UNIVERSITY OF CALIFORNIA,
IRVINE

Computing Nash Equilibria in Adversarial Stochastic Team Games

THESIS

submitted in partial satisfaction of the requirements
for the degree of

MASTER OF SCIENCE

in Electrical and Computer Engineering

by

Stelios Andrew Stavroulakis

Thesis Committee:
Professor Athina Markopoulou, Chair
Assistant Professor Ioannis Panageas
Assistant Professor Yanning Shen

2022

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ALGORITHMS

ACKNOWLEDGMENTS

This thesis has been supported by NSF Awards 1939237, 1956393, 1900654. I would like to express my deepest appreciation to Professor Athina Markopoulou, Professor Ioannis Panageas and Professor Yanning Shen for supervising my thesis. It was also a pleasure working with Salma Elmalaki on Reinforcement Learning applications during this period. I would also like to extend my deepest gratitude to Fivos Kalogiannis, Ioannis Anagnostides, Manolis Vlatakis and Vaggos Chatziafratis for the insightful suggestions and fruitful conversations. Special thanks to Bill, Evi, and Fozzie for the unparalleled support when I most needed it. Particularly helpful to me was George and I am extremely thankful for having him next to me. I'm also extremely appreciative of all the love and encouragement my parents and my brothers have given me. Finally, words can not express how grateful I am to Renata for her unwavering support throughout this journey.

ABSTRACT OF THE THESIS

Computing Nash Equilibria in Adversarial Stochastic Team Games

By

Stelios Andrew Stavroulakis

Master of Science in Electrical and Computer Engineering

University of California, Irvine, 2022

Professor Athina Markopoulou, Chair

Computing Nash equilibrium policies in Multi-agent Reinforcement Learning (MARL) is a fundamental question that has received a lot of attention both in theory and in practice. However, beyond the single-agent setting, provable guarantees for computing stationary Nash equilibrium policies only exist for limited settings, e.g., two-player zero-sum stochastic games and Markov Potential Games. This thesis investigates what happens when a team of players faces an adversary in the absence of coordination. It is an initial step for understanding non-cooperative two-team zero-sum stochastic games, even in this case, where one of the teams is composed of only one agent. In this thesis we consider the aforementioned setting. The contributions are twofold. We prove the existence of Nash equilibria in these settings and also design a *decentralized* algorithm for computing Nash equilibria in such games. One of the main technical challenges in the proof of correctness of the algorithm is the analysis of a *non-linear* program with *non-convex constraints* that arises when applying the Bellman equations; the analysis makes use of the so-called *constraint qualifications* and *Lagrange duality*.

Chapter 1

Introduction

At its core, the appeal of artificial intelligence (AI) relies on the potential to solve advanced real-world problems. Machine learning methods have enabled data-based decision-making across many industries, including manufacturing, education, healthcare, and marketing. Nevertheless, there is still a myriad of unsolved problems that require progressively more autonomous and adaptive decision-making to be solved.

This field has seen great success by leveraging Reinforcement Learning (RL), a subset of machine learning, to achieve excellent empirical performance in many tasks [51], [52], [10]. The foundational idea behind RL is to enable learning by interactions with an environment – by trial and error as follows: the learner (agent) uses feedback from its actions and experiences to find, autonomously, solutions that maximize a numerical reward signal. In this paradigm, instead of training examples that indicate the correct output for a given input, which can be very inefficient or impossible for some problems, success is reached by enabling agents to explore their environment while exploiting actions with immediate reward. Balancing these exploration/exploitation trade-offs is an important point of RL.

Reinforcement learning (RL) has recently demonstrated impressive results in a multitude of domains ranging from biology [31], robotics [34], and video games [41], [44]. Many of the aforementioned RL success stories relied on the rapid concurrent theoretical progress in the field, check [27], [3], [35], [38], and references therein. However, many problems involve multiple agents interacting with each other in the same, shared, environment and are game-theoretic in nature. The general learning framework used for these particular settings is the generalization of the single-agent reinforcement learning framework [11], known also as Stochastic Games, first introduced by Shapley [50]. A comprehensive understanding of the multi-agent setting remains elusive.

In a stochastic game, multiple agents co-exist in a discrete-time dynamical system. All players take actions and at every step the environment trantitions to a new state, based on the actions of all players. The transition also produces rewards, different for each player, that are distributed to all players at every state transition. The goal inevitably shifts from learning optimal policies to equilibrium policies, as simultaneous optimization of multiple functions is not attainable. Various setups within traffic control, robotics, economics, warehouse scheduling, drone missions, auctions are captured by the aforementioned setting. Stochastic games have been studied in a plethora of domains such as game-theory [43], [54], machine learning [37], [11] and numerous other fields due to their broad applicability.

Current State-of-the-art

The fascinating problem that multi-agent RL (MARL) tackles is the ever-changing nature of these non-stationary environments since all agents are constantly interacting in a coupled environment. For that reason provable guarantees for MARL have been rather sparse.

Centralized VS Decentralized Settings In an effort to organize MARL algorithms, we can distinguish two large categories. Centralized/coordinated algorithms and independent/decoupled algorithms [69]. The former includes a centralized controller whose objective function is defined with respect to the joint policies of all agents. Examples of centralized/coordinated algorithms include self-play [32], [23] and are mostly used in setups where the number of player and interaction type (competition/cooperation) is known beforehand. The impressive practical results of centralized algorithms [44], [51], [59] have been recently backed by solid theoretical results, such as finite-sample guarantees [64], [6], [67].

Decentralized settings do not contain a centralized controller. All agents act independently and myopically with respect to their respective loss function and optimization objective. Independent/decoupled algorithms capture settings where agent have their own observations and have their own actions and rewards. The key distinction with the former is that independent/decoupled algorithms can be deployed in a decentralized fashion in environments where the number of agents is not defined a-priori. This setup is much more flexible but at the same time much more challenging since coordination should be learned from interaction instead of communication. Providing similar guarantees for decentralized settings seems to be particularly tricky as independent algorithms might fail altogether, even in the simplest of multi-agent settings [17], [16]. The reason for this is that independently updating policies introduce distribution shift that break the stationarity assumptions enjoyed by most of the single-agent algorithms. This phenomenon is considered one of the grand challenges of MARL [40], [24].

Potential Games VS Zero-Sum Games Provable guarantees for efficiently computing Nash equilibria have been thus far limited to either fully competitive settings, such as two-player zero-sum games [18],[65], [48], [12], [49], or environments in which agents are endeavoring to coordinate towards a common global objective [33], [21], [70]. The two setups

with multiple agents have orthogonal shortcomings. One has the number of players fixed to 2 while allowing competition between the players, the other has an arbitrary number of players but requires cooperation. Figure 1.1 pictorially illustrates the two different settings.

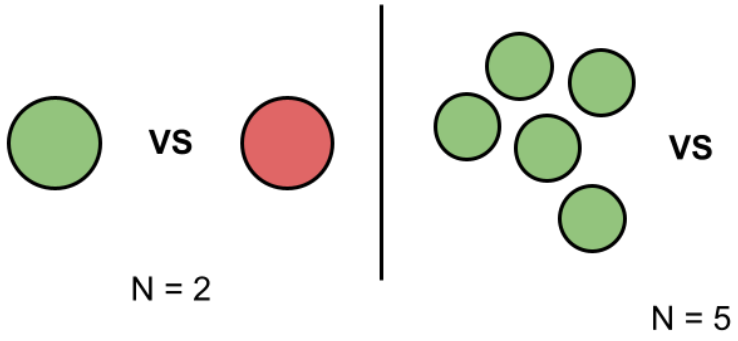

Figure 1.1: On the left we see the two-player, zero-sum, competitive setting [18]. On the right the N-player, fully cooperative setting [33].

Many real-world strategic interactions feature both shared and competing interests between the agents. Efficient algorithms for computing Nash equilibria in such settings are much more scarce. In fact, even in stateless two-player (normal-form) games, computing Nash equilibria is computationally intractable [19], [47], [13], [22] —subject to well-believed complexity-theoretic assumptions. As a result, it is common to investigate equilibrium concepts that are more permissive than Nash equilibria, such as coarse correlated equilibria (CCE) [5], [42], [28]. Unfortunately, recent work has established strong lower bounds even for computing approximate (stationary) CCEs in turn-based stochastic two-player games [20], [29].

Scope of research

The recent negative results discussed above raise the following central question:

Are there natural multi-agent environments incorporating both competing and shared interests for which we can establish efficient algorithms for computing (stationary) Nash equilibria?

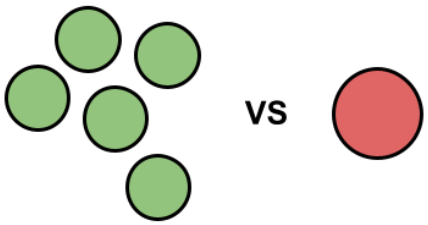

Figure 1.2: The setting considered. A team of N agents facing an adversary. This setting includes cooperation and competition.

The work presented in the following thesis makes concrete progress in this fundamental direction. Specifically, we establish an efficient decentralized algorithm leading to Nash equilibria in adversarial team stochastic games, an important setting in which a team of agents with a common objective faces a competing adversary. This result pushes the boundary of what was considered possible and unifies prior results that only applied to either two-player zero-sum games or Markov potential games. The proposed algorithm has a complexity that is only linearly dependent on the number of players, breaking the "curse of multi-agents", an inherent problem in most multi-agent algorithms where the running time usually scales exponentially with the number of players in the environment. Moreover, the proposed algorithm is fully decentralized and requires no communication between the players. This thesis provides the proof of the algorithm and also experimentally verifies the theoretical findings.

Thesis Outline

Chapter 1 introduces the topic and provides a high-level description of the work.

Chapter 2 includes an overview of the single-agent reinforcement learning literature and all the basic building blocks needed for the main result.

Chapter 3 extends the analysis to the multi-agent case and presents some fundamental challenges in this domain.

Chapter 4 presents the results and the main Theorem.

Chapter 5 presents the experimental results. The multi-agent reinforcement learning environment explicitly created for these tasks is publicly accessible and can be found on GitHub.

Chapter 6 summarizes the findings and proposes future directions.

Chapter 2

Single-Agent Reinforcement Learning

2.1 Markov Decision Processes (MDP)

Real-world problems are often sequential. Moving from a state to another requires taking an action, and taking action now affects the trajectory of events and experiences into the future. The Markov decision process formalism captures these two aspects and provides a general framework for sequential decision-making [57]. One way of formalizing the decision-making process is through a discrete-time dynamical system. This system is reactive to the actions taken by the agent, as shown in figure 2.1.

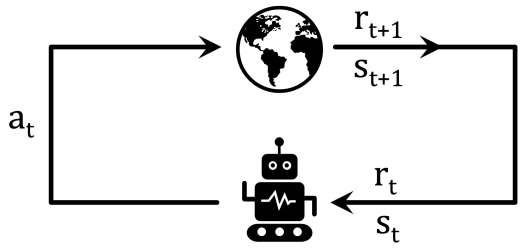

Figure 2.1: A discrete-time dynamical system describing the interaction of the agent with the environment.

The transition dynamics interaction function $p(s' \mid s, a)$. Given a state s in the set of all possible states $\mathcal{S}$ and an action a in the set of all possible actions $\mathcal{A}$, the function p returns the probability of transitioning to state s'.

$$p = \sum_{s' \in \mathcal{S}} p(s' \mid s, a) = 1, \ \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$$

Allowing for probability distributions in each state-action pair enables stochastic transitions between state-action pairs and new states. Otherwise, transitions would be deterministic. The reward at every time-step is a function $R(s', s, a,)$ that assigns the transition from the state-action tuple (s, a) to the state s' with a reward signal r experienced by the agent. Formally,

$$s'_{t+1} \sim p(\cdot \mid s_t, a_t)$$

$$r_{t+1} = R(s_t, a_t, s_{t+1})$$

This discrete-time transition from one state to the next is captured by the subscripts of the states, rewards and actions and can also be pictorially shown in the figure below:

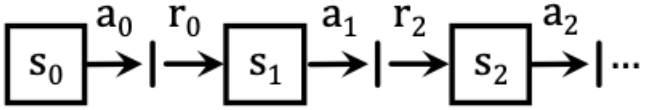

Figure 2.2: A sequence of states, actions, and rewards, defining a trajectory in the MDP.

Formally, an MDP captures the sequential interaction of the agent and the environment [1] in the form of a tuple $M = (S, A, P, r, \gamma, \mu)$ that contains the following:

- A finite action space $\mathcal{A}$.
- A state space $\mathcal{S}$ that is assumed to be finite or countably infinite.
- A transition function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ translating to a distribution of all states $\mathcal{S}$ to all other states in the state space.

- A reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to [0, 1]$. Rewards can be assumed to be upper-bounded by 1 without loss of generality.
- A discount factor γ defining the horizon of the problem.
- An initial distribution $\mu \in \Delta(\mathcal{S})$ that specifies the initial state distribution and how the initial state s_0 is generated

2.2 Objective Function

2.2.1 Finite Horizon

Given the transition probability matrix and the reward received for each transition, there are various ways to behave in such an environment. One arbitrary strategy would be to maximize the immediate reward meaning to act greedily with respect to the reward received at time-step t. This, however, does not account for actions that lead to negative results in the long run, even if they seem promising short-term. One way to get around this is to consider the incremental aggregated sum of rewards collected throughout the trajectory. We define the return at time-step t as:

$$G_t := R_{t+1} + R_{t+2} + ... \tag{2.1}$$

The return G is a random variable since the transitions are stochastic in nature and might not always yield the same return. For this reason, instead of the return G, the goal of an agent in an MDP that accounts for future rewards as well as the stochasticity of the environment should consider the expected sum of rewards.

$$\mathbb{E}[G_t] := \mathbb{E}[R_{t+1} + R_{t+2} + ...] \tag{2.2}$$

The expected sum should be finite for the expectation to be well defined. Since every reward $R_t \in \mathbb{R}$ is finite, the sum of a finite series is also finite. Since the sequence ends, this class of events is called episodic. Each sequence defines one episode.

2.2.2 Infinite Horizon

In the infinite horizon case, the expectation in equation 2.2 might become infinite, as there is no terminal state. A discount factor γ is introduced to bound the return G even if the terms are infinite.

$$\begin{aligned} G_t &:= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \\ &:= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \end{aligned} \tag{2.3}$$

which is bounded as $G_t \leq \sum_{k=0}^{\infty} \gamma^k R_{max} = R_{max} \cdot \sum_{k=0}^{\infty} \gamma^k = R_{max} \cdot \frac{1}{1-\gamma} \in \mathbb{R}$ since $0 \leq \gamma < 1$. The discount factor intuitively works as a toggle controlling how "myopic" an agent is with respect to the rewards it receives over time. Incorporating the discount factor into the return function G allows for the return to be written recursively, as shown in equation 2.4, something that can be exploited by learning algorithms to provide convergence guarantees.

$$G_t := R_{t+1} + \gamma G_{t+1} \tag{2.4}$$

Maximizing the expected sum of a cumulative scalar reward is enough to formulate a goal in a sequential decision-making environment, according to the reward hypothesis [53].

2.3 Policies

The reinforcement learning problem is a sequential decision-making problem under uncertainty. The agent's role is to make decisions in each state of the sequence to optimize some long-term reward signal. The policy π of the agent is a mapping of states to actions for each state $s \in \mathcal{S}$ and $a \in \mathcal{A}$. Each state of an MDP has, by definition, all the necessary information required for decision making. We call this property the Markovian property [39]. The agent's policy should only consider the current state of decision-making and recommend an action either deterministically or stochastically.

2.3.1 Stochastic Policies

In each state, the policy recommends an action. A stochastic policy is a distribution of all available actions assigned to each state s. The agent selects an action by sampling from the probability distribution. This is done mainly to account for stochasticity in the environment. Since the policy $\pi(a \mid s)$ is a probability distribution, it follows the properties of a probability distribution, shown below.

$$\begin{aligned} &\sum_{a \in \mathcal{A}} \pi(a \mid s) = 1 \\ &\pi(a \mid s) \geq 0, \forall a \in \mathcal{A}, s \in \mathcal{S} \end{aligned} \tag{2.5}$$

2.3.2 Deterministic Policies

Deterministic policies are special cases of stochastic policies where all the mass of the probability density function is concentrated on one action, and the distribution collapses to an action selection with probability 1. In this case, $\pi(s) = a$ is the action taken by the policy π in state s.

2.4 Value Function

Value functions are integral components of the reinforcement learning paradigm. They allow the agent to query the quality of their state without having to experience a trajectory under their current policy. Referencing the value of a state allows an agent to make decisions even if rewards are not currently available and even if transitions are stochastic. Value functions work as an estimated performance measure of the policy π. Values can be assigned to states or actions taken in each state.

2.4.1 Action-value Function

An action-value function is by definition the expected return obtained in state s if action a is taken, and policy π is followed thereafter.

$$Q_\pi(s, a) := \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right] \tag{2.6}$$

Equation 2.6 returns the value of choosing action a in state s and then behaving according the the policy π.

2.4.2 State-value Function

Given a state s, the state-value function of that state is by definition the expected return of the agent if policy π is followed thereafter.

$$V_\pi(s) := \mathbb{E}_\pi \left[G_t \mid S_t = s \right] \tag{2.7}$$

Intuitively, the state-value function aggregates all action-value functions into a single value for each state $s \in \mathcal{S}$. The formal connection between the action-state function and the state-value function is presented below:

$$\begin{aligned} V_\pi(s) &= \sum_{a \in \mathcal{A}} \pi(a \mid s) Q_\pi(s, a) \\ Q_\pi(s, a) &= \sum_{s' \in \mathcal{S}} p(s' \mid s, a) \left[R(s, a, s') + \gamma V_\pi(s') \right] \end{aligned} \tag{2.8}$$

2.5 Bellman Equations

The Bellman equations enable expressing the value at each state s is a function of the value of every possible successive state s_{t+1}. This is the key that reduces the consideration of an infinite number of possible trajectories to a set of linear equations with dimensions at most the number of states. The solution of this linear system provides the state values directly.

2.5.1 State-value Equations

The Bellman equation of the state-value equation can be derived in the following manner:

$$\begin{aligned} V_\pi(s) &:= \mathbb{E}_\pi \left[G_t \mid S_t = s \right] \\ V_\pi(s) &= \mathbb{E}_\pi \left[R_{t+1} + \gamma G_{t+1} \mid S_t = s \right] \\ V_\pi(s) &= \sum_{a \in \mathcal{A}} \pi(a \mid s) \sum_{s \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a) \Big[r + \gamma \mathbb{E}_\pi \left[G_{t+1} \mid S_{t+1} = s' \right] \Big] \\ V_\pi(s) &= \sum_{a \in \mathcal{A}} \pi(a \mid s) \sum_{s \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a) \Big[r + \gamma V_\pi(s') \Big] \end{aligned} \tag{2.9}$$

2.5.2 Action-value Equations

Similarly, the action-value equations can be derived from the definition of the action-value function as follows:

$$
\begin{aligned}
Q_\pi(s,a) &:= \mathbb{E}_\pi\left[G_t \mid S_t = s, A_t = a\right] \\
Q_\pi(s,a) &= \sum_{s' \in \mathcal{S}} \sum_{r \in R} p(s', r \mid s, a) \Big[r + \gamma \mathbb{E}_\pi\left[G_{t+1} \mid S_{t+1} = s'\right] \Big] \\
Q_\pi(s,a) &= \sum_{s' \in \mathcal{S}} \sum_{r \in R} p(s', r \mid s, a) \Big[r + \gamma \sum_{a' \in \mathcal{A}} \pi(a' \mid s') \mathbb{E}_\pi\left[G_{t+1} \mid S_{t+1} = s', A_{t+1} = a'\right] \Big]
\end{aligned}
$$

Here, we express the state-action value $Q_\pi(s,a)$ with respect to the next state-action value $Q_\pi(s',a')$. The final equation becomes:

$$
Q_\pi(s,a) = \sum_{s' \in \mathcal{S}} \sum_{r \in R} p(s', r \mid s, a) \Big[r + \gamma \sum_{a' \in \mathcal{A}} \pi(a' \mid s') Q_\pi(s', a') \Big] \tag{2.10}
$$

Equations of the state-value 2.9 and action-value 2.10 functions are now written recursively. They can be used to efficiently estimate value functions by constructing and solving a system of equations with the state-values or state-action values as unknowns.

2.6 Optimal Policies/Value Functions

To define an optimal policy, it makes sense to define an ordering of policies based on their value in all states. Since policies might have different values in different states, we can only claim that $\pi_1 \succ \pi_2 \iff V_{\pi_1}(s) > V_{\pi_2}(s), \forall s \in \mathcal{S}$.

The best policy $\pi^\star$ in an MDP achieves the best value in every state. The best value is

defined as:

$$V^\star = V_{\pi^\star}(s) = \max_\pi V_\pi(s) := \mathbb{E}_{\pi^\star}\Big[G_t \mid S_t = s\Big], \forall s \in \mathcal{S} \tag{2.11}$$

The max operator considers all possible policies for a given state and outputs a value $V^\star$ which is the same for potentially multiple optimal policies that achieve the same value. Every optimal policy also mapped to an action-value function called $Q^\star$ and is defined as:

$$Q^\star = Q_{\pi^\star}(s,a) = \max_\pi Q_\pi(s,a) := \mathbb{E}_{\pi^\star}\Big[G_t \mid S_t = s, A_t = a\Big], \forall s \in \mathcal{S}, a \in \mathcal{A} \tag{2.12}$$

2.6.1 Bellman Optimality Equations

The Bellman equations 2.9 and 2.10 hold for every policy. If we substitute the optimal policy $\pi^\star$, equation 2.9 becomes:

$$V_{\pi^\star}(s) = \sum_{\boldsymbol{a \in \mathcal{A}}} \boldsymbol{\pi^\star(a \mid s)} \sum_{s \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a)\Big[r + \gamma V_\pi(s')\Big]$$

Since an optimal deterministic policy always exists, we can replace the distribution over actions induced by the policy with the action that yields the highest reward.

$$V_{\pi^\star}(s) = \max_a \sum_{s \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a)\Big[r + \gamma V_\pi(s')\Big] \tag{2.13}$$

Similarly, for the action-value function:

$$\begin{aligned} Q_\pi(s,a) &= \sum_{s' \in \mathcal{S}} \sum_{r \in R} p(s', r \mid s, a)\Big[r + \gamma \sum_{\boldsymbol{a' \in \mathcal{A}}} \boldsymbol{\pi(a' \mid s')} Q_\pi(s', a')\Big] \\ Q_\pi(s,a) &= \sum_{s' \in \mathcal{S}} \sum_{r \in R} p(s', r \mid s, a)\Big[r + \gamma \max_a Q_\pi(s', a')\Big] \end{aligned} \tag{2.14}$$

2.6.2 Existence of Optimal Policies

Providing some intuition behind why there always exists a deterministic policy $\pi^\star$ that achieves the best value $V^\star$ is helpful to understand the crux of the Bellman equations.

Definition : The Bellman operator $\mathcal{T}$ acting on an MDP is defined as:

$$\mathcal{T}V(s) := \max_a \Big((1-\gamma)r(s,a) + \gamma \sum_{s'} p(s' \mid s,a)V(s')\Big),\ \forall\ s \in \mathcal{S}, \forall\ V \in \mathbb{R}^{|S|}$$

For a fixed policy π, the Bellman operator acts on the state-value function as follows:

$$\mathcal{T}_\pi V(s) := \sum_{a \in \mathcal{A}} \pi(a \mid s)\Big((1-\gamma)r(s,a) + \gamma \sum_{s'} p(s' \mid s,a)V(s')\Big),\ \forall\ s \in \mathcal{S}, \forall\ V \in \mathbb{R}^{|S|}$$

The Bellman operator maps the value vector $V(s)$ to another vector $V(s')$ and works to effectively do a one-step look-ahead at the following state-value function to determine a slightly better value for the current state s. This operator $\mathcal{T}$ has some essential and convenient properties that are used as building blocks in proving the existence of optimal deterministic and stationary policies.

Contraction Mapping For any pair of value vectors $V(s), V'(s)$, the infinity norm of the difference of the norms evaluated by the Bellman operator is bounded above by the infinity norm of the difference of the values:

$$\|\mathcal{T}V - \mathcal{T}V'\|_\infty \leq (1-\gamma)\|V - V'\|_\infty$$

Monotonicity The Bellman operator preserves the ordering of the values:

$$V > V' \implies \mathcal{T}V > \mathcal{T}V',\ \forall\ V \in \mathbb{R}^{|S|}$$

Given these 2 properties, we can show that the optimal state-value $V^\star(s)$ is the fixed point of the operator $\mathcal{T}$ and the policy induced by the value $V^\star(s)$ is the optimal deterministic policy $\pi^\star$.

2.7 Computing Optimal Policies

The Bellman optimality equations 2.13 and 2.14 unfortunately are not linear. Therefore, we can no longer use standard linear algebra methods to determine optimal values directly. One way to circumvent the non-linearity would be to construct a linear set of equations to find $V^\star$ by substituting $\pi^\star$ and replacing the $max()$ operator with the action that yields the maximum expected return, but finding $\pi^\star$ is the fundamental problem we are trying to solve.

Fortunately enough, inducing the optimal policy from the optimal value is straightforward. It only requires a one-step look-ahead of the state-value function and can be derived directly if given the action-value function. Note that the transition probability matrix is also required. From 2.13 we can scan through the available actions and choose the action that yields that particular value.

$$\pi^\star(s) = \arg\max \sum_{s' \in S} \sum_{r' \in R} p(s', r \mid s, a) \Big[r + \gamma V^\star(s') \Big] \tag{2.15}$$

Given the optimal action-value function, the optimal policy is induced trivially:

$$\pi^\star(s) = \arg\max Q^\star(s, a) \tag{2.16}$$

This is because the action-value function caches the one-step look-ahead result of the state-action value function. Calculating the policy that induces the maximum value is as simple as taking the $\arg\max()$ of the optimal action-value function. Next, we will present a collection of different algorithms for deriving the best policy and solving the reinforcement learning

problem.

2.7.1 Value-Based Methods

Value Iteration

The first algorithm is straightforward because it exploits the contractive property of the Bellman operator. Select any value vector $V(s)$, associate every state with an arbitrary value and apply the Bellman operator on the value vector again and again. Eventually, the values in each state will stabilize and reach their optimal values.

Algorithm 1 Value Iteration

```
V ← V_0
k = 0
while V_{k-1} − V_k > δ do          ▷ Or equivalent convergence criterion
    V_{k+1} ← 𝒯V_k
end while
```

We apply this algorithm because we know that:

$$\|V_k - V^\star\|_\infty \leq \gamma^k \|V_0 - V^\star\|_\infty, \ \forall k$$

Even though $\lim_{k \to \infty} V_k = V^\star$, the speed of convergence is exponential because of the contraction property. This exponential speed implies that after k steps the error will be $\frac{\log(1/\epsilon)}{\log(1/\gamma)}$

The new policy induced by the algorithm at every iteration is

$$\pi_k = \arg\max_a (1-\gamma) r(s,a) + \gamma \sum_{s'} p(s' \mid s,a) V_k(s')$$

The time complexity of the algorithm grows at least linearly with the size of the transition probability matrix $p : |\mathcal{S}|^2 \times |\mathcal{A}|$ and is of the order of $\mathcal{O}\left(\frac{|\mathcal{S}|^2|\mathcal{A}|}{1-\gamma} \cdot \log\left(\frac{1}{(1-\gamma)\epsilon}\right)\right)$. [14]

Q-Learning

One of the most popular value-based algorithms in the literature is the Q-learning [63] algorithm. Its popularity is mainly attributed to the recent break-through results in games like Atari [41] and has demonstrated impressive results despite its simplicity. This algorithm effectively uses Bellman's optimality equations for action values 2.14 as it tries to select a better action at each state. The algorithm is as follows:

Algorithm 2 Q-Learning

$\alpha \in (0, 1]$

$\epsilon > 0$

$Q(s, a) \leftarrow$ instantiate arbitrarily

for each episode **do**

  Initialize state s

  while step in episode **do**

    Choose action a using $Q(s, a)$

    Take action a and observe s', r

    $Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha \left[r + \gamma \max_a Q(s', a) - Q(s, a) \right]$

    $s \leftarrow s'$

  end while

end for

The Q-learning algorithm solves the state-action Bellman optimality equation 2.14 by using samples from the environment. Directly optimizing $Q(s, a)$ enables Q-learning to directly learn $Q^\star$ instead of constantly switching between policy evaluation and improvement steps. Q-learning is a sample-based version of value iteration that applies the state-action Bellman optimality equation repeatedly until it samples enough data to approximate $Q^\star$. There are 3 requirements [26] for $Q(s, a)$ to converge to $Q^\star$.

1. Memory equal to the cardinal product of the state and action space $|\mathcal{S}| \times |\mathcal{A}|$ to save all pairs of Q values

2. A decaying learning rate α_t

3. Coverage of the entirety of the state-space $\mathcal{S}$.

The last requirement is usually taken care of by using the $\epsilon-$greedy exploration procedure that introduces a small exploratory term that makes the agent choose a random action at every state to achieve the coverage mentioned above.

2.7.2 Policy-Based Methods

An alternative approach would be to search in policy space and find the optimal policy. Policy-based methods have better convergence properties and can learn distributions over actions and not explicitly deterministic actions.

Policy Iteration

Policy iteration takes a step in policy space towards a better policy that expects a slightly higher accumulative reward. This requires first evaluating the policy produced and then improving the policy in the right direction (in policy space). We know that policy evaluation can be done through a system of linear equations. Once we compute the policy's value, we can act greedily with respect to the one-step look-ahead value induced by the policy.

Policy iteration seems slightly more complicated than value iteration. However, the main advantage of this method is that we only need to consider the policy function class to optimize upon and directly optimize in policy space. It is also reasonable to argue that the policy space is (for most practical cases) a smaller space than the space of value functions. In

Algorithm 3 Policy Iteration

$\pi \leftarrow \pi_0$
$k = 0$
while $V_{k-1} - V_k > \delta$ **do** ▷ Or equivalent convergence criterion
$\pi_{k+1}(s) \in \arg\max_{a \in \mathcal{A}} \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V^{\pi_k}(s') \right]$
end while

practice, dynamic programming techniques are implemented to solve both evaluation and improvement steps if we were given access to the transition dynamics p [7].

Policy Gradient Methods

One of the most important policy-based methods for finding the optimal policy was initially presented by Sutton et al. [58]. This straightforward idea updates the gradient along the direction that yields higher potential long-term rewards. Fortunately, the gradient of the expected reward has a closed-form:

$$\nabla V_{\pi_\theta} = \mathbb{E}\left[\nabla \log \pi_\theta(a \mid s) Q_{\pi_\theta}(s,a)\right] \tag{2.17}$$

The $Q_{\pi_\theta}(s,a)$ as mentioned earlier are some type of one-step look-ahead values estimating the expected reward for taking a particular action, based on the current policy π_θ. The expected value can be estimated with sample-based methods such as Monte-Carlo simulations. The last piece missing to calculate the gradient of the value function is the gradient of the logarithm of the policy.

Direct Parameterization

Policy gradient methods require creating a class of parameterized functions to take define gradients and take steps toward a particular direction in policy space. Usually, the opti-

mization method used to decrease the cost function of interest (usually the value function $V^{\pi_\theta}(s)$) is gradient ascent:

$$\theta \leftarrow \theta + \eta \nabla V^{\pi_\theta}(s_0)$$

One parameterization class called the direct parameterization class, associates each state-action pair with a parameter $\theta_{s,a}$.

$$\pi(a \mid s) = \theta_{s,a}$$

Luckily, in that parameterization class [2] the gradient of the parameterization is well-defined $\frac{\partial \theta_{(s,a)}}{\partial (s,a)}$ and has a closed-form. These parameters denote the probability of selecting action a in state s. Changing individual parameters in any gradient ascent step might throw the vector π_s out of simplex, meaning that the vector might no longer be a probability distribution, something that is not allowed by the definition of the policy. Policies should always have the two requirements listed below:

- $\sum\limits_{a \in \mathcal{A}} \theta_{s,a} = 1$
- $\theta_{s,a} \geq 0, \ \forall s \in \mathcal{S}, a \in \mathcal{A}$

In order to keep the probability vector in probability space, after each gradient step the new vector is projected back to the simplex by the projection operator $P_{\Delta(\mathcal{A})^S}$. The projected projected gradient ascent (PGA) update rule at each time-step t is defined below:

$$\pi(a \mid s) = P_{\Delta(\mathcal{A})}\Big(\pi(a \mid s) + \eta \nabla_{\pi_\theta} V^{\pi_\theta}(s)\Big)$$

The completeness of the class parameterization allows for some solid theoretical convergence guarantees [2]. Calculating the gradient of 2.17 can be done through sampling as values can also be calculated by some repeated random sampling method, also referenced as Monte Carlo [30] in the literature. In reinforcement learning, one can sample a trajectory of states,

actions, and rewards and estimate the value function without knowing explicitly the transition probability matrix $p(s' \mid s, a)$ that was responsible for the sequence of state, action, reward tuples along the trajectory. As stated in 2.7, the value of a state under policy π is an expectation over future returns. More samples lead to a better estimation of that expectation, according to the theory of large numbers. This method of estimating the gradient through sampling while also doing projected gradient ascent is called stochastic projected gradient ascent [**?**].

2.7.3 Linear Programming

One last set of methods used to derive the optimal policy and solve the Bellman equations is linear programming. The Bellman optimality equations can be expressed without leveraging properties of the Bellman operator but rather framing them as linear programs.

Specifically, equations 2.13 and 2.14 can be written as a $|\mathcal{S}| \cdot |\mathcal{A}| \times |\mathcal{S}|$ linear program where the inequality constraints replace the max operator:

$$\min_{V} \sum_{s \in \mathcal{S}} \xi_0(s) V(s)$$

$$\text{s.t. } V(s) \geq \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s, a) V(s') + r, \ \forall s \in \mathcal{S}$$

The term $\xi_0(s)$ corresponds to the initial distributions over states as we are now optimizing for all states simultaneously. Writing the Bellman equations as a linear program also allows us to consider the dual linear program with some surprisingly exciting interpretations. The dual program is expressed below:

$$\max_{\lambda} \langle \lambda, r \rangle$$

$$\text{s.t. } \sum_{a \in \mathcal{A}} \lambda_a = \xi_0 + \gamma \sum_{a \in \mathcal{A}} p_\alpha(s' \mid s) \lambda_\alpha$$

Now, we are trying to optimize for λ and find $\lambda^\star$ which is called discounted cumulative occupancy measure under the optimal policy:

$$\lambda(s,a)^\star = \mathbb{E}_{\pi^\star}\left[\sum_{t=0}^{\infty}\gamma^t 1(S_t = s, A_t = a)\right]$$

This measure informs us about the long-term stationary state-action distribution induced by the optimal policy. Searching in this abstract space of possible occupancy measures is equivalent to searching for the optimal value as the inner product $\langle \lambda, r\rangle$ in the argument of our maximization objective is the value function.

There are numerous results utilizing interior point methods to provide convergence rates, that are unfortunately slower than value/policy iteration due to the lack of Bellman operator exploitation. One such algorithm [68] guarantees a super-linear rate with respect to the input size $|\mathcal{S}|^2 \cdot |\mathcal{A}|$ of $\mathcal{O}\left(|\mathcal{S}|^4|\mathcal{A}|^4 \log\left(\frac{|\mathcal{S}|}{\epsilon}\right)\right)$.

Chapter 3

Multi-Agent Reinforcement Learning

Most of the recent advances in reinforcement learning have been made in the single-agent domain introduced in Chapter 2 where a single agent is guaranteed to learn a policy that maximizes the expected sum of discounted returns. Some problems are game-theoretic in nature and involve multiple agents interacting, each with special incentives. Setups with numerous agents are a natural extension of the single-agent case and are called stochastic or Markov games. Recently, OpenAI [45] demonstrated superhuman performance in such multi-player settings by utilizing extensive amounts of computation without any theoretical guarantees of (global) convergence. In this chapter we introduce the extension of MDPs for multiple players and explain the numerous challenges that need to be overcome when designing efficient decentralized algorithms for team games.

3.1 Markov Games

The first formal description of these setups came from Shapley in 1952 [55], and they are known to be a generalization of the single-agent MDP. As a model, stochastic games capture

dynamic interactions between multiple agents and broaden the consideration of actions based on the existence of other rational agents operating simultaneously in the same environment.

In this framework, the transitions depend on all actions of all players. All players experience different reward signals that are all functions of all players' actions again.

$$s_{t+1} \sim p(\cdot \mid s_t, a_t^1, \ldots, a_t^N)$$

$$r_{t+1}^i = R^i(s_t, a_t^1, \ldots, a_t^N, s_{t+1})$$

The stochastic game framework addresses sequential decision-making problems for multiple agents. In this coupled setting, $N > 1$ agents all co-exist in an environment, take actions, traverse to states and receive rewards. A pictorial illustration of the extension is provided below:

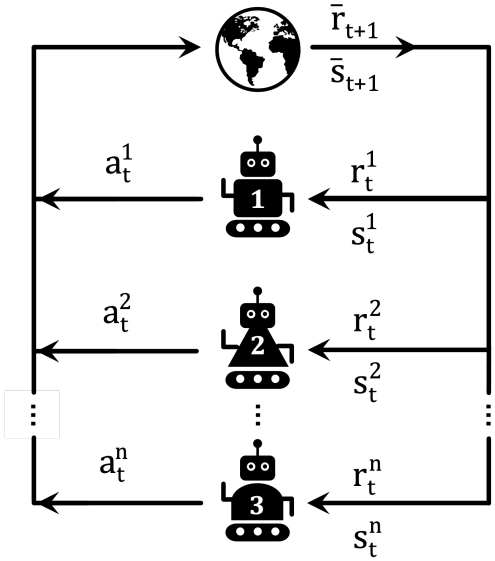

Figure 3.1: A discrete-time dynamical system describing the interaction of multiple agents with the environment.

Formally, a Markov game is defined as a tuple $\mathcal{G} = \Big(N, \mathcal{S}, \{\mathcal{A}^i\}_{i \in N}), \mathcal{P}, \{\mathcal{R}^i\}_{i \in \mathcal{N}}, \gamma\Big)$ that

represent the following:

- N is the number of players/agents in the environment.
- $\mathcal{S}$ is the state-space of the environment.
- $\mathcal{A}^i$ is the action set of each agent i and the total action space is the Cartesian product of action spaces $\mathcal{A} := \mathcal{A}^1 \times \mathcal{A}^2 \times \cdots \times \mathcal{A}^N$
- The transition probability matrix $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ transitions the environment from a state s to another state s'.
- The reward function $\mathcal{R}^i : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ broadcasts an immediate reward to each agent in the environment, based on the transition from state s to state s'. It is important to note that the reward is different for each agent.
- γ is the discount factor added as a parameter to capture the infinite-horizon case.

MARL Challenges

The multi-agent setup has many additional challenges compared to the single-agent case. Interesting dynamics emerge in figure 3.1 as the action of each agent affects the transition of the environment to different states that ultimately affect the rewards of all agents. Each agent has to now optimize over actions that, on top of the stochasticity of the environment, should account for the actions of other players in the environment. This property, by itself, makes the theoretical single-agent convergence guarantees to no longer hold. All of the guarantees enjoyed by the single-agent framework case rely on the stationarity of the environment that does not contain any adaptive agents.

Furthermore, optimization algorithms within the general MARL scheme might have conflicting objectives. Loss functions are multi-dimensional and sometimes mutually contradicting.

This tension leads to shifting the main focus from finding the optimal policy to finding equilibrium points in a multi-dimensional space of strategies/policies.

On top of that, the joint action-space of all the agents grows exponentially with respect to the number of agents in the environment. Each agent adds one more dimension to the action space, introducing what is known as the combinatorial nature of MARL, otherwise known as the "curse of multi-agents."

Additionally, the multi-agent extension of the Markov decision process has a rather complicated information structure. We see from figure 3.1 that each agent takes action based on different obesrvation and experiences different rewards based on everyone's actions. This might lead to sub-optimal local results as far as each agent is concerned. This notion is well studied in game theory and is called the price of anarchy [15]. This measure quantifies performance degradation when transitioning from a centralized to a decentralized algorithm where agents are trying to locally optimize with respect to their individual loss function.

3.2 Solution Concept

Simultaneously optimizing a set of functions for the same value is not possible. The maximization of one function might not imply the maximization of any other function from our pool of functions. This inherent obstacle leads the research direction towards approximating Nash equilibria as a natural solution concept to the problem.

3.2.1 Nash Equilibrium

Before introducing Nash policies and Nash equilibria, it is important to understand why Nash equilibria are considered to be the appropriate solution concept for multi-agent reinforcement

learning. Finding Nash equilibria might be complicated for the following reasons:

- Nash equilibria do not guarantee optimality. A game might have a globally optimal solution with players preferring to deviate from their current strategy unilaterally and behave greedily with respect to their local payoff.
- There might be a set of individual policies that achieve a solution with the same quality. It is, therefore, difficult to determine which combination of policies to choose from.

Luckily, pure Nash equilibria are not the only solution concept in game theory. Algorithm designers have also crafted algorithms that converge to other notions of equilibria. Some examples are correlated equilibria and coarse correlated equilibria.

Definition A Nash equilibrium of the stochastic game G is a set of policies $\pi^\star = (\pi_1^\star, \pi_2^\star, \dots, \pi_N^\star)$ that given the other players policies, they have no incentive unilaterally deviating from their policy as they payoff they will experience will be less than their current one. The value of each agent is defined as:

$$V_s^i(\pi^i, \pi^{-i}) := \mathbb{E}\Big[\sum_{t\geq 0} \gamma^t R^i(s_t, a_t, s_{t+1}) \mid a_t^i \sim \pi^i(\cdot \mid s_t), s_0 = s\Big] \tag{3.1}$$

No incentive for unilateral deviation equivalently means that the policy of every agent i is the best response of the set of the remaining set of policies $\pi_{-i}^\star$. For a Nash policy $\pi_i^\star$, it holds that:

$$V_s^i(\pi_i^\star, \pi_{-i}^\star) \geq V_s^i(\pi_i, \pi_{-i}^\star),\ \forall\ \pi_i \in \Delta(\mathcal{A}_i)^s), \text{and } \forall s \in \mathcal{S} \tag{3.2}$$

3.2.2 Existence of Nash policies

One remarkable fact about stochastic games is that Nash policies always exist. Shapley proved [55] that the minimax theorem applies in a two-player zero-sum stochastic game. A well-defined notion of value exists. Hence, if agents execute policy gradient independently, they converge to the Nash policy $(\pi_1^\star, \pi_2^\star)$. For every policy $\pi_i^\star$:

$$V_\rho(\pi_i, \pi_{-i}^\star) \leq V_\rho(\pi_i^\star, \pi_{-i}^\star),\ \forall\ i \in [N]$$

Nash equilibria always exist but might not necessarily be unique. Many of the MARL algorithms are set to converge to these points. In stark contrast with the single-agent case, an optimal deterministic policy does not exist. Agents should be able to converge to stochastic policies since they account for the actions of other agents in the environment. Markov games are capable of capturing all three fundamental MARL settings.

3.2.3 Cooperative Setting

One of the most common settings in multi-agent reinforcement learning is the fully collaborative setting. All agents have to collaborate to achieve a common goal. All agents share the same reward $R = R_1 = R_2 = \cdots = R_N$. This model is widely used in warehouse environments [56] and is also referenced as Markov team games in the literature [62]. Cooperative games are also interesting because they fall under the umbrella of Markov potential games [33], a class of games that has provable convergence guarantees if all agents optimize their local value function by using independent policy gradient methods. Part of this important result will be leveraged to extend the results to a more general setup where a team of agents collaborates against an adversary.

3.2.4 Competitive Setting

Competition between players arises when the rewards received are opposite. From a game-theoretic perspective, these games are zero-sum because the sum of rewards for all players is zero.

3.2.5 Mixed Setting

Games where coordination and cooperation are involved are called general-sum games [25]. Rewards might be aligned for some and misaligned for others. This setting is challenging due to the lack of structure in the game. Computing Nash equilibria is notoriously hard in these settings. It has been shown that even for two-player general sum stateless games (called normal-form), finding the Nash equilibrium is intractable [13]. Many known algorithms in the literature fail in stochastic MDPs. Cyclic behavior has been observed when applying value iteration in mixed settings [71]. Therefore, experts studying algorithms should exploit any structure of the game when designing efficient algorithms to find Nash equilibria, especially in mixed settings.

Chapter 4

Main Results

In this section we first introduce our MARL setting, state our results and lay down the techniques used in our work.

4.1 Adversarial Team Stochastic Games

An Adversarial Team Stochastic Game (or Adversarial Team Markov Decision Process) is the stochastic game extension of static, normal-form adversarial team games [60]. The game takes place in an infinite-time horizon and future rewards are discounted by some positive constant shared across all agents. A team of agents that share the same utility win what the adversary loses (utilities sum to zero). The game is represented by a tuple $\mathcal{G} = (\mathcal{S}, \mathcal{N}, \mathcal{A}, \mathcal{B}, \{r_k\}_{k\in\mathcal{N}}, P, \gamma, \rho)$ whose constituents are:

- $\mathcal{S}$ a finite set of states, with size $S = |\mathcal{S}|$ and an initial state distribution ρ over states $\rho \in \Delta(\mathcal{S})$ over all states $\Delta(\mathcal{S})$
- a set of n team agents and one adversary $\mathcal{N} = \mathcal{N}_A \cup \mathcal{N}_B = \{1, \ldots, n\} \cup \{n+1\}$

- the action-space, i.e. a finite, non-empty set of actions for each team-agent k, $\mathcal{A}_k$ ($\mathcal{A}_{-k}$ denotes the action-space of all team-agents apart from k); respectively, by $\mathcal{B}$ we denote the adversary's set of actions ; also, $A_k = |\mathcal{A}_k|$ and respectively $B = |\mathcal{B}|$ stand for the cardinality of the corresponding action-space

- a deterministic reward function $r_k : \Delta(\mathcal{S}) \times \Delta(\mathcal{A}) \times \Delta(\mathcal{B}) \to [-1, 1]$ for the k-th player of the team and a reward function $r_{adv} : \Delta(\mathcal{S}) \times \Delta(\mathcal{A}) \times \Delta(\mathcal{B}) \to [-1, 1]$ for the adversary, all mapping distributions of states and actions to instantaneous rewards

- a transition probability function $P : \Delta(\mathcal{S}) \times \mathcal{A} \to [0, 1]$, such that $P(s'|\mathbf{a}, b, s)$ denotes the probability of moving to state s' from state s when the team selects an action profile $\mathbf{a} \in \mathcal{A}$ and the adversary picks an action $b \in \mathcal{B}$

- $\gamma \in [0, 1)$ a discount factor shared among all agents for the future rewards of the game

What makes the latter an adversarial team game is the fact that all team-agents $k \in \mathcal{N}_A$ get the same reward $r_{team} = r_k,\ \forall k \in \mathcal{N}_A$, opposite to the adversary's reward $r = -r_{team}$. We note that when time $t \geq 0$ is relevant, we denote it with a superscript over the corresponding variable. Ultimately, at time t and state $s^{(t)}$, team-agent k $\big($resp. the adversary$\big)$ picks action $a_k^{(t)}$ $\big($resp. $b^{(t)}\big)$ obtains a reward $r^{(t)} = r(s^{(t)}, \mathbf{a}^{(t)}, b^{(t)})$, and the game transitions to a new state s^{t+1} drawn from a probability distribution $P(\cdot|s^{(t)}, \mathbf{a}^{(t)}, b^{(t)})$. Furthermore, if we set $\mathbf{r}^{(t)}$ to be a vector whose every entry is each agent's reward, we can use $\tau = (s^{(t)}, \mathbf{r}^{(t)}, \mathbf{a}^{(t)}, b^{(t)})$ to notate a given trajectory of the game.

Direct Parametrization For our analysis we assume all players use direct policy parametrization. As such we replace $\Delta(\mathcal{A}_k) \to \mathcal{X}_k$, $\big($resp. $\Delta(\mathcal{B}) \to \mathcal{Y}\big)$, $\pi_k \to \mathbf{x}_k$, $\big($resp. $\pi_{adv} \to \mathbf{y}\big)$, $\pi_k(a|s) \to x_{k,s,a}$, $\big($resp. $\pi_{adv}(a|s) \to y_{s,a}\big)$.

4.2 Policies, Values, Solution Concept

A stationary, deterministic policy for agent k (i.e. time-invariant and picking a certain action at each state with probability equal to 1) is a function $\pi : \mathcal{S} \to \mathcal{A}_k$. A natural extension of the latter is a stationary, stochastic policy that maps a given state to a distribution over available actions, $\pi k : \mathcal{S} \to \Delta(\mathcal{A}_k)$ $\big($resp. $\pi_{adv} : \mathcal{S} \to \Delta(\mathcal{B})\big)$. Finally, by $\Pi_{team} : \mathcal{S} \to \Delta(\mathcal{A})$ and $\Pi_{adv} : \mathcal{S} \to \Delta(\mathcal{B})$ we denote the policy-space of the team and the adversary correspondingly, while by $\Pi : \mathcal{S} \to \Delta(\mathcal{A}) \times \Delta(\mathcal{B})$ we denote the policy-space of all agents.

We define the value function $V_s : \Pi \to \mathbb{R}$ as the expected sum discounted reward experienced by the adversary. In the case that the initial state is s_0 with probability equal to 1, the value function is defined as:

$$V_s(\pi, \pi_{adv}) = \mathbb{E}_{(\pi,\pi_{adv})}\left[\sum_{t=0}^{\infty} \gamma^t r_k^{(t)} \big| s_0 = s\right] \tag{4.1}$$

Whereas, when s_0 is drawn from a probability distribution ρ over states, the value function takes the form $V_\rho(\pi, \pi adv) = \mathbb{E}_{s\sim\rho}\Big[V_s(\pi, \pi adv)\Big]$. The adversary's objective is to maximize V_ρ, while the team strives to minimize it. Subsequently, the solution concept we are concerned with is the Nash equilibrium (NE) in infinite-horizon discounted stochastic games.

Nash Equilibrium

A joint policy profile $\big(\pi^*, \pi^*_{adv}\big) = \big((\pi k^*; \pi - k^*), \pi^*_{adv}\big)$ is said to be a Nash equilibrium when:

$$\begin{cases} V_\rho\big((\pi^*_k; \pi^*_{-k}), \pi^*_{adv}\big) \;\leq\; V_\rho\big((\pi'_k; \pi^*_{-k}), \pi^*_{adv}\big), \;\; \forall k, \forall \pi'_k \in \Delta(\mathcal{A}_k)^S \\ V_\rho\big(\pi^*, \pi^*_{adv}\big) \quad \geq \quad V_\rho\big(\pi^*, \pi'_{adv}\big), \quad \forall \pi'_{adv} \in \Delta(\mathcal{B})^S \end{cases}$$

4.3 Results

We study an infinite-horizon stochastic (Markov) game with a finite state space $\mathcal{S}$ in which a team of agents $\mathcal{N}_A$ with a common objective function is competing against a single adversary with opposing interests. Every agent $k \in \mathcal{N}_A$ has a (finite) set of available actions $\mathcal{A}_k$, while $\mathcal{B}$ represents the adversary's set of actions. We also let γ be the discounting factor. Our goal will be to compute a Nash equilibrium; that is, a strategy profile so that neither player can improve via a unilateral deviation (see Definition 4.2 for a formal description). In this context, our primary contribution is summarized with Theorem 1. Below we present algorithm 4.

Algorithm 4 Independent Policy GradientMax (IPGmax)

1: **for** $t \leftarrow 1, 2, \ldots$ **do**
$\mathbf{y}^{(t)} \leftarrow \arg\max_{\mathbf{y}} V_\rho\big(\mathbf{x}^{(t-1)}, \mathbf{y}^{(t-1)}\big)$
$\mathbf{x}_i^{(t)} \leftarrow \mathrm{Proj}_{\mathcal{X}_i}\left\{\mathbf{x}_i^{(t-1)} - \eta \nabla_{\mathbf{x}_i} V_\rho\big(\mathbf{x}^{(t-1)}, \mathbf{y}^{(t)}\big)\right\}$ ▷ for all agents i
2: **end for**
3: $\hat{\mathbf{x}} \leftarrow \mathbf{x}^{(\infty)}$
4: $\hat{\mathbf{y}} \leftarrow \texttt{AdvNashPolicy}(\hat{\mathbf{x}})$
5: **return** $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$

4.3.1 Techniques

Our decentralized Algorithm 4 (IPGmax) works in turns. At each iteration, the adversary chooses his/her optimal policy (plays best response) and then each player of the team runs one step of independent policy gradient on their value function. The algorithm we suggest exploits the machinery provided by [36] for the analysis of GDmax. GDmax is gradient descent on the function $\max_{\mathbf{y}\in\mathcal{Y}} f(\cdot, \mathbf{y})$. [36] showed that GDmax converges to a point $\big(\hat{\mathbf{x}}, \mathbf{y}^\star(\hat{\mathbf{x}})\big)$ in which $\hat{\mathbf{x}}$ is a stationary point of the Moreau envelope of $\max_{\mathbf{y}\in\mathcal{Y}} f(\mathbf{x}, \mathbf{y})$ and $\mathbf{y}^*(\hat{\mathbf{x}})$ is a best

response to $\hat{\mathbf{x}}$. If the function $f(\mathbf{x}, \cdot)$ is strongly-concave, by Dantzig's theorem, one can show that $\big(\hat{\mathbf{x}}, \mathbf{y}^*(\mathbf{x})\big)$ is a stationary point of f. However, if the function f is not strongly-concave in $\mathbf{y}$ (concavity does not suffice), $\big(\hat{\mathbf{x}}, \mathbf{y}^*(\mathbf{x})\big)$ does not necessarily translate to a stationary point of $f(\mathbf{x}, \mathbf{y})$; this is the case for our setting since the value function $V_\rho(\pi_{team}, \pi_{adv})$ is nonconvex-nonconcave.

Using an analysis similar to that of GDmax, IPGmax is guaranteed to converge to a policy profile $\big(\hat{\pi}_{team}, \pi^*_{adv}(\hat{\pi}_{team})\big)$ at which the adversary plays best response and $\hat{\pi}_{team}$ is guaranteed to be a first-order stationary point of $\max_{\pi_{adv}} V(\pi_{team}, \pi_{adv})$ with respect to π_{team}. In this context, we are able to show the following characterization.

If $\hat{\pi}_{team}$ is a first-order stationary point of $\max_{\pi_{adv}} V(\pi_{team}, \pi_{adv})$, then there exists a policy of the adversary $\hat{\pi}_{adv}$ that can be extended to a Nash equilibrium policy $(\hat{\pi}_{team}, \hat{\pi}_{adv})$.

In the special case of normal-form games, a weaker extension theorem was established in the seminal work of [60]. In particular, that result is derived by employing fairly standard linear programming techniques, analyzing the structural properties of the optimal of a suitable LP. On the other hand, our setting introduces several new challenges, not least due to the non-convexity non-concavity of the objective function. Indeed, our method leverages more refined techniques from non-convex programming:

Respectively, while our work makes use of standard policy gradient properties similar to the single-agent Markov Decision Process setting [1, 66], the analysis will not rely on the gradient-dominance property (see [8]), as it does not appear to hold in a team-wise sense. Instead, our technique is inspired by an alternative proof of Shapley's theorem [50] for two-person zero-sum stochastic games that utilizes mathematical programming [61]. In particular, we adopt concepts from the literature of mathematical programming and optimization, namely the guarantees of Karush-Kuhn-Tucker (KKT) conditions and *constraint qualifications* (CQs) when non-convex constraints are in run. Generally, Constraint Qualifications are used to

prove that all feasible (local) optima of the objective function are contained in the set of KKT points. In a nutshell, we prove the extensibility of stationary points of $\max_{\pi_{adv}} V_\rho(\cdot, \pi_{adv})$ to Nash equilibria (rendering them efficient to compute) using a modification of Arrow-Hurwiz-Uzawa CQs [4] and Lagrange duality of the NLP [46]. Then, we observe that the structure of our model (Adversarial Team Stochastic Games) supplies a natural meaning (within the context of the model) to a subset of the Lagrange multipliers. That is, a part of this subset can be used to define a Nash policy for the adversary whereas another part of it is intrinsically related to the value of the corresponding Nash Equilibrium.

4.4 Main Theorem

Theorem 1. *Suppose that both agents of the team $\mathcal{N}_A$ and the adversary follow the IPGmax (Algorithm 4) with direct parametrization. If the learning rates are sufficiently small, the fixed points of IPGmax, let $(\hat{x}, \hat{y})$, are guaranteed to be Nash equilibria policies.*

Proof Sketch

This proof shows that in the setting of adversarial team games, any local minimum of a nonlinear program describing the payoff of the team can be extended to a Nash Equilibrium through the use of Lagrange Duality. The argument is simple, after showing that any local minimum satisfies the Arrow-Hurwicz-Uzawa constraint qualification, the non-negativity of the Lagrange multipliers is leveraged to *i*) demonstrate relations between the multipliers and the value of the objective function *ii*) form an optimal strategy vector for the adversary *iii*) show that the local minimum w.r.t. to the team extends to a Nash Equilibrium.

Proof of Theorem 1

Extendibility of team-local minima to NE in normal-form games through Mathematical Programming

This section contains the proof for Theorem 1. This proof boils down to proving that $\hat{x}$ can be extended to $(\hat{x}, \hat{y})$ so that $(\hat{x}, \hat{y})$ is a Nash equilibrium policy for the game.

Let a game between a team of n players with (mixed) strategy vectors $\mathbf{x}_k$ for each player k, playing against an adversary with a strategy vector $\mathbf{y}$.

Let $C_b(\mathbf{x})$ denote the payoff the team gets when they play $\mathbf{x} = (\mathbf{x}_1, \cdots, \mathbf{x}_n)$ and the adversary plays her b-th pure strategy.

The payoff of the adversary is $U(\mathbf{x}, \mathbf{y}) = \sum_b y_b C_b(\mathbf{x})$.

A (local) team-minmax strategy for the team, is given by the following nonlinear program (NLP):

$$\begin{aligned}
\min \quad & u && \text{(4.2a)}\\
\text{s.t.} \quad & C_b(\mathbf{x}) \leq u && \text{(4.2b)}\\
& \mathbf{x}_k^\top \mathbf{1} = 1 && \text{(4.2c)}\\
& x_{k,a} \geq 0 && \text{(4.2d)}
\end{aligned}$$

We denote the constraints:

$$h_b = C_b(\mathbf{x}) - u \tag{4.3}$$

$$h_k = \mathbf{x}_k^\top \mathbf{1} - 1 \tag{4.4}$$

$$h'_k = 1 - \mathbf{x}_k^\top \mathbf{1} \tag{4.5}$$

$$h_{(k,a)} = -x_{k,a} \tag{4.6}$$

Let $\boldsymbol{e}_a$ denote the one-hot vector with 1 in the a-th entry. Also, by multi-linearity, see that:

$$C_b(\mathbf{x}) = \sum_a x_{k,a} C_b(\boldsymbol{e}_a; \mathbf{x}_{-k}) \tag{4.7}$$

Now, it is easy to observe that:

$$\frac{\partial C_b(\mathbf{x})}{\partial x_{k,a}} = C_b(\boldsymbol{e}_a; \mathbf{x}_{-k}) \tag{4.8}$$

The Lagrangian of the NLP and the Karush-Kuhn-Tucker optimality conditions

Write the Lagrangian of the NLP:

$$\begin{aligned}\mathcal{L}\big((\mathbf{x}, u), \boldsymbol{\mu}\big) = u + \sum_b \mu_b\big(C_b(\mathbf{x}) - u\big) + \sum_k \mu_k\big(\mathbf{x}_k^\top \mathbf{1} - 1\big) + \\ \sum_k \mu'_k\big(1 - \mathbf{x}_k^\top \mathbf{1}\big) + \sum_{(k,a)} \mu_{(k,a)}\big(-x_{k,a}\big)\end{aligned} \tag{4.9}$$

Rename the Lagrangian multipliers for convenience:

$$\underbrace{\lambda(b)}_{\mu_b}, \underbrace{\omega(k)}_{\mu_k}, \underbrace{\psi(k)}_{\mu'_k}, \underbrace{\gamma(k,a)}_{\mu_{(k,a)}} \tag{4.10}$$

Write the derivatives of the Lagrangian and of the constraints:

- For all $\lambda(b)$:

  W.r.t. u:

$$\frac{\partial h_b}{\partial u} = -1 \tag{4.11}$$

  W.r.t. $x_{k,a}$:

$$\frac{\partial h_b}{\partial x_{k,a}} = C_b(\boldsymbol{e}_a; \mathbf{x}_{-k}) \tag{4.12}$$

- For $\omega(k)$ and $\psi(k)$: W.r.t. u:

$$\frac{\partial h_k}{\partial u} = \frac{\partial h'_k}{\partial u} = 0 \tag{4.13}$$

  W.r.t. $x_{k,a}$:

$$\frac{\partial h_k}{\partial x_{\overline{k},a}} = \begin{cases} 1, & \text{if } k = \overline{k} \\ 0, & \text{else} \end{cases} \tag{4.14}$$

  and

$$\frac{\partial h'_k}{\partial x_{\overline{k},a}} = \begin{cases} -1, & \text{if } k = \overline{k} \\ 0, & \text{else} \end{cases} \tag{4.15}$$

- For $\gamma(k, a)$: W.r.t. u:

$$\frac{\partial h_{(k,a)}}{\partial u} = 0 \tag{4.16}$$

W.r.t. $x_{k,a}$:

$$\frac{\partial h_k}{\partial x_{\overline{k},\overline{a}}} = \begin{cases} 1, & \text{if } (k,a) = (\overline{k},\overline{a}) \\ 0, & \text{else} \end{cases} \tag{4.17}$$

- For the Lagrangian:

  W.r.t. u:

$$\frac{\partial \mathcal{L}}{\partial u} = 1 - \sum_b \lambda(b) \tag{4.18}$$

  W.r.t. $x_{k,a}$:

$$\frac{\partial \mathcal{L}}{\partial x_{k,a}} = \sum_b \lambda(b) C_b(\boldsymbol{e}_a; \mathbf{x}_{-k}) + \omega(k) - \psi(k) - \gamma(k,a) \tag{4.19}$$

We can now write the Karush-Kuhn-Tucker optimality conditions for the program above:

$$\begin{aligned} \nabla_{(\mathbf{x},u)}\Big(u\Big) + \sum_b \lambda(b) \nabla_{(\mathbf{x},u)}\Big[C_b(\mathbf{x}) - u\Big] + \sum_k \omega(k) \nabla_{(\mathbf{x},u)}\Big[\mathbf{x}_k^\top \mathbf{1} - 1\Big] + \\ + \sum_k \psi(k) \nabla_{(\mathbf{x},u)}\Big[1 - \mathbf{x}_k^\top \mathbf{1}\Big] + \sum_{(k,a)} \gamma(k,a) \nabla_{(\mathbf{x},u)}\Big[- x_{k,a}\Big] = \mathbf{0} \end{aligned} \tag{4.20}$$

$$\lambda(b)\Big(C_b(\mathbf{x}) - u\Big) = 0, \quad \forall b \tag{4.21}$$

$$\omega(k)\Big(\mathbf{x}_k^\top \mathbf{1} - 1\Big) = 0, \quad \forall k \tag{4.22}$$

$$\psi(k)\Big(1 - \mathbf{x}_k^\top \mathbf{1}\Big) = 0, \quad \forall k \tag{4.23}$$

$$-\gamma(k,a) x_{k,a} = 0, \quad \forall (k,a) \tag{4.24}$$

$$\lambda(b), \omega(k), \psi(k), \gamma(k) \geq 0 \tag{4.25}$$

Under the assumption that a constraint qualification is satisfied, there do exist non-negative Lagrange multipliers
$\lambda(b), \omega(k), \psi(k), \gamma(k)$ satisfying the above system of equations.

Goal With $U(\mathbf{x}, \mathbf{y})$ denote $U(\mathbf{x}, \mathbf{y}) = \sum_b y_b C_b(\mathbf{x})$ Let $(\hat{\mathbf{x}}, \hat{u})$ be a local minimum of the NLP. We are going to try to show that:

- There exists $\hat{\mathbf{y}}$:
$$\hat{\mathbf{y}} = \boldsymbol{\lambda} \cdot \frac{1}{\sum_b \lambda(b)}$$
that is optimal for the adversary when the team plays $\hat{\mathbf{x}}$.
- $\mathbf{x}, \mathbf{y}$ is a NE, i.e.:
$$U(\hat{\mathbf{x}}, \mathbf{y}) \leq U(\hat{\mathbf{x}}, \hat{\mathbf{y}}) \leq U(\mathbf{x}_k, \hat{\mathbf{x}}_{-k}, \hat{\mathbf{y}})$$

Idea Follow Vrieze's proof for the two-player zero-sum stochastic game proof for the existence of a unique value.

Connect $\hat{u}$ with the Lagrange multipliers

$$\hat{x}_{k,a} \cdot \frac{\partial \mathcal{L}}{\partial_{x_{k,a}}}\bigg|_{\mathbf{x}=\hat{\mathbf{x}}} = \hat{x}_{k,a}\Big[\sum_b \lambda(b) C_b(\boldsymbol{e}_a; \hat{\mathbf{x}}_{-k}) + \omega(k) - \psi(k) - \gamma(k,a)\Big] = 0 \Rightarrow \tag{4.26}$$

$$\sum_b \lambda(b)\hat{x}_{k,a} C_b(\boldsymbol{e}_a; \hat{\mathbf{x}}_{-k}) + \hat{x}_{k,a}\Big(\omega(k) - \psi(k)\Big) - \hat{x}_{k,a}\gamma(k,a) = 0 \Rightarrow \tag{4.27}$$

$$\sum_b \lambda(b)\hat{x}_{k,a} C_b(\boldsymbol{e}_a; \hat{\mathbf{x}}_{-k}) + \hat{x}_{k,a}\Big(\omega(k) - \psi(k)\Big) = 0 \tag{4.28}$$

Sum for all a:

$$\sum_a \Big[\sum_b \lambda(b) \hat{x}_{k,a} C_b(\boldsymbol{e}_a; \hat{\mathbf{x}}_{-k}) + x_{k,a} \Big(\omega(k) - \psi(k) \Big) \Big] = 0 \Rightarrow \tag{4.29}$$

$$\sum_b \lambda(b) C_b(\hat{\mathbf{x}}_k; \hat{\mathbf{x}}_{-k}) + \Big(\omega(k) - \psi(k) \Big) = 0 \tag{4.30}$$

Compare the latters equation with $\lambda(b) \Big(C_b(\hat{\mathbf{x}}) - \hat{u} \Big) = 0 \Rightarrow \sum_b \lambda(b) \Big(C_b(\hat{\mathbf{x}}) - \hat{u} \Big) = 0$ to see that:

$$\hat{u} = \frac{\psi(k) - \omega(k)}{\sum_b \lambda(b)} \tag{4.31}$$

Defining a strategy vector for the adversary by using the Lagrange multipliers $\lambda(b)$

Now, divide the partial derivative of the Lagrangian w.r.t. $x_{k,a}$ by $\sum_b \lambda(b)$:

$$\frac{1}{\sum_b \lambda(b)} \frac{\partial \mathcal{L}}{\partial x_{k,a}} = 0 \Rightarrow \tag{4.32}$$

$$\frac{1}{\sum_b \lambda(b)} \left(\sum_b \lambda(b) C_b(\boldsymbol{e}_a; \hat{\mathbf{x}}_{-k}) + \omega(k) - \psi(k) - \gamma(k,a) \right) = 0 \Rightarrow \tag{4.33}$$

$$\sum_b \frac{\lambda(b)}{\sum_{b'} \lambda(b')} C_b(\boldsymbol{e}_a; \hat{\mathbf{x}}_{-k}) + \frac{\omega(k) - \psi(k)}{\sum_{b'} \lambda(b')} - \frac{\gamma(k,a)}{\sum_{b'} \lambda(b')} = 0 \Rightarrow \tag{4.34}$$

$$\sum_b \frac{\lambda(b)}{\sum_{b'} \lambda(b')} C_b(\boldsymbol{e}_a; \hat{\mathbf{x}}_{-k}) - \hat{u} - \frac{\gamma(k,a)}{\sum_{b'} \lambda(b')} = 0 \Rightarrow \tag{4.35}$$

$$\sum_b \frac{\lambda(b)}{\sum_{b'} \lambda(b')} C_b(\boldsymbol{e}_a; \hat{\mathbf{x}}_{-k}) - \hat{u} \geq 0 \Rightarrow \tag{4.36}$$

$$\sum_b \frac{\lambda(b)}{\sum_{b'} \lambda(b')} C_b(\boldsymbol{e}_a; \hat{\mathbf{x}}_{-k}) \geq \hat{u} \tag{4.37}$$

Observe that, from $\frac{\partial \mathcal{L}}{\partial u} = 1 - \sum_b \lambda(b) = 0$ w get that $\sum_b \lambda(b) = 1 > 0$. Hence, $\frac{1}{\sum_{b'} \lambda(b)}$ is well-defined. Let $\hat{\mathbf{y}} = \frac{\lambda(b)}{\sum_{b'} \lambda(b')}$ and we get:

$$\sum_b \hat{y}_b C_b(\boldsymbol{e}_a; \hat{\mathbf{x}}_{-k}) \geq \hat{u} \tag{4.38}$$

Deviations of team-players yield worse outcomes when adversary plays $\hat{\mathbf{y}}$

Take an arbitrary convex combination (with coefficients $x_{k,a}$) of the above equation to form an inequality for a corresponding $\mathbf{x}_k$:

$$\sum_b \hat{y}_b C_b(\mathbf{x}_k; \hat{\mathbf{x}}_{-k}) \geq \hat{u} \Leftrightarrow \tag{4.39}$$

$$U\big((\mathbf{x}_k; \hat{\mathbf{x}}_{-k}), \hat{\mathbf{y}}\big) \geq \hat{u} \tag{4.40}$$

Deviations of the adversary yield smaller value

By feasibility of $(\hat{\mathbf{x}}, \hat{u})$

$$C_b(\hat{\mathbf{x}}) \leq \hat{u} \tag{4.41}$$

Take an arbitrary convex combination of the inequality (with coefficients y_b) to get:

$$\sum_b y_b C_b(\hat{\mathbf{x}}) \leq \hat{u} \Leftrightarrow \tag{4.42}$$

$$U(\hat{\mathbf{x}}, \mathbf{y}) \leq \hat{u} \tag{4.43}$$

Show that $\hat{u} = U(\hat{\mathbf{x}}, \hat{\mathbf{y}})$

It is now straightforward to see that from:

$$U(\hat{\mathbf{x}}, \mathbf{y}) \leq \hat{u} \leq U\big((\mathbf{x}_k; \hat{\mathbf{x}}_{-k}), \hat{\mathbf{y}}\big) \Rightarrow \tag{4.44}$$

$$\hat{u} = U(\hat{\mathbf{x}}, \hat{\mathbf{y}}) \tag{4.45}$$

Show that Arrow-Hurwicz-Uzawa Constraint Qualification are satisfied for any local minimum of the NLP

Denote $A(\hat{\mathbf{x}})$ the set of all inequality constraints that are active at a local minimum $\hat{\mathbf{x}}$.

There exist at least one vector $\boldsymbol{w} \in \mathbb{R}^d$, $d = \underbrace{1}_{u} + \underbrace{\sum_{k}^{n} m_k}_{m_k \text{ avail. moves to player } k}$, s.t. for all $h_i \in A(\hat{\mathbf{x}})$:

$$\begin{cases} \nabla_{\boldsymbol{z}} h_i(\boldsymbol{z})^\top \boldsymbol{w} < 0 & \text{if } h_i \text{ non-(pseudo-)concave} \\ \nabla_{\boldsymbol{z}} h_i(\boldsymbol{z})^\top \boldsymbol{w} \leq 0 & \text{if } h_i \text{ (pseudo-)concave} \end{cases} \tag{4.46}$$

W.l.o.g. assume that the following constraints are active:

$$\begin{cases} \nabla_{(\mathbf{x},u)} h_b(\mathbf{x}, u)^\top \boldsymbol{w} & < 0 \\ \nabla_{(\mathbf{x},u)} h_k(\mathbf{x}, u)^\top \boldsymbol{w} & \leq 0 \\ \nabla_{(\mathbf{x},u)} h'_k(\mathbf{x}, u)^\top \boldsymbol{w} & \leq 0 \\ \nabla_{(\mathbf{x},u)} h_{(k,a)}(\mathbf{x}, u)^\top \boldsymbol{w} & \leq 0 \end{cases} \tag{4.47}$$

Index every entry or sub-sequence of the vector with the corresponding variable (i.e. $u, x_{k,a}$)

and write the inner product of each inequality in a simpler fashion:

$$\begin{cases} -1 \cdot w_u + \nabla_{\mathbf{x}} h_b(\mathbf{x}, u)^\top \boldsymbol{w}_{\mathbf{x}} & < 0 \\ 0 \cdot w_u + \nabla_{\mathbf{x}} h_k(\mathbf{x}, u)^\top \boldsymbol{w}_{\mathbf{x}} & \leq 0 \\ 0 \cdot w_u + \nabla_{\mathbf{x}} h'_k(\mathbf{x}, u)^\top \boldsymbol{w}_{\mathbf{x}} & \leq 0 \\ 0 \cdot w_u + \nabla_{\mathbf{x}} h_{(k,a)}(\mathbf{x}, u)^\top \boldsymbol{w}_{\mathbf{x}} & \leq 0 \end{cases} \tag{4.48}$$

For any $\boldsymbol{w}^\top = [w_u \mid \boldsymbol{w}_{\mathbf{x}}^\top]$, s.t. $w_u > 0$ and $\boldsymbol{w}_{\mathbf{x}} = 0$, the AHU constraint qualification is satisfied.

Chapter 5

Experimental Evaluation

In this section, we lay down all the implantation details of the experimental setup and show the convergence of policies over time. We created a multi-agent team-vs-adversary zero-sum grid-world environment. The setting includes two players and an adversary coexisting in the same shared environment. The team members are assigned an objective that requires collaboration (reaching two different locations on the map at the same time). Every time they complete the task, positive and equal rewards are distributed to the team members, and simultaneously a negative reward is experienced by the adversary. The sum of all rewards is always zero; hence, the game is zero-sum. The adversary's objective is not to let the team win by reaching any target in the grid-world. In that case, the adversary wins, and the team members get punished with a negative reward equal in magnitude to the reward experienced by the adversary. Code snippets of this setting are publicly available on GitHub. We implemented Theorem 1 on a c2-standard-4 machine type CPU platform on Google Cloud ~ 3.5 days.

5.1 Multi-Agent Environment

OpenAI standardized reinforcement learning environment creation in 2016 with the release of what is known as the OpenAI Gym [9]. Gym is a python library used to decouple environments from algorithms and help the research community test different algorithms in different environments. We created a gym-compatible multi-agent reinforcement learning environment to run our decentralized algorithms and support the theoretical findings of our work. A Gym environment requires instantiating the observation and action space upon creation. Both are pictorially shown below:

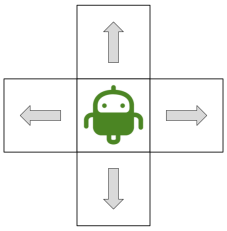

Figure 5.1: Action space of each agent

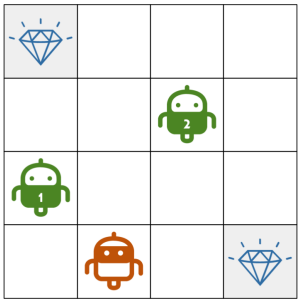

Figure 5.2: Environment sketch.

Game In a 4×4 grid-world environment, we place 2 agents and an adversary. The agents and the adversary can move in all cardinal directions. The team's goal is to capture the 2 diamonds in the environment before the adversary has time to grab any. If the team successfully reaches both diamonds before the adversary, they are victorious. If the adversary finds a diamond before the agents achieve their goal, he wins the game.

States Each positional configuration of the agents on the grid maps to a distinct state in the MDP. Figure 5.2 shows a configuration of the agents in the grid as well as the 2 landmarks at the 2 opposite corners of the grid. Each state has an encoding of the coordinates of the team and the adversary.

$$|\mathcal{S}| = 4^6$$

The size of the state space is 4^6 as we have to save 3 pairs of coordinates, and each coordinate can take discrete values from $[0-3]$.

Actions Each agent has 4 actions mapping to all cardinal directions. The agents take actions simultaneously, and transitions happen synchronously. Actions are passed into the environment as a list of actions containing all players' decisions.

Reward A reward signal of $+1$ is distributed to the team if every member covers a landmark. If this happens before the adversary reaches any landmark, the adversary is rewarded with -1. However, if the adversary reaches any landmark before all team members complete their objective, he receives a reward of $+1$, and the team experiences a reward of -1.

Nash Equilibria To show convergence, we plot the Frobenius norm of consecutive joint policies and the average estimated value of the team. Below are some screenshots of agents in a multi-grid environment demonstrating the behavior.

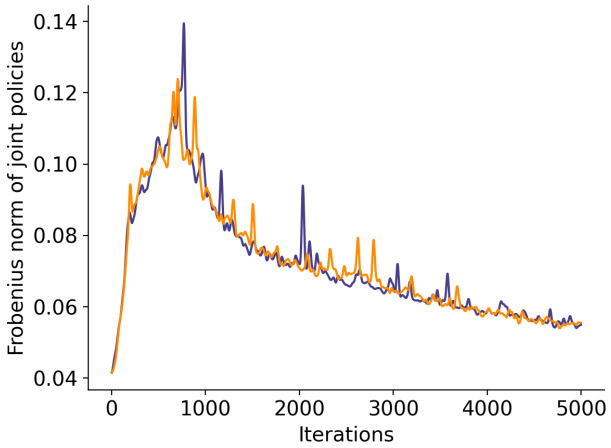

Figure 5.3: We plot the Frobenius norm of the joint policies over time.

The Frobenius norm decreases over time, demonstrating that the agents approach Nash equilibrium. Next, we provide some sequences of screenshots from roll-out trajectories after

training.

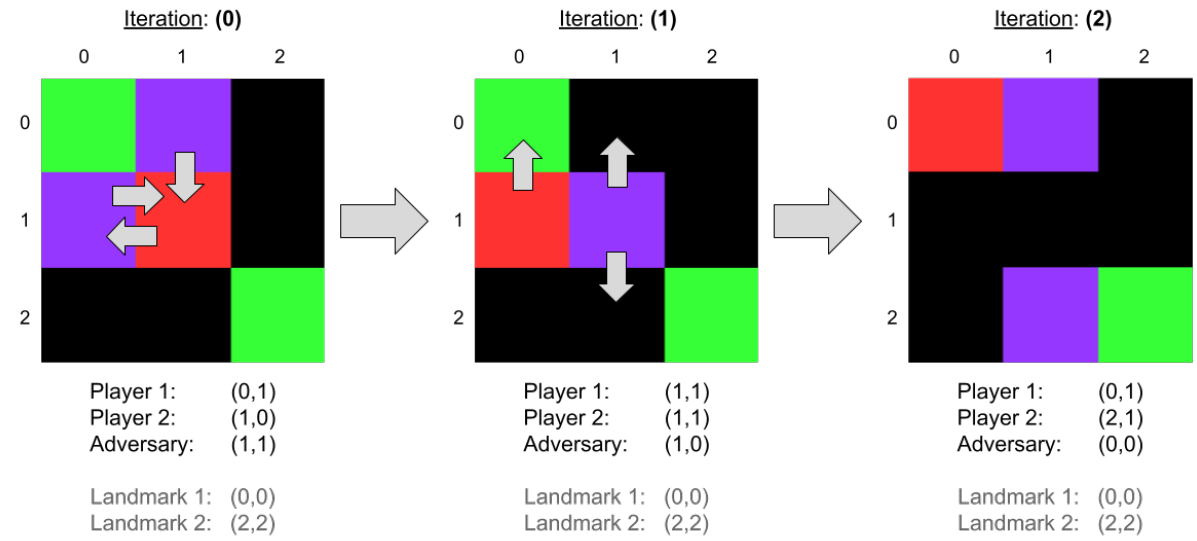

Figure 5.4: A replay after training. Agents (in purple) in iteration 0 both go towards the middle, showing that they are unaware of the actions of their team-mate and can't decide (due to equal distance to the bottom-right target) on how to coordinate.

Converging to a Nash equilibrium is a restrictive class of equilibria where correlated policies are not allowed. The agents in iteration 0 cannot coordinate and separate tasks optimally. They converge to independent probability distributions of actions. Sometimes they break ties correctly, and sometimes they choose to go towards the same target since they are unaware of what their team-mate will do in tie-breaking situations.

5.2 Discussion

In the above toy experiment, we see how agents implementing projected policy gradient in tandem, followed by the best response from the adversary, asymptotically leads to all players reaching the Nash equilibrium. Experiments were also conducted for 4×4 and 5×5 grids, as well as environments with three players in the team. Many GIF files with replays are available online, showcasing Nash-equilibrium type behavior. Due to the nested loop

in the algorithm, computation demand grows significantly with every player added to the environment. At every iterate of policy gradient update from the team, the adversary. Speed of convergence decreases, but asymptotic behavior is still guaranteed with any number of players in the same team by Theorem 1.

Chapter 6

Concluding Remarks

This thesis shows the existence of Nash equilibria in settings where multiple players try to collaborate, given the presence of an adversary that is against them. A decentralized algorithm that provably converges to Nash equilibria was designed to compute Nash equilibria in such settings. The results are also experimentally tested and empirically confirmed. This setting is the first in which decentralized algorithms provably converge to Nash equilibria. This positive result is also the first step toward designing algorithms that provably converge to near-equilibrium policies in non-cooperative two-team zero-sum stochastic games that simultaneously contain cooperative and adversarial elements.

Bibliography

[1] A. Agarwal, N. Jiang, and S. M. Kakade. Reinforcement learning: Theory and algorithms. 2019.

[2] A. Agarwal, S. M. Kakade, J. D. Lee, and G. Mahajan. On the theory of policy gradient methods: Optimality, approximation, and distribution shift, 2019.

[3] A. Agarwal, S. M. Kakade, J. D. Lee, and G. Mahajan. Optimality and approximation with policy gradient methods in markov decision processes. In *Conference on Learning Theory, COLT 2020, 9-12 July 2020*, volume 125 of *Proceedings of Machine Learning Research*, pages 64–66. PMLR, 2020.

[4] K. J. Arrow, L. Hurwicz, and H. Uzawa. Constraint qualifications in maximization problems. *Naval Research Logistics Quarterly*, 8(2):175–191, 1961.

[5] R. Aumann. Subjectivity and correlation in randomized strategies. *Journal of Mathematical Economics*, 1:67–96, 1974.

[6] Y. Bai and C. Jin. Provable self-play algorithms for competitive reinforcement learning, 2020.

[7] D. P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 2nd edition, 2000.

[8] J. Bhandari and D. Russo. Global optimality guarantees for policy gradient methods. *arXiv preprint arXiv:1906.01786*, 2019.

[9] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

[10] N. Brown and T. Sandholm. Superhuman ai for multiplayer poker. *Science*, 365(6456):885–890, 2019.

[11] 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)*, 38(2):156–172, 2008.

[12] S. Cen, Y. Wei, and Y. Chi. Fast policy extragradient methods for competitive games with entropy regularization. In *Advances in Neural Information Processing Systems 34:*

Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, pages 27952–27964, 2021.

[13] X. Chen, X. Deng, and S.-H. Teng. Settling the complexity of computing two-player nash equilibria, 2007.

[14] Y. Chen and M. Wang. Lower bound on the computational complexity of discounted markov decision problems, 2017.

[15] G. Christodoulou. *Price of Anarchy*, pages 665–667. Springer US, Boston, MA, 2008.

[16] C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, AAAI '98/IAAI '98, page 746–752, USA, 1998. American Association for Artificial Intelligence.

[17] A. Condon. On algorithms for simple stochastic games. In *Advances In Computational Complexity Theory*, 1990.

[18] C. Daskalakis, D. J. Foster, and N. Golowich. Independent policy gradient methods for competitive reinforcement learning. *Advances in neural information processing systems*, 33:5527–5540, 2020.

[19] C. Daskalakis, P. W. Goldberg, and C. H. Papadimitriou. The complexity of computing a nash equilibrium. *SIAM J. Comput.*, 39(1):195–259, 2009.

[20] C. Daskalakis, N. Golowich, and K. Zhang. The complexity of markov equilibrium in stochastic games. *arXiv preprint arXiv:2204.03991*, 2022.

[21] D. Ding, C.-Y. Wei, K. Zhang, and M. R. Jovanović. Independent policy gradient for large-scale markov potential games: Sharper rates, function approximation, and game-agnostic convergence. *arXiv preprint arXiv:2202.04129*, 2022.

[22] K. Etessami and M. Yannakakis. On the complexity of nash equilibria and other fixed points. *SIAM J. Comput.*, 39(6):2531–2597, 2010.

[23] J. Heinrich and D. Silver. Deep reinforcement learning from self-play in imperfect-information games, 2016.

[24] P. Hernandez-Leal, M. Kaisers, T. Baarslag, and E. M. de Cote. A survey of learning in multiagent environments: Dealing with non-stationarity, 2017.

[25] J. Hu and M. P. Wellman. Nash q-learning for general-sum stochastic games. *J. Mach. Learn. Res.*, 4(null):1039–1069, dec 2003.

[26] T. Jaakkola, M. I. Jordan, and S. P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6(6):1185–1201, 1994.

[27] C. Jin, Z. Allen-Zhu, S. Bubeck, and M. I. Jordan. Is q-learning provably efficient? In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018*, pages 4868–4878, 2018.

[28] C. Jin, Q. Liu, Y. Wang, and T. Yu. V-learning–a simple, efficient, decentralized algorithm for multiagent rl. *arXiv preprint arXiv:2110.14555*, 2021.

[29] Y. Jin, V. Muthukumar, and A. Sidford. The complexity of infinite-horizon general-sum stochastic games, 2022.

[30] A. Johansen. Monte carlo methods. In P. Peterson, E. Baker, and B. McGaw, editors, *International Encyclopedia of Education (Third Edition)*, pages 296–303. Elsevier, Oxford, third edition edition, 2010.

[31] J. M. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Zídek, A. Potapenko, A. Bridgland, C. Meyer, S. A. A. Kohl, A. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. A. Reiman, E. Clancy, M. Zielinski, M. Steinegger, M. Pacholska, T. Berghammer, S. Bodenstein, D. Silver, O. Vinyals, A. W. Senior, K. Kavukcuoglu, P. Kohli, and D. Hassabis. Highly accurate protein structure prediction with alphafold. *Nature*, 596:583 – 589, 2021.

[32] M. Lanctot, V. Zambaldi, A. Gruslys, A. Lazaridou, K. Tuyls, J. Perolat, D. Silver, and T. Graepel. A unified game-theoretic approach to multiagent reinforcement learning, 2017.

[33] S. Leonardos, W. Overman, I. Panageas, and G. Piliouras. Global convergence of multiagent policy gradient in markov potential games, 2021.

[34] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies, 2015.

[35] Y. Li, R. Wang, and L. F. Yang. Settling the horizon-dependence of sample complexity in reinforcement learning. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021*, pages 965–976. IEEE, 2021.

[36] T. Lin, C. Jin, and M. Jordan. On gradient descent ascent for nonconvex-concave minimax problems. In *International Conference on Machine Learning*, pages 6083–6093. PMLR, 2020.

[37] M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In W. W. Cohen and H. Hirsh, editors, *Machine Learning Proceedings 1994*, pages 157–163. Morgan Kaufmann, San Francisco (CA), 1994.

[38] Y. Luo, H. Xu, Y. Li, Y. Tian, T. Darrell, and T. Ma. Algorithmic framework for model-based deep reinforcement learning with theoretical guarantees. In *7th International Conference on Learning Representations, ICLR 2019*. OpenReview.net, 2019.

[39] A. A. Markov. The theory of algorithms. *Journal of Symbolic Logic*, 18(4):340–341, 1953.

[40] L. Matignon, G. J. Laurent, and N. Le Fort-Piat. Independent reinforcement learners in cooperative Markov games: a survey regarding coordination problems. *Knowledge Engineering Review*, 27(1):1–31, Mar. 2012.

[41] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. 2013. cite arxiv:1312.5602Comment: NIPS Deep Learning Workshop 2013.

[42] H. Moulin and J.-P. Vial. Strategically zero-sum games: The class of games whose completely mixed equilibria cannot be improved upon. *International Journal of Game Theory*, 7:201–221, 1978.

[43] A. Neyman and S. Sorin. *Stochastic Games and Applications*. Kluwer Academic Publishers, nato asi series edition, 2003.

[44] OpenAI. Openai five. `https://blog.openai.com/openai-five/`, 2018.

[45] OpenAI. Dota 2 with large scale deep reinforcement learning, 2019.

[46] R. T. Rockafellar. Lagrange multipliers and optimality. *SIAM review*, 35(2):183–238, 1993.

[47] A. Rubinstein. Settling the complexity of computing approximate two-player nash equilibria, 2016.

[48] M. Sayin, K. Zhang, D. Leslie, T. Basar, and A. Ozdaglar. Decentralized q-learning in zero-sum markov games. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 18320–18334. Curran Associates, Inc., 2021.

[49] M. O. Sayin, F. Parise, and A. Ozdaglar. Fictitious play in zero-sum stochastic games. *arXiv preprint arXiv:2010.04223*, 2020.

[50] L. S. Shapley. Stochastic games. *Proceedings of the national academy of sciences*, 39(10):1095–1100, 1953.

[51] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[52] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, 2017.

[53] D. Silver, S. Singh, D. Precup, and R. S. Sutton. Reward is enough. *Artificial Intelligence*, 299:103535, 2021.

[54] E. Solan and N. Vieille. Stochastic games. *Proceedings of the National Academy of Sciences*, 112(45):13743–13746, 2015.

[55] E. Solan and N. Vieille. Stochastic games. *Proceedings of the National Academy of Sciences*, 112(45):13743–13746, 2015.

[56] S. Stavroulakis and B. Sengupta. Reinforcement learning for location-aware scheduling, 2022.

[57] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[58] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, page 1057–1063, Cambridge, MA, USA, 1999. MIT Press.

[59] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, J. Quan, S. Gaffney, S. Petersen, K. Simonyan, T. Schaul, H. van Hasselt, D. Silver, T. Lillicrap, K. Calderone, P. Keet, A. Brunasso, D. Lawrence, A. Ekermo, J. Repp, and R. Tsing. Starcraft ii: A new challenge for reinforcement learning, 2017.

[60] B. Von Stengel and D. Koller. Team-maxmin equilibria. *Games and Economic Behavior*, 21(1-2):309–321, 1997.

[61] O. J. Vrieze. Stochastic games with finite state and action spaces. *CWI tracts*, 1987.

[62] X. Wang and T. Sandholm. Reinforcement learning to play an optimal nash equilibrium in team markov games. In *Proceedings of the 15th International Conference on Neural Information Processing Systems*, NIPS'02, page 1603–1610, Cambridge, MA, USA, 2002. MIT Press.

[63] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.

[64] C.-Y. Wei, Y.-T. Hong, and C.-J. Lu. Online reinforcement learning in stochastic games. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[65] C.-Y. Wei, C.-W. Lee, M. Zhang, and H. Luo. Last-iterate convergence of decentralized optimistic gradient descent/ascent in infinite-horizon competitive markov games. In M. Belkin and S. Kpotufe, editors, *Proceedings of Thirty Fourth Conference on Learning Theory*, volume 134 of *Proceedings of Machine Learning Research*, pages 4259–4299. PMLR, 15–19 Aug 2021.

[66] L. Xiao. On the convergence rates of policy gradient methods. *arXiv preprint arXiv:2201.07443*, 2022.

[67] Q. Xie, Y. Chen, Z. Wang, and Z. Yang. Learning zero-sum simultaneous-move markov games using function approximation and correlated equilibrium, 2020.

[68] Y. Ye. A new complexity result on solving the markov decision problem. *Mathematics of Operations Research*, 30(3):733–749, 2005.

[69] K. Zhang, Z. Yang, and T. Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms, 2019.

[70] R. Zhang, Z. Ren, and N. Li. Gradient play in stochastic games: stationary points, convergence, and sample complexity. *arXiv preprint arXiv:2106.00198*, 2021.

[71] M. Zinkevich, A. Greenwald, and M. Littman. Cyclic equilibria in markov games. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems*, volume 18. MIT Press, 2005.