, and L. J. Ratliff, "Stackelberg actor-critic: A game-theoretic perspective," *AAAI Reinforcement Learning in Games Workshop*, 2020.

[23] L. Zheng, T. Fiez, Z. Alumbaugh, B. Chasnov, and L. J. Ratliff, "Stackelberg actor-critic: Game-theoretic reinforcement learning algorithms," 2021.

[24] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *International Conference on Machine Learning*, pp. 1329–1338, 2016.

[25] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, pp. 1889–1897, 2015.

[26] 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.

[27] K. P. Wabersich and M. N. Zeilinger, "Linear model predictive safety certification for learning-based control," in *2018 IEEE Conference on Decision and Control (CDC)*, pp. 7130–7135, IEEE, 2018.

[28] G. Dalal, K. Dvijotham, M. Vecerik, T. Hester, C. Paduraru, and Y. Tassa, "Safe exploration in continuous action spaces," *arXiv preprint arXiv:1801.08757*, 2018.

[29] Z. Li, U. Kalabić, and T. Chu, "Safe reinforcement learning: Learning with supervision using a constraint-admissible set," in *2018 Annual American Control Conference (ACC)*, pp. 6390–6395, IEEE, 2018.

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

[31] J. F. Fisac, A. K. Akametalu, M. N. Zeilinger, S. Kaynama, J. Gillula, and C. J. Tomlin, "A general safety framework for learning-based control in uncertain robotic systems," *IEEE Transactions on Automatic Control*, vol. 64, no. 7, pp. 2737–2752, 2018.

[32] K. P. Wabersich and M. N. Zeilinger, "Safe exploration of nonlinear dynamical systems: A predictive safety filter for reinforcement learning," *arXiv preprint arXiv:1812.05506*, 2018.

[33] R. Cheng, G. Orosz, R. M. Murray, and J. W. Burdick, "End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 3387–3395, 2019.

[34] A. Taylor, A. Singletary, Y. Yue, and A. Ames, "Learning for safety-critical control with control barrier functions," in *Learning for Dynamics and Control*, pp. 708–717, PMLR, 2020.

[35] W. Cui and B. Zhang, "Reinforcement learning for optimal frequency control: A lyapunov approach," *arXiv preprint arXiv:2009.05654*, 2020.

[36] W. Cui and B. Zhang, "Lyapunov-regularized reinforcement learning for power system transient stability," *arXiv preprint arXiv:2103.03869*, 2021.

[37] B. Grünbaum, *Convex polytopes*, vol. 221. Springer Science & Business Media, 2013.

[38] F. Blanchini and S. Miani, *Set-theoretic methods in control*. Springer, 2008.

[39] C. Gokcek, P. T. Kabamba, and S. M. Meerkov, "An lqr/lqg theory for systems with saturating actuators," *IEEE Transactions on Automatic Control*, vol. 46, no. 10, pp. 1529–1542, 2001.

[40] E. G. Gilbert and K. T. Tan, "Linear systems with state and control constraints: The theory and application of maximal output admissible sets," *IEEE Transactions on Automatic control*, vol. 36, no. 9, pp. 1008–1020, 1991.

[41] I. Kolmanovsky and E. G. Gilbert, "Theory and computation of disturbance invariant sets for discrete-time linear systems," *Mathematical problems in engineering*, vol. 4, 1998.

[42] F. Blanchini, "Set invariance in control," *Automatica*, vol. 35, no. 11, pp. 1747–1767, 1999.

[43] H. R. Tiwary, "On the hardness of computing intersection, union and minkowski sum of polytopes," *Discrete & Computational Geometry*, vol. 40, no. 3, pp. 469–479, 2008.

[44] V. Broman and M. Shensa, "A compact algorithm for the intersection and approximation of n-dimensional polytopes," *Mathematics and computers in simulation*, vol. 32, no. 5-6, pp. 469–480, 1990.

[45] J. P. Hespanha, *Linear systems theory*. Princeton university press, 2018.

[46] G. Shani, D. Heckerman, and R. I. Brafman, "An mdp-based recommender system," *Journal of Machine Learning Research*, vol. 6, no. Sep, pp. 1265–1295, 2005.

[47] J. Leike, M. Martic, V. Krakovna, P. A. Ortega, T. Everitt, A. Lefrancq, L. Orseau, and S. Legg, "Ai safety gridworlds," *arXiv preprint arXiv:1711.09883*, 2017.

[48] T. M. Moldovan and P. Abbeel, "Safe exploration in markov decision processes," in *Proceedings of the 29th International Coference on International Conference on Machine Learning*, pp. 1451–1458, 2012.

[49] 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.

[50] E. Altman, *Constrained Markov decision processes*, vol. 7. CRC Press, 1999.

[51] S. Bhatnagar and K. Lakshmanan, "An online actor–critic algorithm with function approximation for constrained markov decision processes," *Journal of Optimization Theory and Applications*, vol. 153, no. 3, pp. 688–708, 2012.

[52] Y. Chow, M. Ghavamzadeh, L. Janson, and M. Pavone, "Risk-constrained reinforcement learning with percentile risk criteria," *Journal of Machine Learning Research*, vol. 18, no. 167, pp. 1–51, 2018.

[53] C. Tessler, D. J. Mankowitz, and S. Mannor, "Reward constrained policy optimization," in *International Conference on Learning Representations*, 2018.

[54] M. Yu, Z. Yang, M. Kolar, and Z. Wang, "Convergent policy optimization for safe reinforcement learning," in *Advances in Neural Information Processing Systems*, pp. 3121–3133, 2019.

[55] S. Paternain, L. Chamon, M. Calvo-Fullana, and A. Ribeiro, "Constrained reinforcement learning has zero duality gap," in *Advances in Neural Information Processing Systems*, pp. 7555–7565, 2019.

[56] J. Achiam, D. Held, A. Tamar, and P. Abbeel, "Constrained policy optimization," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 22–31, JMLR. org, 2017.

[57] T.-Y. Yang, J. Rosca, K. Narasimhan, and P. J. Ramadge, "Projection-based constrained policy optimization.," in *ICLR*, 2020.

[58] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause, "Safe model-based reinforcement learning with stability guarantees," in *Advances in Neural Information Processing Systems*, pp. 908–918, 2017.

[59] T. Koller, F. Berkenkamp, M. Turchetta, and A. Krause, "Learning-based model predictive control for safe exploration," in *2018 IEEE Conference on Decision and Control (CDC)*, pp. 6059–6066, IEEE, 2018.

[60] A. Wachi, Y. Sui, Y. Yue, and M. Ono, "Safe exploration and optimization of constrained mdps using gaussian processes," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[61] Y. Chow, O. Nachum, E. Duenez-Guzman, and M. Ghavamzadeh, "A lyapunov-based approach to safe reinforcement learning," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 8103–8112, 2018.

[62] T. Jaksch, R. Ortner, and P. Auer, "Near-optimal regret bounds for reinforcement learning," *Journal of Machine Learning Research*, vol. 11, no. Apr, pp. 1563–1600, 2010.

[63] M. El Chamie, Y. Yu, B. Açıkmeşe, and M. Ono, "Controlled markov processes with safety state constraints," *IEEE Transactions on Automatic Control*, vol. 64, no. 3, pp. 1003–1018, 2019.

[64] W. Ding, T. Qin, X.-D. Zhang, and T.-Y. Liu, "Multi-armed bandit with budget constraint and variable costs," in *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

[65] D. P. Zhou and C. J. Tomlin, "Budget-constrained multi-armed bandits with multiple plays," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[66] M. Joseph, M. Kearns, J. H. Morgenstern, and A. Roth, "Fairness in learning: Classic and contextual bandits," in *Advances in Neural Information Processing Systems*, pp. 325–333, 2016.

[67] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[68] D. G. Luenberger, Y. Ye, *et al.*, *Linear and nonlinear programming*, vol. 2. Springer, 1984.

[69] S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh, and M. Lee, "Natural actor–critic algorithms," *Automatica*, vol. 45, no. 11, pp. 2471–2482, 2009.

[70] K.-M. Chung, H. Lam, Z. Liu, and M. Mitzenmacher, "Chernoff-hoeffding bounds for markov chains: Generalized and simplified," in *STACS'12 (29th Symposium on Theoretical Aspects of Computer Science)*, vol. 14, pp. 124–135, LIPIcs, 2012.

[71] I. Osband and B. Van Roy, "On lower bounds for regret in reinforcement learning," *arXiv preprint arXiv:1608.02732*, 2016.

[72] Y. Efroni, S. Mannor, and M. Pirotta, "Exploration-exploitation in constrained mdps," *arXiv preprint arXiv:2003.02189*, 2020.

[73] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[74] T. Başar and G. J. Olsder, *Dynamic noncooperative game theory*. SIAM, 1998.

[75] Y. Yang and J. Wang, "An overview of multi-agent reinforcement learning from game theoretical perspective," *Studies in Systems, Decision and Control Handbook on RL and Control*, 2020.

[76] C. Zhang and V. Lesser, "Multi-agent learning with policy prediction," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 24, 2010.

[77] J. Foerster, R. Y. Chen, M. Al-Shedivat, S. Whiteson, P. Abbeel, and I. Mordatch, "Learning with opponent-learning awareness," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '18, p. 122–130, 2018.

[78] A. Letcher, J. Foerster, D. Balduzzi, T. Rocktäschel, and S. Whiteson, "Stable opponent shaping in differentiable games," in *International Conference on Learning Representations*, 2018.

[79] M. Prajapat, K. Azizzadenesheli, A. Liniger, Y. Yue, and A. Anandkumar, "Competitive policy optimization," *arXiv preprint arXiv:2006.10611*, 2020.

[80] A. Rajeswaran, I. Mordatch, and V. Kumar, "A game theoretic framework for model based reinforcement learning," in *International conference on machine learning*, 2020.

[81] M. Hong, H.-T. Wai, Z. Wang, and Z. Yang, "A two-timescale framework for bilevel optimization: Complexity analysis and application to actor-critic," *arXiv preprint arXiv:2007.05170*, 2020.

[82] S. M. Kakade, "A natural policy gradient," *Advances in neural information processing systems*, vol. 14, pp. 1531–1538, 2001.

[83] J. Peters and S. Schaal, "Natural actor-critic," *Neurocomputing*, vol. 71, no. 7-9, pp. 1180–1190, 2008.

[84] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[85] Z. Shen, A. Ribeiro, H. Hassani, H. Qian, and C. Mi, "Hessian aided policy gradient," in *International Conference on Machine Learning*, pp. 5729–5738, 2019.

[86] V. Tangkaratt, A. Abdolmaleki, and M. Sugiyama, "Guide actor-critic for continuous control," in *International Conference on Learning Representations*, 2018.

[87] L. J. Ratliff, S. A. Burden, and S. S. Sastry, "Genericity and structural stability of non-degenerate differential nash equilibria," in *American Control Conference*, pp. 3990–3995, IEEE, 2014.

[88] E.-V. Vlatakis-Gkaragkounis, L. Flokas, and G. Piliouras, "Poincaré recurrence, cycles and spurious equilibria in gradient-descent-ascent for non-convex non-concave zero-sum games," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019*, 2019.

[89] L. Flokas, E.-V. Vlatakis-Gkaragkounis, and G. Piliouras, "Solving min-max optimization with hidden structure via gradient descent ascent," *arXiv preprint arXiv:2101.05248*, 2021.

[90] S. C. Chan, S. Fishman, A. Korattikara, J. Canny, and S. Guadarrama, "Measuring the reliability of reinforcement learning algorithms," in *International Conference on Learning Representations*, 2019.

[91] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[92] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, pp. 1057–1063, 2000.

[93] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1291–1307, 2012.

[94] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, pp. 1928–1937, 2016.

[95] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proceedings of the 35th International Conference on Machine Learning* (J. Dy and A. Krause, eds.), vol. 80 of *Proceedings of Machine Learning Research*, (Stockholmsmässan, Stockholm Sweden), pp. 1861–1870, PMLR, 10–15 Jul 2018.

[96] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, 2016.

[97] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[98] G. Ramponi and M. Restelli, "Newton optimization on helmholtz decomposition for continuous games," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 11325–11333, 2021.

[99] V. S. Borkar, *Stochastic approximation: a dynamical systems viewpoint*, vol. 48. Springer, 2009.

[100] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[101] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, IEEE, 2012.

[102] P. Stone and M. Veloso, "Multiagent systems: A survey from a machine learning perspective," *Autonomous Robots*, vol. 8, no. 3, pp. 345–383, 2000.

[103] L. Busoniu, R. Babuska, and B. De Schutter, "A comprehensive survey of multiagent reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 38, no. 2, pp. 156–172, 2008.

[104] J. Filar and K. Vrieze, *Competitive Markov decision processes*. Springer Science & Business Media, 2012.

[105] E. Mazumdar, L. J. Ratliff, and S. S. Sastry, "On gradient-based learning in continuous games," *SIAM Journal on Mathematics of Data Science*, vol. 2, no. 1, pp. 103–131, 2020.

[106] R. Lowe, Y. WU, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," *Advances in Neural Information Processing Systems*, vol. 30, pp. 6379–6390, 2017.

[107] B. Amos and J. Z. Kolter, "Optnet: Differentiable optimization as a layer in neural networks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 136–145, JMLR. org, 2017.

[108] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter, "Differentiable convex optimization layers," *Advances in Neural Information Processing Systems*, vol. 32, pp. 9562–9574, 2019.

[109] B. Amos, I. D. J. Rodriguez, J. Sacks, B. Boots, and J. Z. Kolter, "Differentiable mpc for end-to-end planning and control," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 8299–8310, 2018.

[110] P. L. Donti, M. Roderick, M. Fazlyab, and J. Z. Kolter, "Enforcing robust control guarantees within neural network policies," in *International Conference on Learning Representations*, 2020.

[111] J. Z. Kolter and G. Manek, "Learning stable deep dynamics models," *Advances in Neural Information Processing Systems*, vol. 32, pp. 11128–11136, 2019.

[112] 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, 2016.

[113] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, "Control barrier functions: Theory and applications," in *2019 18th European Control Conference (ECC)*, pp. 3420–3431, IEEE, 2019.

[114] A. J. Taylor, V. D. Dorobantu, S. Dean, B. Recht, Y. Yue, and A. D. Ames, "Towards robust data-driven control synthesis for nonlinear systems with actuation uncertainty," *arXiv preprint arXiv:2011.10730*, 2020.

[115] C. Szepesvári, "Constrained mdps and the reward hypothesis," *https://readingsml.blogspot.com/2020/03/constrained-mdps-and-reward-hypothesis.html*, 2020.

[116] D. Ding, X. Wei, Z. Yang, Z. Wang, and M. R. Jovanović, "Provably efficient safe exploration via primal-dual policy optimization," *arXiv preprint arXiv:2003.00534*, 2020.

[117] S. Qiu, X. Wei, Z. Yang, J. Ye, and Z. Wang, "Upper confidence primal-dual optimization: Stochastically constrained markov decision processes with adversarial losses and unknown transitions," *arXiv preprint arXiv:2003.00660*, 2020.

[118] K. C. Kalagarla, R. Jain, and P. Nuzzo, "A sample-efficient algorithm for episodic finite-horizon mdp with constraints," *arXiv preprint arXiv:2009.11348*, 2020.

[119] A. HasanzadeZonuzy, D. Kalathil, and S. Shakkottai, "Learning with safety constraints: Sample complexity of reinforcement learning for constrained mdps," *arXiv preprint arXiv:2008.00311*, 2020.

[120] T. D. Simão, N. Jansen, and M. T. Spaan, "Alwayssafe: Reinforcement learning without safety constraint violations during training," in *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 1226–1235, 2021.

[121] O. Vinyals and D. Povey, "Krylov subspace descent for deep learning," in *Artificial Intelligence and Statistics*, pp. 1261–1268, PMLR, 2012.