Fictitious Mean-field Reinforcement Learning for Distributed Load Balancing

by

Fatemeh Fardno

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2022

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

In this work, we study the application of multi-agent reinforcement learning (RL) in distributed systems. In particular, we consider a setting in which strategic clients compete over a set of heterogeneous servers. Each client receives jobs at a fixed rate. For each job, clients choose a server to run the job. The objective of each client is to minimize its average wait time. We model this setting as a Markov game and theoretically prove that the game becomes in the limit a Markov potential game (MPG). We further propose a novel mean-field reinforcement learning algorithm, combining mean-field Q-learning and fictitious play. Through rigorous experiments, we show that our algorithm outperforms naive deployment of single-agent RL, and in some cases, performs comparably to the Nash Q-learning [61], while being less complex in terms of memory and computation. We also empirically analyze the convergence of our proposed algorithm to a Nash equilibrium and study its performance in four benchmark examples.

Acknowledgements

I would like to express my gratitude to my supervisor, Dr. Seyed Majid Zahedi, who guided me throughout this project. The completion of this study could not have been possible without his expertise.

I wish to extend my special thanks to Professor Wojciech Golab and Professor Mark Crowley for taking the time to read my thesis.

Lastly, I wish to show my appreciation to my family. I am forever thankful for their unconditional love and support throughout this entire process.

Dedication

This thesis is dedicated to my parents, for their endless love and support.

Table of Contents

List of Figures

Chapter 1

Introduction

Uncoordinated load balancing is challenging when a set of strategic agents compete over a set of shared resources. In game theory literature, this is often modeled as congestion games [84, 57]. In congestion games, each agent's utility depends on the resources it chooses and the number of other agents using that resource. If all agents choose the most efficient resource, that resource becomes congested and all agents suffer. As a result, agents might have preference to choose less capable resources to avoid congested ones. Another tool used in game theory to model uncoordinated competition over shared resources is the multi-agent, multi-armed bandit games [105, 118]. In such games, a set of independent decision-makers sequentially choose among a group of arms with the goal of maximizing their expected gain, without having any prior knowledge about the properties of each arm.

A critical limitation of the aforementioned models is the assumption that subsequent rounds of these games are independent. In other words, the outcome of these games at any given round has no effect on the outcome of future rounds [38]. However, there is a carryover effect between rounds in queuing systems. The carryover effect is caused by

the jobs remaining in the queues. In fact, the average wait time of each queue at round r depends not only on the number of tasks sent to it at r but also on the number of jobs stored on the queue from previous rounds.

Krishnasamy et al. [73] study a variant of the multi-armed bandit problem with the carryover effect which is suitable for queuing applications. They consider a bandit setting where jobs queue for service, and service rates of different queues are unknown. The authors study a queuing system with a single queue and K servers. Arrivals to the queue and service offered by servers are according to a Bernoulli distribution. At any given time, the queue is served by at most one server and the problem is to schedule a server at every time slot. They further evaluate the performance of different scheduling policies against the no-regret[1] policy.

Gaitonde and Tardos (2020) [38] study the multi-agent version of the queuing system in [73]. They consider a system where each client has a queue and servers process jobs with a fixed time-independent probability. All unprocessed jobs are then sent back to their respective queues. In settings with centralized scheduler, queues remain stable if the sum of arrival rates is less than the sum of service rates. However, this condition does not guarantee the stability of queues in settings with self-interested and independent clients. Gaitonde and Tardos (2020) [38] show that if the capacity of servers is high enough[2], and agents use no-regret learning algorithms, then the system remains stable. In a follow up work [39], the same authors show that if agents choose strategies that maximize their long-run success rate, the extra capacity can be improved[3].

Nash equilibrium always exists for finite-space infinite-horizon discounted stochastic

[1]The policy that schedules the optimal server in every time slot.

[2]To allow a centralized coordinator to get all jobs done even when the job arrival rate is doubled.

[3]The stability can be guaranteed even with $\frac{e}{e-1} \approx 1.58$ extra capacity.

games [30]. However, finding a Nash equilibrium is challenging [21, 117, 23]. To address this challenge, prior works have used machine learning [76, 34, 80, 61, 9, 46, 98]. Machine learning is successfully used to find the optimal policy in single-agent environments. For instance, Q-learning converges to the optimal Q-value function almost surely under some certain conditions [123, 122, 20]. However, learning in multi-agent environments is complicated as agents not only interact with the environment, but also with each other [14]. A key challenge in applying single-agent RL algorithms in such settings is that multiple agents learn concurrently, causing the environment faced by each one of them to be non-stationary. This invalidates the stationarity assumption for the convergence of single-agent RL algorithms.

In recent years, there has been a growing interest in extending single-agent RL to multi-agent RL (MARL) [34, 76, 129, 114, 113]. However, most MARL algorithms do not scale when there many agents in the game [128, 34, 61]. Therefore, their application is limited. This challenge is addressed in game theory by modelling games as mean-field games (MFG). In such games, each agent's effect on the overall environment is considered to be negligible. However, the interaction with other agents can be captured by the average action, or the empirical distribution over actions. Therefore, each agent only needs to find the best response to the mean-field action rather than tracking actions of all other agents.

Mean-field reinforcement learning is a method to learn Nash equilibrium in MFGs using multi-agent reinforcement learning. In this approach, interactions within the population of agents are approximated by those between a single agent and the average effect from the overall population or neighboring agents [126, 109]. In such algorithms, each agent only considers the average effect of the overall population or the neighboring agents. However, this approach requires each agent to track the neighboring agents' policies, which can also be intractable in the case of a large population of agents. In order to solve this issue, we

combine mean-field reinforcement learning algorithm in [126] with fictitious play. Fictitious play is a game in which the empirical distribution of other agents' actions is considered to be the belief about their mixed strategy. However, using mean-field reinforcement learning is not necessary in potential games. In recent years, Leonardos et al. [76] and Macua et al. [80] show the convergence of naive development of single-agent RL algorithms in potential games. We aim to improve the state-of-the-art and also extend the related work by adding fictitious play to the learning process.

In this work, we study the application of multi-agent reinforcement learning (RL) in distributed systems. In particular, we consider the setting in which a set of strategic clients compete over a set of servers. Each server has a unique service rate. Clients have probabilistic job arrivals[4] and each job has a probabilistic size[5]. Clients select a server for each new job. In each server, jobs are queued to be served. At any given time, the wait time of the job queue on each server depends on its queue lenght and service rate. The objective of each client is to optimally distribute its jobs among servers to minimize its average latency.

We aim to model our setting as an infinitely repeated game, where the system has a global[6] state. State is a vector containing the number of jobs held by each server (length of queues). State changes as agents take actions, and the reward of each agent depends not only on the joint action but also on the state. Our setting is a dynamic game with multiple agents; therefore, we can not use congestion games or bandit models, as they have no notion of state. We model this setting as a Markov game and theoretically prove that when the number of clients goes to infinity, the game becomes a Markov potential game

[4] At each round, each client receives a new job with a fixed, time-independent probability.

[5] Each job size comes from a geometric distribution with a fixed, time-independent parameter.

[6] All agents have access to the state.

(MPG). We also study the use of machine learning in such systems and propose a novel reinforcement learning algorithm to find the game's Nash equilibria.

To the best of our knowledge, this is the first work to combine mean-field reinforcement learning and fictitious play. Through rigorous experiments, we show that our algorithm outperforms naive development of single-agent RL. In some cases, our algorithm performs comparably to the Nash Q-learning [61], while being significantly less complex in terms of memory and computation. We also empirically analyze the convergence of our proposed algorithm to Nash equilibrium and study its performance in four benchmark examples[7].

Through experiments, we present some scenarios that adding fictitious play to other RL algorithms such as projected stochastic gradient ascent (PSGA) [76] and actor-critic (AC) improves their performance by 167% and 550% on average, respectively. In short, our contributions are as follows:

- We prove that our game is a Markov potential game when the number of clients goes to infinity (Section 4).
- We develop a novel reinforcement learning algorithm, combining mean-field reinforcement learning and fictitious play (Section 5.2).
- We implement a simulation environment where we can compare the performance of different learning algorithms for arbitrary scenarios (Section 6).
- Using simulation, we suggest that our proposed algorithm outperforms the naive development of single-agent RL by 307% on average and up to 1900% (Section 6).

[7] All examples are designed to be Markov potential games.

Chapter 2

Background

In this chapter, we provide some background on the topics related to our research. We begin by introducing stochastic games. Subsequently, we review potential games and then move to Q-learning. In addition, we discuss regret minimization. We then review multi-agent reinforcement learning and lastly, we discuss mean-field Q-learning.

2.1 Stochastic Games

Markov decision process (MDP) [60] is a mathematical concept for discrete-time stochastic processes. It provides a framework for studying decision-making under uncertainty, i.e., situations where the outcome not only depends on the decision maker's action, but on the environment as well:

Definition 2.1.1. ***Markov decision process (MDP)*** *[61]. A Markov Decision Process is a tuple* $<\mathcal{S}, \mathcal{A}, r, p>$*, where* $\mathcal{S}$ *is the discrete state space,* $\mathcal{A}$ *is the discrete action space,*

$r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function of the agent, and $p : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the transition function, where $\Delta(\mathcal{S})$ is the set of probability distributions over state space $\mathcal{S}$.

In an MDP, the agent acts according to a policy. In general, a policy is the representation of agent's behavior. A stationary policy is a policy that depends only on the current state of the agent. Stationary policies can either be deterministic or stochastic. A deterministic, stationary policy $\pi : \mathcal{S} \to \mathcal{A}$, determines the action of the agent at each state, i.e., $\pi(s) = a \in \mathcal{A}$, while a stochastic, stationary policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$, specifies a probability distribution over the action space for each state $s \in \mathcal{S}$. In this case, the action of the agent at time t, comes from a probability distribution that depends on the agent's current state, i.e., $a_t \sim \pi(s_t)$. A strategy can also be a behavioral strategy, in which the agent's decision may depend on the game's history.

Stochastic games[1] extend MDPs to the case where there are multiple agents interacting with each other. It is a framework for studying multi-agent systems:

Definition 2.1.2. ***Stochastic game*** *[116]. An n-agent stochastic game Γ is a tuple $<S, A^1, ..., A^n, r^1, ..., r^n, p>$, where S is the state space, A^i is the action space of agent $i (i = 1, ..., n)$, $r^i : S \times A^1 \times ... \times A^n \to \mathbb{R}$ is the payoff function for agent i, $p : S \times A^1 \times ... \times A^n \to \Delta(S)$ is the transition probability map, where $\Delta(S)$ is the set of probability distributions over state space S.*

At each state s, agents independently and simultaneously take actions $a^1, ...a^n$ and receive rewards $r^i(s, a^1, ...a^n), i = 1, ..., n$[2]. The state then transitions to a new state s', with probability $p(s'|s, a^1, ...a^n)$, where $\sum_{s' \in S} p(s'|s, a^1, ..., a^n) = 1$.

The nature of agents' interaction can be cooperative[3], competitive, or mixed. The

[1] Also called Markov games.

[2] Note that the reward of each agent is a function of state and joint action of all agents.

[3] Also known as collaborative.

cooperative setting, which is also the most studied one, assumes that all agents' actions improve the collective reward, also referred to as social welfare. Team games, i.e., games in which all agents receive the same reward, are an example of cooperative games. On the other hand, in a competitive setting, agents compete with each other. In this case, each agent's objective is to maximize its own utility function, which is not necessarily beneficial for the social welfare. Zero-sum games, i.e., games in which the sum of utilities of all agents is zero, are purely competitive games. In the case of two-agent zero-sum game, if an agent wins, the other one loses, and the net change in utilities is zero. Constant-sum games, i.e., games where the combined utilities of the agents are constant, are another example of purely competitive games. There are games that do not belong to any of the above categories, such as general-sum games, which are neither competitive nor cooperative.

In game theory, agents are considered to be self-interested. In stochastic games, each agent's objective is to maximize its own discounted expected sum of rewards. We can define the exact value function as:

$$v^i(s, \pi^1, ..., \pi^n) = \sum_{t=0}^{\infty} \gamma^t \mathbb{E}(r_t^i | \pi^1, ..., \pi^n, s_0 = s), \tag{2.1}$$

where r_t^i is the instantaneous reward of agent i at time t, γ is the discount factor[4], s is the initial state, and π^j, $j = 1, ..., n$ is the policy of agent j. The expectation is taken over random policies and rewards and state transitions. Given the definition of v, the concept of Nash equilibrium can be defined. A Nash equilibrium is a joint strategy where each agent's strategy is the best response to others'. In game theory, the best response is the strategy (or strategies) which produces the optimal outcome for an agent, given other agents' strategies [8].

Definition 2.1.3. ***Nash equilibrium*** *[61]. In stochastic game Γ, a Nash equilibrium*

[4]The discount factor determines how much agents care about future rewards.

point is a tuple of n strategies $(\pi_*^1, ...\pi_*^n)$ *such that for all* $s \in S$ *and* $i = 1, ..., n$,

$$v^i(s, \pi_*^1, ..., \pi_*^n) \geq v^i(s, \pi_*^1, ..., \pi_*^{i-1}, \pi^i, \pi_*^{i+1}, ..., \pi_*^n),$$

for all $\pi^i \in \Pi^i$, *where* Π^i *is the set of strategies available to agent* i.

All stochastic games have at least one Nash equilibrium in stationary strategies [31]. We write $\pi = \prod_{i \in \mathcal{N}} \pi^i$ and $\pi^{-i} = \prod_{j \neq i} \pi^j$ to denote the joint strategies of all agents and strategies of all agents except agent i, respectively. Similarly, a joint strategy $\pi_* = (\pi_*^i)_{i \in \mathcal{N}}$ is an $\epsilon-$Nash policy if there exists an $\epsilon > 0$ such that for each agent i:

$$v^i(s, \pi_*^i, \pi_*^{-i}) \geq v^i(s, \pi^i, \pi_*^{-i}) - \epsilon, \tag{2.2}$$

for all $\pi^i \in \Pi^i$ and for all $s \in S$.

We next define Markov potential games (MPGs).

2.2 Markov Potential Games

Monderer and Shapley (1996) [92] formally define a class of games called potential games. In such games, there exists a global[5] function called potential function such that if any agent changes its policy unilaterally, the change in its reward equals the change in the potential function. Although potential games are not necessarily cooperative, they embrace a notion of cooperation, as all agents' utilities are aligned with the shared potential function:

Definition 2.2.1. ***Potential game*** *[92]. Let* $\Gamma(u^1, u^2, ...u^n)$ *be a game in strategic form with finite number of agents, where* $\mathcal{N} = \{1, 2, ..., n\}$ *is the set of agents,* Y^i *is the set of*

[5]Shared by all agents.

strategies of agent i, $u^i : Y \to \mathbb{R}$ *is the payoff function of agent* i *where* $Y = Y^1 \times Y^2 \times ... \times Y^n$ *is the set of strategy profiles. A function* $\phi : Y \to \mathbb{R}$ *is a potential function for* Γ*, if for every* $i \in \mathcal{N}$ *and for every* $y^{-i} \in Y^{-i}$*:*

$$u^i(y^{-i}, x) - u^i(y^{-i}, z) = \phi(y^{-i}, x) - \phi(y^{-i}, z), \quad \textit{for all } x, z \in Y^i$$

A trivial class of potential games is team games, i.e., games where all agents receive the same reward. In this case, the reward function itself is the potential function. Although Shapley defines potential games for stateless games, Leonardos et al. (2021) [76] introduce an extension of potential games to Markov games called Markov potential games (MPGs). Markov potential games are suitable frameworks for modeling sequential decision making.

Definition 2.2.2. ***Markov potential game*** *[76]. A Markov Decision Process (MDP),* $\mathcal{G}$*, is called a Markov Potential Game (MPG) if there exits a state-dependent function* $\phi_s : \Pi \to \mathbb{R}$ *for* $s \in \mathcal{S}$ *so that*

$$\phi_s(\pi_i, \pi_{-i}) - \phi_s(\pi_i', \pi_{-i}) = v^i(s, \pi_i, \pi_{-i}) - v^i(s, \pi_i', \pi_{-i}),$$

for all agents $i \in \mathcal{N}$*, all states* $s \in \mathcal{S}$ *and all policies* $\pi_i, \pi_i' \in \Pi_i, \pi_{-i} \in \Pi_{-i}$*.*

Monderer and Shapley (1996) [92] prove that all normal-form potential games have a deterministic Nash policy profile. Leonardos et al. (2021) [76] prove that all Markov potential games also have a deterministic Nash policy, i.e., there exists a Nash policy π^* such that for all agents $i \in \mathcal{N}$ and for all states $s \in \mathcal{S}$, there exists an action $a_i \in A_i$, so that $\pi_i^*(a_i|s) = 1$.

2.3 Q-learning

Each agent's objective in MDP (see 2.1.1) is to maximize the sum of its discounted expected rewards:

$$v^{\pi}(s) = \sum_{t=0}^{\infty} \gamma^t \mathbb{E}(r_t|\pi, s_0 = s),\ s \in \mathcal{S}, \tag{2.3}$$

where s is an arbitrary initial state, r_t is the instantaneous reward of the agent at time t, and γ is the discount factor. The discount factor determines how much agents care about future rewards, e.g., $\gamma = 0$ shows that they only care about the instantaneous reward.

In equation 2.3, $v^{\pi}(s)$ is the expected sum of the accumulated reward of agent when it starts from state s and follows policy π. This is also called the value function. It basically represents the value of state s under policy π. The optimal value function is defined as:

$$v^*(s) = \max_{\pi} v^{\pi}(s). \tag{2.4}$$

The agent's objective is to find the optimal policy π^*, i.e., $\pi^* = \arg\max_{\pi} v(s, \pi)$ for all arbitrary initial states $s \in \mathcal{S}$. The optimal policy can be found by an iterative search method to find the fixed point of the corresponding *Bellman* equation (see Appendix B.1) for π^*:

$$v^{\pi^*}(s) = \max_{a} \{r(s, a) + \gamma \sum_{s'} p(s'|s, a) v^{\pi^*}(s')\}, \tag{2.5}$$

where $p(s'|s, a)$ is the probability of transitioning to state s' from s, given action a. All MDPs have an optimal policy π^*, which is not necessarily unique [110]. All optimal policies achieve the optimal value function, i.e., $v^{\pi^*}(s) = v^*(s)$ [110].

We can similarly define the action-value function, also called Q-function as:

$$Q^{\pi}(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) v(s'). \tag{2.6}$$

$Q^\pi(s,a)$ is the total discounted expected reward of taking action a at state s and following policy π afterward. We can rewrite the value function in terms of the Q-function as:

$$v^\pi(s) = \mathbb{E}_{a\sim\pi(.|s)}[Q^\pi(s,a)] = \sum_{a\in\mathcal{A}} \pi(a|s)Q^\pi(s,a). \tag{2.7}$$

The state transition probabilities or the reward function are not always known. To address this challenge, model-free reinforcement learning can be used to learn the optimal policy directly without considering a model for the environment, i.e., without knowing either the state transition probabilities or the reward function. Q-learning is an example of such learning algorithm.

The basic idea behind Q-learning is that if we know the optimal Q-function for each action-state pair, $Q^*(s,a)$, we can simply find the optimal policy π^* by identifying the action that maximizes $Q^*(s,a)$ in each state. So the problem of finding the optimal policy boils down to the problem of finding $Q^*(s,a)$ for all action-state pairs. In the case of small state and action spaces, we can present the Q-function as a table, which is referred to as the tabular setting. Suppose the agent takes action a at state s, receives r as reward, and goes to s'. Given the sampled experience (s,a,r,s'), the vanilla Q-learning update is:

$$Q^{t+1}(s,a) = Q^t(s,a) + \alpha_t(s,a)(r + \gamma \max_{a'} Q^t(s',a') - Q^t(s,a)). \tag{2.8}$$

where α is the learning rate. See Appendix B.2 for further explanation.

Generally, reinforcement learning algorithms can be grouped into three main categories: value iteration, policy iteration, and policy gradient. We provide more information about these methods in Appendix B.3.

2.4 Regret Minimization

The main idea behind regret minimization algorithms is to learn the best fixed action in hindsight [64]. The average overall regret of agent i at time T is:

$$R_i^T = \frac{1}{T} \max_{\pi_i^*} \sum_{t=1}^{T} \left(r_i(\pi_i^*, \pi_{-i}^t) - r_i(\pi_i^t, \pi_{-i}^t) \right). \tag{2.9}$$

This measures the performance of an algorithm compared to the best hindsight static strategy. The objective of regret minimization algorithms is to find policies that minimize the overall regret.

We can also define regret in the context of queuing systems. Let Q_t be the queue length of a server in a queuing system at time t, and let Q_t^* be the corresponding queue length under a policy that always schedules the optimal server. Krishnasamy et al. [73] define the queue-regret as:

$$\psi(t) := \mathbb{E}[Q_t - Q_t^*]. \tag{2.10}$$

Regret minimization algorithms are guaranteed to converge to the equilibria of certain games [10, 51, 132]. EXP3 [7] and follow-the-leader (FTL) [52] are two examples of such learning algorithms.

2.5 Multi-agent Reinforcement Learning

Most reinforcement learning applications involve the participation of more than one agent. Multi-agent reinforcement learning (MARL) addresses sequential decision-making problems with more than one agent. In particular, it considers settings where the evolution of the state and the reward of each agent are influenced by the joint action of all agents. Stochastic games (see 2.1) are theoretical frameworks for MARL.

The naive development of single-agent RL algorithms, also called independent learning, may fail to converge in multi-agent environments [33]. Learning in multi-agent environments is highly nontrivial as agents are learning concurrently, causing the environment faced by each one of them to be non-stationary. Action taken by each one of the agents affects the reward of other agents and state transitions. This violates the stationarity assumption in single-agent reinforcement learning algorithms, an assumption that guarantees the convergence of such algorithms. However, there are some cases in which applying single-agent RL algorithms to multi-agent settings would converge to the set of Nash equilibria [76, 91, 37, 80].

To handle non-stationarity, each agent may need to account for the joint action in their learning process. Joint-action-learners [22], observe the action of all other agents. Each agent assumes that other agents are selecting actions according to a stationary policy. Therefore, each agent estimates other agents' policies from their actions. They then play optimally with respect to this learned estimate. Minimax-Q algorithms [78, 62, 61] observe both the actions and rewards of the all other agents and learn a Nash equilibrium explicitly. Such algorithms learn and play the equilibrium independent of the behavior of other agents. LOLA [34] is another instance of MARL algorithms. A LOLA agent optimizes the expected return after the rest of the population update their policy with one learning step, instead of optimizing the expected return under the current parameters.

The downside of the aforementioned algorithms is that each individual agent has to account for the joint action of all agents, whose dimension increases with the number of agents. One way to tackle the scalability issue with a massively large number of agents is to use the mean-field games (MFGs) models. In such games, each agent's effect on the overall environment is negligible. However, the interaction with other agents can be captured by the average action, or the empirical distribution over actions. Therefore, each agent only

needs to find the best response to the mean-field.

2.6 Mean-field Q-learning

Mean-field reinforcement learning is another method to tackle the multi-agent reinforcement learning problem with a large number of agents. In this approach, interactions within the population of agents are approximated by those between a single agent and the average effect from the overall population or neighboring agents [126]. Yang et al. (2018) [126] propose *mean-field Q-learning*, an approach which adds the mean action of neighboring agents to the Q-function of each agent. In this case, the Q-function of agent $i \in \mathcal{N}$ is written as $Q^i(s, a^i, a^{-i})$, where a^{-i} is the mean action of all neighboring agents and is calculated using the following equation:

$$\bar{a}^j = \frac{1}{N^j} \sum_k a^k, \; a^k \sim \pi_t^k(.|s, \hat{a}^k), \tag{2.11}$$

where N^j is the number of neighbors of agent j, and a^k is sampled from the policy of agent k, which depends on the state and previous mean action of agent k's neighbors. Policies of neighbouring agents, π_t^k, is parametrized by agents' previous actions, $\hat{a}^k$. The main assumption here is that the Q function can be approximated using only the pairwise local interactions:

$$Q^j(s, \mathbf{a}) = \frac{1}{N^j} \sum_{k \in \mathcal{N}(j)} Q^j(s, a^j, a^k), \tag{2.12}$$

where $\mathcal{N}(j)$ is the index set of neighboring agents of agent j.

Therefore, each agent has to track all its neighbors' policies in order to be able to calculate $\bar{a}^j$. This is a drawback of MF-Q, as keeping track of neighboring agents' policies can be very expensive, especially when number of agents are massively large. A solution

would be using fictitious action instead of mean action. We will explain this solution further in next chapters.

Chapter 3

Problem Formulation

We study the following discrete-time queuing system. Suppose $\mathcal{N} = \{1, ..., n\}$ is the set of heterogeneous clients, and $M = \{1, ..., m\}$ is the set of heterogeneous servers. Each server is characterized by its fixed processing rate μ_j, $j \in \{1, ..., m\}$. During each time step, each client $i \in N$ receives a new task with probability λ_i $(0 \leq \lambda_i \leq 1)$. Clients that receive a job choose a server to send their task to. Server j processes jobs at rate μ_j. Tasks are queued in servers and processed according to the first-come, first-served (FCFS) discipline, while ties are broken in favor of clients with a smaller id numbers.

All arrival times and task sizes are assumed to be independent of one another. In the case of constant task sizes, each server can be modeled as an M/D/1 queuing system, i.e., Poisson arrivals and deterministic task sizes. Each server can also be modeled as an M/M/1 queuing system, i.e., Poisson arrivals and geometric task sizes, if task sizes come from a geometric distribution, as it is the only memory-less discrete distribution.

As in [38], we can write Q_t^j as the number of unprocessed tasks at the beginning of time

step t in server j:

$$Q_{t+1}^j = Q_t^j + B_t^j - S_t^j, \tag{3.1}$$

where B_t^j is the number of tasks received by server j at time t, and S_t^j is the number of tasks served by server j at time t. Both S_t^j and B_t^j are random variables. Note that S_t^j is necessarily zero if $Q_t^j + B_t^j = 0$. We initialize $Q_0^j = 0$.

Definition 3.0.1. ***System stability*** *[38]. The above system is strongly stable under some given dynamics if, for any fixed $r \geq 0$, the random process Q_t^j satisfies $\mathbb{E}[(Q_t^j)^r] \leq C_r$ for some absolute constant C_r depending only on r and parameters $\lambda = \sum_{i=1}^n \lambda_i$ and $\mu = \sum_{j=1}^m \mu_i$, but not on t.*

In other words, the queuing system is strongly stable if all moments of the queue length of all servers are bounded. In order for the system to be strongly stable under a centralized scheduling policy, the total arrival rate must be less than the total processing rate of the system, i.e., $\lambda < \mu$ [38].

The problem faced by each client is how to distribute its jobs between servers to operate optimally. Each client should find the probability of sending their task to servers so that the expected execution time of its tasks are minimized.

We formulate the problem as a non-cooperative stochastic game. The game is non-cooperative, as each client acts in a selfish manner and minimizes the expected wait time for its own tasks. At each round, each client that has sent a task receives a reward proportional to the inverse of the expected wait time of the chosen server. In addition to rewards, clients also have access to a global state, which measures each server's queue length at the beginning of each time step. The game is stochastic[1], as state transitions are controlled by all agents' current state and action.

[1] Markovian.

We denote the state at round t by vector $s_t = (s_t^1, s_t^2, ..., s_t^m)$ and the state space by S, where $s_0 = (0, ..., 0)$.

At the beginning of the game, the action space of client i, $i \in \{1, .., n\}$ is $\mathcal{A}_i = \{e_1, ..., e_m, e_0\}$, where e_j is a vector in which all elements are zero except the i-th element, which is one. It represents the action of choosing server j, $j \in \{1, .., m\}$. The vector $e_0 = (0, ..., 0)$ represents no action when client i does not receive a task.

It is important to consider that state space size is proportional to the number of clients. There is a capacity on the maximum number of tasks in each server. Clients can no longer send their tasks to servers with full queues. These servers are not in the clients' action space until their queue length becomes less than the capacity. The state space size will therefore be $capacity^m$ since we have m servers, and each one of them can have a queue size of 0 to $capacity - 1$.

The reward received by client i when choosing server j at time t is:

$$R_i(a_i, a_{-i}, s) = \frac{1}{w_j(s, N_j(a))^2}\,, \tag{3.2}$$

where $w_j(s, k) \triangleq$ is the expected wait time on server j at state s, given that k tasks are sent to it, and $N_j(a) \triangleq$ is the number of tasks sent to server j given joint action a. We can also define the value function as:

$$v^i(s, \pi_i, \pi_{-i}) = \sum_{t=0}^{\infty} \gamma^t \mathbb{E}(r_t^i | \pi, s_0 = s),\ s \in \mathcal{S}, \tag{3.3}$$

where r_t^i is the instantaneous reward of agent i at time t and γ is the discount factor. The value function is the expected accumulative reward of agent i when starting from state s and following policy π_i, while other agents act according to π_{-i}. Note that all clients have the same reward function but do not receive the same reward. Furthermore, note

that each agent's reward, not only depends on its own action but on the action of all other agents as well. It also depends on the state of the system.

The action taken by one agent affects the reward of other agents and the evolution of the state. Therefore, each agent should account for how other agents behave and adapt to the collective behavior accordingly [128].

We can now formally define the stochastic game. The following notation is standard and mainly follows [76]. Using common conventions, we will write $X = |\mathcal{X}|$ and $\Delta(\mathcal{X})$ to denote all probability distributions' size and space over any set $\mathcal{X}$, respectively.

There are n agents who repeatedly choose actions in a Markov Decision Process (MDP). Each agent's goal is to maximize its value function $v^i(s, \pi_i, \pi_{-i})$. The n-agent stochastic game can then be defined as a tuple $\mathcal{G} = (\mathcal{S}, \mathcal{N}, \{\mathcal{A}_i, \mathcal{R}_i\}, P, \gamma)$, where:

- $\mathcal{S}$ is the finite state space with size $|\mathcal{S}| = capacity^m$. $\Delta(\mathcal{S})$ denotes the set of all probability distributions over states.
- $\mathcal{N} = \{1, ..., n\}$ is the set of agents.
- $\mathcal{A}_i$ is the finite action space for agent $i \in \mathcal{N}$, with generic elements $a_i \in \mathcal{A}_i$. Note that we can write $\mathcal{A} = \prod_{i \in \mathcal{N}} \mathcal{A}_i$ and $\mathcal{A}_{-i} = \prod_{j \neq i} \mathcal{A}_j$ to denote the joint action spaces of all agents and the joint action spaces of all agents except agent i with generic elements $\mathbf{a} = (a_i)_{i \in \mathcal{N}}$ and $\mathbf{a}_{-i} = (a_j)_{j \neq i}$, respectively.
- $\mathcal{R}_i : \mathcal{S} \times \mathcal{A} \to [0, 1]$ is the reward function of agent i, i.e., $\mathcal{R}_i(s, a_i, \mathbf{a}_{-i})$ is the instantaneous reward of agent i, when it takes action a_i, and other agents take $\mathbf{a}_{-i}$ at state s.
- P is the state transition probability function, i.e., $P(s'|s, \mathbf{a})$ is the probability of

transitioning from s to s', when all agents take the joint action $\mathbf{a}$. In our game, state transitions are deterministic, i.e., $P(s'|s, \mathbf{a}) \in \{0, 1\}$.

- γ is the discount factor for future rewards.

We can index all the above terms with time t. At each time-step $t \geq 0$, all agents observe the state s_t, simultaneously take the joint action $\mathbf{a}_t = (a_{1,t}, ..., a_{n,t})$ and receive the reward $\mathcal{R}_i(s_t, \mathbf{a}_t)$, $i \in \mathcal{N}$. The state then transitions to $s_{t+1} \sim P(.|s_t, \mathbf{a}_t)$.

Our goal is to solve the game $\mathcal{G}$ by finding the Nash equilibrium strategies. In the next chapter, we prove that $\mathcal{G}$ becomes a Markov potential game (MPG) in the limit. We then provide an algorithm to find the optimal Nash equilibrium.

Chapter 4

Potentiality Proof

In this chapter, we prove that our game becomes a Markov potential game (MPG) in the limit. A key challenge of learning in stochastic games is that multiple agents learn concurrently. The action taken by one agent not only affects the state's evolution but also impacts the reward of other agents. This causes the environment faced by each individual agent to become non-stationary, which violates the stationarity assumption for convergence of single-agent reinforcement algorithms. However, Leonardos et al. (2021) [76] prove that projected stochastic gradient ascent (PSGA) converges to ϵ-approximate Nash policy in Markov potential games. The intuition behind the proof is that running gradient ascent on each agent's value function is equivalent to running gradient ascent on the potential function. So instead of finding the ϵ-stationary point of all agents' value functions, all we have to do is to find the ϵ-stationary point of the potential function.

When a game is an MPG, running independent gradient ascent for each agent guarantees convergence to the Nash policy [76]. The only drawback of naively applying independent gradient ascent is that the number of steps to reach the ϵ-approximate Nash policy is

proportional to $\frac{1}{\epsilon^6}$ which makes the convergence too slow.

Not all games that are potential at each state are MPGs. According to [76], the sufficient conditions for games that are potential at each state to be MPGs are as follows:

- C1. *Agent-independent Transitions*: State transitions do not depend on the joint action, i.e., $p(s'|s, \mathbf{a}) = p(s'|s)$.
- C2. *Equality of Individual Dummy Terms*: $p(s'|s, \mathbf{a})$ is arbitrary but the dummy terms of all agents, i.e., $u_s^i(a_{-i})$, are equal across all the states, i.e., there exists a function $u_s^i : \mathcal{A}_{-i} \to \mathbb{R}$ such that $R_i(a_i, a_{-i}, s) = \phi_s(a_i, a_{-i}) + u_s^i(a_{-i})$, and

$$\nabla_{\pi_i(s)} \mathbb{E}_{\tau \sim \pi}[\sum_{k=0}^{T} \gamma^k u_{s_k}^i(a_{-i,k}) | s_0 = s] = c_s \mathbf{1}, \tag{4.1}$$

  for all states $s', s \in \mathcal{S}$, $c_s \in \mathbb{R}$ and $\mathbf{1} \in \mathbb{R}^{\mathcal{A}_i}$, where $\pi_i(s)$ is the policy of agent i at state s.

If either C1 or C2 are true, the game is an MPG [76].

The game described in Chapter 3 is potential at each state since we can write the individual reward function (3.2) as:

$$R_i(a_i, a_{-i}, s) = \phi_s(a_i, a_{-i}) + u_s^i(a_{-i}) \,, \tag{4.2}$$

where:

$$\phi_s(a_i, a_{-i}) \triangleq \sum_{j=1}^{m} \sum_{k=1}^{N_j(a)} \frac{1}{w_j(s,k)^2} \,, \; and \tag{4.3}$$

$$u_s^i(a_{-i}) \triangleq -\sum_{j=1}^{m} \sum_{k=1}^{N_j(a_{-i})} \frac{1}{w_j(s,k)^2} \,. \tag{4.4}$$

In 4.3 and 4.4, $N_j(a)$ and $N_j(a_{-i})$ are the number of tasks sent to server j given the joint actions a and a_{-i}[1], respectively. So if agent i deviates to a'_i:

$$\begin{aligned} R_i(a_i, a_{-i}, s) - R_i(a'_i, a_{-i}, s) &= \phi_s(a_i, a_{-i}) + u^i_s(a_{-i}) - \phi_s(a'_i, a_{-i}) - u^i_s(a_{-i}) \\ &= \phi_s(a_i, a_{-i}) - \phi_s(a'_i, a_{-i}). \end{aligned} \tag{4.5}$$

The distributed load-balancing game does not satisfy C1, as state transitions are clearly dependent on the joint action of all agents. However, we can show that as the number of agents goes to infinity, $\nabla_{\pi_i(s^*)}\mathbb{E}_{\tau\sim\pi}[\sum_{k=0}^{T}\gamma^k u^i_{s_k}(a_{-i,k})|s_0 = s]$[2] goes to zero. In this case, C2 holds for $c_s = 0$ and hence, the game is an MPG. We are now ready to prove that our game is a Markov potential game in the limit.

Assumption 1. *Each client has an arrival rate of* $\frac{\lambda}{n}$[3] *and since* $\lim_{n\to\infty} n(\frac{\lambda}{n}) = \lambda$*, it is theoretically possible to have a feasible system with infinite number of clients, as long as* $\sum_{j=1}^{m}\mu_j > \lambda$.

We want to show that $\nabla_{\pi_i(s^*)}\mathbb{E}_{\tau\sim\pi}[\sum_{k=0}^{T}\gamma^k u^i_{s_k}(a_{-i,k})|s_0 = s]$ goes to zero as n goes to infinity. For finite horizon T, we define $\mathbb{P}^\pi(s', s)$ to be the probability of transitioning from s to s' under policy π. We now define $v^i_t(s)$ for $0 \le t \le T$ as:

$$v^i_t(s) \triangleq \mathbb{E}_{\tau\sim\pi}[\sum_{k=0}^{t}\gamma^k u^i_{s_k}(a_{-i,k})|s_0 = s]^4. \tag{4.6}$$

where the expectation is taken over random policies. Using dynamic programming, we can rewrite $v^i_t(s)$ as:

$$v^i_t(s) = \mathbb{E}_{\tau\sim\pi}[u^i_s(a_{-i,0})] + \sum_{s'}\mathbb{P}^\pi(s', s)\,\mathbb{E}_{\tau\sim\pi}[\sum_{k=1}^{t}\gamma^k u^i_{s_k}(a_{-i,k})|s_0 = s']. \tag{4.7}$$

[1] Action of all agents except agent i.

[2] s and s^* are two arbitrary states.

[3] Note that n is the number of agents.

[4] The expectation is taken over random policies.

Since $u_s^i(a_{-i,0})$ is independent from $\pi_i(s)$, we have $\mathbb{E}_{\tau\sim\pi}[u_s^i(a_{-i,0})] = \mathbb{E}_{a_{-i}\sim\pi_{-i}(s)}[u_s^i(a_{-i,0})]$. We can also factorize a γ from the second term:

$$v_t^i(s) = \mathbb{E}_{a_{-i}\sim\pi_{-i}(s)}[u_s^i(a_{-i,0})] + \gamma \sum_{s'} \mathbb{P}^\pi(s', s)\mathbb{E}_{\tau\sim\pi}[\sum_{k=0}^{t-1} \gamma^t u_{s_k}^i(a_{-i,k})|s_0 = s'], \tag{4.8}$$

where according to 4.6, we have $v_{t-1}^i(s') = \mathbb{E}_{\tau\sim\pi}[\sum_{k=0}^{t-1} \gamma^k u_{s_k}^i(a_{-i,k})|s_0 = s']$. So we can write:

$$v_t^i(s) = \mathbb{E}_{a_{-i}\sim\pi_{-i}(s)}[u_s^i(a_{-i,0})] + \gamma \sum_{s'} \mathbb{P}^\pi(s', s)v_{t-1}^i(s'). \tag{4.9}$$

Our goal is to show that $\nabla_{\pi_i(s^*)}v_T^i(s)$ goes to zero as n goes to infinity. $\nabla_{\pi_i(s^*)}v_T^i(s)$ measures the change in $v_T^i(s)$, as agent i modifies its policy at state s^*. Intuitively, we expect $v_T^i(s)$ not to change much when there are infinite number of agents in the game. We can write $\nabla_{\pi_i(s^*)}v_t^i(s)$ by taking derivative of 4.9 as:

$$\nabla_{\pi_i(s^*)}v_t^i(s) = \nabla_{\pi_i(s^*)}\left(\mathbb{E}_{a_{-i}\sim\pi_{-i}(s)}[u_s^i(a_{-i,0})]\right) + \nabla_{\pi_i(s^*)}\left(\gamma \sum_{s'} \mathbb{P}^\pi(s', s)v_{t-1}^i(s')\right). \tag{4.10}$$

Since $\mathbb{E}_{a_{-i}\sim\pi_{-i}(s)}[u_s^i(a_{-i,0})]$ only depends on action of other agents, we have:

$$\nabla_{\pi_i(s^*)}\mathbb{E}_{a_{-i}\sim\pi_{-i}(s)}[u_s^i(a_{-i,0})] = 0. \tag{4.11}$$

Therefore, we can write:

$$\begin{aligned}\nabla_{\pi_i(s^*)}v_t^i(s) &= \nabla_{\pi_i(s^*)}\left(\gamma \sum_{s'} \mathbb{P}^\pi(s', s)v_{t-1}^i(s')\right) \\ &= \gamma \sum_{s'} \mathbb{P}^\pi(s', s)\nabla_{\pi_i(s^*)}v_{t-1}^i(s') + \gamma \sum_{s'} v_{t-1}^i(s')\nabla_{\pi_i(s^*)}\mathbb{P}^\pi(s', s).\end{aligned} \tag{4.12}$$

If we show that $\nabla_{\pi_i(s^*)}v_t^i(s)$ is bounded, and both bounds go to zero as n goes to infinity, we can conclude the main theorem 4.0.4. The following lemmas are essential for proving the main theorem. We postpone the proofs to Appendix C for more readability.

In order prove the main theorem 4.0.4, we first show in lemma 4.0.1 that $u_s^i(a_{-i})$ has a lower bound and therefore, $v_t^i(s)$ (4.6) is also lower bounded.

Lemma 4.0.1. *$u_s^i(a_{-i})$ (4.4) is lower bounded by $-\frac{2\pi^2}{3}\sum_{j=1}^m \mu_j^2$, where m is the number of servers and μ_j is the service rate of server j.*

Lemma 4.0.2 shows that $\nabla_{\pi_i(s^*)} v_t^i(s)$ is lower bounded by $\sum_{i=1}^t \gamma^i \frac{\lambda}{n} \hat{v}_{t-i}\mathbf{1}$, which goes to zero as $n \to \infty$. We prove this lemma using lemma 4.0.1 and proof by induction.

Lemma 4.0.2. *For all $s \in \mathcal{S}$, $\sum_{i=1}^t \gamma^i \frac{\lambda}{n} \hat{v}_{t-i}\boldsymbol{1} \leq \nabla_{\pi_i(s^*)} v_t^i(s)$ (4.12), where $\hat{v}_{t-i}$ is the minimum of $v_{t-i}^i(.)$ (4.9) over all states.*

In lemma 4.0.3, we use proof by induction to show that $\nabla_{\pi_i(s^*)} v_t^i(s)$ is upper bounded by zero (see the complete proof in Appendix C).

Lemma 4.0.3. *For all $s \in \mathcal{S}$, $\nabla_{\pi_i(s^*)} v_t^i(s)$ (4.12) is upper bounded by 0.*

We are now ready to prove the main theorem, which can be directly concluded from lemma 4.0.2 and 4.0.3.

Theorem 4.0.4. *For all $s \in \mathcal{S}$, $\nabla_{\pi_i(s^*)} v_T^i(s)$ (4.12) goes to zero as n goes to infinity.*

Proof. According to lemmas 4.0.2 and 4.0.3, $\sum_{i=1}^t \gamma^i \frac{\lambda}{n} \hat{v}_{t-i} 1 \leq \nabla_{\pi_i(s^*)} v_t^i(s) \leq 0$. As n goes to infinity, both lower and upper bounds of $\nabla_{\pi_i(s^*)} v_T^i(s)$ go to zero. Using squeeze (sandwich) theorem, we can conclude that $\nabla_{\pi_i(s^*)} v_t^i(s)$ goes to zero as well. □

Chapter 5

Algorithm

In this chapter, we propose our novel reinforcement-learning algorithm ,which is a combination of mean-field Q-learning and fictitious play.

5.1 MF-Q + FP

One drawback of MF-Q (see 2.6) is that keeping track of neighboring agents' policies can be very expensive, especially when the number of agents is large. To address this challenge we combine fictitious play (FP) with MF-Q. Fictitious play is a process in which all agents take the empirical distribution of other agents' actions as a belief about their mixed strategies. In fictitious play, each agent plays the best response in accordance to their belief about other agents' strategies.

Strategies are first initialized arbitrarily at $t = 0$, and then agents use the following rule to update the average strategy at time t subsequently [40]:

$$f_i^t = \frac{(t-1)f_i^{t-1} + a_i^t}{t}, \tag{5.1}$$

where f_i^{t-1} is an $m \times 1$ vector representing the average strategy of agent i until time $t-1$, and a_i^t is an $m \times 1$ vector with only one non-zero (one) element. a_i^t is the best response action of agent i to the profile f_{-i}^{t-1} of the other agents played at time $t-1$. Other agents take f_i^t as the mixed strategy of agent i at time t and best respond to it. Although fictitious play was first defined for stateless games, we can extend it to Markov games. In this case, the average strategy would be defined per state:

$$f_i^t(s) = \frac{(t-1)f_i^{t-1}(s) + a_i^t(s)}{t}, \tag{5.2}$$

where s is the state at time t.

In fictitious play, each agent i has to keep track of the average strategy of all other agents at all states, which can be summarized into matrix f_{-i}:

$$f_{-i} = \begin{pmatrix} f_1(s_1) & \cdots & f_{i-1}(s_1) & f_{i+1}(s_1) & \cdots & f_n(s_1) \\ f_1(s_2) & \cdots & f_{i-1}(s_2) & f_{i+1}(s_2) & \cdots & f_n(s_2) \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ f_1(s_{|\mathcal{S}|}) & \cdots & f_{i-1}(s_{|\mathcal{S}|}) & f_{i+1}(s_{|\mathcal{S}|}) & \cdots & f_n(s_{|\mathcal{S}|}) \end{pmatrix}, \tag{5.3}$$

where $|\mathcal{S}|$ is the size of state space.

As the number of agents increases, $|\mathcal{S}|$ also grows, making f_{-i} intractable. To solve this issue, we can use the same reasoning as in mean-field games literature [28]. As each agent is small compared to the population size, its effect on the whole system can be neglected. Therefore, each representative agent can best respond to the flow of the entire population. In this case, each agent can see all other agents as a single entity. In other words, each agent i plays in a two-agent game: i and $\mathcal{N} - \{i\}$.

Instead of tracking a separate fictitious action for each agent, we can track the average strategies of all agents. This would significantly reduce the dimensions of f.

The average strategy update rule will now be:

$$f^t(s) = \frac{(t-1)f^{t-1}(s) + a^t(s)}{t}, \tag{5.4}$$

where $f^{t-1}(s)$ is the average strategy of all agents in state s until time $t-1$ and $a^t(s)$ is the mean action of all agents in state s at time t. So now each agent has to keep track of $f(s)$, for all $s \in \mathcal{S}$.

If the state space is not huge, keeping track of f requires far less memory than keeping track of all agents' policies. Each agent's policy is normally parameterized by a parameter vector θ. The size of f would be $|\mathcal{S}| \times m$, where m is the size of action space, and the size of the matrix keeping all agents' policies would be $|\theta| \times n$, where $|\theta|$ is the size of policy parameter vector and n is the number of agents. $|\theta|$ usually is more than m, but mainly depends on how the basis function is defined.

We are now ready to introduce our algorithm, which is a combination of mean-field Q-learning and fictitious play.

5.2 Fictitious Mean-field Reinforcement Learning Algorithm

We use linear approximation of the Q function with a second-order basis, i.e., for each agent i, $Q_{\theta^i}(s, a^i, f(s)) = \Psi(s, a^i, f(s))^T \theta^i$. Where θ^i is the parameter vector of agent i. We define the basis function $\Psi(s, a^i, f(s))$ to be:

$$\Psi(s, a^i, f(s)) = \begin{bmatrix} sa^i \\ sfa^i \\ 1 \end{bmatrix}, \tag{5.5}$$

where sa^i is a vector of size m^2, where each element is the multiplication of elements in s and a^i, i.e., $\{s\}_j\{a^i\}_k$, $\forall j,k \in \{1,...,m\}$. Also, sfa^i is a vector of size m^3, where each element is the multiplication of elements in s, f and a^i, i.e., $\{s\}_j\{f\}_k\{a^i\}_h$, $\forall j,k,h \in \{1,...,m\}$. Here, 1 is the constant bias used to adjust the learning procedure.

The Boltzmann policy is then determined for each agent i that:

$$\pi_t^i(a^i|s, f(s)) = \frac{\exp(\beta Q_{\theta^i}(s, a^i, f(s)))}{\sum_{a\in A^i} \exp(\beta Q_{\theta^i}(s, a, f(s)))}, \tag{5.6}$$

where β is the exploration rate. As the name suggests, β determines the amount of exploration by policy, e.g., when β is large, the policy would become almost deterministic, and vice versa; when β is close to zero, the policy would almost always explore all actions with nearly the same probability.

The main advantage of using the Boltzmann policy is that it is Greedy in the Limit with Infinite Exploration (GLIE). Such policies satisfy the two following properties:

- If a state is visited infinitely often, then all actions in that state are also chosen infinitely often.
- As $t \to \infty$, the learning policy is greedy with respect to the learned Q-function.

One of the main assumptions for the convergence of MFQ is that the policy should be GLIE. By iterating equations 5.4 and 5.6, the average strategy and the policies for all agents improve alternatively [126].

The Q-function can now be updated in a recurrent manner for each agent i as:

$$Q_{t+1}^i(s, a^i, f) = (1-\alpha)Q_t^i(s, a^i, f) + \alpha[r^i + \gamma v_t^i(s')], \tag{5.7}$$

where α denotes the learning rate and γ is the discount factor.

Here, $y^i = r^i + \gamma v_t^i(s')$ is the target mean-field value, and the final goal is to update Q parameters such that $Q_{\theta^i}(s, a^i, f(s)) = r^i + \gamma v_t^i(s')$, where $v_t^i(s')$ is calculated by:

$$v_t^i(s') = \sum_{a^i} \pi_t^i(a^i|s', f) Q_t^i(s', a^i, f(s')). \tag{5.8}$$

We update the Q parameters by minimizing the loss function $\mathcal{L}(\theta^j) = \frac{1}{K}\sum(y^j - Q_{\theta^j}(s^j, a^j))$ using gradient descent. The psudo code of the algorithm is as follows:

Algorithm 1: Fictitious Mean-field Reinforcement Learning

Initialize θ^j and $f(s)$ for all $j \in \{1, ..., n\}$ and $s \in \mathcal{S}$
while *training not finished* **do**
  for k = *1,...,batch size* **do**
    For each agent j, sample action a^j from Q_{θ^j} by equation 5.6, with the current fictitious action f and the exploration rate β
    For each agent j, update the mean action using equation 5.4
    Take the joint action $\mathbf{a} = [a^1, ..., a^n]$ and observe the joint reward $\mathbf{r} = [r^1, ..., r^n]$ and the next state s'
    Store $< s, \mathbf{a}, \mathbf{r}, s', f >$ in a replay buffer $\mathcal{D}$
  for j = *1,...,n* **do**
    Sample a mini batch of K experiences from $\mathcal{D}$
    Set $y^j = r^j + \gamma v_{\theta^j}^{MF}(s')$ by equation 5.8
    Update Q parameters by minimizing the loss $\mathcal{L}(\theta^j) = \frac{1}{K}\sum(y^j - Q_{\theta^j}(s^j, a^j))$

Chapter 6

Experiments

In this chapter, we first apply our algorithm to four benchmark examples to empirically show the effectiveness of our proposed algorithm. We then apply our algorithm to the game presented in Chapter 3, and compare its performance against the state-of-the-art.

Each of four tests are designed to be potential games, as we have previously proven our game to be Markov potential for many agents. All tests are played by 100 homogeneous agents, batch size is set to 100, $\beta = 0.01$ and $\gamma = 0.99$. The learning rate is initially set to 1 and then geometrically decays by the discount factor with a minimum of 0.001.

We implement algorithm 1 and compare it against two baseline models:

1. Independent Q-learner (IQL), the simplest multi-agent version of Q-learning that does not consider the action taken by other agents [113], and
2. Nash Q-learning (NQL) [61], where the action of all other agents is added to the Q-function.

We also implement actor-critic (AC) and projected stochastic gradient ascent (PSGA) [76], once as an independent learner and once with fictitious action. Therefore, we can evaluate the effect of adding fictitious action to learning algorithms other than Q-learning.

It is known that a Nash equilibrium exists in all finite games, and through these benchmark examples we empirically show that our iterative algorithm converges to an approximation of the optimal Nash equilibrium. Each one of the tests has an infinite number of mixed Nash equilibria; however, we empirically show that our proposed algorithm converges to the optimal equilibrium.

6.1 Test I (two states, two actions team game)

Setting. This scenario has two states, s_1 and s_2, where state transitions are independent of agents' actions. Every agent can take two possible actions in both states, a_1 and a_2.

If all agents take a_1 at state s_1, they all get a reward of 100. If one or more of them deviates, they all get 1. The same happens for s_2, if all agents take a_2 in s_2, they all get 100 and if one or more of them deviates, they all get 1.

The reward function would be $\forall i \in \mathcal{N}$:

$$R_i(\mathbf{a}, s_1) = \begin{cases} 100, & \text{if } \mathbf{a} = (a_1, ..., a_1), \\ 1, & \text{otherwise,} \end{cases} \tag{6.1}$$

and

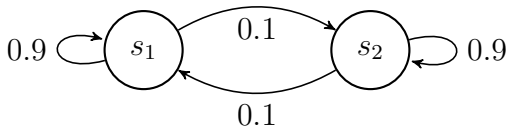
$$R_i(\mathbf{a}, s_2) = \begin{cases} 100, & \text{if } \mathbf{a} = (a_2, ..., a_2), \\ 1, & \text{otherwise.} \end{cases} \tag{6.2}$$

State transitions however, are independent from actions:

$$\mathbb{P}(s_2|s_1, \mathbf{a}) = \mathbb{P}(s_2|s_1) = 0.1, \tag{6.3}$$

$$\mathbb{P}(s_1|s_2, \mathbf{a}) = \mathbb{P}(s_1|s_2) = 0.1, \tag{6.4}$$

Bellow is the visual representation of the Markov chain:



This game is a team game (see Chapter 2), as all agents receive the same reward at each state; therefore, it's a potential game at each state. As state transitions are agent-independent, C1 (see Chapter 4) is satisfied; therefore, the game is a Markov potential game, where the potential function is the same as the reward function.

The only optimal deterministic Nash policy in s_1 is all agents playing a_1 with probability one, as none of the agents get a higher utility by unilaterally deviating from it. By the same reasoning, the deterministic Nash policy in s_2 would be all agents playing a_2 with probability one. This strategy maximizes the potential function, and as we discussed earlier, Nash equilibria are the optimal points of the potential function in potential games.

Results. Figure 6.1 illustrates the result, comparing the performance of algorithm 1 to two different baselines, independent Q-learner and Nash Q-learning. Nash Q-learning has the best performance, as it calculates the Nash policy at each stage of the game and updates Q-functions according to that.

Nash Q-learning is very expensive in terms of both space complexity and run time [61]. In this algorithm, each agent has to keep track of all other agents Q-functions and also

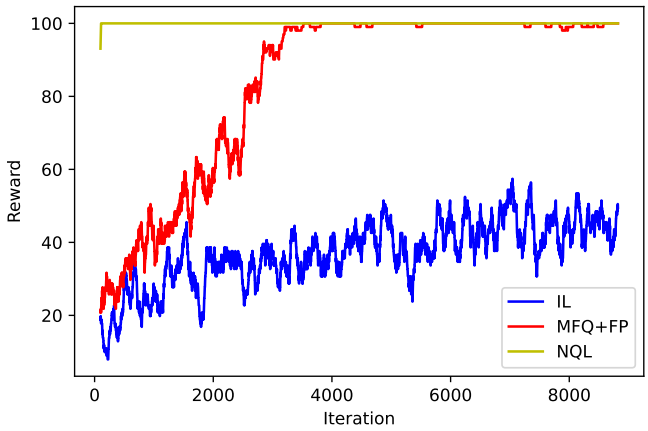


Figure 6.1: Test I; Comparing reward of Nash Q-learning, independent learner and MFQ+F.

has to calculate the Nash policy at every stage of the game. Regarding space complexity, the algorithm is linear in the number of states, polynomial in the number of actions, but exponential in the number of agents. On the other hand, the algorithm's running time is mainly dominated by the calculation of Nash policy in each state. The computational complexity of finding an equilibrium in matrix games is unknown [61]. Overall, implementing Nash Q-learning for settings with a large number of agents or general-sum matrix games can be very complex and infeasible. Therefore, we only implement it for the benchmark examples.

Algorithm 1, on the other hand, is far less complex in terms of time and memory and has comparable performance with NQL. As expected, it also outperforms the independent learner.

Figure 6.2 illustrates the performance of vanilla PSGA and PSGA with fictitious action added. Leonardos et al. (2021) [76] prove that PSGA converges to the Nash equilibrium,

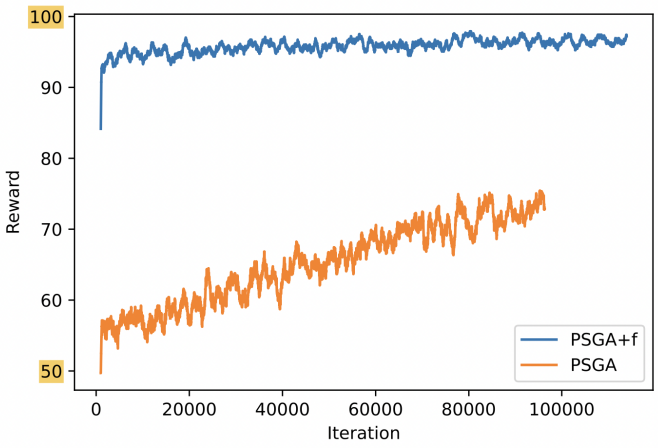


Figure 6.2: Test I; Comparing reward of vanilla PSGA and PSGA with fictitious action.

but as mentioned before, the convergence is very slow. Naive policy gradient methods perform poorly in simple multi-agent settings, which is also supported in our experiment. They are also known to exhibit high variance gradient estimates, which is exacerbated in multi-agent settings. As shown, adding f improves the performance dramatically.

Figure 6.3 illustrates the performance of vanilla actor-critic and actor-critic with fictitious action added. As shown, independent actor-critic does not converge. Adding fictitious action is beneficial, especially when the number of agents is high and collaboration is necessary between a large population of agents. Independent actor-critic can perform well with a small number of agents, but it cannot compete with AC+f when there are many agents. As illustrated in figure 6.4, independent learner can perform well when we only have ten agents in the game.

In the figures below, the y axis is the average reward of all agents, which is equal to the instantaneous reward of each agent.

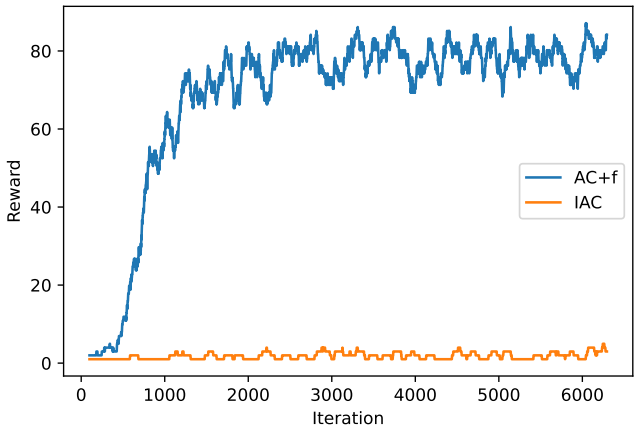


Figure 6.3: Test I; Comparing reward of vanilla actor-critic and actor-critic with fictitious action.

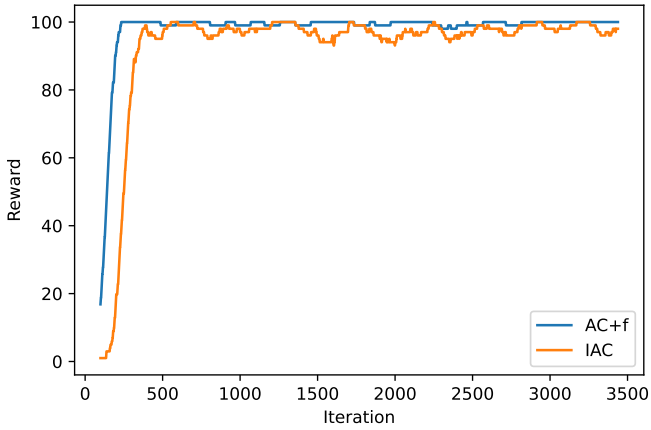


Figure 6.4: Test I; Comparing reward of independent actor-critic and actor-critic with fictitious action with 10 agents.

6.2 Test II (one state, two actions team game)

Setting. This scenario has one state, s_1. Every agent can take either of two possible actions, a_1 and a_2. If all agents take the same action, i.e., all of them take a_1 or all of them take a_2, they get a reward of 100. Otherwise their reward would be 1.

The reward function would be $\forall i \in \mathcal{N}$:

$$R_i(\mathbf{a}, s_1) = \begin{cases} 100, & \text{if } \mathbf{a} = (a_1, ..., a_1), \\ 100, & \text{if } \mathbf{a} = (a_2, ..., a_2), \\ 1, & \text{otherwise.} \end{cases} \tag{6.5}$$

This game is also a Markov potential game, with the same reasoning as test I. The deterministic Nash policy would be all agents taking action a_1 with probability one, or all of them taking a_2 with probability one.

Results. We expect the independent learner to perform poorly in this setting, as co-operation is necessary to achieve the optimal outcome.

Figure 6.5 illustrates the result, comparing the performance of algorithm 1 to two different baselines, independent Q-learner and Nash Q-learning. As illustrated, MFQ+FP performs comparable to NQL, while being less expensive. As expected, independent learner does not learn the optimal policy in this setting. The results for PSGA and actor-critic can be seen in figures 6.6 and 6.7 respectively.

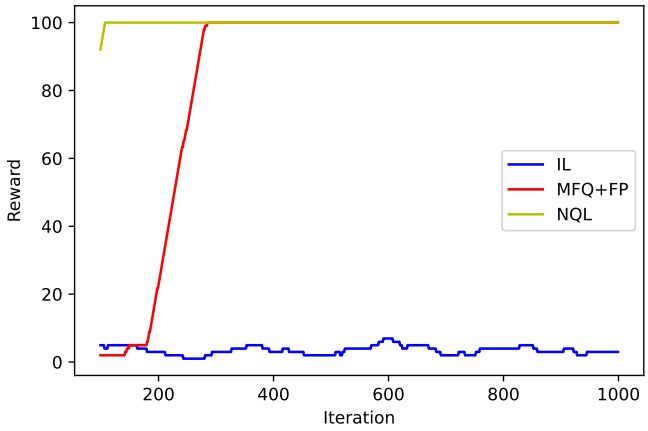


Figure 6.5: Test II; Comparing reward of Nash Q-learning, independent learner and MFQ+F.

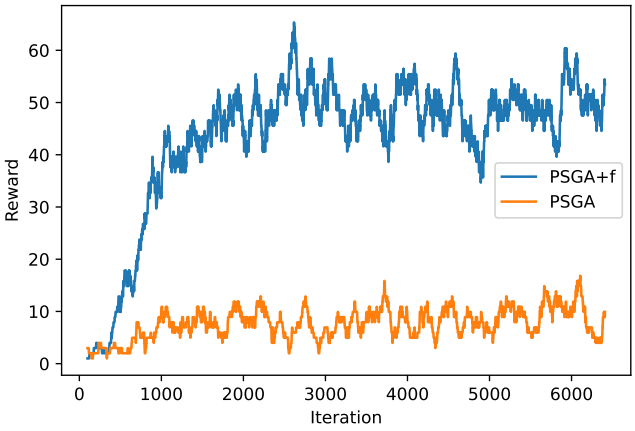


Figure 6.6: Test II; Comparing reward of vanilla PSGA and PSGA with fictitious action.

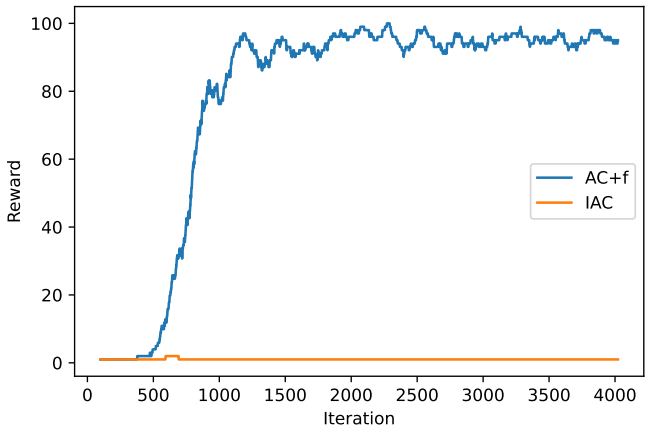


Figure 6.7: Test II; Comparing reward of vanilla actor-critic and actor-critic with fictitious action.

6.3 Test III (two states, two actions team game)

Setting. This scenario has two states, s_1 and s_2, where state transitions are independent of agents' actions. Every agent can take either of two possible actions in both states, a_1 and a_2.

If all agents take the same action at state s_1, i.e., they all take a_1 or they all take a_2, they all get a reward of 100. If only one of them deviates, they all get 1. The reverse happens for s_2, if all agents take the same action at state s_2, they all get 1 and if at least one of them deviates, they all get 100.

The reward function would be $\forall i \in \mathcal{N}$:

$$R_i(\mathbf{a}, s_1) = \begin{cases} 100, & \text{if } \mathbf{a} = (a_1, ..., a_1), \\ 100, & \text{if } \mathbf{a} = (a_2, ..., a_2), \\ 1, & \text{otherwise.} \end{cases} \tag{6.6}$$

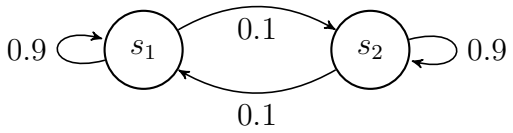
and

$$R_i(\mathbf{a}, s_2) = \begin{cases} 1, & \text{if } \mathbf{a} = (a_1, ..., a_1), \\ 1, & \text{if } \mathbf{a} = (a_2, ..., a_2), \cdot \\ 100, & \text{otherwise.} \end{cases} \tag{6.7}$$

State transitions however, are independent from actions:

$$\mathbb{P}(s_2|s_1, \mathbf{a}) = \mathbb{P}(s_2|s_1) = 0.1, \;\; \mathbb{P}(s_1|s_2, \mathbf{a}) = \mathbb{P}(s_1|s_2) = 0.1. \tag{6.8}$$

Bellow is the visual representation of the Markov chain:



Results. As in test II, we expect the independent learner to perform poorly in this setting, as cooperation is necessary to achieve the optimal outcome.

The nature of interaction between agents in all previous tests is collaborative, i.e., all agents' objective is to maximize a common reward function. So far, we can conclude that adding fictitious action is beneficial in collaborative settings, especially with massive number of agents.

Test IV, on the other hand, is neither collaborative nor competitive; each agent's reward is independent of other agents' actions or states. Also, each agent has a separate state. Therefore, test iv is not at all a Markov game.

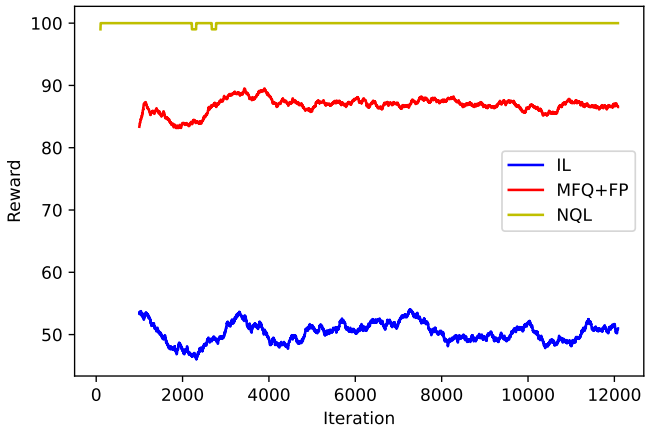


Figure 6.8: Test III; Comparing reward of Nash Q-learning, independent learner and MFQ+F.

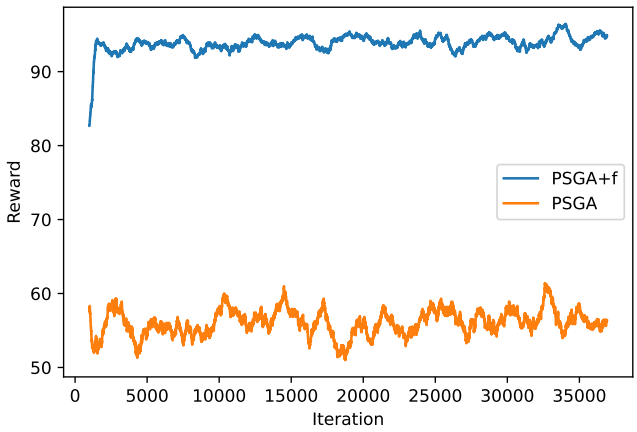


Figure 6.9: Test III; Comparing reward of vanilla PSGA and PSGA with fictitious action. Independent PSGA converges to the Nash equilibrium very slowly.

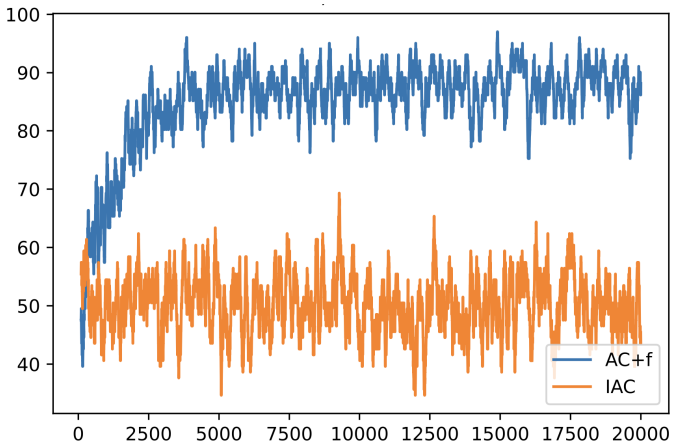


Figure 6.10: Test III; Comparing reward of vanilla actor-critic and actor-critic with fictitious action.

6.4 Test IV (two states, two actions mixed game)

Setting. This scenario has two states, s_1 and s_2. Every agent can take two possible actions in both states, a_1 and a_2, and agents have separate states.

If agent i, $i \in \mathcal{N}$ selects action a_1 in s_1, it receives a reward of 80 and stays in s_1 with probability 0.9. Otherwise, it gets a reward of 100 and moves to s_2 with probability 1. In s_2, all actions yield a reward of 1.

The reward function would be $\forall i \in \mathcal{N}$:

$$R_i(a_i, s_1) = \begin{cases} 80, & \text{if } a_i = a_1, \\ 100, & \text{if } a_i = a_2, \end{cases} \tag{6.9}$$

and

$$R_i(a_i, s_2) = 1. \tag{6.10}$$

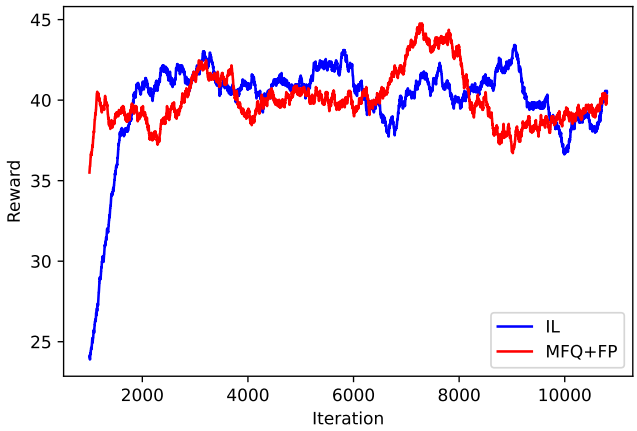


Figure 6.11: Test IV; Comparing reward of independent learner and MFQ+F. As expected, independent learner performs as good as our proposed algorithm.

State transitions would therefore be:

$$\mathbb{P}(s_2|s_1, a_1) = 0.1, \quad \mathbb{P}(s_2|s_1, a_2) = 1. \tag{6.11}$$

This game is also potential, as we can define the potential function as the sum of all agents' value functions; as change of each agent's policy does not affect the value function of all other agents.

The optimal policy for each agent would be $\pi_i(a_1|s_1) = 1$ and $\pi_i(a_1|s_2) = \pi_i(a_2|s_2) = \frac{1}{2}$, $\forall i \in \mathcal{N}$. We expect the independent learner to have a good performance in this case, as agents' rewards are independent from each other. Each agent's action has no impact on other agents' rewards or state transitions. Therefore, the independent learner can learn the optimal policy without cooperating with other agents.

6.5 Load Balancing Game

We can model the distributed load-balancing game as a collaborative or competitive game. In the competitive setting, strategic agents compete over shared resources, which in this case are the servers. A critical question in such settings is whether a mechanism with fair allocation of resources exists, despite agents' strategic behavior. The objective function used for measuring the fairness of an allocation is the geometric mean of the agents' values, also known as the Nash social welfare (NSW) [11].

In the collaborative setting, however, strategic agents maximize a common reward function. Therefore, the best objective function for such settings would be the common reward function, also known as social welfare.

From the server side, we care about balancing the load between servers. In order to measure the performance of each algorithm, we plot each server's average wait time and compare them. Algorithms with closer average wait times perform better. We also plot the mean average wait time across all servers.

We have three different configurations for each setting:

I. number of servers = 2, clients total arrival rate = 4, servers service rates = 6, 2,

II. number of servers = 3, clients total arrival rate = 10, servers service rates = 4, 4, 4, and

III. number of servers = 3, clients total arrival rate = 10, servers service rates = 6, 6, 4.

All configurations are with 100 agents, and job sizes are set to be 1. The capacity is set to 25 for each server, the batch size is 100, and $\beta = 1$. We show results for mean-field

Q-learning and actor-critic, with and without the fictitious action. We also present the results for both competitive and collaborative versions of each algorithm.

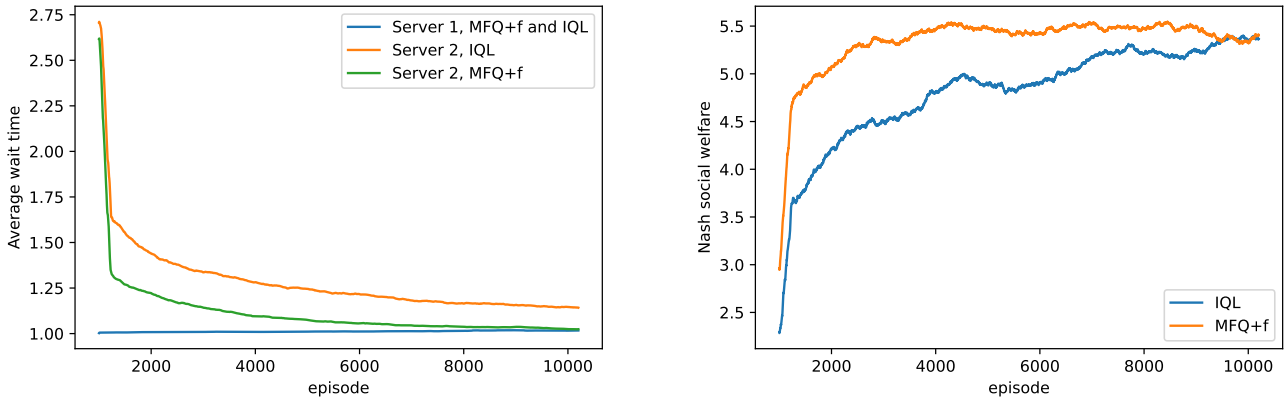
6.5.1 Competitive Setting

Results. For the competitive setting we have:

- **Configuration I**: In this configuration server 1 is way faster than server 2, and can solely handle the system's arrival. However, servers have a limited capacity, and there's also a chance that at some time step, the input to the system is more than server one's service rate, but overall we expect the clients to choose server 1 with high probability. As illustrated in figure 6.12a, servers are load balanced when fictitious action is added to the learning procedure. 6.12b also suggests that Nash social welfare converges faster to the optimal solution with MFQ+f. It is also clear from 6.12a that the mean average wait time of servers is less when the fictitious action is added.

  As illustrated in figure 6.13a, servers are more load balanced when fictitious action is added. Also the mean average wait time across both servers is lower in AC+f, 6.13b. As shown in 6.13c, the Nash social welfare in AC+f is higher at first, but eventually the naive learner would also converge. Also, note that server one is so fast; therefore, it always has an average wait time close to one.

- **Configuration II**: There are three identical servers in this configuration. We expect the clients to send tasks equally to all servers. We also expect the independent learner to perform well in this setting, as agents start the learning process with the optimal strategy. As shown in figure 6.14, independent learner performs comparably to MFQ+f. It is even performing better in terms of mean average wait time. In



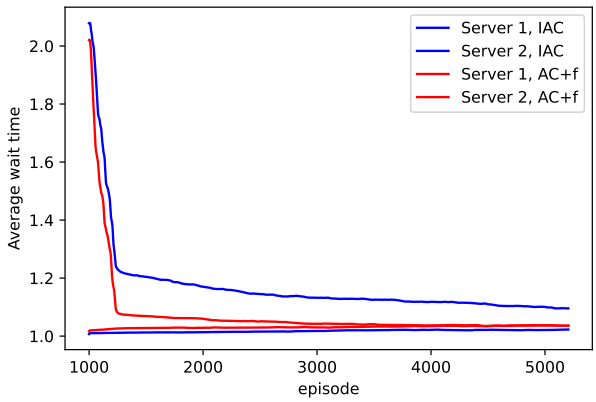
(a) (b)

Figure 6.12: Competitive load balancing; Configuration I; Comparing performance of naive Q-learner and MFQ+f.

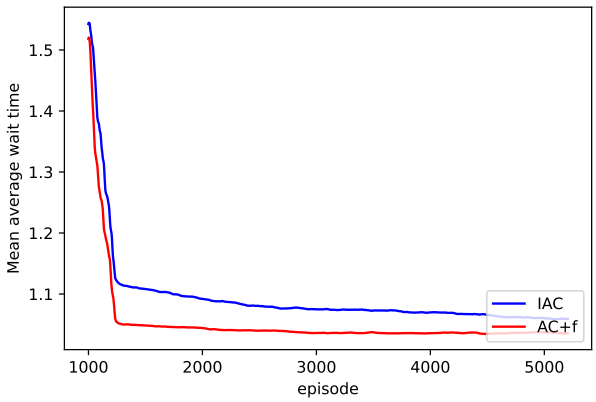
terms of Nash social welfare, IQL performs better at first, but eventually, MFQ+f has a better performance. In general however, both algorithms have almost the same performance. Moreover, as we can see in figure 6.15, AC+f is outperforming the independent learner in terms of Nash social welfare. In terms of mean average wait time, the independent learner performs better by iteration 2500, but eventually, AC+f outperforms the independent learner.

- **Configuration III**: In this configuration, server one and two are faster than server 3 and can handle the arrival of the system. We expect the learning algorithm to learn to choose the first two servers with high probability.

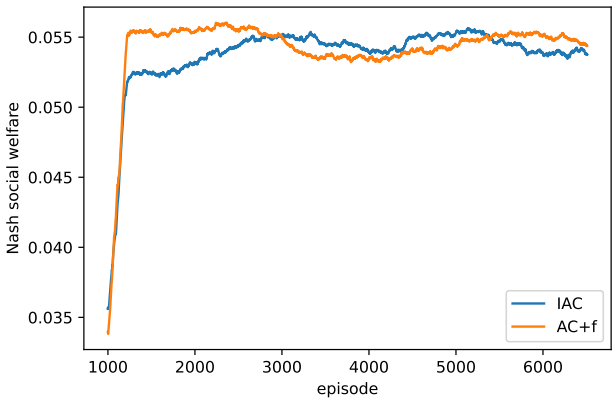
  As illustrated in figure 6.16, the average wait time of server three, the slowest server, is going down by iteration 2000. Still, there is a difference with the average wait time of the other two servers for both independent learner and MFQ+f. The load on



(a)

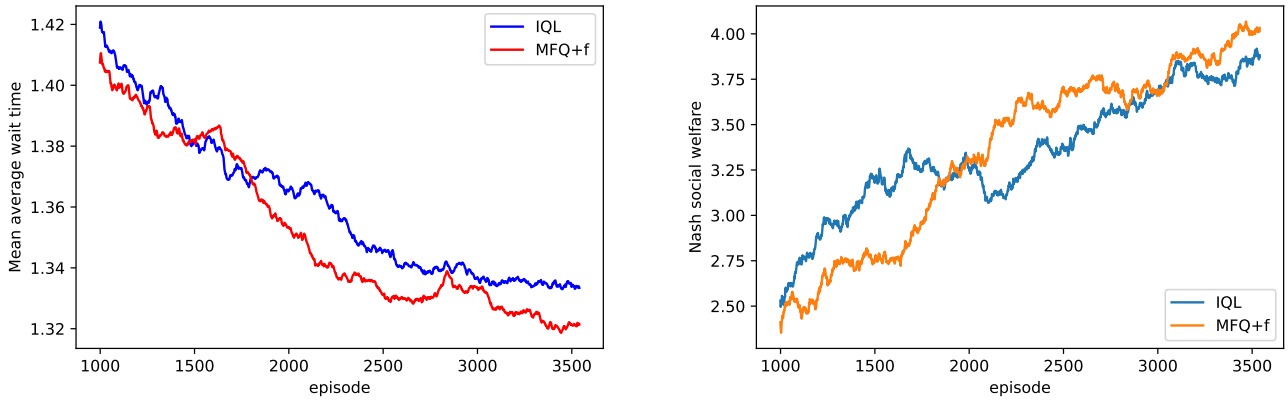


(b)



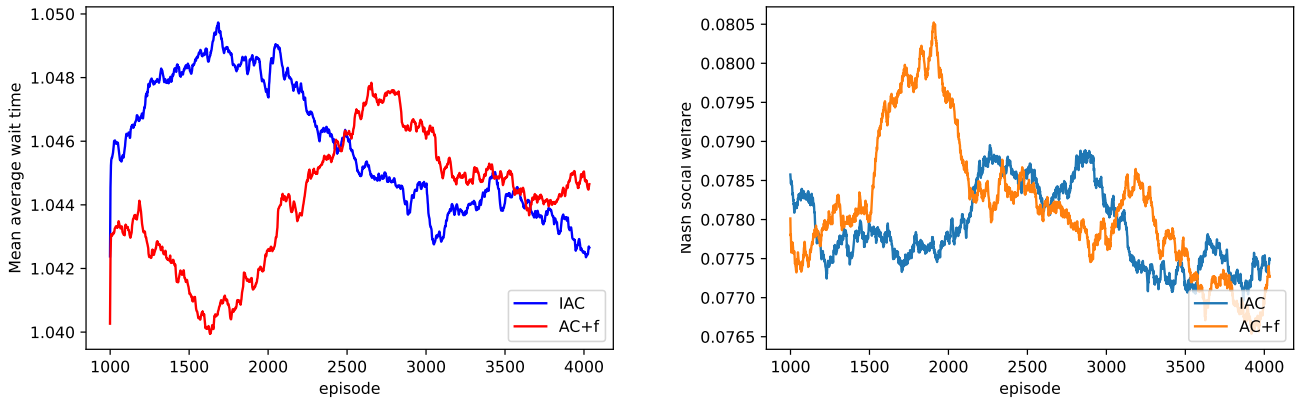
(c)

Figure 6.13: Competitive load balancing; Configuration I; Comparing performance of naive actor-critic and AC+f.



(a) (b)

Figure 6.14: Competitive load balancing; Configuration II; Comparing performance of naive Q-learner and MFQ+f.



(a) (b)

Figure 6.15: Competitive load balancing; Configuration II; Comparing performance of naive learner and AC+f.

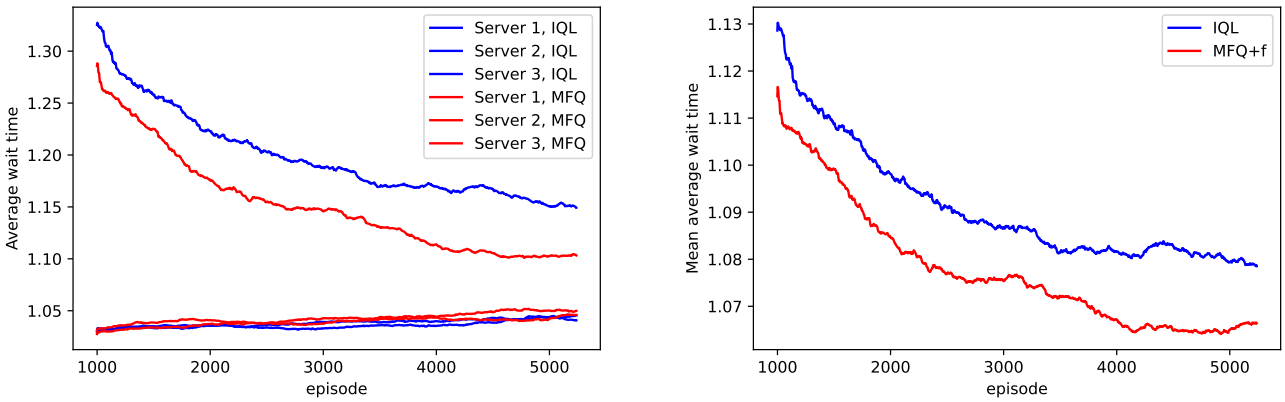
other servers is one and remains one. The mean average wait time is decreasing in both algorithms. However, the mean average wait time is less in MFQ+f. As shown in the figure, the Nash social welfare is also higher in MFQ+f. We can also see in figure 6.17 that adding fictitious action to actor-critic, is beneficial to the Nash social welfare and also the average wait time of servers. In this case, the independent learner performs comparably to AC+f by iteration 3500, but eventually, AC+f outperforms the independent learner.

We now present the results for the collaborative setting. In this setting, social welfare is defined as the sum of all agents' rewards. Note that the evaluation criteria is social welfare and no longer the NSW. Configurations are the same as before, and we present the results for both mean-field Q-learning and actor-critic, with and without the fictitious action. Also as configuration 2 does not highlight the effect of adding fictitious action, we do not present it here.

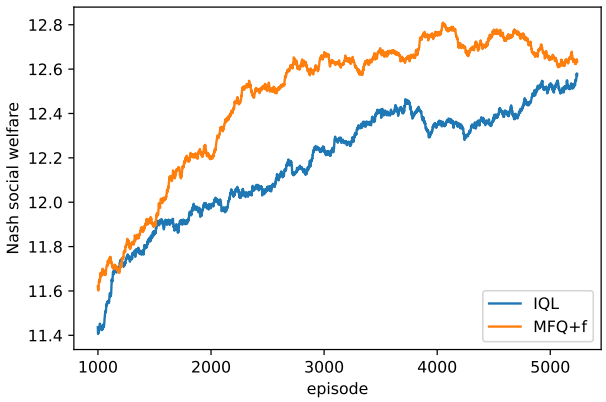
6.5.2 Collaborative Setting

Results. For the collaborative setting we have:

- **Configuration I**: As we can see in figure 6.19, the average wait time on server 2 goes down by iteration 2000 in both algorithms. The average wait time of both servers become the same by iteration 3500 in MFQ+f. However, they still have a difference by iteration 4000 when using independent Q-learning. The mean average wait time is also lower in MFQ+f. As it can be seen, adding fictitious action is also beneficial to the social welfare. We can also see in figure 6.20 that adding fictitious action to actor-critic, is beneficial to social welfare and also the average wait time of servers.

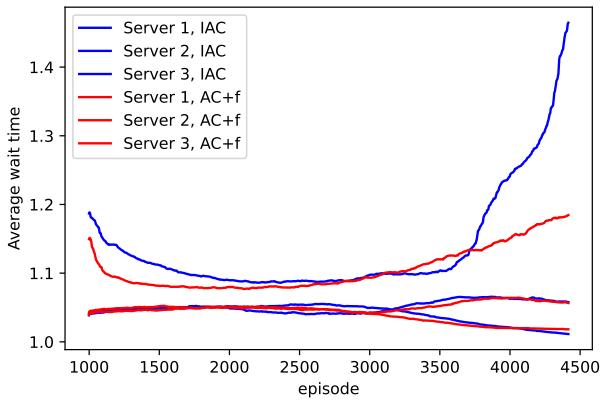


(a) (b)

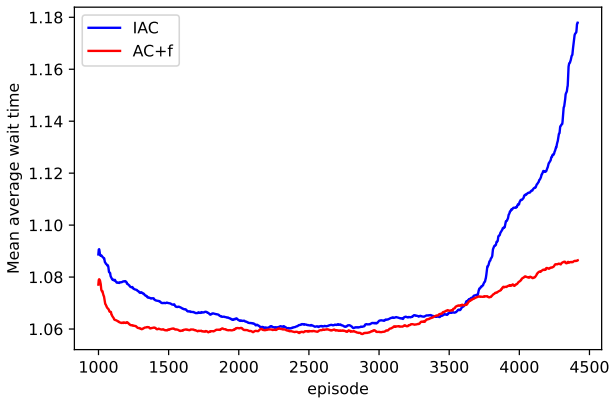


(c)

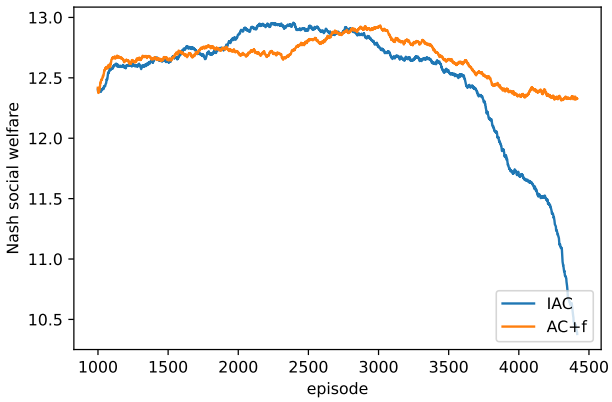
Figure 6.16: Competitive load balancing; Configuration III; Comparing performance of naive Q-learner and MFQ+f.



(a)

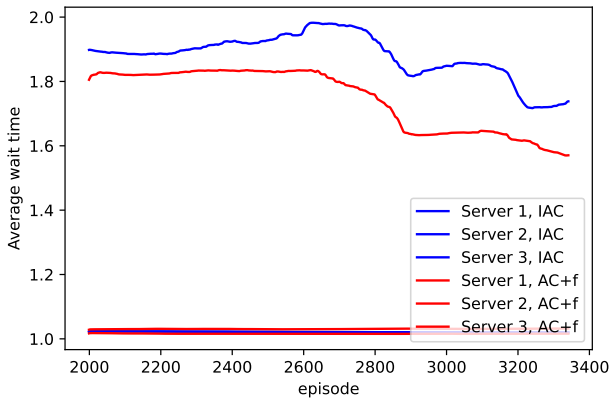


(b)

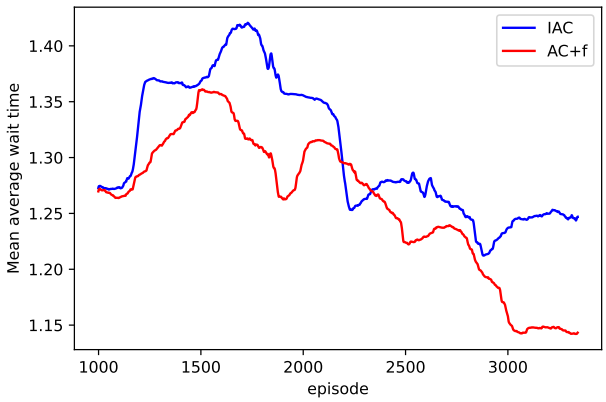


(c)

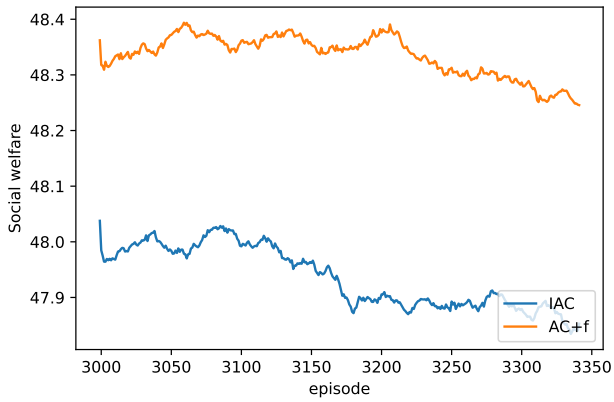
Figure 6.17: Competitive load balancing; Configuration III; Comparing performance of naive learner and AC+f.



(a)



(b)

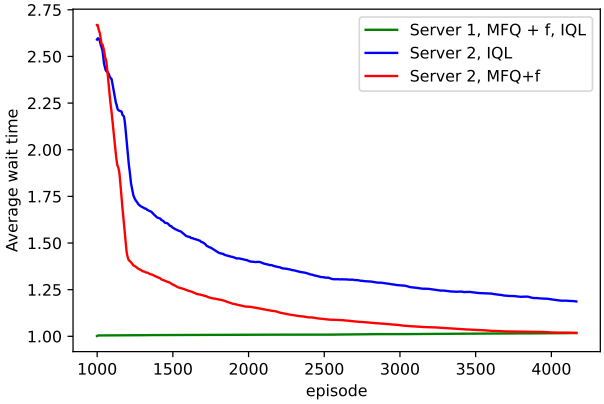


(c)

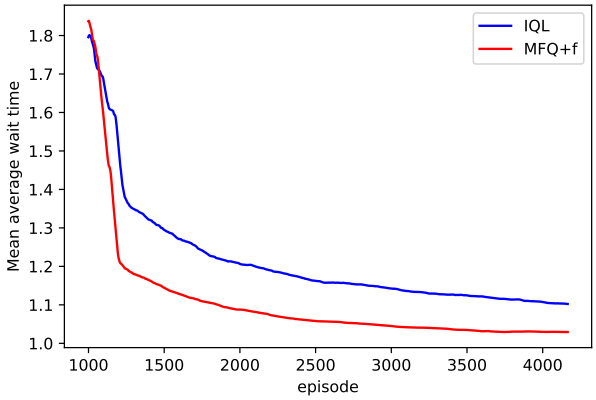
Figure 6.18: Collaborative load balancing; Configuration III; Comparing performance of naive learner and AC+f.

Unlike AC+f, the average wait time of server two in IAC maintains a difference with the average wait time of server one. Moreover, the mean average wait time is lower in AC+f and the social welfare is higher. Also as shown in 6.19a and 6.20a, adding fictitious action has helped the system to become more load balanced. Also, note that server one is so fast; therefore, it always has an average wait time close to one.

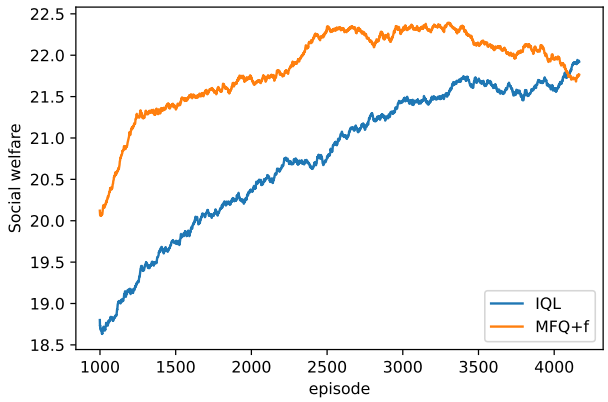
- **Configuration iii**: As illustrated in figure 6.21, the average wait time of server three becomes almost the same with the other two servers in MFQ+f. However, it maintains a difference in IQL. The mean average wait time is less in MFQ+f and social welfare is also higher in MFQ+f. We can also see in figure 6.18 that adding fictitious action to actor-critic, is beneficial to social welfare and also the average wait time of servers. The average wait time of server three is closer to one in AC+f. Moreover, the mean average wait time is significantly less in AC+f and the social welfare is higher. In Q-learning, we have also shown in figure 6.21a that the system becomes more load balanced when the fictitious action is added.



(a)

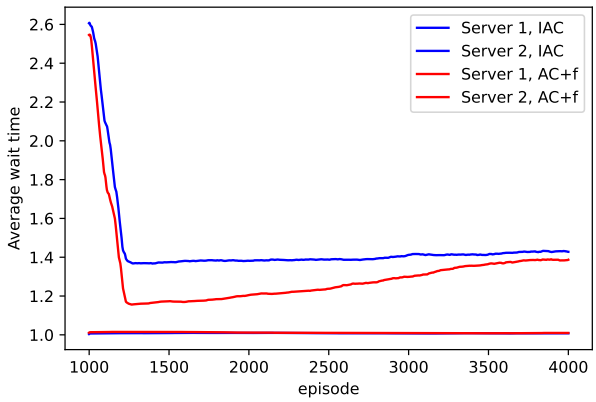


(b)

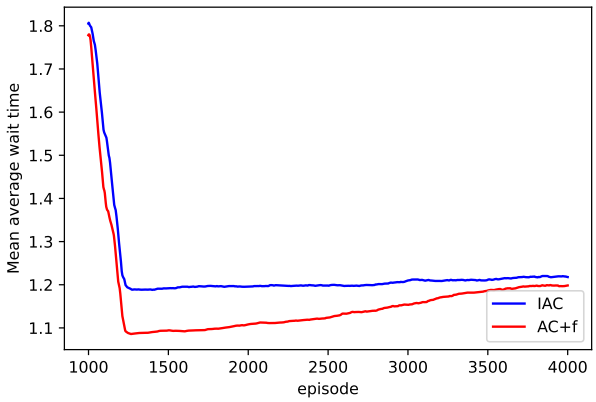


(c)

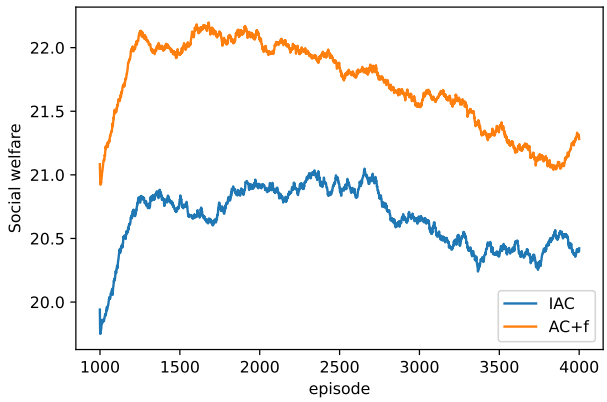
Figure 6.19: Collaborative load balancing; Configuration I; Comparing performance of naive Q-learner and MFQ+f.



(a)

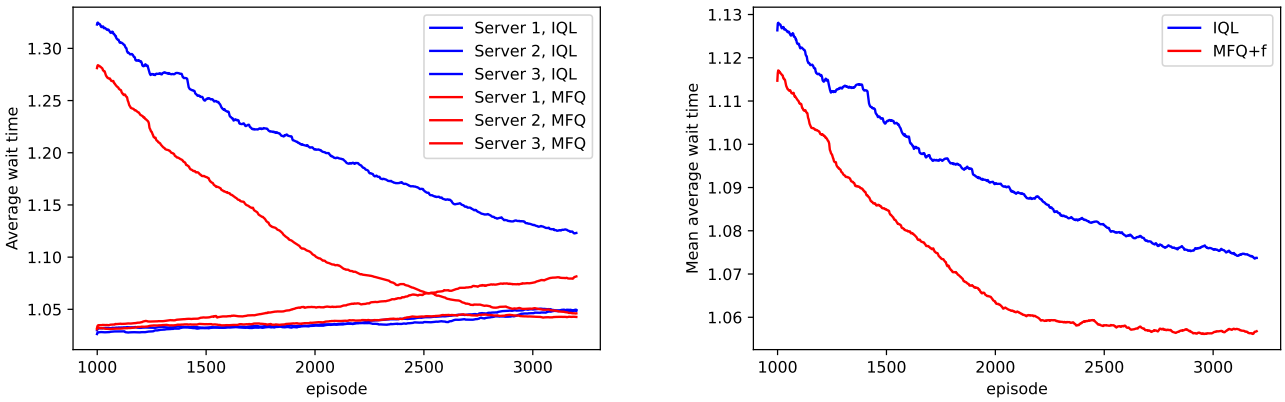


(b)



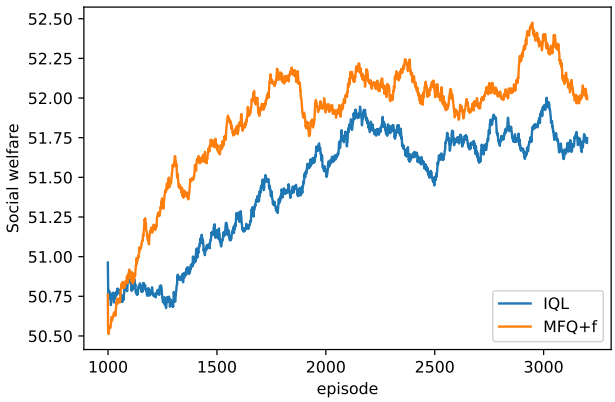
(c)

Figure 6.20: Collaborative load balancing; Configuration I; Comparing performance of naive learner and AC+f.



(a)

(b)



(c)

Figure 6.21: Collaborative load balancing; Configuration III; Comparing performance of naive Q-learner and MFQ+f.

Chapter 7

Related work

In this chapter, we begin by going through studies on load balancing, a large number of them elaborating on distributed systems. Only a few instances within this broad literature have been devoted to studying such systems from a game theoretic approach. Subsequently, we review works related to fictitious play, e.g., the original definition, the fictitious play property, and fictitious play in stochastic games. Later, we move to studies on potential games. In addition, we review the body of literature studying conditions for stochastic games to be Markov potential games. We then review works related to mean-field games and Q-learning. Lastly, we review the large body of literature on multi-agent reinforcement learning and mean-field reinforcement learning.

7.1 Load Balancing

A large number of literature has studied load balancing in distributed systems. Most of them study systems with a centralized scheduler [130, 53, 59, 86, 77]. Join-shortest-queue

(JSQ) is one the best-studied load balancing disciplines [47, 29, 35, 69, 68, 42, 124]. In this method, jobs are sent to the server with the shortest queue length. Join-idle-queue (JIQ) [79, 87, 65] is another load balancing policy where jobs are sent to idle servers, if there is any, or to a randomly selected server. These methods might not perform well as the queue length is not a good indicator of the servers' wait times. Power-of-d-choices [86, 125, 41, 96, 102, 94, 131] queries d randomly selected servers and sends the jobs to the least loaded ones. This method is proved to yield short queues in homogeneous systems, but might be unstable in heterogeneous systems.

Studying distributed load balancing through a game-theoretic lens is a natural approach. However, there exist only a few studies on the game theoretic approach for the load balancing problem [67, 100, 93, 27, 26, 95, 1, 70, 71, 82, 101]. Grosu and Chronopoulos (2005) [44] model the optimal load balancing problem in heterogeneous systems as a non-cooperative, repeated finite game and find a structure for the Nash equilibrium in pure strategies. They later introduce a distributed greedy algorithm to reach the equilibrium. As their setting is stateless, the proposed algorithm does not necessarily perform well in dynamic games. Moreover, Grosu et al. [45] model the load balancing problem as a cooperative game among agents.

Schaerf et al. (1994) [103] study the distributed load balancing problem using multi-agent reinforcement learning methods. Their framework is highly similar to our setting, but differs on one element: Each server simultaneously works on multiple tasks, and the efficiency with which the server handles the job depends on its capacity and the number of tasks it handles. They model the problem as a stochastic system, where local information is available to each agent. Each agent's evaluation of servers' efficiency is summarized in a vector called an efficiency estimator, which can also be the state of those agents. Each agent's policy is a function of the state. They compare the performance of their algorithm

to other existing algorithms such as non-adaptive algorithms, i.e., algorithms that agents use servers in a predetermined manner, while revealing that naive communication between agents will not necessarily improve system's efficiency.

Krishnasamy et al. [73] study a variant of the multi-armed bandit problem with the carryover effect which is suitable for queuing applications. They consider a bandit setting where jobs queue for service and service rates of different queues are unknown. The authors study a queuing system with a single queue and K servers. Arrivals to the queue and service offered by servers are according to a Bernoulli distribution. At any given time, the queue is served by at most one server and the problem is to schedule a server at every time slot. They further evaluate the performance of different scheduling policies against the no-regret[1] policy.

Gaitonde and Tardos (2020) [38] study the multi-agent version of the queuing system in [73]. Their setting is slightly different from ours. In our setting, each server has a separate queue and processes tasks with a fixed service rate. However, Gaitonde and Tardos (2020) [38] describe a system where each client has a queue and servers process jobs with a fixed timed-independent probability. All unprocessed jobs are then sent back to their respective queues. Queues also receive bandit feedback on whether their packet cleared at their chosen server. One concern in such settings is whether the system remains stable, despite the selfish behavior of the learning agents. Such concern does not arise in settings with a centralized scheduler, as they remain stable if the sum of arrival rates is less than the sum of service rates. However, this condition does not guarantee the stability in settings with strategic clients. Gaitonde and Tardos (2020) [38] show that if the capacity of the servers is high enough to allow a centralized coordinator to get all jobs done even with double the job arrival rate, and agents use no-regret learning algorithms, then the system remains

[1]The policy that schedules the optimal server in every time slot.

stable. A no-regret learning algorithm minimizes the expected difference in queue sizes to that of a strategy that knows the optimal server. They later show in [39] that the stability can be guaranteed even with $\frac{e}{e-1} \approx 1.58$ extra capacity, only if agents choose strategies that maximizes their long-run success rate.

7.2 Fictitious Play

Brown proposed fictitious play in 1951 [12]. The original version of the fictitious play depicts a two-agent, finite, repeated game where at every round, each agent simultaneously plays a myopic pure best response to the empirical strategy distribution of her opponent [5]. Each agents takes the empirical distribution of other agents' actions to be her belief about their mixed strategy. A game has the fictitious play property (FPP) if every fictitious play process converges in beliefs to the set of Nash equilibria [91]. While Robinson (1951) [99] proves that every 2-agent zero-sum game has the fictitious play property, Miyasawa (1961) [88] extends it to all two-agent 2×2 games, and Monderer and Shapley (1996) [91] go on to convey convergence for identical interest games. Shapley (1964) [106] conveys an example of a 2-agent 3×3 game as well, but without the fictitious play property. A significant number of papers in the literature attempt to identify classes of games with FPP, including [85, 6, 56, 92, 58, 49, 104, 25, 4, 115]. There also exists works that find classes of games without the FPP, e.g., [24, 66, 43, 90, 36, 72].

An issue is that Fictitious play requires each agent to have complete knowledge of other agents' strategies. To solve this issue, Perolat et al. (2018) [97] develop a decentralized online algorithm inspired by actor-critic that approximates the fictitious play process. They prove the convergence of such an algorithm towards a Nash equilibrium in both cases of zero-sum two-agent multistage games and cooperative multistage games. It is worth

mentioning that this study focuses only on multistage games that can be modeled by a tree, meaning that the interaction proceeds from stage to stage without going back to a previously visited state. Even though this model represents a broad class of multi-agent sequential decision processes where the interaction never reverts back to the same state, it also does not apply to many settings.

Fictitious play can also be extended to stochastic games, but the main challenge is to define and compute the best response to empirical observations. Given what other agents do, calculating the total discounted payoff is not always straightforward. To address this issue, Baudin (2021) [2] combines ideas from Q-learning and fictitious play; a standard method of reinforcement learning, and a standard game theory method. Baudin defines three reinforcement learning algorithms and proves their convergence to the set of Nash equilibria in identical interest discounted stochastic games for both continuous and discrete-time systems. In this case, identical interest translates to all agents having the same reward function, which is different from how it is defined in [91]. Monderer and Shapley (1996) [91] define an identical interest game as a game which is best response equivalent in mixed strategies to a game with identical payoff functions, which can also be interpreted as a potential game.

7.3 Potential Games

Monderer and Shapley (1996) [92] introduce the concept of potential games and their properties. Potential games are one of the most important and best-studied classes of games, representing multi-agent coordination, as all agents' utilities are perfectly aligned with each other via a potential function. To put in simply, cooperation is desirable in such games. The authors prove that fictitious play converges to equilibrium in a class of games

that contains the finite (weighted) potential games [91].

Although the concept of potential games was first defined for single-stage games, Leonardos et al (2021) [76] extend it to stochastic games. They carefully study the definition of Markov potential games and establish its structural properties. The authors provide sufficient conditions for Markovian/stochastic games that are potential at each state to be Markov potential games. They also prove convergence to the ϵ-NE in MPGs with finite state space when agents independently run policy gradient on their own policy. Fox et al. (2022) [37] also reveal that in a Markov potential game, with all agents using independent natural policy gradient, their policies converge to the equilibrium.

Mguni et al. (2021) [83] provide sufficient conditions for a stochastic game to be an MPG. However, in this thesis we provide a counterexample for proposition four. The authors use this proposition to prove of the main theorem. Our example has conditions stated in the paper but we show that it is not a Markov potential game. The counterexample is in Appendix A.

Macua et al. (2018) [80], on the other hand, provide necessary and sufficient conditions on agents' reward functions for a stochastic game to be an MPG. Moreover, they provide two methods of obtaining the potential function and convey that since the game is potential, the Nash equilibrium can be found by solving only one optimal control problem instead of several, as in one for each agent.

7.4 Mean-field Games

Mean-field game (MFG) is a framework dedicated to the analysis of games with an infinite number of identical agents [75, 63]. For such large population of agents, it is unrealistic

for an agent to collect information about all the other agents. Mean-field theory explains that an agent only needs to implement strategies based on the distribution of all other agents. Since all agents are identical and indistinguishable, individual interaction between agents is redundant, and only the distribution of agents over states matters [16, 3]. In such games, two dynamics determine the system's evolution: state dynamics and mean-field flow dynamics. Hence finding the Nash equilibrium boils down to identifying the equilibrium distribution of the population and the best response of a representative agent to the mean-field flow of the population. Studying mean-field games can be easier, as the impact of one single agent on others is negligible. Furthermore, one can use the optimal policy for the MFG as an approximate Nash equilibrium for the game with a finite number of agents [16, 3, 19].

There is not much literature on multi-agent reinforcement learning in mean-field games, especially in non-cooperative settings. Hadikhanloo (2018) [48] shows that fictitious play converges in first-order monotone mean-field games. Cardaliaguet and Hadikhanloo (2017) [17] also introduce a learning procedure similar to fictitious play and prove its convergence when the mean-field game is potential. The downside, however, is that they only consider the Nash equilibrium between fully informed agents, assuming that agents know the exact best response and take action according to that. Only a few papers have considered the realistic version in which agents learn game dynamics and rewards as they play. For instance look at [48, 28, 127, 18]. Elie et al. (2019) [28] focus on the realistic setting where agents have no prior information about the game and eventually learn their best response by a reinforcement learning algorithm. Having only the classical assumptions on MFGs, they prove the convergence of first-order mean-field game and provide some numerical results.

7.5 Q-learning

The main goal of reinforcement learning is to find a policy that maximizes each agent's accumulated reward without having explicit information about the reward structure or the dynamics of the environment. Q-learning is one of the most-studied reinforcement learning algorithms. In a nutshell, Q-learning estimates the value of each action at each state. Therefore, it can be used to find the optimal policy in a dynamic environment by choosing the action with the highest Q-value at each state. Under particular conditions, i.e., finite state and action spaces and bounded reward, Q-learning is proven to converge to the optimal policy [122, 111].

However, Q-learning does not perform well in MDPs with large state and action spaces. As the amount of time and memory required for exploring all state-action pairs and updating the Q-table becomes unrealistic. It also does not work in settings with continuous state or action spaces. In order to solve this issue, Google DeepMind [89] introduce deep Q-learning. In this algorithm, instead of having a Q-table, there is a neural network for approximating the Q-values. The state is given as input to the neural network and the Q-values of all possible actions are the outputs. However, this algorithm requires a considerable amount of data before reaching a practical experience, and the performance may be abysmal during training. Hester et al. (2018) [55] introduce an improved algorithm to solve this issue.

Q-learning is not a good fit for multi-agent environments because of the high number of possible states. A straightforward approach to extend Q-learning for multi-agent settings is independent Q-learning (IQL) [113]. In IQL each agent independently runs Q-learning to learn its own policy while treating other agents as part of the environment. One crucial issue is that agents' policies are changing during the training, and the environment be-

comes non-stationary from the perspective of each individual agent. This means that the dynamics that produced the data for past experience replay in Deep Q-network (DQN) no longer represent the dynamics that agents are currently learning. Therefore using past experience, which is crucial for stabilizing deep Q-learning, is no longer helpful in this situation. However, Matignon et al. (2012) [81] show with empirical evidence that IQL often works well in some cooperative settings, but there is no guarantee for its performance.

One way to tackle this issue is to limit the use of experience replay to short buffers [89] or disable the experience replay [32]. "Hyper Q-learning" [114] is a different approach that avoids non-stationarity in IQL by having each agent learn a policy that is dependent on an estimate of other agents' mixed policies (rather than base actions) as well its own policy, while other agents' strategies are estimated from observed actions via Bayesian inference. Technically, this method reduces each agent's learning problem to a single-agent problem in a stationary environment. The author did not provide any proof for convergence, but the results show that Hyper Q-learning performs well in rock-paper-scissors.

Foerster et al. (2017) [33] propose two approaches for effectively using experience replay in multi-agent RL. The first approach augments each tuple in the experience replay with the probability of the joint action in that tuple, using the current policy. The second approach is inspired by Hyper Q-learning. The main difficulty of Hyper Q-learning is that it increases the dimensionality of the Q-function, which makes learning the Q-function to be unrealistic, specially when other agents' policies are high dimensional deep neural networks. The authors show that doing so is feasible as each agent can only conditions on a low dimensional fingerprint, which only disambiguates where in the experience replay was a sample from. Tampuu et al. (2017) [112] also use deep Q-learning approach to train competing pong agents.

Hu and Wellman (2003) [61] propose Nash Q-learning, an extension of Q-learning to

general-sum stochastic games. One way to adapt Q-learning to multi-agent context is to augment the Q-function with the joint action of all agents. Given the multi-agent version of Q-function, the authors define *Nash Q-value*, as the expected sum of discounted rewards when agents follow Nash equilibrium strategies from the next time-step on. The Q-function is then updated with future Nash equilibrium payoffs, instead of agent's own maximum payoff. In this algorithm, each learning agent should maintain all other agents' Q-functions, assuming that it can observe other agents' rewards and actions. Therefore, the algorithm's complexity, in terms of space, is exponential in the number of agents. Also, the algorithm's running time is dominated by the calculation of Nash equilibrium. The computational complexity of calculating an equilibrium in multi-agent, general-sum matrix games is unknown. Consequently, Nash Q-learning is not a good fit for games with massively large number of agents.

7.6 Multi-agent Reinforcement Learning

Most reinforcement learning applications involve the participation of more than one agent. Multi-agent reinforcement learning (MARL) studies the behavior of multiple learning agents interacting in a shared environment. There is a huge body of literature on MARL [14, 13, 62, 113, 78, 108, 119, 74, 22, 54, 9, 121, 54, 22, 78, 62]. Zhang et al. (2021) [128] review theoretical results of multi-agent reinforcement learning in stochastic and extensive-form games. Learning in multi-agent environments is highly nontrivial as agents are learning concurrently, causing the environment faced by each one of them to be non-stationary. Action taken by each one of the agents affects the reward of other agents and, in the case of stochastic games, will also affect the state transitions. This violates the stationarity assumption in single-agent reinforcement learning algorithms, an assumption

that guarantees the convergence of such algorithms.

The naive development of single-agent RL algorithms, also called independent learning, may fail to converge in multi-agent environments [33]. In this learning method, each agent independently learns its own policy and treats other agents as a part of non-stationarity in the environment. However, there are some cases in which applying single-agent RL algorithms to multi-agent settings would converge to the set of Nash equilibria [76, 91, 37, 80].

Foerster et al. (2017) [34] introduce learning with opponent-learning awareness (LOLA). LOLA includes an additional term that accounts for the impact of one agent's policy on the learning step of the other agents. Instead of optimizing the expected return under the current parameters, a LOLA agent optimizes the expected return after the rest of the population update their policy with one learning step. The authors show that iterated prisoner's dilemma lead to cooperation if both agents are LOLA agents. Applied to infinitely repeated matching pennies, LOLA agents converge to the Nash equilibrium. The authors support the algorithm's performance with empirical results, but no proof is provided.

Zhang et al. (2018) [129] study the problem of multi-agent reinforcement learning (MARL) in a setting where the agents are located at the nodes of a time-varying communication network. Each agent's objective is to maximize the globally averaged return over the network by communicating with their neighbors. The setting is fully decentralized as each agent makes individual decisions and agents can only access their own rewards. The authors propose two decentralized actor-critic algorithms, one of them updates the Q-function and the other one updates the value function. They also use function approximation, which makes the algorithms suitable for settings with massively large number of agents. With linear function approximation, they provide proof for convergence of both algorithms.

7.7 Mean-field Reinforcement Learning

Most MARL algorithms have scalability issue. In such algorithms, each agent has to account for the joint action of all agents, whose dimension increases with the number of agents. One way to tackle the scalability issue in MARL algorithms with a huge number of agents is to use mean-field games models. Mean-field reinforcement learning is a method to tackle the MARL problem with a large number of agents [126]. In this algorithm, each agent only considers the average effect of the overall population or the neighboring agents. The authors develop mean-field Q-learning (MF-Q) and mean-field actor-critic (MF-AC) algorithms. The authors also provide a proof for the convergence of MF-Q to Nash equilibrium under three assumptions, and the most strict one is that all Nash equilibria in each stage game should have the same value. The downside of MF-Q is that it requires each agent to track the neighboring agents' policies, which can be intractable in the case of many agents.

Subramanian et al. (2020) [109] introduce the concept of multiple types to model agent diversity, as agents belonging to a single type have similar strategies and goals. Since agents within each type are identical, the mean-field approximation is reasonable within types. In this case, many agent interactions is reduced to N agent interactions, where each agent represents a type. They also consider two different kinds of mean-field games with types. First, games where agents have predefined types, and it is known a priori. Second, games where the type of each agent is unknown and must be learned during the game. The authors suggested novel algorithms for both types of mean-field multiple-type games, which outperform the state-of-the-art algorithms that assume all agents belong to the same type.

Chapter 8

Conclusion and Future Work

Within the course of this thesis, we set to study the application of multi-agent reinforcement learning (RL) in distributed systems. In particular, we consider a setting in which strategic clients compete over a set of heterogeneous servers. In such an environment, each client receives jobs at a fixed rate, and has to choose an appropriate server to run each job. The main goal of each client then is to minimize the average wait time. This setting is modeled as a Markov game, and we theoretically prove that the game becomes asymptotically a Markov potential game (MPG). We also further propose a novel mean-field reinforcement learning algorithm which combines mean-field Q-learning and fictitious play. Lastly, we attempt to demonstrate through rigorous experiments that our algorithm outperforms the naive development of single-agent RL, and in some cases, performs even on par with the Nash Q-learning, while also being less complex in terms of memory and computation. In addition, we also offer an empirical analysis of the convergence of our proposed algorithm as compared to the Nash equilibrium and study its performance in four benchmark examples. Many different tests and experiments is left for the future due to lack of time. Future work

concerns deeper analysis of particular scenarios. Moreover, there's still no proof provided for the convergence of our proposed algorithm to the set of Nash equilibria.

References

[1] Eitan Altman, Tamer Basar, Tania Jiménez, and Nahum Shimkin. Routing into two parallel links: Game-theoretic distributed algorithms. *Journal of Parallel and Distributed Computing*, 61(9):1367–1381, 2001.

[2] Lucas Baudin. Best-response dynamics and fictitious play in identical interest stochastic games. *arXiv preprint arXiv:2111.04317*, 2021.

[3] Alain Bensoussan, Jens Frehse, Phillip Yam, et al. *Mean field games and mean field type control theory*, volume 101. 2013.

[4] Ulrich Berger. Fictitious play in 2× n games. *Journal of Economic Theory*, 120(2):139–154, 2005.

[5] Ulrich Berger. Brown's original fictitious play. *Journal of Economic Theory*, 135(1):572–578, 2007.

[6] Ulrich Berger. Learning in games with strategic complementarities revisited. *Journal of Economic Theory*, 143(1):292–301, 2008.

[7] Ilai Bistritz, Zhengyuan Zhou, Xi Chen, Nicholas Bambos, and Jose Blanchet. Online exp3 learning in adversarial bandits with delayed feedback. *Advances in Neural Information Processing Systems*, 32, 2019.

[8] Lawrence E Blume. The statistical mechanics of best-response strategy revision. *Games and Economic Behavior*, 11(2):111–145, 1995.

[9] Vivek S Borkar. Reinforcement learning in markovian evolutionary games. *Advances in Complex Systems*, 5(01):55–72, 2002.

[10] Michael Bowling. Convergence and no-regret in multiagent learning. *Advances in neural information processing systems*, 17, 2004.

[11] Simina Brânzei, Vasilis Gkatzelis, and Ruta Mehta. Nash social welfare approximation for strategic agents. In *Proceedings of the 2017 ACM Conference on Economics and Computation*, pages 611–628, 2017.

[12] George W Brown. Iterative solution of games by fictitious play. *Act. Anal. Prod Allocation*, 13(1):374, 1951.

[13] Lucian Busoniu, Robert Babuska, and Bart 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.

[14] Lucian Buşoniu, Robert Babuška, and Bart De Schutter. Multi-agent reinforcement learning: An overview. *Innovations in Multi-agent Systems and Applications-1*, pages 183–221, 2010.

[15] Dan Calderone and S Shankar Sastry. Markov decision process routing games. In *2017 ACM/IEEE 8th International Conference on Cyber-Physical Systems (ICCPS)*, pages 273–280, 2017.

[16] Pierre Cardaliaguet. Notes on mean field games. Technical report, 2010.

[17] Pierre Cardaliaguet and Saeed Hadikhanloo. Learning in mean field games: the fictitious play. *ESAIM: Control, Optimisation and Calculus of Variations*, 23(2):569–591, 2017.

[18] Pierre Cardaliaguet and Charles-Albert Lehalle. Mean field game of controls and an application to trade crowding. *Mathematics and Financial Economics*, 12(3):335–363, 2018.

[19] René Carmona, François Delarue, et al. *Probabilistic theory of mean field games with applications I-II*. 2018.

[20] Diogo Carvalho, Francisco S Melo, and Pedro Santos. A new convergent variant of q-learning with linear function approximation. *Advances in Neural Information Processing Systems*, 33:19412–19421, 2020.

[21] Krishnendu Chatterjee, Rupak Majumdar, and Marcin Jurdziński. On nash equilibria in stochastic games. In *International Workshop on computer science logic*, pages 26–40, 2004.

[22] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. *AAAI/IAAI*, 1998(746-752):2, 1998.

[23] Vincent Conitzer and Tuomas Sandholm. Complexity results about nash equilibria. *arXiv preprint cs/0205074*, 2002.

[24] Stuart Gordon Cowan. *Dynamical systems arising from game theory*. 1992.

[25] Ross Cressman, Christopher Ansell, and Ken Binmore. *Evolutionary dynamics and extensive form games*, volume 5. 2003.

[26] Anastasios A Economides and John A Silvester. A game theory approach to cooperative and non-cooperative routing problems. In *SBT/IEEE International Symposium on Telecommunications*, pages 597–601, 1990.

[27] Anastasios A Economides, John A Silvester, et al. Multi-objective routing in integrated services networks: A game theory approach. In *Infocom*, volume 91, pages 1220–1227, 1991.

[28] Romuald Elie, Julien Pérolat, Mathieu Laurière, Matthieu Geist, and Olivier Pietquin. Approximate fictitious play for mean field games. 2019.

[29] Patrick Eschenfeldt and David Gamarnik. Join the shortest queue with many servers. the heavy-traffic asymptotics. *Mathematics of Operations Research*, 43(3):867–886, 2018.

[30] Jerzy Filar and Koos Vrieze. *Competitive Markov decision processes*. Springer Science & Business Media, 2012.

[31] Arlington M Fink. Equilibrium in a stochastic n-person game. *Journal of Science of the Hiroshima University, series ai (mathematics)*, 28(1):89–93, 1964.

[32] Jakob Foerster, Ioannis Alexandros Assael, Nando De Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 29, 2016.

[33] Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Triantafyllos Afouras, Philip HS Torr, Pushmeet Kohli, and Shimon Whiteson. Stabilising experience replay for deep multi-agent reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning(ICML)*, pages 1146–1155, 2017.

[34] Jakob N Foerster, Richard Y Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, and Igor Mordatch. Learning with opponent-learning awareness. *arXiv preprint arXiv:1709.04326*, 2017.

[35] Robert D Foley and David R McDonald. Join the shortest queue: stability and exact asymptotics. *Annals of Applied Probability*, pages 569–607, 2001.

[36] Dean P Foster and H Peyton Young. On the nonconvergence of fictitious play in coordination games. *Games and Economic Behavior*, 25(1):79–96, 1998.

[37] Roy Fox, Stephen M Mcaleer, Will Overman, and Ioannis Panageas. Independent natural policy gradient always converges in markov potential games. In *Proceedings of the 25th International Conference on Artificial Intelligence and Statistics(AISTATS)*, pages 4414–4425, 2022.

[38] Jason Gaitonde and Éva Tardos. Stability and learning in strategic queuing systems. In *Proceedings of the 21st ACM Conference on Economics and Computation*, pages 319–347, 2020.

[39] Jason Gaitonde and Eva Tardos. Virtues of patience in strategic queuing systems. In *Proceedings of the 22nd ACM Conference on Economics and Computation*, pages 520–540, 2021.

[40] Sam Ganzfried. Fictitious play outperforms counterfactual regret minimization. *arXiv preprint arXiv:2001.11165*, 2020.

[41] Kristen Gardner, Mor Harchol-Balter, Alan Scheller-Wolf, Mark Velednitsky, and Samuel Zbarsky. Redundancy-d: The power of d choices for redundancy. *Operations Research*, 65(4):1078–1094, 2017.

[42] Kristen Gardner and Cole Stephens. Smart dispatching in heterogeneous systems. *ACM SIGMETRICS Performance Evaluation Review*, 47(2):12–14, 2019.

[43] Andrea Gaunersdorfer and Josef Hofbauer. Fictitious play, shapley polygons, and the replicator equation. *Games and Economic Behavior*, 11(2):279–303, 1995.

[44] Daniel Grosu and Anthony T Chronopoulos. Noncooperative load balancing in distributed systems. *Journal of Parallel and Distributed Computing*, 65(9):1022–1034, 2005.

[45] Daniel Grosu, Anthony T Chronopoulos, and Ming-Ying Leung. Load balancing in distributed systems: An approach using cooperative games. In *In Proceedings of the 16th IEEE International Parallel and Distributed Processing Symposium*, pages 10–pp, 2002.

[46] Hongyi Guo, Zuyue Fu, Zhuoran Yang, and Zhaoran Wang. Decentralized single-timescale actor-critic on zero-sum two-player stochastic games. In *International Conference on Machine Learning*, pages 3899–3909. PMLR, 2021.

[47] Varun Gupta, Mor Harchol Balter, Karl Sigman, and Ward Whitt. Analysis of join-the-shortest-queue routing for web server farms. *Performance Evaluation*, 64(9-12):1062–1081, 2007.

[48] Saeed Hadikhanloo. *Learning in mean field games*. PhD thesis, 2018.

[49] Sunku Hahn. The convergence of fictitious play in 3× 3 games with strategic complementarities. *Economics Letters*, 64(1):57–60, 1999.

[50] Sergiu Hart and Andreu Mas-Colell. A simple adaptive procedure leading to correlated equilibrium. *Econometrica*, 68(5):1127–1150, 2000.

[51] Sergiu Hart and Andreu Mas-Colell. A reinforcement procedure leading to correlated equilibrium. In *Economics essays*, pages 181–200. 2001.

[52] Elad Hazan, Amit Agarwal, and Satyen Kale. Logarithmic regret algorithms for online convex optimization. *Machine Learning*, 69(2):169–192, 2007.

[53] Tim Hellemans and Benny Van Houdt. On the power-of-d-choices with least loaded server selection. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2(2):1–22, 2018.

[54] Pablo Hernandez-Leal, Michael Kaisers, Tim Baarslag, and Enrique Munoz de Cote. A survey of learning in multiagent environments: Dealing with non-stationarity. *arXiv preprint arXiv:1707.09183*, 2017.

[55] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. Deep q-learning from demonstrations. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[56] Josef Hofbauer. Stability for the best response dynamics. *Unpublished Manuscript, University of Vienna*, 1995.

[57] Ron Holzman and Nissan Law-Yone. Strong equilibrium in congestion games. *Games and Economic Behavior*, 21(1-2):85–101, 1997.

[58] Shlomit Hon-Snir, Dov Monderer, and Aner Sela. A learning approach to auctions. *Journal of Economic Theory*, 82(1):65–88, 1998.

[59] Illés Antal Horváth, Ziv Scully, and Benny Van Houdt. Mean field analysis of join-below-threshold load balancing for resource sharing servers. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 3(3):1–21, 2019.

[60] Ronald A Howard. Dynamic programming and markov processes. 1960.

[61] Junling Hu and Michael P Wellman. Nash q-learning for general-sum stochastic games. *Journal of Machine Learning Research*, 4(Nov):1039–1069, 2003.

[62] Junling Hu, Michael P Wellman, et al. Multiagent reinforcement learning: theoretical framework and an algorithm. In *ICML*, volume 98, pages 242–250, 1998.

[63] Minyi Huang, Roland P Malhamé, and Peter E Caines. Large population stochastic dynamic games: closed-loop mckean-vlasov systems and the nash certainty equivalence principle. *Communications in Information & Systems*, 6(3):221–252, 2006.

[64] Amir Jafari, Amy Greenwald, David Gondek, and Gunes Ercal. On no-regret learning, fictitious play, and nash equilibrium. In *ICML*, volume 1, pages 226–233, 2001.

[65] Brendan Jennings and Rolf Stadler. Resource management in clouds: Survey and research challenges. *Journal of Network and Systems Management*, 23(3):567–619, 2015.

[66] James S Jordan. Three problems in learning mixed-strategy nash equilibria. *Games and Economic Behavior*, 5(3):368–386, 1993.

[67] Hisao Kameda, Jie Li, Chonggun Kim, and Yongbing Zhang. *Optimal load balancing in distributed computer systems*. 2012.

[68] Marios Kogias and Edouard Bugnion. Hovercraft: Achieving scalability and fault-tolerance for microsecond-scale datacenter services. In *Proceedings of the Fifteenth European Conference on Computer Systems*, pages 1–17, 2020.

[69] Marios Kogias, George Prekas, Adrien Ghosn, Jonas Fietz, and Edouard Bugnion. {R2P2}: Making {RPCs} first-class datacenter citizens. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 863–880, 2019.

[70] Yannis A Korilis, Aurel A Lazar, and Ariel Orda. Capacity allocation under noncooperative routing. *IEEE Transactions on Automatic Control*, 42(3):309–325, 1997.

[71] Elias Koutsoupias and Christos Papadimitriou. Worst-case equilibria. In *Annual symposium on theoretical aspects of computer science*, pages 404–413, 1999.

[72] Vijay Krishna and Tomas Sjöström. On the convergence of fictitious play. *Mathematics of Operations Research*, 23(2):479–511, 1998.

[73] Subhashini Krishnasamy, Rajat Sen, Ramesh Johari, and Sanjay Shakkottai. Regret of queueing bandits. *Advances in Neural Information Processing Systems*, 29, 2016.

[74] Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. *Advances in Neural Information Processing Systems*, 30, 2017.

[75] Jean-Michel Lasry and Pierre-Louis Lions. Jeux à champ moyen. i–le cas stationnaire. *Comptes Rendus Mathématique*, 343(9):619–625, 2006.

[76] Stefanos Leonardos, Will Overman, Ioannis Panageas, and Georgios Piliouras. Global convergence of multi-agent policy gradient in markov potential games. *arXiv preprint arXiv:2106.01969*, 2021.

[77] Hwa-Chun Lin and Cauligi S Raghavendra. A dynamic load-balancing policy with a central job dispatcher (lbc). *IEEE Transactions on Software Engineering*, 18(2):148, 1992.

[78] Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning Proceedings 1994*, pages 157–163. 1994.

[79] Yi Lu, Qiaomin Xie, Gabriel Kliot, Alan Geller, James R Larus, and Albert Greenberg. Join-idle-queue: A novel load balancing algorithm for dynamically scalable web services. *Performance Evaluation*, 68(11):1056–1071, 2011.

[80] Sergio Valcarcel Macua, Javier Zazo, and Santiago Zazo. Learning parametric closed-loop policies for markov potential games. *arXiv preprint arXiv:1802.00899*, 2018.

[81] Laetitia Matignon, Guillaume J Laurent, and Nadine Le Fort-Piat. Independent reinforcement learners in cooperative markov games: a survey regarding coordination problems. *The Knowledge Engineering Review*, 27(1):1–31, 2012.

[82] Marios Mavronicolas and Paul Spirakis. The price of selfish routing. In *Proceedings of the thirty-third annual ACM Symposium on Theory of Computing*, pages 510–519, 2001.

[83] David H Mguni, Yutong Wu, Yali Du, Yaodong Yang, Ziyi Wang, Minne Li, Ying Wen, Joel Jennings, and Jun Wang. Learning in nonzero-sum stochastic games with potentials. In *Proceedings of the 38th International Conference on Machine Learning(ICML)*, pages 7688–7699, 2021.

[84] Igal Milchtaich. Congestion games with player-specific payoff functions. *Games and Economic Behavior*, 13(1):111–124, 1996.

[85] Paul Milgrom and John Roberts. Adaptive and sophisticated learning in normal form games. *Games and Economic Behavior*, 3(1):82–100, 1991.

[86] Michael Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1094–1104, 2001.

[87] Michael Mitzenmacher. Analyzing distributed join-idle-queue: A fluid limit approach. In *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 312–318, 2016.

[88] Koichi Miyasawa. On the convergence of the learning process in a 2 x 2 non-zero-sum two-person game. Technical report, 1961.

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

[90] Dov Monderer and Aner Sela. A2× 2game without the fictitious play property. *Games and Economic Behavior*, 14(1):144–148, 1996.

[91] Dov Monderer and Lloyd S Shapley. Fictitious play property for games with identical interests. *Journal of Economic Theory*, 68(1):258–265, 1996.

[92] Dov Monderer and Lloyd S Shapley. Potential games. *Games and Economic Behavior*, 14(1):124–143, 1996.

[93] John F Nash Jr. The bargaining problem. *Econometrica: Journal of the Econometric Society*, pages 155–162, 1950.

[94] Muhammad Anis Uddin Nasir, Gianmarco De Francisci Morales, David Garcia-Soriano, Nicolas Kourtellis, and Marco Serafini. The power of both choices: Practical load balancing for distributed stream processing engines. In *2015 IEEE 31st International Conference on Data Engineering*, pages 137–148, 2015.

[95] Ariel Orda, Raphael Rom, and Nahum Shimkin. Competitive routing in multiuser communication networks. *IEEE/ACM Transactions on Networking*, 1(5):510–521, 1993.

[96] Kay Ousterhout, Patrick Wendell, Matei Zaharia, and Ion Stoica. Sparrow: distributed, low latency scheduling. In *Proceedings of the twenty-fourth ACM symposium on operating systems principles*, pages 69–84, 2013.

[97] Julien Perolat, Bilal Piot, and Olivier Pietquin. Actor-critic fictitious play in simultaneous move multistage games. In *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics(AISTATS)*, pages 919–928, 2018.

[98] Julien Pérolat, Florian Strub, Bilal Piot, and Olivier Pietquin. Learning nash equilibrium for general-sum markov games from batch data. In *Artificial Intelligence and Statistics*, pages 232–241, 2017.

[99] Julia Robinson. An iterative method of solving a game. *Annals of Mathematics*, pages 296–301, 1951.

[100] Tim Roughgarden. Stackelberg scheduling strategies. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 104–113, 2001.

[101] Tim Roughgarden and Éva Tardos. How bad is selfish routing? *Journal of the ACM (JACM)*, 49(2):236–259, 2002.

[102] Amitabha Roy, Laurent Bindschaedler, Jasmina Malicevic, and Willy Zwaenepoel. Chaos: Scale-out graph processing from secondary storage. In *Proceedings of the 25th Symposium on Operating Systems Principles*, pages 410–424, 2015.

[103] Andrea Schaerf, Yoav Shoham, and Moshe Tennenholtz. Adaptive load balancing: A study in multi-agent learning. *Journal of Artificial Intelligence Research*, 2:475–500, 1994.

[104] Aner Sela et al. Fictitious play in 2× 3 games. *Games and Economic Behavior*, 31(1):152–162, 2000.

[105] Shahin Shahrampour, Alexander Rakhlin, and Ali Jadbabaie. Multi-armed bandits in multi-agent networks. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2786–2790, 2017.

[106] Lloyd Shapley. Some topics in two-person games. *Advances in Game Theory*, 52:1–29, 1964.

[107] Lloyd S Shapley. Stochastic games. *Proceedings of the National Academy of Sciences*, 39(10):1095–1100, 1953.

[108] Yoav Shoham, Rob Powers, and Trond Grenager. Multi-agent reinforcement learning: a critical survey. Technical report, 2003.

[109] Sriram Ganapathi Subramanian, Pascal Poupart, Matthew E Taylor, and Nidhi Hegde. Multi type mean field reinforcement learning. *arXiv preprint arXiv:2002.02513*, 2020.

[110] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. 2018.

[111] Csaba Szepesvári and Michael L Littman. A unified analysis of value-function-based reinforcement-learning algorithms. *Neural Computation*, 11(8):2017–2060, 1999.

[112] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent cooperation and competition with deep reinforcement learning. *PloS One*, 12(4):e0172395, 2017.

[113] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the 10th International Conference on Machine Learning (ICML)*, pages 330–337, 1993.

[114] Gerald Tesauro. Extending q-learning to general adaptive multi-agent systems. *Advances in Neural Information Processing Systems*, 16, 2003.

[115] Lars Thorlund-Petersen. Iterative computation of cournot equilibrium. *Games and Economic Behavior*, 2(1):61–75, 1990.

[116] Frank Thuijsman. *A survey on optimality and equilibria in stochastic games*. 1997.

[117] Michael Ummels and Dominik Wojtczak. The complexity of nash equilibria in simple stochastic multiplayer games. In *International Colloquium on Automata, Languages, and Programming*, pages 297–308, 2009.

[118] Joannes Vermorel and Mehryar Mohri. Multi-armed bandit algorithms and empirical evaluation. In *European conference on machine learning*, pages 437–448, 2005.

[119] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko

Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[120] Bernhard Von Stengel. Computing equilibria for two-person games. *Handbook of Game Theory with Economic Applications*, 3:1723–1759, 2002.

[121] Xiaofeng Wang and Tuomas Sandholm. Reinforcement learning to play an optimal nash equilibrium in team markov games. *Advances in neural information processing systems*, 15, 2002.

[122] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

[123] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

[124] Wayne Winston. Optimality of the shortest line discipline. *Journal of applied probability*, 14(1):181–189, 1977.

[125] Qiaomin Xie, Xiaobo Dong, Yi Lu, and Rayadurgam Srikant. Power of d choices for large-scale bin packing: A loss model. *ACM SIGMETRICS Performance Evaluation Review*, 43(1):321–334, 2015.

[126] Yaodong Yang, Rui Luo, Minne Li, Ming Zhou, Weinan Zhang, and Jun Wang. Mean field multi-agent reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pages 5571–5580, 2018.

[127] Huibing Yin, Prashant G Mehta, Sean P Meyn, and Uday V Shanbhag. Learning in mean-field oscillator games. In *49th IEEE Conference on Decision and Control (CDC)*, pages 3125–3132, 2010.

[128] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of Reinforcement Learning and Control*, pages 321–384, 2021.

[129] Kaiqing Zhang, Zhuoran Yang, Han Liu, Tong Zhang, and Tamer Basar. Fully decentralized multi-agent reinforcement learning with networked agents. In *Proceedings of the 35th International Conference on Machine Learning*, pages 5872–5881. PMLR, 2018.

[130] Xingyu Zhou, Fei Wu, Jian Tan, Kannan Srinivasan, and Ness Shroff. Degree of queue imbalance: Overcoming the limitation of heavy-traffic delay optimality in load balancing systems. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2(1):1–41, 2018.

[131] Hang Zhu, Kostis Kaffes, Zixu Chen, Zhenming Liu, Christos Kozyrakis, Ion Stoica, and Xin Jin. {RackSched}: A {Microsecond-Scale} scheduler for {Rack-Scale} computers. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 1225–1240, 2020.

[132] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. *Advances in neural information processing systems*, 20, 2007.

APPENDICES

Appendix A

A.1 Counterexample

In this section, we provide a counterexample of proposition 4 in [83]. Our example's setting is as follows:

- There are only two players, player 1 and 2.
- The game is infinite horizon.
- The state space is continuous and each state is a number between zero and 1.
- Action space of both players is continuous and each action is a number between zero and 1.
- State transitions according to action of player 1, e.g., if player 1 plays x, state transitions to x.

We define the reward functions such that for all $a_1, a_2 \in [0, 1]$ and $s \in [0, 1]$ we have:

$$R_1(a_1, a_2, s) = 0, \; and \tag{A.1}$$

$$R_2(a_1, a_2, s) = a_2. \tag{A.2}$$

We define the ϕ function as: $\phi(a_1, a_2, s) = R_2(a_1, a_2, s) = a_2$, so that the game is both c-SPG and state transitive:

$$R_1(a_1, a_2, s) - R_1(a_1', a_2, s) = \phi(a_1, a_2, s) - \phi(a_1', a_2, s) = a_2 - a_2 = 0, \tag{A.3}$$

$$R_1(a_1, a_2, s) - R_1(a_1, a_2, s') = \phi(a_1, a_2, s) - \phi(a_1, a_2, s') = a_2 - a_2 = 0, \tag{A.4}$$

$$R_2(a_1, a_2, s) - R_2(a_1', a_2, s) = \phi(a_1, a_2, s) - \phi(a_1', a_2, s) = 0, \tag{A.5}$$

$$R_2(a_1, a_2, s) - R_2(a_1, a_2, s') = \phi(a_1, a_2, s) - \phi(a_1, a_2, s') = 0. \tag{A.6}$$

Suppose under policy π, both players play x at state x, where $x \in [0, 1]$. Also policy π' is such that player 1 always plays 1 and player two acts the same as policy π. We also define $V_1^\pi(s)$ as the value function of player 1, starting at state s under policy π. We now have:

$$V_1^\pi(0) - V_1^{\pi'}(0) = R_1(0,0,0)(1+\gamma+\gamma^2+...) - (R_1(1,0,0)+\gamma R_1(1,1,1)+\gamma^2 R_1(1,1,1)+...) = 0, \tag{A.7}$$

$$\begin{aligned} B^\pi(0) - B^{\pi'}(0) &= \phi(0,0,0)(1+\gamma+\gamma^2+...) - (\phi(1,0,0)+\gamma\phi(1,1,1)+\gamma^2\phi(1,1,1)+...) \\ &= -(\gamma+\gamma^2+...) \\ &= -\frac{\gamma}{1-\gamma}\,. \end{aligned} \tag{A.8}$$

It is now trivial that $V_1^\pi(0) - V_1^{\pi'}(0) \neq B^\pi(0) - B^{\pi'}(0)$.

Now suppose distribution $P(.|)$ puts all the weight on state zero and nothing on other states:

$$\mathbb{E}_{s\backsim P(.|)}[V_1^\pi(s) - V_1^{\pi'}(s)] = V_1^\pi(0) - V_1^{\pi'}(0), \tag{A.9}$$

$$\mathbb{E}_{s\backsim P(.|)}[B^\pi(s) - B^{\pi'}(s)] = B^\pi(0) - B^{\pi'}(0), \tag{A.10}$$

So we have

$$\mathbb{E}_{s\backsim P(.|)}[V_1^{\pi}(s) - V_1^{\pi'}(s)] \neq \mathbb{E}_{s\backsim P(.|)}[B^{\pi}(s) - B^{\pi'}(s)]). \tag{A.11}$$

which contradicts proposition 4.

Appendix B

B.1 Bellman Equation

We can decompose the value function into two parts, the instantaneous reward and the discounted expected future value. This is the idea behind the *Bellman* equation. We can solve a complex optimization problem using *Bellman* equation by decomposing it into simpler recursive sub-problems and finding their optimal solutions. The *Bellman* equation for the value function will then be:

$$v^{\pi}(s) = \sum_{a} \pi(a|s) \left(r(s,a) + \sum_{s'} p(s'|s,a) v^{\pi}(s') \right). \tag{B.1}$$

We can now define the *Bellman* operator $T^{\pi} : \mathbb{R} \to \mathbb{R}$, in the following way:

$$\left(T^{\pi} v\right)(s) = \sum_{a} \pi(a|s) \left(r(s,a) + \sum_{s'} p(s'|s,a) v(s') \right). \tag{B.2}$$

In this way, we can rewrite the *Bellman* equation as:

$$T^{\pi} v(s) = v(s). \tag{B.3}$$

The above equation has a unique solution, $v^{\pi}(s)$.

B.2 Q-learning

We can no not use the tabular setting in case of large state or action spaces. In this case, we can represent the Q-function using a class of parameterized function approximators $\mathcal{Q} = \{Q_w | w \in \mathbb{R}^p\}$, where p is the number of parameters. We can now approximate the Q-function as:

$$Q_w(s, a) = w^T \phi(s, a), \tag{B.4}$$

where ϕ is called the feature function. The update in the function approximation setting is:

$$w^{t+1} = w^t + \alpha_t(s, a)(r + \gamma \max_{a'} Q_w^t(s', a') - Q_w^t(s, a))\nabla_w Q_w(s, a). \tag{B.5}$$

It is basically gradient descent on the loss function:

$$l(w) = \mathbb{E}_{s,a,r,s'}(r + \gamma \max_{a'} Q_w(s', a') - Q_w(s, a))^2. \tag{B.6}$$

When taking the gradient, we assume that $\max_{a'} Q_w(s', a')$ is a fixed target and is independent of w. Therefore, we can have a target Q-network to approximate $\max_{a'} Q_w(s', a')$ and update its parameters every once in a while with the original Q parameters. This will stabilize the learning procedure. This setting is also called deep Q-learning since we are parametrizing the Q-function using a neural network.

Q-learning is proved to converge to the optimal Q-function with probability one as long as

$$\sum_t \alpha_t(s, a) = \infty \, , \sum_t \alpha_t^2(s, a) < \infty. \tag{B.7}$$

We also have to ensure that $\forall t$, $0 \leq \alpha_t \leq 1$, so that all action-state pairs are visited infinitely many times. The first convergence proof is outlined in [123], and the complete proof is in [122]. [20] also proposes conditions that guarantee the convergence of deep Q-learning.

B.3 RL Algorithms

Value iteration algorithms are algorithms that iteratively determine the value of each state-action pair and assume that agent(s) take the best possible action under the current estimate of the state-action pair value. Q-learning for instance is a value iteration algorithm.

On the other hand, policy iteration algorithms first initialize a value function, $v(s)$, and a policy, $\pi(a|s)$. These estimations are then improved using two main steps, policy evaluation, and policy improvement, by iteratively fixing the policy using the value function and fixing the value function using policy.

Actor-critic is probably one of the most studied policy iteration reinforcement learning algorithms. In this setting, the value function, called the critic, estimates the value of each state, which is almost a measurement of the actor's performance. The actor, responsible for policy, then improves the performance using policy gradient. The critic also gets updated using gradient ascent. The online update of critic reduces the variance of the policy gradient and, therefore, leads to a better performance than the policy gradient.

Unlike the policy iteration method, the policy gradient learns a parametrized policy directly and without relying on the value function. The policy gradient method can be seen as an optimization problem with the objective function as:

$$J(\theta) = \mathbb{E}_{a\sim\pi^{\theta}(s)}[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s]. \tag{B.8}$$

In policy gradient, we want to find the policy parameters θ that maximize the expected future reward. We can then update θ by applying gradient ascent to maximize $J(\theta)$:

$$\theta_{t+1} = \theta_t + \alpha \nabla_{\theta} J(\theta). \tag{B.9}$$

Different policy gradient algorithms depend on how they estimate $\nabla_{\theta} J(\theta)$. Most policy gradient algorithms are known to have high variance.

Appendix C

C.1 Proof of lemma 4.0.1

Proof. With a fixed k, $w_j(s,k)$[1] is the smallest when there are no tasks in the queue. In this case, the wait time of all tasks is $\{\frac{1}{\mu_j}, \frac{2}{\mu_j}, ..., \frac{k}{\mu_j}\}$, and the average wait time is $\frac{k+1}{2\mu_j}$. So each $w_j(s,k)$ is lower bounded by $\frac{k+1}{2\mu_j}$. We now have:

$$u_s^i(a_{-i}) \triangleq -\sum_{j=1}^{m}\sum_{k=1}^{N_j(a_{-i})}\frac{1}{w_j(s,k)^2} \geq -\sum_{j=1}^{m}\sum_{k=1}^{N_j(a_{-i})}\frac{4\mu_j^2}{(k+1)^2}.$$

By rearranging the terms we get:

$$u_s^i(a_{-i}) \geq -\sum_{j=1}^{m}\sum_{k=1}^{N_j(a_{-i})}\frac{4\mu_j^2}{(k+1)^2} = -\sum_{j=1}^{m}4\mu_j^2\sum_{k=1}^{N_j(a_{-i})}\frac{1}{(k+1)^2}.$$

We know the summation of the reciprocals of the squares of the natural numbers is equal to $\frac{\pi^2}{6}$. Then:

$$u_s^i(a_{-i}) \geq -\sum_{j=1}^{m}4\mu_j^2\sum_{k=1}^{N_j(a_{-i})}\frac{1}{(k+1)^2} \geq -\sum_{j=1}^{m}4\mu_j^2(\frac{\pi^2}{6}) = -\frac{2\pi^2}{3}\sum_{j=1}^{m}\mu_j^2.$$

□

[1]The expected wait time on server j at state s, given that k tasks are sent to it.

C.2 Proof of lemma 4.0.2

Proof. Suppose $s^* = s$. At each time step, player i receives a packet with probability $\frac{\lambda}{n}$. If it does not receive a packet, the action of other players will change the state. Also, suppose the action of players is an integer in $\{1, .., m\}$, which determines the chosen server. We can write:

$$\mathbb{P}^{\pi}(s', s) = (1-\frac{\lambda}{n})\sum_{a_{-i}} \pi(a_{-i}|s)\mathbb{P}(s', s, a_{-i}) + \frac{\lambda}{n}\sum_{a_i} \pi(a_i|s) \sum_{a_{-i}} \pi(a_{-i}|s)\mathbb{P}(s', s, a_i, a_{-i}) \quad \text{(C.1)}$$

By taking derivative of $\pi_i(s^*)$ we get:

$$\nabla_{\pi_i(s^*)}\mathbb{P}^{\pi}(s', s) = \frac{\lambda}{n}\begin{bmatrix} \sum_{a_{-i}} \pi(a_{-i}|s)\mathbb{P}(s', s, 1, a_{-i}) \\ \vdots \\ \sum_{a_{-i}} \pi(a_{-i}|s)\mathbb{P}(s', s, m, a_{-i}) \end{bmatrix} \quad \text{(C.2)}$$

$$\begin{aligned} \sum_{s'} \nabla_{\pi_i(s^*)}\mathbb{P}^{\pi}(s', s) &= \frac{\lambda}{n}\sum_{s'} \begin{bmatrix} \sum_{a_{-i}} \pi(a_{-i}|s)\mathbb{P}(s', s, 1, a_{-i}) \\ \vdots \\ \sum_{a_{-i}} \pi(a_{-i}|s)\mathbb{P}(s', s, m, a_{-i}) \end{bmatrix} \\ &= \frac{\lambda}{n}\begin{bmatrix} \sum_{s'}\sum_{a_{-i}} \pi(a_{-i}|s)\mathbb{P}(s', s, 1, a_{-i}) \\ \vdots \\ \sum_{s'}\sum_{a_{-i}} \pi(a_{-i}|s)\mathbb{P}(s', s, m, a_{-i}) \end{bmatrix} \\ &= \frac{\lambda}{n}\begin{bmatrix} \sum_{a_{-i}} \pi(a_{-i}|s)\sum_{s'}\mathbb{P}(s', s, 1, a_{-i}) \\ \vdots \\ \sum_{a_{-i}} \pi(a_{-i}|s)\sum_{s'}\mathbb{P}(s', s, m, a_{-i}) \end{bmatrix} = \frac{\lambda}{n}\begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}. \end{aligned} \quad \text{(C.3)}$$

If $s^* \neq s$, the change in strategy has no effect on transitions from state s. We therefore have $\nabla_{\pi_i(s^*)}\mathbb{P}^{\pi}(s', s) = 0$. From 4.10 we have:

$$\nabla_{\pi_i(s^*)} v_T^i(s) = \gamma \sum_{s'} \mathbb{P}^{\pi}(s', s)\nabla_{\pi_i(s^*)} v_{T-1}^i(s') + \gamma \sum_{s'} v_{T-1}^i(s')\nabla_{\pi_i(s^*)}\mathbb{P}^{\pi}(s', s)). \quad \text{(C.4)}$$

Using C.3, we can write the second term as:

$$\sum_{s'} v^i_{T-1}(s')\nabla_{\pi_i(s^*)}\mathbb{P}^{\pi}(s',s) \geq \hat{v}_{T-1}\sum_{s'}\nabla_{\pi_i(s^*)}\mathbb{P}^{\pi}(s',s) \geq \frac{\lambda}{n}\hat{v}_{T-1}\mathbf{1}\,. \tag{C.5}$$

Where $\hat{v}_{T-1}$ is the minimum of $v^i_{T-1}(.)$ over all states. This lower bound exists since u has a lower bound (see lemma 4.0.1).

So we have:

$$\nabla_{\pi_i(s^*)}v^i_T(s) \geq \gamma\sum_{s'}\mathbb{P}^{\pi}(s',s)\nabla_{\pi_i(s^*)}v^i_{T-1}(s') + \gamma\frac{\lambda}{n}\hat{v}_{T-1}\mathbf{1}. \tag{C.6}$$

We now use proof by induction. For t=0, according to 4.11:

$$\nabla_{\pi_i(s^*)}v^i_0(s) = \nabla_{\pi_i(s^*)}\mathbb{E}_{a_{-i}\sim\pi_{-i}(s)}[u^i_s(a_{-i,0})] = 0. \tag{C.7}$$

So the statement holds. For t=1, according to C.6:

$$\nabla_{\pi_i(s^*)}v^i_1(s) \geq \gamma\sum_{s'}\mathbb{P}^{\pi}(s',s)\nabla_{\pi_i(s^*)}v^i_0(s') + \gamma\frac{\lambda}{n}\hat{v}_0\mathbf{1}. \tag{C.8}$$

Again by using 4.11 we have:

$$\nabla_{\pi_i(s^*)}v^i_1(s) \geq \gamma\frac{\lambda}{n}\hat{v}_0\mathbf{1}. \tag{C.9}$$

Therefore, the base case is proved. Now suppose the statement holds for an arbitrary t, i.e., $\nabla_{\pi_i(s^*)}v^i_t(s) \geq \sum_{i=1}^{t}\gamma^i\frac{\lambda}{n}\hat{v}_{t-i}1$, we now want to show that it also holds for t+1. By the induction hypothesis we have:

$$\begin{aligned}
\nabla_{\pi_i(s^*)}v^i_{t+1}(s) &\geq \gamma\sum_{s'}\mathbb{P}^{\pi}(s',s)\nabla_{\pi_i(s^*)}v^i_t(s') + \gamma\frac{\lambda}{n}\hat{v}_t\mathbf{1} \\
&\geq \gamma\sum_{s'}\mathbb{P}^{\pi}(s',s)\sum_{i=1}^{t}\gamma^i\frac{\lambda}{n}\hat{v}_{t-i}\mathbf{1} + \gamma\frac{\lambda}{n}\hat{v}_t\mathbf{1} \\
&= \gamma\sum_{i=1}^{t}\gamma^i\frac{\lambda}{n}\hat{v}_{t-i}\mathbf{1} + \gamma\frac{\lambda}{n}\hat{v}_t\mathbf{1} \\
&= \sum_{i=1}^{t+1}\gamma^i\frac{\lambda}{n}\hat{v}_{t+1-i}\mathbf{1}.
\end{aligned} \tag{C.10}$$

This shows that the statement holds for t+1. Therefore, by proof by induction, we can conclude the statement is true for all $0 \leq t \leq T$.

□

C.3 Proof of lemma 4.0.3

Proof. We use proof by induction. Since $\nabla_{\pi_i(s^*)} v_0^i(s) = \nabla_{\pi_i(s^*)} \mathbb{E}_{a_{-i} \sim \pi_{-i}(s)}[u_s^i(a_{-i,0})] = 0$ (4.11), the base case is proved. Now suppose the statement holds for an arbitrary t, i.e., $\nabla_{\pi_i(s^*)} v_t^i(s) \leq 0$. We want to show that it also holds for t+1. Similar to 4.10, we have:

$$\nabla_{\pi_i(s^*)} v_{t+1}^i(s) = \gamma \sum_{s'} \mathbb{P}^\pi(s', s) \nabla_{\pi_i(s^*)} v_t^i(s') + \gamma \sum_{s'} v_t^i(s') \nabla_{\pi_i(s^*)} \mathbb{P}^\pi(s', s)). \tag{C.11}$$

But we know that $v_t^i(s') \leq 0$, and from induction hypothesis we have $\nabla_{\pi_i(s^*)} v_t^i(s') \leq 0$. Moreover, $\mathbb{P}^\pi(s', s)$ and $\nabla_{\pi_i(s^*)} \mathbb{P}^\pi(s', s)$ are always positive. Therefore, all the terms on the right side of C.11 are non-positive. This means:

$$\nabla_{\pi_i(s^*)} v_{t+1}^i(s) = \gamma \sum_{s'} \mathbb{P}^\pi(s', s) \nabla_{\pi_i(s^*)} v_t^i(s') + \gamma \sum_{s'} v_t^i(s') \nabla_{\pi_i(s^*)} \mathbb{P}^\pi(s', s)) \leq 0. \tag{C.12}$$

This shows that the statement holds for t+1. Therefore, by proof by induction, we can conclude that the statement is true for all $0 \leq t \leq T$. □