A Modified Q-Learning Algorithm in Games

by

Yatao Wang

A thesis submitted in conformity with the requirements for the degree of Master of Applied Science
Graduate Department of Electrical and Computer Engineering
University of Toronto

© Copyright 2014 by Yatao Wang
Abstract

A Modified Q-Learning Algorithm in Games

Yatao Wang

Master of Applied Science
Graduate Department of Electrical and Computer Engineering
University of Toronto
2014

This thesis presents a modified Q-learning algorithm and provides conditions for convergence to a pure Nash equilibrium in potential games. In general Q-learning schemes, convergence to a Nash equilibrium may require decreasing step-sizes and long learning time. In this thesis, we consider a modified Q-learning algorithm based on constant step-sizes, inspired by Joint Strategy Fictitious Play (JSFP). When compared to JSFP, the Q-learning with constant step-sizes requires less information aggregation, but only reaches an approximation of a Nash equilibrium. We show that by appropriately choosing frequency dependent step-sizes, sufficient exploration of all actions is ensured and the estimated equilibrium approaches a Nash equilibrium.
Acknowledgements

First and foremost, I would like to thank my supervisor, Professor Lacra Pavel. Her insightful suggestions and passion for research have greatly inspired me in numerous ways. The constant support and guidance from her throughout the course is what made this thesis possible.

Next, I wish to express my sincere gratitude to my parents and brother, who have always stood by me in times of distress. Their continuous encouragement is what motivated, and will continue to motivate me to keep going.

Finally I would like to thank the Department of Electrical and Computer Engineering for the opportunity and support of the completion of this thesis. I have gained invaluable experience from it.
## Contents

List of Tables vii

List of Figures viii

List of Symbols ix

1 Introduction 1
   1.1 Motivation ................................................. 1
   1.2 Literature Review .......................................... 3
   1.3 Contribution .............................................. 4
   1.4 Organization ............................................. 5

2 Background 6
   2.1 Learning Game Theory ................................. 6
      2.1.1 Non-cooperative finite strategic-form games .......... 6
      2.1.2 Potential Games ...................................... 9
      2.1.3 Repeated games ...................................... 11
      2.1.4 Special Example: Congestion Games ................. 11
   2.2 Algorithms of Learning Game Theory .................. 12
      2.2.1 Fictitious Play .................................... 15
      2.2.2 Joint Strategy Fictitious Play .................... 16
      2.2.3 Perturbed Reinforcement Learning .................. 17
      2.2.4 Individual Q-Learning ............................. 19
      2.2.5 Summary ........................................... 20
B.3 Simulation Code for Reinforcement Learning Algorithm . . . . . . . . . . 75
B.4 Simulation Code for Individual Q-Learning Algorithm with Multiple-Timescales 79
B.5 Simulation Code for Plot Generation . . . . . . . . . . . . . . . . . . . . 82
## List of Tables

2.1 Learning Algorithms in Games Comparison ........................................ 21

6.1 Statistic Results of Potential Ratio and Total Utility Ratio between Q-Learning and JSFP in 1000 Different Congestion Games ..................... 53

6.2 Potential and Total Utility at Time 200 between Q-Learning and JSFP in One Sample Congestion Game .................................................. 53
List of Figures

6.1 Unperturbed MQL with $\mu = 0.97$, $\alpha = 0.5$ ........................................ 52
6.2 FMJSFP-I with $\mu = 0.03$, $\alpha = 0.5$ .................................................. 52
6.3 Perturbed MQL with $\zeta = 0.9999$, $\xi = 0.01$ ........................................ 55
6.4 RL with $\nu = 0.78$ ................................................................. 55
6.5 IQL with Multi-Timescales .......................................................... 56
6.6 (Expected) Potential Value of All Learning Process ....................... 57
List of Symbols

\( \mathcal{G} \) \hspace{1cm} A finite strategic-form game
\( N \) \hspace{1cm} Number of players in \( \mathcal{G} \)
\( \mathcal{I} = \{1, \ldots, N\} \) \hspace{1cm} Set of all \( N \) players
\( \mathcal{A}_i \) \hspace{1cm} Finite action set of player \( i \)
\( \mathcal{M}_i = \{1, \ldots, |\mathcal{A}_i|\} \) \hspace{1cm} Index set of actions of player \( i \)
\( \Delta_i \) \hspace{1cm} The probability simplex of player \( i \) in \( \mathbb{R}^{|\mathcal{A}_i|} \)
\( a_i(k) \in \mathcal{A}_i \) \hspace{1cm} Action of player \( i \) at time \( k \)
\( \mathbf{e}_{ij} \in \Delta_i \) \hspace{1cm} Pure strategy of player \( i \), \( j \)-th unit vector, \( j \in \mathcal{M}_i \)
\( x_i(k) \in \Delta_i \) \hspace{1cm} Mixed-strategy of player \( i \) at time \( k \)
\( \mathbf{1}_{ij}/|\mathcal{A}_i| \in \Delta_i \) \hspace{1cm} Mixed-strategy of player \( i \) based on uniform distribution on each action
\( \mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_N \) \hspace{1cm} Joint action set
\( a \in \mathcal{A} \) \hspace{1cm} Joint action (pure-strategy profile)
\( \Delta = \Delta_1 \times \cdots \times \Delta_N \) \hspace{1cm} Joint probability simplex in \( \mathbb{R}^{|\mathcal{A}_i| \times N} \)
\( x \in \Delta \) \hspace{1cm} Mixed-strategy profile
\( a^* \in \mathcal{A} \) \hspace{1cm} Pure Nash equilibrium
\( x^* \in \Delta \) \hspace{1cm} Mixed-strategy Nash equilibrium
\( \mathcal{U}_i(a) \in \mathbb{R} \) \hspace{1cm} Utility of player \( i \) for joint action \( a \)
\( \mathcal{U}_i(x) \in \mathbb{R} \) \hspace{1cm} Expected utility of player \( i \) for joint mixed-strategy (profile) \( x \)
\( P(a) \in \mathbb{R} \) \hspace{1cm} Potential value for joint action \( a \)
\( Q_i(a) \in \mathbb{R}^{|\mathcal{A}_i|} \) \hspace{1cm} Q-vector of player \( i \) for joint action \( a \)
Chapter 1

Introduction

For centuries, human beings have been facing choices and learning from previous experiences. Nowadays, not only behavioral scientists are interested in the learning behavior of human beings, researchers in engineering also imitate human learning behaviors in the field of machine learning, [24]. The topic of this thesis is a modified Q-Learning algorithm, modeling a “myopic” learning process under the game theoretical framework, with minimum information gathered. In this chapter, we will first discuss the motivation of our study of the modified Q-Learning algorithm problem. Then we will briefly review the literature on different learning algorithms in games. Finally, we will state the contribution and the organization of this thesis.

1.1 Motivation

To find a perfect candidate for a position, we always want someone experienced. But how can we make machines “experienced” about the repetitive work they do? The idea to learn by interacting with the environment is probably the first to occur when one thinks about the nature of learning. Evolution has equipped animals, and even some plants with varies type of sensations to interact with and adapting to the changes of the surroundings. An infant could gradually learn to interact with its environment by touching and observing, without any instructions. Therefore, it comes handy and intuitive that when we want to learn about learning, we learn from the mother nature,
and we learn by interacting with the environment.

Before we start to design a learning algorithm for machines, we need a computational model so that machines can comprehend the signals in the environment. We picked value-based learning, because it is a simple, straightforward approach to understand the sophisticated environment, that is justified by recent neuroscience studies, as well as our common sense. Neuro-physiological data from rhesus monkeys trying to perform a repetitive task suggests that value-based learning is actually employed in nature. In reality, people build up survey/review applications for hotels, restaurants, and even applications to learn about the general utilities/ratings of users.

With value-based learning, the large-scale repeated games is ideal for designing distributed algorithms that allow machines to interact with each other and achieve global optimization over the network. However, the main challenges must be addressed.

First, to achieve an optimal solution, information gathering is a must. As we might have learned from our experience, the ratings of other people can only be considered as a reference. The data collected could be biased, misleading, unavailable or even false. In a large-scale game players are inherently faced with limited observational and computational capabilities. An open research problem is how to design learning algorithms with minimum information requirements for each player. Therefore, when performing a repetitive task, it is sometimes more energy-economic and space-efficient to build a “review system” based on the observations of previous observations. Especially today, as the scale and volume of data transporting over the networks keeps growing exponentially.

Second, to achieve a fast convergence, parameters need to be chosen carefully. Not only the form of the algorithm affects the learning time, the step-sizes are also crucial influence. Most prior results in Q-learning primarily focused on diminishing step-sizes. This allows the application of powerful results in stochastic approximation for studying the long-term stochastic processes’ behavior via their ODE approximations, [12]. However, with accurate measure of the utility received, the noise averaging effect of diminishing step-sizes is not helpful. By choosing appropriate constant step-sizes, we might gain faster convergence as well as adaptability introduced by “fading memory”.

Last, in every learning algorithm, a fine balance of exploration and exploitation is
Chapter 1. Introduction

essential. The perturbation functions are usually carefully designed to suit the needs of the learning algorithm. A fine perturbation function preferably is decoupled from dynamics of the learning algorithm, adjustable and have minimum effect on perturbing the optimal solution.

1.2 Literature Review

A variety of algorithms have been proposed as well as analysis of the long-term behavior and convergence to Nash equilibria in large-scale repeated games. Guaranteed convergence to a Nash equilibrium in potential games has been shown for adaptive play and the broad class of finite-memory better-reply processes [24], [25]. However, the assumption is that the agents’ rewards (utilities) for different joint action profiles is a-priori known. In many large-scale games this assumption on the utilities is not realistic.

Among the most studied learning algorithms is the well-known fictitious-play (FP), [4], and it is known to be convergent to a Nash equilibrium in potential games.

Recently, an elegant variant of Fictitious Play called Joint Strategy Fictitious Play (JSFP) has been proposed in [17], as a plausible decision making model in large-scale potential games. The authors showed its beneficial features when applied to the large-scale congestion game. However, while reducing the information requirement, players still have to monitor the joint actions and need to know their own utility so as to find their optimal actions.

Players often have also limited information on the analytical structure of their own utility function, an even more challenging problem. Such problem setting of games with unknown utilities (rewards) and unobserved opponent actions, is a natural setup for Reinforcement Learning (RL) algorithms, or payoff-based dynamics, [23]. In [18], the authors investigate payoff-based dynamics that converge to pure-strategy Nash equilibria in weakly acyclic games, one of which, sample experimentation dynamics, can admit perturbations in agents’ rewards.

Although Q-learning and the action adaptation processes are well understood independently, the combined problem of learning Nash equilibria in games with unknown
Chapter 1. Introduction

reward functions is less well understood. In [7], the authors specify a joint action learner (JAL), in which each player keeps track of the frequency of other players’ actions, while updating the utility (reward) estimate for the joint action played. However, the authors do not provide convergence conditions for their algorithm. Their investigation relies on experimental evidence of convergence, and, furthermore, it is restricted to team games, i.e., with a common utility function. Convergence is proved for identical interest games in [13] assuming that learning takes place at multiple time scales. In [14], players use variants of the Q-learning procedure independent of each other, oblivious of the effects of changes in other players’ actions on their own payoffs. Convergence results to a Nash distribution were developed in [14] under the assumption of multiple time-scales. Specifically, almost sure convergence to a Nash distribution was shown in two-player zero-sum games and in N-player partnership games. In [11], [9] a continuous approximation of the discrete time system was introduced to study the dynamics of Q-learning in several examples.

Diminishing step-sizes are also used in the recent Q-learning schemes in [5], where convergence results are obtained without the ODE approximation. However the size of the learning problem faced by the agents grows exponentially with the number of players, thereby reducing the usefulness in large games. Moreover, diminishing step-sizes result in a long learning time and slow convergence.

1.3 Contribution

Inspired by the work mentioned above, this thesis represents an effort in designing of modified Q-Learning algorithm in games and its convergence analysis. Specifically, our contribution is an algorithm that combines the strengths of Q-learning in terms of minimal information requirements, while at the same time achieving faster convergence, albeit to a near-optimal (approximate) Nash equilibrium. Our standing assumption is that players do not have information about the actions of the other players, and, moreover, they do not have complete information of their own payoff structure. We consider a modified Q-learning algorithm with constant step-size and develop some convergence results for
potential games. The trade-off is convergence to near-optimal Nash equilibrium.

We achieve a faster convergence for the modified Q-learning algorithm by introducing non-negligible, constant step-sizes to reach a sub-optimal state in a reasonably short time, and then approach a Nash equilibrium via a slightly modified perturbation function as in [6]. The main challenge is proving convergence to Nash equilibria without the averaging effect of stochastic approximation, as this results in long convergence time. Our analysis techniques are similar to those used in the JSFP case [17]. However, when compared to Fictitious Play algorithms, the setup here is complicated by players’ lack of information on the analytical structure of their own utility.

1.4 Organization

The thesis is organized as follows. Chapter 2 introduces important definitions, theorems, and algorithms in learning in games used in this thesis. Then Chapter 3 introduces a modified Q-learning scheme with a state-based perturbation function. Chapter 4 provides analysis on the converging behavior of the algorithm. In Chapter 5 we introduce a continuous-time approximation of the algorithm and analyze its stationary points. Chapter 6 presents numerical results for a traffic congestion game, and finally in Chapter 7, we summarizes the important findings and conclude the thesis.
Chapter 2

Background

In this chapter, we provide some fundamental concepts and results of learning algorithms in games mentioned in the previous chapter. They will help formulate the modified Q-Learning algorithm under a game theoretical framework in a rigorous manner. We start with the first of the two essential parts of learning in games: the game theory framework. Game theory framework has proven to be a powerful tool for analyzing the system dynamics of multi-agent systems.

2.1 Learning Game Theory

Learning in games abandons the demanding assumptions of classical game theory on players’ rationality and beliefs. However, learning game theory assumes that individual players adapt, learning over time about the game and the behavior of others (e.g. through reinforcement, imitation, or belief updating). Therefore, instead of immediately playing a perfect move, players adapt their strategy based on the outcomes of previous matches, hence a classical game with learning, or learning in games.

2.1.1 Non-cooperative finite strategic-form games

A non-cooperative finite strategic-form game $\mathcal{G}$ involves a set $\mathcal{I}$ of $N$ players, $\mathcal{I} = \{1, \ldots, N\}$, where each player $i \in \mathcal{I}$ has a finite action set $\mathcal{A}_i$ and a utility function...
Chapter 2. Background

$U_i : \mathcal{A} \to \mathbb{R}$, with $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_N$ denoting the joint-action set.

Let $a_i \in \mathcal{A}_i$ be an action of player (agent) $i \in \mathcal{I}$, and $a = (a_1, a_2, \ldots, a_N) \in \mathcal{A}$ the joint action profile of all players. Let $|\mathcal{A}_i|$ denote the cardinality of the set $\mathcal{A}_i$, and $\mathcal{M}_i = \{1, \ldots, |\mathcal{A}_i|\}$, the index set of player’s $i$ actions. Sometimes we use index notation and write $a_i = e_{ij}$, $j \in \mathcal{M}_i$, to indicate the $j^{th}$ action selected by player $i$. Let $a_{-i}$ denote the profile of actions for players other than player $i$, i.e., $a_{-i} = (a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_N)$.

With this notation, we often write a joint action profile $a$ as $\left( a_i, a_{-i} \right)$, or as $\left( e_{ij}, a_{-i} \right)$, if index notation is used. Similarly, we write $U_i(a)$ as $U_i(a_i, a_{-i})$ or $U_i(e_{ij}, a_{-i})$. A player’s goal is to maximize its own utility conditional on the choices of its opponents. A best response (BR) correspondence $\beta_i(a_{-i})$ is the set of optimal strategies for player $i$, given the strategy profile of its opponents,

$$\beta_i(a_{-i}) = \{ a_i^* \in \mathcal{A}_i : U_i(a_i^*, a_{-i}) = \max_{a_i \in \mathcal{A}_i} U_i(a_i, a_{-i}) \},$$

or, in index notation,

$$\beta_i(a_{-i}) = \{ e_{ij}^* \in \mathcal{M}_i : U_i(e_{ij}^*, a_{-i}) = \max_{j \in \mathcal{M}_i} U_i(e_{ij}, a_{-i}) \}.$$  \hfill (2.1)

**Definition 2.1.1.** Given a game $\mathcal{G}$, a strategy $N$-tuple (profile) $a^* = (a_1^*, \ldots, a_N^*)$ is said to be a pure Nash equilibrium if and only if

$$U_i(a_i^*, a_{-i}^*) \geq U_i(a_i, a_{-i}^*), \quad \forall a_i \in \mathcal{A}_i, \forall i \in \mathcal{I}.$$  

In a Nash equilibrium, each agent plays a best response, $a_i^* \in \beta_i(a_{-i}^*)$, for all $i \in \mathcal{I}$. Hence, no individual player has an incentive to unilaterally change its strategy. A Nash equilibrium ($N$-tuple) profile, $a^* \in \mathcal{A}$, is a fixed-point $a^* \in \beta(a^*)$ of the overall $N$-tuple best-response correspondence, where $\beta := (\beta_1, \ldots, \beta_N)$. Thus $a^* \in \mathcal{A}$ is called a pure Nash Equilibrium, if for all players $i \in \mathcal{I}$, $U_i(a_i^*, a_{-i}^*) = \max_{a_i \in \mathcal{A}_i} U_i(a_i, a_{-i}^*)$.

Sometimes, a mixed strategy is used where each player randomly chooses between several actions. Let $x_{ia_i}$ or $x_{ij} \in [0,1]$ denote the probability that player $i \in \mathcal{I}$ selects action $a_i = e_{ij}$ in the action set $\mathcal{A}_i$. Then $x_i = (x_{i1}, \ldots, x_{i|\mathcal{A}_i|})$ is a probability distribution...
on its action set $\mathcal{A}_i$, or a mixed-strategy for player $i \in \mathcal{I}$. Hence, a mixed strategy $x_i$ is an element of $\Delta_i$, where

$$
\Delta_i = \{ x_i \in \mathbb{R}^{\left|\mathcal{A}_i\right|} | 1^T x_i = 1, x_{ij} \geq 0, \forall j \in \mathcal{M}_i \},
$$

with 1 denoting the all ones vector, is the set of probability distributions over $\mathcal{A}_i$. The vertices of $\Delta_i$ are the unit vectors $e_{ij}$. When $x_i = e_{ij}$, player $i$ chooses the $j^{th}$ action with probability one and such a strategy is called a pure strategy. Hence, using index notation, $a_i$ can simply be identified by unit vectors $e_{ij}$ in $\Delta_i$, and the action set $\mathcal{A}_i$ by the vertices of the simplex $\Delta_i$.

Likewise, we denote by $x = (x_1, x_2, \ldots, x_N) \in \Delta$ the mixed-strategy profile of all players, where $\Delta = \Delta_1 \times \Delta_2 \times \cdots \times \Delta_N$, and $x = (x_i, x_{-i})$. All the players choose their actions based on $x_i$ independently. Therefore, given a mixed-strategy profile $x \in \Delta$, the expected utility of player $i$, also denoted $U_i$, is the multi-linear extension from $\mathcal{A}$ to $\Delta$, given as

$$
U_i(x) = \sum_{a \in \mathcal{A}} (\prod_{i \in \mathcal{I}} x_{sa_s}) U_i(a_i, a_{-i}),
$$

where each element $x_{sa_s}$ is the probability of player $s$ playing $a_s$. Equivalently, $U_i(x) = \sum_{j \in \mathcal{M}_i} x_{ij} U_i(e_{ij}, x_{-i})$, where $U_i(e_{ij}, x_{-i}) = \sum_{a_{-i} \in \mathcal{A}_{-i}} (\prod_{s \in \mathcal{I}/\{i\}} x_{sa_s}) U_i(e_{ij}, a_{-i})$ and $\mathcal{I}/\{i\}$ denotes the set of players other than player $i$.

Similarly, a mixed-strategy profile $x^* \in \Delta$ is called a mixed-strategy Nash Equilibrium, if for all players $i \in \mathcal{I}$

$$
U_i(x_i^*, x_{-i}^*) \geq U_i(x_i^*, x_{-i}^*), \quad \forall x_i \in \Delta_i, x_i \neq x_i^*.
$$

In the case when the inequality is strict, the Nash equilibrium is called a strict Nash equilibrium. Such a Nash equilibrium is a fixed-point of the mixed-strategy best-response extension, i.e., $x_i^* \in \beta_i(x_{-i}^*)$, for all $i \in \mathcal{I}$, where

$$
\beta_i(x_{-i}) = \{ x_i^* \in \Delta_i : U_i(x_i^*, x_{-i}) \geq U_i(x_i, x_{-i}), \forall x_i \in \Delta_i \},
$$
where $\beta_i(x_{-i})$ is the best-response set. To avoid set-valued mappings (multiple elements in the best-response set), a smooth best-response can be used, e.g.

$$\sigma_i(x_{-i}) = \arg \max_{x_i \in \Delta_i} \left\{ \sum_{j \in M_i} x_{ia} U_i(e_{ij}, x_{-i}) + \tau v_i(x_i) \right\},$$

where $\tau > 0$ is a temperature parameter and $v_i : \Delta_i \to \mathbb{R}$ is a player-dependent smoothing function. One example is Boltzmann selection, $v_i(x_i) = -\sum_{j \in M_i} x_{ij} \log x_{ij}$. This results in the smooth best-response (SBR) function

$$\sigma_i(x_{-i}) = \frac{e^{\tau^{-1}U_i(e_{ij}, x_{-i})}}{\sum_{j \in M_i} e^{\tau^{-1}U_i(e_{ij}, x_{-i})}}. \quad (2.2)$$

As the temperature parameter $\tau \to 0$, this set approaches the set of best responses.

### 2.1.2 Potential Games

Potential games, [19], are an important class of games that can be used as the template for the design of decentralized algorithms in large-scale problems, [22], [1]. One of the most known examples is the problem of distributed traffic routing modelled as a large-scale congestion game, [21]. In such a game, a large number of vehicles or agents makes daily routing decisions to optimize their own objectives in response to their own observations.

A potential game is characterized by a single function, called the potential function, which specifies the players’ joint preference over outcomes [19].

**Definition 2.1.2.** A function $P : \mathcal{A} \to \mathbb{R}$ is a potential for the game $\mathcal{G}$ if for all $i \in \mathcal{I}$, for all $a_{-i} \in \mathcal{A}_{-i}$,

$$P(a_i, a_{-i}) - P(a'_i, a_{-i}) = U_i(a_i, a_{-i}) - U_i(a'_i, a_{-i}), \quad (2.3)$$

for all $a_i, a'_i \in \mathcal{A}_i$, or, equivalently,

$$P(e_{ij}, a_{-i}) - P(e_{ij}', a_{-i}) = U_i(e_{ij}, a_{-i}) - U_i(e_{ij}', a_{-i})$$
for all $j,j' \in M_i$. A game that admits such a potential function is called a potential game.

The local optima of the potential function are Nash equilibria of the game. Intuitively, a potential is a function of action profiles such that the difference induced by a unilateral deviation equals the change in the deviator’s payoff. The existence of a potential function for a game implies a strict joint preference ordering over game outcomes, which in turn, ensures that the game has a number of useful properties.

**Theorem 2.1.1.** Every finite potential game possesses at least one pure-strategy equilibrium [19].

**Proof.** Suppose $P$ is a potential for a game $G$. The equilibrium set of $G$ corresponds to the set of local maxima of $P$. That is, $a^*$ is an equilibrium point for $G$ if and only if for every $i \in I$,

$$P(a^*) \geq P(a_i', a_{-i}) \quad \forall a_i' \in A_i.$$  

Consequently, if $P$ admits a maximal value in $A$ (which is true by definition for a finite $A$), then $P$ possesses a pure strategy Nash equilibrium. \hfill \Box

A step in a game $G$ is a change in one player’s strategy. An improvement step in $G$ is a change in one player’s strategy such that its utility is improved. A path in $G$ is a sequence of steps, $(a(0), a(1), \ldots, a(T))$ in which exactly one player changes their strategy at each step. A path has an initial point, $a(0)$, and if it is of finite length $T$, a terminal point $a(T)$. A path is an improvement path in $G$ if for all $k$, $U_i(a(k-1)) < U_i(a(k))$ for the deviating player $i$ at step $k$. A game $G$ is said to have the finite improvement property if every improvement path is finite.

**Theorem 2.1.2.** Every improvement path in an ordinal potential game is finite [19].

**Proof.** For every improvement path $\phi = (a_0, a_1, a_2, \ldots)$ we have,

$$P(a_0) < P(a_1) < P(a_2) < \ldots.$$
Since $\mathcal{A}$ is a finite set, the sequence $\phi$ must be finite.

The finite improvement property ensures convergence to a Nash equilibrium in finite time for players who choose “better responses”. These results ensure that a number of simple adaptive processes converge to a pure-strategy Nash equilibrium.

2.1.3 Repeated games

The problem of distributed traffic routing can be modeled as a large-scale repeated game. In a repeated game, agents or players update and adapt their strategies depending on the opponents’ actions in the previous stage(s) of the game. The dynamics of learning in repeated games is an area of active interest, [8], [25]. In this section, we review some key definitions and theorems in non-cooperative games, and give a special discussion of congestion games.

In a repeated version of the game $\mathcal{G}$, at every iteration $k \in \{0, 1, 2, \ldots \}$, each player $i \in \mathcal{I}$ selects an action $a_i(k) \in \mathcal{A}_i$ and receives the utility $U_i(a(k))$, a function of the joint actions $a(k) = (a_1(k), \ldots, a_N(k))$ of all players. Each player $i \in \mathcal{I}$ chooses action $a_i(k)$ according to the probability distribution $x_i(k)$. This selection is a function of the information and observations available to player $i \in \mathcal{I}$ up to iteration $k$. Both the action selection function and the available information depend on the learning process.

2.1.4 Special Example: Congestion Games

Congestion games are a special type of games that are closely related to potential games. In the following part, we briefly discuss key definitions and propositions of congestion games.

A congestion game $\mathcal{CG}(\mathcal{I}, \mathcal{R}, \mathcal{A}, \mathcal{U})$ is defined with one more set than general strategic games. $\mathcal{R}$ denotes the set of facilities, and $a_i \in \mathcal{A}_i$, the route, is actually a nonempty subset of $\mathcal{R}$. 
Chapter 2. Background

\[ U_i = \sum_{r \in A_i} c_r(n_r(A)), \]

where \( c_r \in \mathbb{R}^{\{1, \ldots, N\}} \) denote the vector of payoffs on each facility, and \( n_r(A) \) denotes the number of users of facility \( r \).

**Proposition 2.1.1.** [19] Every congestion game defined as \( CG \), is a potential game.

*Proof.* For each \( a \in A \), define

\[ P(a) = \sum_{r \in \bigcup_{i=1}^N a_i} \left( \sum_{n=1}^{n_r(A)} c_r(n) \right). \]

The proof that \( P \) is a potential for \( CG \) can be deduced from [21].

It is worth noting that every finite potential game is isomorphic to a congestion game[19]. We use a congestion game setup for simulations to compare the performance of algorithms in Chapter 6.

## 2.2 Algorithms of Learning Game Theory

In this section, we introduce some representative algorithms of learning game theory (LGT) discussed in the previous chapter, to further illustrate their properties and theorems. For convenience, we use similar terms as introduced in the previous section, and those to be introduced in Chapter 3.

If the player knows the functional form (analytical structure) of its utility function \( U_i \) and can observe the actions of the other players at each step, then the fictitious play (FP) algorithm can be used. In a FP algorithm, each player computes the empirical frequency vectors \( q_{-i}(k) \) of its opponents, and chooses its action as the best-response against this vector. Hence

\[ \beta_i(q_{-i}(k)) := \{ a_i^* \in A_i : U_i(a_i^*, q_{-i}(k)) = \max_{a_i \in A_i} U_i(a_i, q_{-i}) \}. \]
In case of a non-unique best-response, a random selection from the set $\beta_i(q_{-i}(k))$ is made as an action selection. Alternatively, the use of a smooth best-response leads to a smooth fictitious-play (sFP) algorithm. The empirical frequencies generated by the FP converge to a Nash equilibrium in potential games [19], but require observations of the individual actions of all other players. This is relaxed in joint strategy FP (JSFP) [17], where each player tracks the empirical frequencies of the joint actions of all other players. In JSFP, the action of a player is based on the assumption that other players are playing randomly but jointly according to their joint empirical frequencies, i.e., each player views all other players as a collective group.

In cases in which the utility function is not known, a useful approach is to use a Q-learning algorithm in order to estimate the reward (utility) function in a recursive manner [23]. In this case, the action selection is based on estimated utilities, or Q-values, which characterize the relative utility of a particular action. Either a best-response or a smooth best-response can be used as the action selection mechanism. The task of a Q-learning agent is to learn a mapping from environment to actions so as to maximize a numerical utility or reward signal. One of the attractive features of Q-learning is the fact that it does not assume knowledge about utility or reward functions, rather these must be learned from the environment. In each step, the player receives a signal from the environment indicating its state and chooses an action. Once the action is performed, it changes the environment, generating a reinforcement signal that is then used to evaluate the quality of the decision by updating the corresponding Q values. The Q-values are estimations of the actual reward and the optimal policy is then followed by selecting the actions where the Q-values are maximum.

In the single-agent case, assuming a stationary environment, i.e., that the probabilities of receiving specific reinforcement signals do not change over time, if each action is executed in each state an infinite number of times and the learning rate is decayed appropriately, the Q-values will converge with probability 1 to the optimal ones, [23].

In the multi-agent setting, of $N$ players playing a game repeatedly, the process of learning Q values by observing actual utilities presents a significantly more complex problem, since the utilities (rewards) available to each player depend on the joint-actions
or strategies of all the other players, and hence are not stationary. In [14], each of
the players are equipped with a standard Q-learning algorithm and learn independently
without considering the presence of each other in the environment. Each player \( i \) selects
an action \( a_i(k) = e_{ij} \) based on strategy \( x_i \), receives a utility (reward) \( U_i(a(k)) \) and then
updates \( Q_i(k) \). For example,

\[
Q_{ij}(k + 1) = (1 - \mu_i(k))Q_{ij}(k) + \mu_i(k)U_i(a(k)),
\]

(2.4)

where \( \mu_i(k) \) are learning rates or step-sizes, assumed to be diminishing. In [14] an extra
normalization was used in (2.4), where the utility error prediction term is divided by the
probability with which \( a_i(k) \) was selected. In (2.4), \( a(k) = (a_i(k), a_{-i}(k)) \) denotes the
joint-action. For the action selection, each player plays according to a Q-value based
smooth best-response, (2.2), based on Boltzmann selection,

\[
x_{ij}(k) = \frac{e^{\tau Q_{ij}(k)}}{\sum_{j' \in M_i} e^{\tau Q_{ij'}(k)}},
\]

(2.5)

closely related to the soft-max exploration method of reinforcement learning [23].

Based on standard theorems of stochastic approximation [2], the behavior of these
learners in 2-player games is analyzed by the corresponding ODE, [14]. The strategy
evolution is closely related to the smooth best response dynamics, the same dynamical
system that characterizes stochastic fictitious play (FP), [3] despite the fact that individual Q-learning uses significantly less information. Using techniques from [13], extension
to \( N \)-player partnership games is studied for player-dependent learning rates. The use
of diminishing learning rates is beneficial and allows one to use stochastic approximation
results, but in general leads to slow convergence time.

We start with the well known Fictitious Play(FP) process, [8]. (The details of the
proof of the main theorems can be found in Appendix A.)
Chapter 2. Background

2.2.1 Fictitious Play

The idea of Fictitious Play (FP) has been around for over half a century. The term “fictitious” refers to the game playing under the unrealistic assumption that the opponent is playing a constant strategy. Hence, such players are fictitious. Each player in a FP setup collects the history of other players’ actions, and plays the best response accordingly.

Define the empirical frequency, $q_{ij}(k)$, as the ratio of iterations in history at which player $i$ has chosen the action $e_{ij} \in A_i$ up to time $k - 1$, i.e.,

$$q_{ij}(k) = \frac{1}{k} \sum_{\kappa=0}^{k-1} I\{a_i(\kappa) = e_{ij}\},$$

where $a_i(k) \in A_i$ is player $i$’s action at time $k$, and $I\{\cdot\}$ is the indicator function. Now the vector form of the empirical frequency is written as

$$q_i(k) = \begin{bmatrix} q_{i1}(k) \\ \vdots \\ q_{i|A_i|}(k) \end{bmatrix}.$$

The expected utility for the action $a_i \in A_i$ is

$$U_i(a_i, q_{-i}(t)) = \sum_{a_{-i} \in A_{-i}} U_i(a_i, a_{-i}).$$

In the FP process, player $i$ uses this expected utility by selecting an action at time $k$ from the best response set

$$\beta_i(q_{-i}(k)) := \{a^*_i \in A_i : U_i(a^*_i, q_{-i}(k)) = \max_{a_i \in A_i} U_i(a_i, q_{-i})\}.$$

It is proven that empirical frequencies generated by FP converge to a Nash equilibrium in potential games [19], [20].
2.2.2 Joint Strategy Fictitious Play

In this section, we discuss the elegant variant of the FP algorithm, the Joint Strategy Fictitious Play (JSFP) algorithm from [16]. As mentioned in Section 1.2, in JSFP, each player tracks the empirical frequencies of the joint actions of all other players. The presumption is that all other players are playing randomly but jointly according to their joint empirical frequencies.

In order to choose best response of the expected utility based on joint empirical frequencies, the players only need to keep track of the predicted utilities for each of its action, which admits the following simple recursion

\[
\bar{U}_{ij}(k + 1) = (1 - \mu(k))\bar{U}_{ij}(k) + \mu(k)U_i(a_i, a_{-i}(k)), \quad \forall a_i \in A_i. \tag{2.6}
\]

where in Fading Memory JSFP with Inertia (FMJSFP-I) process, \(\mu(k)\) is chosen as a parameter for all times \(k \geq 1\) such that, \(0 < \mu \leq 1\). \(1 - \mu\) is considered as the discount factor, or “inertia”, indicating how the empirical frequency is weighted along the process.

With expected utility calculated at each iteration, players choose their actions according to the following rules:

- If the action \(a_i(k - 1)\) chosen by player \(i\) at time \(k - 1\) belongs to \(\beta(\bar{U}(k))\), then \(a_i(k) = a_i(k - 1)\)

- Otherwise, player \(i\) chooses an action, \(a_i(k)\), at time \(k\) according to the probability distribution

\[
\alpha_i(k)\beta_i(k) + (1 - \alpha_i(k))e_{i\alpha_i(k-1)}, \tag{2.7}
\]

where \(\beta_i(k) \in \Delta_i\) is any probability distribution whose support is contained in the set \(\beta_i(\bar{U}_i(k))\)

There are several assumptions and propositions following FMJSFP-I.

**Assumption 2.2.1.** [16] There exist constants \(\varepsilon\) and \(\bar{\varepsilon}\) such that for all time \(k \geq 0\) and
for all players $i \in I$

$$0 < \xi < \alpha_i(k) < \bar{\epsilon} < 1.$$  

This assumption guarantees that the players are always willing to optimize with some nonzero inertia.

**Assumption 2.2.2.** [16] Player utilities satisfy the following: for all players $i \in I$, actions $a_i, a'_i \in A_i$, $a_i \neq a'_i$, and joint actions $a_{-i} \in A_{-i}$

$$U_i(a_i, a_{-i}) \neq U_i(a'_i, a_{-i}).$$

The following proposition establishes convergence to Nash equilibrium for FMJSFP-I.

**Proposition 2.2.1.** [16] In any finite generalized ordinal potential game in which no player is indifferent between distinct strategies as in Assumption 2.2.2, the action profiles $a(k)$ generated by a FMJSFP-I process satisfying Assumption 2.2.1 converge to a pure Nash equilibrium almost surely.

### 2.2.3 Perturbed Reinforcement Learning

Another algorithm that derives conditions for convergence to Nash equilibria in potential games is perturbed Reinforcement Learning (RL), in [6]. A Perturbed Linear Reward-Inaction Scheme is introduced, in which the decisions of each agent are slightly perturbed.

Given a strategy profile $x \in \Delta$, the expected payoff vector of each agent $i$, $\bar{U}_i : \Delta \to \mathbb{R}^{|A_i|}$, is computed by

$$\bar{U}_i(x) = \sum_{j \in A_i} e_j \sum_{a_{-i} \in A_{-i}} (\prod_{s \in -i} x_{sa}) U_i(j, a_{-i}).$$

The probability that agent $i$ selects action $j$ at time $k$ with perturbation is

$$\chi_{ij}(k) = (1 - \rho_i(x_i(k), \xi)) * x_{ij}(k) + \rho_i(x_i(k), \xi)/|A_i|, \quad (2.8)$$
for some perturbation function $\rho_i : \Delta(|A_i|) \times [0, 1] \rightarrow [0, 1]$ and the probability vector $x_i(k)$ is updated according to the recursion

$$x_i(k+1) = \Pi_\Delta \{x_i(k) + \mu(k)U(a(k))[e_{ij}(k) - x_i(k)]\}, \quad (2.9)$$

where the step-size sequences

$$\mu(k) = \frac{1}{k^\nu + 1} \quad (2.10)$$

for some $\nu \in (1/2, 1]$. For these values of $\nu$, the following conditions can easily be verified

$$\sum_{k=0}^{\infty} \mu(k) = \infty, \quad \sum_{k=0}^{\infty} \mu(k)^2 < \infty.$$

The utility function satisfies the following assumption.

**Assumption 2.2.3.** [6] For every $i \in \mathcal{I}$, the utility function satisfies $U_i(a) > 0$ for all $a \in A$.

Furthermore, the following class of perturbation functions are considered

**Assumption 2.2.4.** [6] The perturbation function $\rho_i : \Delta(|A_i|) \times [0, 1] \rightarrow [0, 1]$ is continuously differentiable. Furthermore, for some $\zeta \in (0, 1)$ sufficiently close to 1, $\rho_i$ satisfies the following properties:

- $\rho_i(x_i, \xi) = 0$ for all $x_i$ such that $|x_i|_\infty < \zeta$ for any $\xi \geq 0$
- $\lim_{|x_i|_\infty \rightarrow 1} \rho_i(x_i, \xi) = \zeta$
- $\lim_{|x_i|_\infty \rightarrow 1} \frac{\partial \rho_i(x_i, \xi)}{\partial \xi} = c$ for some $c > 0$
- $\lim_{|x_i|_\infty \rightarrow 1} \frac{\partial \rho_i(x_i, \xi)}{\partial x_{ij}} = 0$ for any $j \in A_i$

One candidate perturbation function is

$$\rho_i(x_i, \xi) = \begin{cases} 0 & |x_i|_\infty < \zeta \\ \frac{\xi}{(1-\zeta^2)(|x_i|_\infty - \zeta)^2} & |x_i|_\infty \geq \zeta \end{cases}$$
Chapter 2. Background

It is straightforward to check that this function satisfies the properties of Assumption 2.2.4 when we select $\zeta \in (0,1)$ sufficiently close to 1.

The convergence to Nash equilibria would require the following property in potential games.

**Property 2.2.1.** [6] There exists a $C^2$ function $P : \Delta \to \mathbb{R}$ such that

$$\nabla x_i P(x) = \vec{U}_i(x).$$

Note that this property is also used to define potential games in population games, where players from an infinite-size population are paired to play the game.

**Proposition 2.2.2.** [6] For the class of games satisfying Property 2.2.1, the Perturbed Linear Reward-Inaction scheme converges to some invariant set in the neighborhood of the Nash equilibria.

### 2.2.4 Individual Q-Learning

Although Q-Learning with Multiple-Timescales is not an algorithm designed for potential games, we will briefly introduce the Individual Q-learning (IQL) algorithm and its conditions for convergence.

Each player $i$ selects an action $a_i(k)$ using the strategy $\beta_i(Q_i(k))$, receives utility $\mathcal{U}_i(a(k))$, and then updates $Q_i(k)$ according to

$$Q_{ij}(k+1) = Q_{ij}(k) + \mu_i(k) \frac{\mathcal{U}_i(a(k)) - Q_{ij}(k)}{\beta_i(Q_i(k))},$$

for each $a_i \in \mathcal{A}_i$, where for each $i$, $\mu_i(k)$ is a deterministic sequence of learning parameters satisfying

$$\sum_{k=0}^{\infty} \mu_i(k) = \infty, \quad \sum_{k=0}^{\infty} \mu_i(k)^2 < \infty,$$
and additionally
\[
\mu_i(k)/\mu_{i+1}(k) \to 0 \quad \text{as} \quad k \to \infty.
\]

We consider the step-size sequences
\[
\mu_i(k) = (k + C)^{-\nu_i}, \quad (2.12)
\]
where \(\nu_i \in (0.5, 1]\) is chosen differently for each player.

Under the following assumption, a Nash distribution can be reached for Individual Q-Learning algorithm.

**Assumption 2.2.5.** [14] For each \(i \in (I - \{1\}\), there exists a function \(\tilde{q}_i : \Delta_1 \times \cdots \times \Delta_{i-1} \to \mathbb{R}^{|A_i|}\) such that, for arbitrary fixed \((Q_1, \ldots, Q_{i-1})\), the ODE
\[
\frac{d}{dt}q_{ij}(t) = u_i(a_i, [\pi_{(<i)}, B_{(>i)}[\pi_{(<i)}, \beta_i(q_i(t))]] - q_{ij}(t), \quad \forall a_i \in A_i
\]
has the globally attracting fixed point \(\tilde{q}_i(\pi_{(<i)})\), where
\[
\pi_{(<i)} = (\beta_1(Q_1), \ldots, \beta_{i-1}(Q_{i-1})).
\]

### 2.2.5 Summary

We now summarize the properties of the algorithms mentioned above.

FP requires that each player can observe the (actions) decisions of all other players. Each player computes the empirical frequencies (i.e. running averages) of these observed decisions. Then, each player updates its strategy in a best-response manner to the empirical frequencies of all the other players.

In JSFP, each player tracks the empirical frequencies of the joint actions of all other players. In contrast to FP, the action of a player is based on the (still incorrect) presumption that other players are playing randomly but jointly according to their joint empirical frequencies, i.e., each player views all other players as a collective group. The
Chapter 2. Background

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FP</td>
<td>BR</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>FMJSFP-I</td>
<td>BR with Inertia</td>
<td>Constant</td>
<td>Weighted Sum</td>
<td>Constant</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>RL</td>
<td>Weighted Sum</td>
<td>Decreasing</td>
<td>-</td>
<td>-</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>IQL</td>
<td>SBR</td>
<td>1</td>
<td>Weighted Sum</td>
<td>Decreasing Multi-Timescales</td>
<td>N</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 2.1: Learning Algorithms in Games Comparison

authors showed its beneficial features when applied to the large-scale congestion game. However, while reducing the information requirement, players still have to monitor the joint actions and need to know their own utility so as to find their optimal actions.

In RL, agents observe only the actual utilities received as a result of the joint actions of their opponents, and use these actual utilities to choose future actions. The algorithm alternates between two phases, exploration and exploitation, and requires that several parameters are set in advance, such as the exploration phase length, exploration rates, and tolerances on payoff difference and switching rates for deciding when to change strategies.

Q-learning is a useful approach for learning Nash equilibria in games with unknown noisy utilities or rewards. Players’ rewards are initially unknown and must be learned or estimated online from actual observations.

For IQL, agents observe the actual utilities received as a result of their own actions, and use this information along with the frequencies of current actions to estimate the expected utilities. The algorithm adopts smoothed best response over the estimated expected utilities, and the “temperature” of Boltzmann selection is set in advance.

From Table 2.1, we compare the properties of different algorithms in LGT. All the algorithms but the RL algorithm use BR/SBR in their probability updating scheme. Both FP and IQL use no inertia to play the (smooth) best response. However, the
FMJSFP-I algorithm introduced inertia in the probability updating scheme as conditions of convergence to Nash equilibria.

Neither the FP algorithm nor the RL algorithm tracks expected utility. Other than using different notations and step-sizes, the Q-value that the IQL algorithm tracks in (2.11) is very similar to the expected utility that the FMJSFP-I algorithm tracks in (2.6).

With complete information of the player’s utility, neither the FP algorithm nor the FMJSFP-I algorithm include any kind of perturbation in its updating scheme, but the RL algorithm is designed with a state base perturbation function, while the IQL algorithm uses the natural perturbation property in SBR.

Although IQL is not designed for potential games, it is one of the most important algorithms that inspires the design of the modified Q-learning algorithm, which we introduce in the next chapter.
Chapter 3

A Modified Q-Learning Algorithm

In this chapter we present the modified Q-learning algorithm we consider. The two components of such an algorithm are the action selection and the Q-value updating rule.

The updating scheme of this modified Q-learning algorithm (MQL) is mainly inspired by Fading Memory Joint Strategy Fictitious Play with Inertia (FMJSFP-I). Without the perturbation function, (3.1) and (3.3) are very similar to (2.7) and (2.6). However, for FMJSFP-I, with the complete information of player’s utility function, each $\bar{U}_{ij}$ is updated at every iteration $k$, while for MQL, the information is out-dated. Hence, only the Q-value corresponding to the active action is updated at time $k$. The relaxation of information inquiry would result in convergence to sub-optimal points. Therefore, a perturbation function for exploration needs to be introduced to fully explore the game and ensure convergence to Nash equilibria.

3.1 Updating Scheme

At each time-step $k > 0$, each player $i \in \mathcal{I}$ chooses an action $a_i(k) = e_{ij}$ based on its mixed-strategy $x_i(k)$ and its Q-value. Its probability vector $x_i(k)$ is updated according to the recursion

$$x_i(k+1) = (1 - \alpha_i(k))x_i(k) + \alpha_i(k)\hat{\beta}_i(Q_i(k)),$$

(3.1)
where $Q_i(k)$ is the Q-value vector, $\hat{\beta}_i$ is defined as

$$
\hat{\beta}_i(Q_i(k)) = \{e_{ij^*} : j^* \in M_i : Q_{ij^*} = \max_{j \in M_i} Q_{ij}(k)\}.
$$

(3.2)

where $0 < \alpha_i(k) < 1$ is the step-size players choose, indicating how aggressive the players are to move from previous action towards the expected action that maximizing the Q-value or numerical utility.

The Q-value of each player $Q_i(k)$ acts as the estimation of $U_i(a(k))$ by following the joint action $a(k) = (a_i(k), a_{-i}(k)) = (e_{ij}(k), a_{-i}(k))$, providing key information in the decision making for each player. This $Q_i(k)$ is a $|A_i|$-dimensional vector with components $Q_{ij}(k + 1), j \in M_i$. Each of its components is updated similar to (2.4) as follows: for the $j$-th component corresponding to the played action at time-step $k$, $a_i(k) = e_{ij}$,

$$
Q_{ij}(k + 1) = (1 - \mu_{ij}(k))Q_{ij}(k) + \mu_{ij}(k)U_i(a(k)),
$$

(3.3)

where $0 < \mu_{ij}(k) < 1$ is the learning rate (step size), while for the other components $j' \in M_i, j' \neq j$ not played at time-step $k$,

$$
Q_{ij'}(k + 1) = Q_{ij'}(k).
$$

(3.4)

Thus in the algorithm we consider, for each player $i \in I$, at time-step $k > 0$, a player chooses action $e_{ij} = a_i(k)$ with probability $x_{ij}(k)$ based on the mixed-strategy $x_i(k)$, and updates $x_i(k)$ and $Q_i(k)$ as in (3.1), (3.2) and (3.3), (3.4).

Furthermore, we have the following assumption.

**Assumption 3.1.1.** Player step-sizes are constant if not specified, i.e. for all players $i \in I$, actions $j \in M_i$,

$$
\mu_{ij}(k) = \mu, \quad \alpha_i(k) = \alpha,
$$

where $0 < \alpha < \mu < 1$.

An important component of Q-learning is the action selection mechanism, responsible
for selecting the actions that the agent performs during the learning process.

Our proposed action selection (3.1), (3.2) is based on greedy selection, when the action with the highest Q-value is selected with some inertia ($\alpha \neq 1$). This is a slightly modified Q-learning algorithm: instead of a Q-value based smooth best-response as in (2.5), the action selection (3.1), (3.2) uses a Q-value based best-response with inertia, inspired by the JSFP in Subsection 2.2.2, [17].

Comparing the Q-value based best-response $\hat{\beta}_i$ in (3.2) to (2.1), it is clear that $\hat{\beta}_i$ acts as an estimated best response. Let $\hat{a}_i^*(k) := e_i \hat{\gamma} = \hat{\beta}_i(Q_i(k))$, $i \in I$. Considering the overall $N$-tuple $\hat{a}^*(k) = (\hat{a}_1^*(k), \ldots, \hat{a}_N^*(k))$, and denoting $\hat{\beta}(Q(k)) := (\hat{\beta}_1(Q_1(k)), \ldots, \hat{\beta}_N(Q_N(k)))$, it follows that $\hat{a}^*(k) = \hat{\beta}(Q(k))$ acts as an estimated equilibrium at time-step $k$. Since it results from a process that lacks utility information, this estimated equilibrium would most likely be sub-optimal. We observe that for the JSFP to converge to a Nash equilibrium, we do not need to estimate the exact value of utilities for each action, but rather a correct order of utilities of all actions. The Q-value of each action are incomplete estimations of joint strategy utility as in JSFP, while the action that has the largest probability to be updated is the estimated equilibrium. We prove in the following chapter that the corresponding Q-values eventually converges to the actual utility if the estimated equilibrium is no longer changing. Therefore, if the action space of the game is well explored, the estimated equilibrium most likely converges to an actual (true) Nash equilibrium.

### 3.2 Perturbation Function

The action selection mechanism allows for a trade-off between exploitation and exploration. In particular, the agents reinforce the evaluation of the actions already known to be good, but also explore new actions. The Boltzmann action selection (2.5) used in a standard Q-learning algorithm, [14], [11] incorporates this trade-off. The greedy action selection, with constant step-sizes in (3.1), (3.2) might generally lead to sub-optimal solutions. In order to incorporate a means of exploring less-optimal strategies, in the second part of the chapter we introduce a perturbation in the Q-learning algorithm.

Specifically, inspired by the perturbing scheme in [6], we assume that each player $i$
Chapter 3. A Modified Q-Learning Algorithm

selects the $j$-th action, $j \in \mathcal{M}_i$ according to a modified strategy with probability

$$
\chi_{ij} = (1 - \rho_i(x_i, \xi))x_{ij} + \rho_i(x_i, \xi)1_{ij}/|\mathcal{A}_i|,
$$

(3.5)

where $\rho_i(x_i, \xi)$ is a perturbation function defined next.

**Assumption 3.2.1.** The perturbation function $\rho_i : \Delta_i \times [\bar{\epsilon}, 1] \to [0, 1]$ is continuously differentiable. Furthermore, for some $\zeta \in (0, 1)$ sufficiently close to 1, $\rho_i$ satisfies the following properties:

- $\rho_i(x, \xi) = 0 \forall x$ such that $\min_{i \in \mathcal{I}}(|x_i|_\infty) < \zeta$ for any $\xi \geq 0$
- $\lim_{\min_{i \in \mathcal{I}}|x_i|_\infty \to 1} \rho_i(x, \xi) = \xi$
- $\lim_{\min_{i \in \mathcal{I}}|x_i|_\infty \to 1} \frac{\partial \rho_i(x, \xi)}{\partial x_{ij}} = 0$ for any $i \in \mathcal{I}$, and any $j \in \mathcal{A}_i$

This perturbation function is slightly modified from the one in [6] and ensures mutation and exploration of all actions. Note that this mechanism is similar to the $\epsilon$-greedy exploration, where it selects a random action with small probability $\rho_i$ and the best action, i.e. the one that has the highest Q-value at the moment, with probability $(1 - \rho_i)$.

Assumption 3.2.1 guarantees that the perturbation function $\rho_i$ of agent $i$ is 0 when its strategy is not close to any boundary of $\Delta_i$ and is $\xi$ when its strategy is at the boundary of $\Delta_i$. The perturbation does not change with $x$ when evaluated at a vertex of $\Delta_i$ and for $\xi = 0$. This perturbation function gives a minimum support of exploration of all actions, and the amount of perturbation can be easily quantified.

### 3.3 Discussion

As discussed in Chapter 2, there are two types of algorithms in learning in game theory. The first type, represented by FP and JSFP, use the complete utility information to help achieve a Nash equilibrium. The other type, such as Reinforcement Learning and Q-learning with Multiple-Timescales, use stochastic approximation to reach a neighborhood of a Nash equilibrium or a Nash distribution.
The modified Q-learning algorithm takes a different approach. We note the similarity between the Q-learning algorithm and JSFP. Therefore, we investigate the idea of introducing non-negligible constant step-sizes and combining the perturbation function with the Q-learning algorithm, to achieve a faster converging results with less information aggregation.

Although the perturbation function is inspired by perturbed reinforcement learning algorithm in [6], there is a small difference when it comes to the usage of the perturbation. In perturbed reinforcement learning, the perturbation is state based. That is, not all players enter the perturbation at the same time. This state based perturbation would require less information gathering and no synchronization on the perturbation of all players.

However, the modified Q-learning algorithm requires convergence to the estimated Nash equilibrium first, and then after all the players converged to a pure strategy, the perturbation is turned on to explore the rest actions. This condition may be relaxed to reduce the information gathering needed for a global perturbation function. For example, the players can track the change of their updating Q-value to see if other players have converged.

In the next chapter, we further discuss the convergence analysis of this modified Q-learning algorithm.
Chapter 4

Convergence Analysis

In this chapter, we give conditions under which the modified Q-learning algorithm (3.1) - (3.4) converges to a pure strategy Nash equilibrium almost surely. In the first part we consider constant learning rates (step-sizes), while in the second part we consider frequency dependent step-sizes based on perturbation (3.5) as a mechanism of exploration. Although the modified Q-Learning algorithm is not designed for mixed-strategy Nash equilibria, for completeness, convergence to mixed-strategy Nash equilibria is discussed in the last section of this chapter.

4.1 Convergence to Estimated Equilibria

In the following, we show that in the absence of a mechanism of exploring all actions, the modified Q-learning algorithm (3.1) - (3.4) converges to an estimated equilibrium. This is partially due to the lack of full utility function information.

**Proposition 4.1.1.** Under Assumption 3.1.1, if for some $K > 0$, an action profile $a(k) = (e_{ij}, a_{-i})$ is repeatedly played in the consequent $K$ iterations, i.e., $a(k + \kappa) = a(k)$, for all $1 \leq \kappa < K$, then

$$Q_{ij}(k + K) = (1 - \mu)^K Q_{ij}(k) + (1 - (1 - \mu)^K)\mathcal{U}_i(a(k)).$$  

(4.1)
Proof. Assume as in the statement that an action profile \( a(k) = (e_{ij}, a_{-i}) \) is repeatedly played, i.e., the \( j^{th} \) action is played by player \( i \). Then from (3.1), (3.3) and (3.4), it follows that if the \( j^{th} \) action is played, the other actions \( j' \in M_i, j' \neq j \) are never played during the following \( K \) iterations, so that each \( Q_{ij} \) stays unchanged. Moreover, for the played \( j^{th} \) action and \( Q_{ij} \), from recursively using (3.3) it follows that

\[
Q_{ij}(k + K) = (1 - \mu)^K Q_{ij}(k) + \mu \frac{1 - (1 - \mu)^K}{1 - (1 - \mu)} U_i(a(k)),
\]

which yields (4.1).

\[ \square \]

**Corollary 4.1.1.** If for some sufficiently large \( K > 0 \), conditions in Proposition 4.1.1 hold, then

\[
\lim_{K \to \infty} Q_{ij}(k + K) = U_i(a(k)).
\]

Corollary 4.1.1 is verified by simply taking the limit of the left hand side of (4.1) as \( K \to \infty \) in Proposition 4.1.1, and using \( \mu < 1 \) from the Assumption 3.1.1.

**Remark 4.1.1.** If the conditions of Proposition 4.1.1 are satisfied, from the description of the Q-learning algorithm (3.1), (3.3) and (3.4), other actions \( j' \in A_i \) are never played during \( K \) iterations, so that each \( Q_{ij'} \) stays unchanged.

The following result shows an absorption property of estimated equilibria in the Q-learning algorithm with constant step-sizes. The proof is similar to the proof of Theorem 3.1 in [17].

**Proposition 4.1.2.** Assume that at some time-step \( k, \hat{a}^* = \hat{a}^*(k) \) is played, where \( \hat{a}^* = (\hat{a}_{ij}^*, \hat{a}_{-i}^*) \), \( \hat{a}_{ij}^* = e_{ij}^* \), and that for every player \( i \in I, \forall j' \in M_i \) and \( j' \neq \hat{j}^* \), \( Q_{ij'}(k) < Q_{i\hat{j}^*}(k), Q_{ij'}(k) < U_i(\hat{a}^*) \). Then at any consequent \( K \)-th iteration, with probability of at least \( \prod_{\kappa=1}^{K} (1 - (1 - \alpha)^{\kappa})^N \), the following holds

\[
Q_{i\hat{j}^*}(k + K + 1) = (1 - \mu)^{K+1} Q_{i\hat{j}^*}(k) + (1 - (1 - \mu)^{K+1}) U_i(\hat{a}^*),
\]

\[
x_i(k + K + 1) = (1 - \alpha)^{K+1} x_i(k) + (1 - (1 - \alpha)^{K+1}) \hat{a}_{ij}^*.
\]
Proof. We prove the result by induction. For $K = 1$, based on the conditions in the statement, since for every player $i$, $\forall j' \in A_i$ and $j' \neq \hat{j}^*$, $Q_{ij'}(k) < Q_{i\hat{j}^*}(k)$, from (3.2) it follows that $\hat{a}^*$ is the estimated best-response at time-step $k$, and $\hat{a}_i^* = e_{\hat{j}^*}$, is the component corresponding to player $i$. Therefore using (3.1) it follows that at time-step $(k + 1)$,

$$x_i(k + 1) = (1 - \alpha)x_i(k) + \alpha e_{\hat{j}^*}.$$ 

Since $e_{\hat{j}^*}$ is the unit vector, this indicates $x_{\hat{j}^*}(k + 1) \geq \alpha$. This holds for every player, and therefore at time-step $(k + 1)$, $\hat{a}^*$ is played with probability of at least $\alpha^N$. When $\hat{a}^*$ is played, hence $\hat{a}_i^* = e_{\hat{j}^*}$, i.e., the $\hat{j}^*$-th action is played by player $i$ at time-step $(k + 1)$. Then from (3.3) it follows that the corresponding Q-value is updated as

$$Q_{i\hat{j}^*}(k + 1) = (1 - \mu)Q_{i\hat{j}^*}(k) + \mu \mathcal{U}_i(\hat{a}^*) > (1 - \mu)Q_{ij'}(k) + \mu Q_{ij'}(k),$$

(4.2)

where the middle inequality follows from the conditions given in the statement. Thus, $Q_{i\hat{j}^*}(k + 1) > Q_{ij'}(k)$, for all $j' \neq j$. Since any other $j'$-th action, $j' \neq \hat{j}^*$ is not played at time-step $(k + 1)$, from (3.4) it also follows that $Q_{ij'}(k + 1) = Q_{ij'}(k)$. Therefore, $Q_{i\hat{j}^*}(k + 1) > Q_{ij'}(k + 1), \forall j' \in M_i, j' \neq \hat{j}^*$. This shows that, at time-step $(k + 1)$, $\hat{a}_i^* = e_{\hat{j}^*}$ remains the estimated best-response component for player $i$. Repeating the above argument for all players, it follows that at time-step $(k + 1)$, $\hat{a}^*$ remains the estimated best-response, and the claim follows for $K = 1$.

As the next step of induction, suppose now that at every consequent $\kappa$-th iterations, $1 \leq \kappa \leq K - 1$, $\hat{a}^*$ is played with probability $\prod_{\kappa=1}^{K-1}(1 - (1 - \alpha)^\kappa)^N$. When $\hat{a}^*$ is played, hence $\hat{a}_i^* = e_{\hat{j}^*}$ is played by player $i$, it follows that at time-step $(k + K)$, $x_i$ is updated as

$$x_i(k + K) = (1 - \alpha)^K x_i(k) + (1 - (1 - \alpha)^K) e_{\hat{j}^*},$$

(4.3)
where $\hat{a}_i^* = e_{\hat{j}^*}$. Moreover, by Proposition 4.1.1,

$$Q_{ij^*}(k + K) = (1 - \mu)^K Q_{ij^*}(k) + (1 - (1 - \mu)^K) U_i(\hat{a}^*).$$  \hspace{1cm} (4.4)$$

Since, for player $i$, any other $j'$-th action other than the $\hat{j}^*$-th, $j' \neq \hat{j}^*$ is not played, it also follows that $Q_{ij'}(k + K) = Q_{ij'}(k)$. Moreover, $Q_{ij'}(k + K) < Q_{ij^*}(k + K)$, and $Q_{ij'}(k + K) < U_i(\hat{a}^*)$, $\forall j' \in A_i$, $j' \neq \hat{j}^*$.

Following the same argument as for $K = 1$, from (4.3) it follows that at time-step $(k + K + 1)$, $\hat{a}^*$ is played with probability of at least $(1 - (1 - \alpha)^K)^N$. Hence from (3.3) and the foregoing two inequalities it follows that

$$Q_{ij^*}(k + K + 1) = (1 - \mu)Q_{ij^*}(k + K) + \mu U_i(\hat{a}^*)$$

$$> (1 - \mu)Q_{ij'}(k + K) + \mu Q_{ij'}(k + K)$$

$$= Q_{ij'}(k + K).$$

Also, note that $Q_{ij'}(k + K + 1) = Q_{ij'}(k + K)$, so that $Q_{ij^*}(k + K + 1) > Q_{ij'}(k + K + 1)$, $\forall j' \in A_i$, $j' \neq \hat{j}^*$, hence the estimated best response for player $i$ is unchanged and is given as $\hat{a}_i^* = e_{\hat{j}^*}$. The same can be shown for all players $i \in I$, and therefore,

$$\hat{\beta}(Q(k + K + 1)) = \hat{\beta}(Q(k)) = \hat{a}^*, $$

which indicates that the estimated equilibrium is unchanged. Substituting the above and (4.3), with $\hat{a}_i^* = e_{\hat{j}^*}$, into (3.1) yields that, at time-step $(k + K + 1)$,

$$x_i(k + K + 1) = (1 - \alpha)^{K+1}x_i(k) + (1 - (1 - \alpha)^{K+1})\hat{a}_i^*. $$

Substituting (4.4) into (3.3) gives at time-step $(k + K + 1)$,

$$Q_{ij^*}(k + K + 1) = (1 - \mu)Q_{ij^*}(k + K) + \mu U_i(\hat{a}^*)$$

$$= (1 - \mu)^{K+1}Q_{ij^*}(k) + (1 - (1 - \mu)^{K+1}) U_i(\hat{a}^*),$$
which completes the induction argument.

The next corollary follows immediately by using $0 < \alpha < \mu < 1$.

**Corollary 4.1.2.** If the conditions in Proposition 4.1.2 hold for some sufficiently large $K > 0$, then with probability $\prod_{\kappa=1}^{\infty} (1 - (1 - \alpha)^{\kappa})^N$ for every player $i$

\begin{align}
\lim_{K \to \infty} Q_{ij^*}(k + K) &= U_i(\hat{a}^*), \\
\lim_{K \to \infty} x_i(k + K) &= \hat{a}_i^*
\end{align}

(4.5) (4.6)

where $\hat{a}^* = (\hat{a}_i^*, \hat{a}_{-i}^*)$, $\hat{a}_i^* = e_i^*_{j^*}$.

**Remark 4.1.2.** Proposition 4.1.2 shows that if for every player, the actual (true) utility of the estimated equilibrium is greater than the numerical (estimated) utility of other actions, the $Q$-value of the estimated equilibrium is likely to converge to the true utility. The sufficient conditions in Proposition 4.1.2 mean that actions other than $\hat{a}_i^*$ are sub-optimal not only for the estimated utility ($Q$-value) (hence not played under the algorithm), but also for the actual (true) utility. While intuitively these are reasonable conditions, we do not have an argument to show how strict they are. Under these conditions, we give a more precise characterization in the next result.

Before we discuss the next theorem, we introduce the following definition of almost sure convergence to an equilibrium $a^*$.

**Definition 4.1.1.** The sequence of random variables $a(k)$ converges almost surely to the random variable $a^*$ if the sequence of functions $a(k)$ converges to the function $a^*$ as $k \to \infty$ for all $a$ in $A$, except possibly on a set of probability zero. That is,

$$P[a(k) \to a^* \text{ as } k \to \infty] = 1.$$ 

**Theorem 4.1.1.** If the conditions in Proposition 4.1.2 hold, then as $K \to \infty$, $x(k + K)$ converges to $\hat{a}^*$, with some strictly positive probability $p_0$, where

$$p_0 = \prod_{\kappa=1}^{\infty} (1 - (1 - \alpha)^{\kappa})^N > 0.$$ 

(4.7)
Chapter 4. Convergence Analysis

Proof. From Proposition 4.1.2 and Corollary 4.1.2, (4.6) holds for every player with the same probability \( \prod_{\kappa=1}^{\infty} (1 - (1 - \alpha)^\kappa)^N \). We show that this probability is strictly positive, using an approach as used in Proposition 6.1 in [6]. The product \( \prod_{\kappa=1}^{\infty} (1 - (1 - \alpha)^\kappa) \) is non-zero if and only if \( \sum_{\kappa=1}^{\infty} \log(1 - (1 - \alpha)^\kappa) > -\infty \), i.e.,

\[
- \sum_{\kappa=1}^{\infty} \log(1 - (1 - \alpha)^\kappa) < \infty. \tag{4.8}
\]

Note that,

\[
\lim_{\kappa \to \infty} -\frac{\log(1 - (1 - \alpha)^\kappa)}{(1 - \alpha)^\kappa} = \lim_{\kappa \to \infty} \frac{1}{1 - (1 - \alpha)^\kappa} = 1,
\]

since \( 0 < (1 - \alpha) < 1 \). Thus, from the limit comparison test, (4.8) holds if and only if \( \sum_{\kappa=1}^{\infty} (1 - \alpha)^\kappa < \infty \). This obviously holds for \( 0 < (1 - \alpha) < 1 \), since

\[
\sum_{\kappa=1}^{\infty} (1 - \alpha)^\kappa = \frac{1}{1 - (1 - \alpha)} = \frac{1}{\alpha} < \infty.
\]

Therefore, indeed

\[
\lim_{K \to \infty} \prod_{\kappa=1}^{K} (1 - (1 - \alpha)^\kappa) > 0,
\]

i.e., as \( K \to \infty \), (4.6) holds and \( x(k + K) \) converges to \( \hat{a}^* \) with strictly positive probability, \( p_0 = \lim_{K \to \infty} \prod_{\kappa=1}^{K} (1 - (1 - \alpha)^\kappa)^N > 0 \).

From Theorem 4.1.1, we conclude the following corollary, indicating \( x(k) \) enters a neighborhood of \( \hat{a}^* \), where perturbation function is non-zero, almost surely. The proof follows a similar argument as in the proof of Theorem 3.1 in [17] and Theorem 6.2 in [25].

**Corollary 4.1.3.** If the conditions in Proposition 4.1.2 hold for all the consequent iterations, then for any \( \zeta \), as \( K \to \infty \), \( \max_{i \in I} (|x_i(k + K) - e_{i\hat{a}^*}|_\infty) < \zeta \), almost surely.

Proof. From (4.3) and Theorem 4.1.1, we achieve

\[
x_i(k + K) - e_{i\hat{a}^*} = (1 - \alpha)^K (x_i(k) - e_{i\hat{a}^*}).
\]
with probability at least $p_0$. Then, following the fact that $e_{j^*}$ is a unit vector and $|x_i(k) - e_{j^*}|_\infty \leq 1$, we achieve

$$|x_i(k + K) - e_{j^*}|_\infty \leq (1 - \alpha)^K.$$ 

with probability at least $p_0$.

If only we pick $K_0 > \log_{1-\alpha} \zeta > 0$, then we have $(1 - \alpha)^K_0 < \zeta$. Therefore, we conclude that $\forall K_0$, $\max_{i \in I}(|x_i(k + \kappa) - e_{j^*}|_\infty) < \zeta$ happens with probability at least $p_0$. Furthermore, $\forall K$, $\max_{i \in I}(|x_i(k + \kappa) - e_{j^*}|_\infty) \geq \zeta$ happens with probability at most $(1 - p_0)$. Since both $K_0$ and $p_0$ are independent of time $k$, at time $k + K$, it follows that at some sufficiently large time $K$, the probability of $\max_{i \in I}(|x_i(k + K) - e_{j^*}|_\infty) \geq \zeta$ is at most $(1 - p_0)\left(\frac{1}{K_0}\right)$. Since $1 - p_0$ is strictly less than 1, $(1 - p_0)\left(\frac{1}{K_0}\right) \to 0$ as $K \to \infty$. That is,

$$P[\max_{i \in I}(|x_i(k + K) - e_{j^*}|_\infty) < \zeta] \to 1 \quad \text{as} \quad K \to \infty.$$ 

By definition, we conclude that for any $\zeta$, as $K \to \infty$, $\max_{i \in I}(|x_i(k + K) - e_{j^*}|_\infty) < \zeta$, almost surely.

**Remark 4.1.3.** In the case when players use the JSFP algorithm (assuming own utility information and observation of opponents’ joint-actions), players reach a Nash equilibrium and stay there with probability $p > 0$ over a finite time $T > 0$, cf. Theorem 3.1 in [17]. As shown in Theorem 4.1.1 when players use the Q-learning algorithm and estimate their own utility function, they are only guaranteed to reach the estimated equilibrium with some probability. However, as long as the conditions in Proposition 4.1.2 hold for all consequent iterations, the players enter the perturbation neighborhood of the estimated equilibrium almost surely, where perturbation would be introduced to further explore actions other than the estimated best response.
Chapter 4. Convergence Analysis

4.2 Convergence to Pure Strategy Nash Equilibrium

In the previous section, we proved that the Q-learning algorithm converges to an estimated equilibrium almost surely. Next we impose some additional assumptions that help to show that the estimated equilibrium can reach an actual Nash equilibrium. Based on the perturbation function in (3.5), we adjust the learning rates of $Q_{ij}$ depending on the probability $x_{ij}$ of action $a_i = e_{ij}$ being played.

Assumption 4.2.1. Player step-sizes of $Q$ are adjusted based on the frequency of a particular action, i.e., for all players $i \in \mathcal{I}$, actions $j \in \mathcal{A}_i$,

$$
\mu_{ij}(k) = (1 - \chi_{ij}(k)),
$$

where $\chi_{ij}$ is defined in (3.5) and Assumption 3.2.1.

Assumption 4.2.2. Player utilities satisfy the following: for all players $i \in \mathcal{I}$, actions $j, j' \in \mathcal{M}_i$, $j \neq j'$, and joint actions $a_{-i} \in \mathcal{A}_{-i}$

$$
\mathcal{U}_i(e_{ij}, a_{-i}) \neq \mathcal{U}_i(e_{ij'}, a_{-i}).
$$

Assumption 4.2.2 means that no player is indifferent between distinct strategies; alternatively we could assume that all pure equilibria are strict.

Assumption 4.2.3. After $|x_i|_\infty > \zeta$, $\forall i \in \mathcal{I}$, i.e., when every player has entered the perturbation zone, no more than one player chooses action $j'$ other than $\hat{j}^*$ at each iteration.

In order to satisfy Assumption 4.2.3, we either force the perturbation to be asynchronous, so that it affects one player at a time, or we choose $\zeta$ in Assumption 3.2.1 to be sufficiently large and $\xi$ to be sufficiently small. Such choices of $\xi, \zeta$ ensure $(1 - \zeta(1 - \xi))^2$ to be sufficiently close to 0.

Next we show that under the perturbation function in Assumption 3.2.1, the estimated equilibrium converges to a Nash equilibrium almost surely.
The following theorem follows a similar structure to the proof of Proposition 2.2.1 in Appendix A.1.

**Theorem 4.2.1.** In potential games, if for some sufficiently large \( K > 0 \), the conditions in Proposition 4.1.2 hold, and in addition Assumption 4.2.1, 4.2.2, 4.2.3 hold, then the estimated equilibrium \( \hat{a}^*(k) \) converges to a Nash equilibrium \( a^* \) almost surely.

**Proof.** From Proposition 4.1.2 and Corollary 4.1.2, it follows that (4.5) and (4.6) holds for player \( m \), that is

\[
\lim_{K \to \infty} Q_m\hat{a}^*_j(k + K) = U_m(\hat{a}^*(k)), \\
\lim_{K \to \infty} x_m(k + K) = e_m\hat{a}^*_j.
\]

Suppose that the perturbation becomes active at some large enough time-step \( \bar{k} \), and player \( m \) choose a different \( j' \)-th action other than the \( \hat{j}^* \)-th one, i.e., chooses \( a'_m(\bar{k}) = e_{mj'} \) other than \( \hat{a}^*_m(\bar{k}) = e_m\hat{a}^*_j \). By Assumption 4.2.2, this is the only player to do so at time \( \bar{k} \). From (3.5), such perturbation happens with probability of at least \( \xi/|A_i| \) for player \( m \). \( \hat{a}^*_m(\bar{k}) \) was the component of player \( m \) in the estimated equilibrium \( \hat{a}^* = (\hat{a}^*_m, \hat{a}^*_m) \).

Let \( a'(\bar{k}) = (a'_m(\bar{k}), \hat{a}^*_m(\bar{k})) \) the new action profile (joint-action) at time-step \( \bar{k} \). Since player \( m \) choose the \( j' \)-th action at \( \bar{k} \), from (3.3) and Assumption 4.2.1, at \( (\bar{k} + 1) \), he would update its \( Q_{mj'} \) to be

\[
Q_{mj'}(\bar{k} + 1) = \chi_{mj'}(\bar{k})Q_{mj'}(\bar{k}) + (1 - \chi_{mj'}(\bar{k}))U_m(a'(\bar{k})). \tag{4.9}
\]

By assumption, the conditions of Proposition 4.1.2 hold, so that for player \( m \), \( Q_{mj'}(\bar{k}) > Q_{mj'}(\bar{k}) \). Now consider the following two cases:

- If \( U_m(a'(\bar{k})) < U_m(\hat{a}^*(\bar{k})) \), then player \( m \) does not find any response that is better.

- If \( U_m(a'(\bar{k})) > U_m(\hat{a}^*(\bar{k})) \), then player \( m \) finds an action \( a'_m(\bar{k}) \) that is a better response than \( \hat{a}^*_m(\bar{k}) \), i.e. the joint-action \( a'(\bar{k}) \) becomes the new estimated best
response and this is denoted by $\hat{a}^*(\bar{k} + 1)$,

$$
\hat{a}^*(\bar{k} + 1) := a'(\bar{k}) = (a'_m(\bar{k}), \hat{a}_{-m}^*(\bar{k}))
$$

(4.10)

and

$$
\neq (\hat{a}_m(\bar{k}), \hat{a}_{-m}^*(\bar{k})) = \hat{a}^*(\bar{k}).
$$

In the first case, i.e. $U_m(a'(\bar{k})) < U_m(\hat{a}^*(\bar{k}))$, i.e., at a failed attempt to improve the utility, players stay at the estimated equilibrium $\hat{a}^*$ almost surely, and the potential stays unchanged.

The second case is a successful attempt to improve the utility, i.e., when $U_m(a'(\bar{k})) > U_m(\hat{a}^*(\bar{k}))$. Consider (4.9) and note that, since actions other than the estimated best response $\hat{j}$ have sufficiently small probability, i.e. $\chi_{mj'}(\bar{k})$ is sufficiently close to 0, and $(1 - \chi_{mj'}(\bar{k}))$ is sufficiently close to 1. Thus, from (4.9), $Q_{mj'}$ is updated to be sufficiently close to $U_m(a'(\bar{k}))$, and therefore the $j'$-th action, $a'_m(\bar{k}) = e_{mj'}$ becomes its new estimated best response. From (2.3) it follows that when the utility of player $m$ is improved, the potential of the game is improved also. Thus, a successful attempt to improve the utility of player $m$, i.e., $U_m(a'_m(\bar{k}), \hat{a}_{-m}^*(\bar{k})) > U_m(\hat{a}_m(\bar{k}), \hat{a}_{-m}^*(\bar{k}))$, results in an improved potential of the game $G$ and a new estimated best response profile $\hat{a}^*(\bar{k} + 1)$ as in (4.10).

In summary, the first case leads to an estimated best response that is unchanged and results in an unchanged potential of the game $G$, while the second case leads to a new estimated best response that improves the potential of the game $G$. Whenever a player $m$ takes a successful attempt and shifts to the new estimated equilibrium, the utility of this player would improve by $U_m(\hat{a}^*(\bar{k} + 1)) - U_m(\hat{a}^*(\bar{k}))$. Hence, by (2.3),

$$
P(\hat{a}^*(\bar{k} + 1)) - P(\hat{a}^*(\bar{k})) = U_m(\hat{a}^*(\bar{k} + 1)) - U_m(\hat{a}^*(\bar{k})).
$$

Therefore, the potential value of the whole profile would also increase by the same amount. By the finite improvement property (Theorem 2.1.2, [19] Lemma 2.3), the estimated equilibrium $\hat{a}^*(\bar{k})$ converges to an actual Nash equilibrium $a^*$ almost surely. 

Remark 4.2.1. Theorem 4.2.1 and Corollary 4.1.2, show that in a potential game, the
Q-learning scheme with a perturbation function as in Assumption 3.2.1 and Assumption 4.2.1 will converge to a Nash equilibrium almost surely, while using less information than JSFP. Instrumental for this is the finite improvement property of potential games. Our analysis techniques are similar to those used in the JSFP case [17]. However, the setup here is complicated by players’ lack of information on the analytical structure of their own utility.

4.3 Convergence to Mixed-Strategy Nash Equilibrium

In the previous two sections, we discussed convergence to pure Nash equilibria of the modified Q-learning algorithm. However, some games may not admit any pure Nash equilibria. Although the modified Q-learning algorithm is not targeted at games with mixed-strategy Nash equilibria, the convergence to mixed-strategy Nash equilibria is discussed in this section.

In order to achieve convergence to a mixed-strategy Nash equilibrium in potential games, out-dated utilities are no longer sufficient. Each player needs to acquire an accurate expected utility of every action in order to calculate the best mixed-strategy. To estimate such expected utility $U_i(x^*)$, the most intuitive approach is to make use of $Q_i(x)$ in the Q-learning algorithm, and introduce a time-scale separation between the learning and the strategy update.

**Assumption 4.3.1.** Players choose the step sizes of $x$ to be much smaller than the step sizes of $Q$, i.e., for all players $i \in \mathcal{I}$, actions $j \in \mathcal{A}_i$,

$$\frac{\mu_i(k)}{\alpha_i(k)} = \frac{\mu}{\alpha} = c|\mathcal{A}_i|,$$

for some $c >> 0$.

Or, let $K = c|\mathcal{A}|$ for some $c >> 0$, and let one of the players $i \in \mathcal{I}$ updates $x_i$ after
every $K$ iterations, i.e., change (3.1) to be

$$x_i(k + K) = (1 - \alpha_i(k))x_i(k) + \alpha_i(k)\hat{x}_i(Q_i(k)), \quad (4.11)$$

while other $i \in \mathcal{I}/i$

$$x_i(k + K) = x_i(k), \quad (4.12)$$

and change (3.3) to be

$$Q_i(k + \kappa + 1) = (1 - \mu_i(k + \kappa))x_i(k + \kappa) + \mu_i(k + \kappa)\mathcal{U}_i(a(k + \kappa)), \quad (4.13)$$

where $\alpha_i(k) = \alpha$, $0 \leq \kappa \leq K$, $\mu_i(k + \kappa) = \frac{1}{k+1}$.

Assumption 4.3.1 in its two versions implies that by the time $x$ has a non-negligible change, and the $Q$ process has already converged. We use the second version of Assumption 4.3.1 for now. Thus, for every player $i \in \mathcal{I}$, every action $j \in \mathcal{A}_i$ we have

$$\mathbb{E}[Q_{ij}(k + K)] = \mathcal{U}_i(e_{ij}, x_{-i}(k)),$$

and, by the weak law of large numbers, for some $\delta > 0$

$$\lim_{c \to \infty} P[|Q_{ij}(k + K) - \mathbb{E}[Q_{ij}(k + K)]| < \delta] = 1,$$

where $\mathbb{E}[Q_{ij}(k + K)] = \mathcal{U}_i(e_{ij}, x_{-i}(k))$ is finite.

Therefore after every $K$ iterations, $x_i$ would get updated towards estimated best-response $\hat{x}^*$, and the potential of the game $P(x)$ would increase by $P(x(k + K)) - P(x(k)) = \mathcal{U}_i(x_i(k + K), x_{-i}(k)) - \mathcal{U}_i(x(k))$. By the finite improvement property of potential games (Theorem 2.1.1, [19]) (4.11), (4.12) and (4.13) converges to a mixed Nash equilibrium almost surely.
Chapter 5

Continuous-Time Approximation

In the previous chapter, we provided the conditions of convergence of the modified Q-learning algorithm to both pure and mixed-strategy Nash equilibria. To further understand the dynamics of the Q-learning algorithm, we propose an ODE approximation of the unmodified Q-learning algorithm with negligible constant step-sizes, and study its stationary points under the perturbation of the smoothed best response.

We start off by defining a continuous potential function for potential games. Similar to $U_i(x)$ and $U_i(e_{ij}, x_{-i})$, an extension of the potential function to the mixed-strategy space can be made, so that

$$P(x) = \sum_{j \in A_i} x_{ij} P(e_{ij}, x_{-i}),$$

where

$$P(e_{ij}, x_{-i}) = \sum_{a_{-i} \in A_{-i}} \left( \prod_{s \in I \setminus \{i\}} x_{sa_s} \right) P(e_{ij}, a_{-i}).$$

As in Property 8.1 in [6], we define potential games as below:

**Definition 5.0.1.** A finite $N$-player game with action sets $\{A_i\}_{i=1}^N$ and utility functions $\{U_i\}_{i=1}^N$ is a potential game if, there exists some continuously differentiable potential
function $P : \Delta \to \mathbb{R}$, such that

$$\nabla x_i P(x) = \tilde{U}_i(x_{-i}),$$

where $\nabla x_i$ denotes $[\frac{\partial}{\partial x_{i1}}, \ldots, \frac{\partial}{\partial x_{i|A_i|}}]$, and $\tilde{U}_i(x_{-i}) = [U_i(e_{i1}, x_{-i}), \ldots, U_i(e_{i|A_i|}, x_{-i})]^T$.

A straightforward calculation shows that such a function $P$ serves as a potential function under the definition 2.1 of [17]

$$P_i(x_i, x_{-i}) - P_i(x'_i, x_{-i}) = \nabla x_i P(x)^T (x_i - x'_i) = \tilde{U}_i(x_{-i})^T (x_i - x'_i).$$

Therefore, it follows that $\forall i \in I$,

$$P(x_i, x_{-i}) - P(x'_i, x_{-i}) = U_i(x_i, x_{-i}) - U_i(x'_i, x_{-i}), \quad (5.1)$$

for all $x_i, x'_i \in \Delta_i$, for all $x_{-i} \in \Delta_{-i}$.

## 5.1 Continuous-Time Approximation

In the unmodified Q-learning algorithm, instead of the best-response, a Boltzmann distribution is applied. Hence, in each iteration, each player plays according to

$$x_{ij}(k) = \frac{e^{r^{-1}Q_{ij}(k)}}{\sum_{j' \in A_i} e^{r^{-1}Q_{ij'}(k)}}. \quad (5.2)$$

This is also known as smoothed best-response (SBR) and ensures a Lipschitz continuous Best Response Dynamics with (3.3). Similar to the derivation in [11], the approximate ODE of the unmodified Q-learning algorithm can be obtained by letting $t = k\mu$ and taking the limit $\mu \to 0$.

**Proposition 5.1.1.** From (3.3) and (5.2), we have the approximate ODE of Q-learning
algorithm is
\[
\frac{dx_{ij}}{dt} = \tau^{-1} x_{ij}(U_i(e_{ij}, x_{-i}(t)) - \sum_{j' \in A_i} x_{ij'}U_i(e_{ij'}, x_{-i}(t))) + x_{ij} \sum_{j' \in A_i} x_{ij'} \ln \left( \frac{x_{ij'}}{x_{ij}} \right) \tag{5.3}
\]

Proof. In the previous sections, we considered non-negligible constant step-sizes in (3.3). In the following, we assume the step-sizes \( \mu \) to be sufficiently small, i.e., \( \mu \to 0 \). Then letting \( t = k\mu \), and \( \mu \to 0 \) in (3.3) it follows that
\[
\lim_{\rho \to 0} \frac{Q_{ij}(t + \rho) - Q_{ij}(t)}{t + \rho - t} = U_i(e_{ij}, x_{-i}(t)) - Q_{ij}(t) \tag{5.4}
\]

Hence, the left-hand side of the above is \( dQ_{ij}(t)/dt \). The updating scheme (5.2) for the mixed-strategy \( x \) can be approximated in continuous time by
\[
x_{ij}(t) = \frac{e^\tau Q_{ij}(t)}{\sum_{j' \in A_i} e^\tau Q_{ij'}(t)} \tag{5.5}
\]

Taking the time derivative in the foregoing yields
\[
\frac{dx_{ij}(t)}{dt} = \frac{d}{dt} \frac{e^\tau Q_{ij}(t)}{\sum_{j' \in A_i} e^\tau Q_{ij'}(t)} = \tau^{-1} x_{ij}(t) \left( \frac{dQ_{ij}(t)}{dt} - \sum_{j' \in A_i} x_{ij'}(t) \frac{dQ_{ij'}(t)}{dt} \right),
\]

so that
\[
\frac{dx_{ij}(t)}{dt} = \tau^{-1} x_{ij}(t) \left( \frac{dQ_{ij}(t)}{dt} - \sum_{j' \in A_i} x_{ij'}(t) \frac{dQ_{ij'}(t)}{dt} \right). \tag{5.6}
\]

From (5.5), it follows that \( x_i(t) \) stays in \( \text{int}(\Delta_i) \), hence it has full support for all \( t \).
Substituting (5.4) in (5.6), yields
\[
\frac{dx_{ij}(t)}{dt} = \tau^{-1} x_{ij}(t) \left( U_i(e_{ij}, a_{-i}(t)) - Q_{ij}(t) - \sum_{j' \in A_i} x_{ij'}(U_i(e_{ij'}, a_{-i}(t)) - Q_{ij'}(t)) \right)
\]
\[
= \tau^{-1} x_{ij}(t) \left( U_i(e_{ij}, x_{-i}(t)) - \sum_{j' \in A_i} x_{ij'} U_i(e_{ij'}, x_{-i}(t)) + \sum_{j' \in A_i} x_{ij'}(Q_{ij}(t) - Q_{ij'}(t)) \right).
\]
(5.7)

From (5.5), \( \forall j, j' \in A_i \)

\[
x_{ij}(t) = \frac{e^{\tau^{-1} Q_{ij}(t)}}{\sum_{j'' \in A_i} e^{\tau^{-1} Q_{ij''}(t)}}, \quad \text{(5.8)}
\]
\[
x_{ij'}(t) = \frac{e^{\tau^{-1} Q_{ij'}(t)}}{\sum_{j'' \in A_i} e^{\tau^{-1} Q_{ij''}(t)}}, \quad \text{(5.9)}
\]

Therefore, dividing (5.9) by (5.8), results in
\[
\frac{x_{ij'}}{x_{ij}} = \frac{e^{\tau^{-1} Q_{ij'}}}{e^{\tau^{-1} Q_{ij}}},
\]
or equivalently,
\[
\ln x_{ij'} - \ln x_{ij} = \ln e^{\tau^{-1} Q_{ij'}} - \ln e^{\tau^{-1} Q_{ij}}
\]
\[
= \tau^{-1} Q_{ij'} - \tau^{-1} Q_{ij}.
\]

After multiplying both sides by \( x_{ij'} \) and summing over \( j' \in A_i \) this yields
\[
\tau^{-1} \sum_{j' \in A_i} x_{ij'}(Q_{ij'}(t) - Q_{ij}(t)) = \sum_{j' \in A_i} x_{ij'} \ln \left( \frac{x_{ij'}}{x_{ij}} \right).
\]

Substituting this into (5.7), yields the ODE (5.3). \( \square \)
5.2 Stationary Points of Approximate ODE

Note that the Q-learning ODE approximation (5.3) can be separated into two parts. The first part is similar to the Replicator Dynamics in game theory, while the second part consists of entropy terms which can be treated as mutation. We denote the first part as \( f_i(x) \), and the second part as \( g_i(x) \). Then we write (5.3) as

\[
\dot{x}_i = f_i(x) + g_i(x), \quad \forall i \in \mathcal{I}.
\] (5.10)

We denote the components of \( f_i(x) \), the replicator dynamics part, as \( f_{ij}(x) \), \( j \in \mathcal{A}_i \). Then from (5.3) and (5.10), it follows that

\[
f_{ij}(x) = \tau^{-1} x_{ij} \left( U_i(e_{ij}, x_{-i}) - \sum_{j' \in \mathcal{A}_i} x_{ij'} U_i(e_{ij'}, x_{-i}) \right).
\] (5.11)

Similarly for the components of \( g_i(x) \), the perturbation part, it follows that

\[
g_{ij}(x) = x_{ij} \sum_{j' \in \mathcal{A}_i} x_{ij'} \ln \left( \frac{x_{ij'}}{x_{ij}} \right).
\] (5.12)

Using the fact that empirical frequencies are defined as probabilities over strategy set, and \( \sum_{j' \in \mathcal{A}_i} x_{ij'} = 1 \), (5.11) can be rewritten as

\[
f_{ij}(x) = \tau^{-1} x_{ij} \sum_{j' \in \mathcal{A}_i} x_{ij'} \left( U_i(e_{ij}, x_{-i}) - U_i(e_{ij'}, x_{-i}) \right).
\]

From (5.1), in a potential game, it follows that

\[
f_{ij}(x) = \tau^{-1} x_{ij} \sum_{j' \in \mathcal{A}_i} x_{ij'} \left( P_i(e_{ij}, x_{-i}) - P_i(e_{ij'}, x_{-i}) \right)
= \tau^{-1} x_{ij} \left( P_i(e_{ij}, x_{-i}) - \sum_{j' \in \mathcal{A}_i} x_{ij'} P_i(e_{ij'}, x_{-i}) \right).
\]

From Definition 5.0.1, \( P(x) \) is continuously differentiable. Moreover, \( f_{ij}(x) \), \( g_{ij}(x) \) are both continuously differentiable on \( \text{int}(\Delta_i) \). Using the fact that \( \sum_{j' \in \mathcal{A}_i} x_{ij'} = 1 \), and
Chapter 5. Continuous-Time Approximation

\[ P_i(x_i, x_{-i}) = \sum_{j \in A_i} x_{ij} P_i(e_{ij}, x_{-i}) \]

we rewrite \( f_{ij} \) and \( g_{ij} \) as

\[
\begin{align*}
    f_{ij}(x) & = \tau^{-1} x_{ij} \left( P_i(e_{ij}, x_{-i}) - \sum_{j' \in A_i} x_{ij'} P_i(e_{ij'}, x_{-i}) \right) \\
    & = \tau^{-1} x_{ij} \left( P_i(e_{ij}, x_{-i}) - P_i(x_i, x_{-i}) \right), \\
    g_{ij}(x) & = x_{ij} \sum_{j' \in A_i} x_{ij'} \ln \left( \frac{x_{ij'}}{x_{ij}} \right) \\
    & = x_{ij} \sum_{j' \in A_i} x_{ij'} \left( \ln x_{ij'} - \ln x_{ij} \right) \\
    & = x_{ij} \sum_{j' \in A_i} x_{ij'} \ln x_{ij'} - x_{ij} \ln x_{ij}.
\end{align*}
\]

The solution of \( f_{ij}(x) = 0, \forall i \in \mathcal{I}, j \in A_i \) is either \( x_{ij} = 0 \) or \( P_i(e_{ij}, x_{-i}) = P_i(x_i, x_{-i}) \).

When \( e_{ij} = x_i \), it is obvious to tell that any pure strategy is a solution for replicator dynamics \( f_{ij}(x) = 0 \). Moreover, for any mixed-strategy Nash equilibrium, we have

\[ P_i(e_{ij}, x^*_{-i}) = P_i(x^*_i, x^*_{-i}), \]

which we then substitute into \( f_{ij}(x) \) to yield

\[
\begin{align*}
    f_{ij}(x^*) & = \tau^{-1} x^*_{ij} \left( P_i(e_{ij}, x^*_{-i}) - P_i(x^*_i, x^*_{-i}) \right) \\
    & = 0.
\end{align*}
\]

Let \( \Delta_{NE} \) denote the set of Nash equilibria, and \( \Delta_f \) the set of the rest points of the replicator dynamics,

\[ \Delta_{NE} \subset \Delta_f. \]

The smooth best response requires \( x \) to be completely mixed. Therefore, we only
need to study the interior of the ∆, denoted by ∆° = int(∆). Now we know

\[ \Delta_{NE} \cap \Delta^o = \Delta^o_{NE} = \Delta^o_f = \Delta_f \cap \Delta^o. \]

The other term \( g_{ij}(x) \) acts as a perturbation at this equilibrium. When \( g_{ij}(x) = 0 \), we have a vanishing perturbation. Conversely, when \( g_{ij}(x) \neq 0 \), we have a non-vanishing perturbation. For the first case, letting \( g_{ij}(x) = 0 \), we have

\[ x_{ij} = 0 \quad \text{or} \quad \sum_{j' \in \mathcal{A}_i} x_{ij'}(\ln x_{ij'} - \ln x_{ij}) = 0, \]

which means \( x_{ij} = 0 \) or

\[ x_{ij'} = 0 \quad \text{or} \quad x_{ij'} = x_{ij} \quad \forall j' \neq j, j' \in \mathcal{A}_i. \]

The solution of \( g_{ij}(x) = 0 \) is

\[ x_{ij} = \frac{1}{|\sup(x_i(0))|} \quad \forall j \in \mathcal{A}_i. \]

Hence, for any pure strategies \( a \in \mathcal{A} \), we have \( g_{ij}(a) = 0, \forall i \in \mathcal{I}, j \in \mathcal{A}_i \). Therefore, the perturbation is vanishing at pure Nash equilibria. On the other hand, for mixed strategies \( x \), unless \( x_{ij} = 1/|\sup(x_i(0))| \), \( j \in \sup(x_i) \) (i.e., \( x_i \) is evenly distributed among all strategies in its support), \( g_{ij}(x) \neq 0 \) hence the perturbation is non-vanishing, as defined in Chapter 9 [10].

However, since the smooth best response only preserves mixed-strategies, none of the pure stable points will be actually achieved in this process. Therefore, the perturbation is non-vanishing at full support of all strategies. The solution of \( g_{ij}(x) = 0 \) is \( x_i = \frac{1}{|\mathcal{A}_i|}, \) for all \( j \in \mathcal{A}_i \).

If we carefully pick \( \tau \) to be small enough, the effect of \( g_{ij}(x) \) will be negligible in comparison to \( f_{ij}(x) \). Let \( \Delta^o_g \) denote the stationary points of ODE (5.3), then \( \Delta^o_g \) is the set of perturbed Nash equilibria.

The following proposition presents the convergence analysis to an invariant level set
which includes unperturbed mixed-strategy Nash equilibria set $\Delta^o$.

**Proposition 5.2.1.** In potential games, the approximate ODE (5.3) of Q-learning (3.3) and (5.2) will converge to an invariant set that includes unperturbed mixed-strategy Nash equilibria $\Delta^o$.

**Proof.** Similar to the proof of Proposition 5.1.1 in [6], we introduce a positive-definite function

$$V(x) = P_{\max} - P(x) \geq 0, \quad x \in \Delta, \quad (5.13)$$

where $P_{\max} = \sup_{x \in \Delta} P(x)$. Note that $\nabla_x V(x) = - \nabla_x P(x) = - \vec{U}_i(x_{-i})$. For the first part involving $f_i(x)$ terms in the ODE (5.3), we have

$$\nabla_x V_i(x)^T f_i(x) = - \vec{U}_i(x_{-i})^T \tau^{-1} x_i (U_i(e_{ij}, x_{-i}) - U_i(x))$$

$$= - \tau^{-1} \sum_{j \in A_i} \sum_{j' \in A_i, j \neq j'} x_{ij} x_{ij'} (U_{ij}(x) - U_{ij'}(x))^2$$

$$\leq 0.$$ 

Therefore, the only occasion when $\nabla_x V_i(x)^T f_i(x) = 0$ is when $U_{ij}(x^*) = U_{ij'}(x^*), \forall i \in I$ and $\forall j \neq j' \in A_i$. That is to say, $x^* \in \Delta^o$ is one of the mixed-strategy Nash equilibria. Since $V(x)$ is positive definite at $x^*$, $V(x^*) = 0$, i.e. $P(x) = P_{\max}$ if and only if $x = x^*$.

By Theorem 3.11 (Specialized Nagumos Theorem 2) in [15], for any potential value $\forall P_0 < P_{\max}$, let $V_0(x) = P_0 - P(x)$. Let set $B_0(\Delta^o) = \{x \in \Delta : V_0(x) \leq 0\}$. Hence, set $B_0(\Delta^o)$ is positively invariant if and only if we have $\forall x \in \partial B_0(\Delta^o), L_{f+g} V_0(x) \leq 0$.

By definition, $x_{ij}$ are bounded in $[0, 1]$, therefore $x_{ij} \ln x_{ij}$ are bounded for all $i \in I$, and $j \in A_i$. In any finite repeated game $U(x)$ are bounded too. Therefore, $\nabla_x V_i(x)^T g_i(x)$ are bounded.

$$\nabla_x V_i(x)^T g_i(x) = - \vec{U}_i(x_{-i})^T (x_i \sum_{j' \in A_i} x_{ij'} \ln x_{ij'} - \text{diag}\{\ln x_{ij}\} x_i)$$
The upper bound of $\nabla_x V_i(x)^T g_i(x)$ is $\max(|\mathcal{U}_i|)|\mathcal{A}_i|e^{-1}$.

By definition, we have $P_0 < P_{\text{max}}$. Therefore, $x^* \in \text{int}B_0(\Delta_j^o)$. All of the boundary points $x \in \partial B_0(\Delta_j^o)$ satisfy

$$\nabla_x V_0(x)^T f(x) < 0.$$ 

Now we only need to prove there exists $0 < \tau < 1$ such that $L_{f+g} V_0(x) = \nabla_x V_0(x)^T f(x) + \nabla_x V_0(x)^T g(x) \leq 0$.

$$\nabla_x V_0(x)^T f(x) + \nabla_x V_0(x)^T g(x)$$

$$\leq -\tau^{-1}(\min_{j \in \mathcal{A}_i} x_{ij})^2(\min_{j \neq j' \in \mathcal{A}_i} (\mathcal{U}_{ij}(x) - \mathcal{U}_{ij'}(x))^2) + \max(|\mathcal{U}_{ij}|)|\mathcal{A}_i|e^{-1}$$

$$\leq 0.$$ 

If only we have

$$\tau^{-1} \geq \frac{\max_{j \in \mathcal{A}_i}(|\mathcal{U}_{ij}|)|\mathcal{A}_i|e^{-1}}{(\min_{j \in \mathcal{A}_i} x_{ij})^2(\min_{j \neq j' \in \mathcal{A}_i} (\mathcal{U}_{ij}(x) - \mathcal{U}_{ij'}(x))^2)},$$

then set $B_0(\Delta_j^o)$ is positively invariant.

Now to prove Proposition 5.2.1, we only need to prove that, such $\tau$ will lead to $(\exists \tau > 0)(\forall x_0 \in \Delta)(\forall t \geq 0)x(t) \in B_0(\Delta_j^o)$.

The following proof follows similar approach of the proof in Theorem 4.1 in [10]

As $B_0(\Delta_j^o)$ is a positively invariant set, the function $t \rightarrow V_0(x(t))$ is non-increasing, non-negative, and continuous. Therefore, there exists $p \in [0, P_0]$ such that $\lim_{t \rightarrow \infty} V_0(x(t)) = p$. We need to show that $p = 0$. Assume by contradiction that $p > 0$, and let $\tau' \in (0, \tau)$ be small enough. Let

$$\gamma = -\max_{x \in \Delta/B_p(\Delta_j^o)} L_{f+g} V_0(x).$$

Since $V_0$ is $C^1$, $L_{f+g} V_0(x)$ is continuous, and strictly negative in $B_p(\Delta_j^o)$. Therefore $\gamma$
exists and \( \gamma > 0 \). For all \( t \geq 0 \), we have

\[
V_0(x(t)) = V_0(x(0)) + \int_0^t \frac{d}{d\tau} [V_0(x(\tau))] d\tau \\
= V_0(x(0)) + \int_0^t Lf + g V_0(x(\tau)) d\tau \\
\leq V_0(x(0)) - \gamma t.
\]

We deduce that for all \( t > (V_0(x(0)) - p/2)/\gamma \), \( V_0(x(t)) < p/2 \) and therefore \( \lim_{t \to \infty} V(x(t)) < p \), which gives a contradiction. We thus have that \( \lim_{t \to \infty} V(x(t)) = 0 \).

In summary the Q-learning Algorithm converges to an invariant set that includes unperturbed mixed-strategy Nash equilibrium set \( \Delta^o_f \) asymptotically.

\[\square\]

**Remark 5.2.1.** Proposition 5.2.1 provides a Lyapunov function candidate and proves convergence to an invariant level set including unperturbed mixed-strategy Nash equilibria set \( \Delta^o_f \). Using the current Lyapunov function candidate, we cannot provide global convergence to perturbed Nash equilibrium set. The reason is that the replicator dynamics is strategy independent for each player. However, the perturbation would remain partially positive and partially negative for different strategies of the same player. Therefore, it is a great challenge to find a proper Lyapunov function that fits for both parts. To apply Theorem 4.4 in [10], either another Lyapunov function will need to be found, or further assumptions imposed.
Chapter 6

Simulations

In this chapter, we present simulation results of the modified Q-learning algorithm compared to algorithms discussed in Section 2.2, for an example of a congestion game in a similar setup as in [17]. In [17], 100 drivers are choosing between 10 different parallel routes to get to the same destination. This simulation setup provides perfect conditions for all the algorithms discussed in Section 2.2, except for FP, due to the fact that the dimension of empirical frequencies is as high as $10^{99}$.

As stated in Section 2.1.4, a typical congestion game consists of a set $\mathcal{I}$ of $N$ players and a set $\mathcal{R}$ of resources. For each player $i$, let the set of pure strategies $\mathcal{A}_i$ be the set of resources. An action $a_i \in \mathcal{A}_i$ reflects a selection of (multiple) resources, $a_i \in \mathcal{R}$. A player $i$ is “using” resource $r$ if $r \in a_i$. For an action profile $a$, let $q_r(a)$ be the number of drivers using road $r$, i.e., $\{i \in \mathcal{I} : r \in a_i\}$. For each resource $r \in \mathcal{R}$ an associated cost function $c_r$ is defined that reflects the cost of using the resource as a function of the number of players using that resource. In a congestion game, the utility of player $i$ using resources indicated by $a_i$ depends only on the total number of players using the same resources, i.e.,

$$U_i(a) = -\sum_{r \in a_i} c_r(q_r(a)),$$

where the negative sign reflects the cost of using a resource and its effect on a utility function. Any such congestion game is a potential game [21].

In the case of distributed routing, consider the simple case of $N = 100$ players seeking
to traverse from node A to node B along 10 different parallel roads, [17]. Each driver
can select any road as a possible route, so that the set of resources is the set of roads,
\( \mathcal{R} \), and each player can select one road. Each road has a quadratic cost function with
positive (randomly chosen) coefficients,

\[ c_r(q) = a_r q^2 + b_r q + c_r, \quad r = 1, \ldots, 10, \]

where \( q \) represent the number of vehicles on that particular road.

In the following sections, when the number of vehicles on each route and utility of
each route are shown, the legend is omitted since routes are parallel and each route is
represented by the same color in all sets of figures. For the figures of utilities, only when
the utilities of routes converge to one another will the drivers stop switching routes,
because they can no longer improve their utilities. This is intuitively true, as one would
not tend to change route if one think other routes are just as crowded.

6.1 Performance of Unperturbed Modified Q-Learning
with Constant Step-sizes

In this section, we present simulation results of the modified Q-learning algorithm (3.1)
and (3.3) without any perturbations, comparing to the performance of the Fading Mem-
ory Joint Strategy Fictitious Play with Inertia algorithm (FMJSFP-I). The parameter \( \alpha \)
are chosen as 0.5 for all days and all players in both algorithms, and \( \mu \) is chosen as 0.97
for the modified Q-learning algorithm and 0.03 for the FMJSFP-I algorithm.

Fig.6.1 shows results for the modified Q-learning algorithm without any perturbations
by implementing the setup stated above, while Fig.6.2, shows corresponding results ob-
tained by implementing the FMJSFP-I algorithm. Intuitively, the pure Nash equilibrium
of this game setup is when all the drivers find changing routes would only make their
utility lower. That is to say, the utilities of routes should converge to a small range that
changing routes is no longer a better option. In both Fig.6.1 and Fig.6.2, the utilities
of routes are converging. However, the FMJSFP-I algorithm has a better result because
Chapter 6. Simulations

Figure 6.1: Unperturbed MQL with $\mu = 0.97$, $\alpha = 0.5$

Figure 6.2: FMJSFP-I with $\mu = 0.03$, $\alpha = 0.5$
Table 6.1: Statistic Results of Potential Ratio and Total Utility Ratio between Q-Learning and JSFP in 1000 Different Congestion Games

<table>
<thead>
<tr>
<th>Percentage</th>
<th>Minimum</th>
<th>Mean</th>
<th>Median</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Potential</td>
<td>1.0000</td>
<td>1.0289</td>
<td>1.0229</td>
<td>1.1923</td>
</tr>
<tr>
<td>Total Cost</td>
<td>0.7765</td>
<td>0.8822</td>
<td>0.8820</td>
<td>0.9924</td>
</tr>
</tbody>
</table>

Table 6.2: Potential and Total Utility at Time 200 between Q-Learning and JSFP in One Sample Congestion Game

<table>
<thead>
<tr>
<th>Value</th>
<th>Potential(JSFP)</th>
<th>Potential(QL)</th>
<th>Total Cost(JSFP)</th>
<th>Total Cost(QL)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.9851×10^3</td>
<td>1.9966×10^3</td>
<td>1.1737×10^3</td>
<td>9.8684×10^4</td>
</tr>
</tbody>
</table>

the differences between the utilities of routes are smaller. Since it is known that the FMJSFP-I algorithm guarantees convergence to pure Nash equilibria, we can conclude that the unperturbed modified Q-learning algorithm only converges to a sub-optimal state. By comparing the number of vehicles on each route between two algorithms, we can tell the sub-optimal state achieved by the modified Q-learning algorithm is quite close to the pure Nash equilibrium achieved by the FMJSFP-I algorithm.

Comparing the two cases, it can be seen that even the modified Q-learning algorithm requires much lower information aggregation and computation load, the two algorithms have similar convergence time. With the great advantage of requiring no information about utility functions or other players actions, the modified Q-learning algorithm achieved a sub-optimal solution that is fairly close to the true Nash equilibrium obtained in FMJSFP-I. When compared to other reinforcement learning algorithms, this is a reasonably good sub-optimal point to stay on while the perturbation function is still trying to optimize the estimated equilibrium to reach a Nash equilibrium.

Moreover, even though the modified Q-learning algorithm has less information requirements, some advantages are demonstrated in the simulations. We can observe that transient fluctuations in strategies and utilities over time are smaller for the modified Q-learning algorithm comparing to the FMJSFP-I algorithm.

Table 6.1 shows statistic results obtained by randomly generating 1000 different congestion games and comparing the final results of potential function value achieved and accumulated total utility of all drivers over time between the modified Q-learning algo-
Chapter 6. Simulations

rithm and the FMJSFP-I algorithm. It can be seen clearly from the table that with constant step-sizes, the modified Q-learning algorithm can get to around 2% - 3% over the optimal value of the potential function within 200 iterations, while saving more than 10% the cost on average compared to the FMJSFP-I algorithm. According to the simulation results, we are 94.46% confident that the expected total utility is strictly smaller than the FMJSFP-I algorithm.

Table 6.2 gives one example in the 1000 randomly generated congestion games. Within 200 iterations, the total cost saved for 100 drivers are to the level of $10^4$, yet the difference between the potential is only to the level of $10^1$. However, if the process is running indefinitely, the cost saved in earlier optimization stage will not make a big difference in the long run. Eventually, after both algorithms have achieved convergence, the difference between sub-optimal stage and optimal one can accumulate to a much larger number than the cost saved by reducing fluctuations in the beginning. For this reason, perturbations must be introduced in the modified Q-learning algorithm to reach a true Nash equilibrium.

6.2 Performance of Modified Q-Learning with Perturbation Compared to Reinforcement Learning

In this section, we present simulation results of the modified Q-learning algorithm (3.1) and (3.3) with perturbations comparing to the performance of Reinforcement Learning (RL) (2.8), (2.9), and Individual Q-learning algorithm (IQL), (2.11). The parameters $\alpha$ and $\mu$ of the modified Q-learning algorithm are chosen the same values as in Section 6.1. The decreasing step-size sequence for RL are chosen according to (2.10) with $\nu = 0.78$, while the decreasing step-size sequences for IQL are chosen according to (2.12), with $\nu_i$ evenly distributed with in the interval $(0.5, 1]$.

For Reinforcement Learning (RL), we adjusted the reward to be positive to fit the setup for Assumption 2.2.3. In Individual Q-learning algorithm (IQL), since the players have identical utility function, the order of convergence does not affect the result. There-
Figure 6.3: Perturbed MQL with $\zeta = 0.9999$, $\xi = 0.01$

Figure 6.4: RL with $\nu = 0.78$

Therefore, we can conclude that Assumption 2.2.5 is satisfied to apply the IQL algorithm.

From the Figure 6.3, Figure 6.4 and Figure 6.5, we observe that the modified Q-learning algorithm outperformed the other two algorithms. Under the perturbation, the modified Q-learning algorithm explores actions that are not in the estimated best response set, and converges almost surely to the pure Nash equilibrium achieved by the FMJSFP-I algorithm. The perturbed reinforcement learning is the slowest algorithm in the sense of convergence. After $10^4$ iterations, the corresponding expected numbers of vehicles and the expected utilities are not even close to the Nash equilibrium acquired by the FMJSFP-I algorithm. The Individual Q-learning algorithm only converge to a Nash distribution, which is a perturbed Nash equilibrium under the smoothed best response.
In this case, many routes have no vehicles on them and are totally wasted. Therefore, the modified Q-learning algorithm achieves best result in this simulation setup, comparing to the other two learning algorithms.

From Figure 6.6, we note the difference between trajectories of the potential function value over time of the four algorithms. The FMJSFP-I algorithm achieves the optimal potential value within shortest time period, yet require more information aggregation and computation than all the other algorithms. The modified Q-learning algorithm without perturbation would achieve as fast convergence to the sub-optimal point as the FMJSFP-I algorithm. But without the perturbation function, the optimization process is stuck at the sub-optimal point forever. Although the perturbation function slows down the modified Q-learning algorithm in the beginning, it gradually makes the process converge to a true Nash equilibrium. The simulation results confirmed that the modified Q-learning algorithm with the perturbation function achieves better performance in the long run. the IQL algorithm has better converging performance than RL, yet it is stuck at a Nash distribution that has higher potential than those potential values achieved by the modified Q-learning algorithm and the FMJSFP-I algorithm.
6.3 Summary

From the simulation results in the previous two sections, we conclude that the modified Q-Learning algorithm has several advantages.

Comparing to the FMJSFP-I algorithm, the modified Q-learning algorithm requires less information aggregation. At each iteration, only Q-values corresponding to the active actions is updated. The inadequacy of information results in that the modified Q-learning algorithm only converges to a sub-optimal point in potential games. However, reducing information aggregation and computation leads to less transient fluctuations and less the accumulated utility over time.

Comparing to the RL algorithm and the IQL algorithm, the unperturbed modified Q-learning algorithm can achieve a sub-optimal point that has a potential value close to optimal within a short time. The simulation results shows that the sub-optimal point achieved by unperturbed modified Q-learning is better than both other learning algorithms. Therefore, we still observe the significant reduction in accumulated cost. The
modified Q-learning algorithm with the perturbation function requires longer time to converge. However, it converges to the actual pure Nash equilibrium while the other two algorithm only converge to a perturbed Nash equilibrium/a Nash distribution.

With rather large step-sizes used in the modified Q-learning algorithm, only pure Nash equilibrium will be achieved, as the algorithm is designed for pure Nash equilibria. If mixed-strategy Nash equilibria are desired using the modified Q-learning algorithm, as stated in Section 4.3, smaller $\alpha$ in (3.1) should be applied. However, the advantage of achieving sub-optimal point within a short period may no longer be possible.
Chapter 7

Conclusions and Future Work

In this thesis, we considered a Q-learning scheme for distributed convergence to Nash equilibria in potential games. The main difference from prior schemes lies in the choice of step-sizes and perturbation function. The non-negligible constant step-sizes result in faster convergence to an estimated equilibrium. This helps reduce the learning cost to a sub-optimal point while searching for the Nash equilibria.

There are multiple ways that the modified Q-learning algorithm can be applied. Without the perturbations, the modified Q-learning algorithm can be used as a quick estimation of the optimal solution in a large-scale, repeated potential game. When applied repeatedly, the sub-optimal solution may converge to the optimal one eventually. However, almost sure convergence to a pure Nash equilibrium is guaranteed at the cost of the perturbations.

7.1 Conclusions

When compared to JSFP, the modified Q-learning algorithm with constant step-sizes requires less information aggregation, but only reaches a sub-optimal state that can be considered an approximation of a Nash equilibrium. We showed that by appropriately choosing frequency dependent step-sizes, sufficient exploration of all actions is ensured and the estimated equilibrium approaches the Nash equilibria.

Several benefits can be achieved from reducing information aggregation. Firstly, mas-
sive amount of memory space is saved and computation loads for the agents are further reduced comparing to JSFP. Secondly, the utility function of each player does not need to be gathered before the optimization starts. Thirdly, as long as the potential property (2.3) is preserved (for example, the game remains a congestion game), the modified Q-learning algorithm under perturbation can cope with slowly changing environments. Lastly, the delay in the network does not affect the system because only local information is gathered.

In Chapter 3, we first introduced the scheme of the modified Q-learning algorithm with the perturbation function. Without such a function, the modified Q-learning scheme greatly resembles FMJSFP-I. However, like every other reinforcement learning algorithm, a balance of exploration and exploitation is needed. Thus, we introduced the perturbation function inspired by [6], and provided proofs of convergence to estimated equilibria, pure Nash equilibrium, and mixed-strategy Nash equilibria in Chapter 4.

Then, in Chapter 5, we used a continuous time ODE to approximate the behavior of the unmodified Q-learning with negligible constant step-sizes. In the following section, we were able to follow similar procedures in [6] to study the stationary points of the Q-learning approximate ODE.

In Chapter 6, the MATLAB simulation results were presented to confirm the analysis by demonstrating convergence of the modified Q-learning algorithm to a pure Nash equilibrium. We were also able to compare the stochastic process of the potential function value and accumulated cost of the modified Q-learning algorithm, FMJSFP-I, RL, and the unmodified Q-learning algorithm. As a result, the modified Q-learning algorithm outperforms all the other algorithms, as expected.

In Summary, this thesis has proposed a modified Q-learning algorithm and taken a rigorous approach to study the convergence behavior of such Q-learning algorithm in potential games. We were able to prove the almost sure convergence of the modified Q-learning algorithm to pure Nash equilibria and convergence of continuous time approximation of the unmodified Q-learning algorithm with negligible constant step-sizes.
7.2 Future Work

Future work will consider relaxing these conditions, as well as possible extensions to other classes of games.

The next logical step is to identify the sub-optimal set of estimated equilibria that is achievable by the modified Q-learning algorithm with constant step-sizes, and to relax the conditions of convergence to these estimated equilibria. We have shown in Chapter 6 that the modified Q-learning algorithm with constant step-sizes can achieve a sub-optimal estimated equilibrium that is quite close to a true Nash equilibrium. Since the modified Q-learning algorithm does not provide the nice property of monotonically decreasing the value of the potential function, it is a great challenge to study the distribution of sub-optimal estimated equilibria. If we could derive conditions and provide rigorous proof that such advantage is guaranteed by the nature of the algorithm or the game, it would help justify the value of the modified Q-learning algorithm design.

Another direction is to relax the convergence conditions of the modified Q-learning algorithm to a priori hypotheses. The current conditions have to be tested along the way to justify the convergence of the algorithm. However, if we could develop relaxed a priori conditions for the modified Q-learning algorithm, we would be able to identify the games that fit our conditions before we start the simulation or optimization.

One more direction is to relax the assumption on the setup of the game. Although the modified Q-learning algorithm does not require the full information of players’ exact utility matrix, it does require knowing the game yields a global potential function. One possible approach is to relax the potential games to the weakly acyclic games. A weakly acyclic game has the property that for every action in its action set, there is at least one better reply path connecting this action to a sink. Such property is preserved in all potential games, therefore every potential game is a weakly acyclic game. Also such property is crucial in proving the convergence of estimated equilibria to pure Nash equilibria, which makes the weakly acyclic games a good candidate for the next step.
Bibliography


Appendix A

Supplementary proofs

A.1 Proof of Proposition 2.2.1

Proof. At time $k$, let $a_0 = a(k)$. There exists a positive constant $K$, independent of $k$, such that if the current action $a_0$ is repeated $K$ consecutive stages, i.e. $a(k) = \cdots = a(k + K + 1) = a_0$, then $\beta(\tilde{U}(k + K)) = \beta_i(\tilde{U}(k_0))$ for all players. The probability of such an event is that at least $(1 - \bar{\epsilon})^{N(K-1)}$. If the joint action $a_0$ is an equilibrium, then the process will stay at the equilibrium and we are done. Otherwise, there must be at least one player $i' \in I$ such that $a_{i'(1)} \notin \beta_i(\tilde{U}_i(1))$ and hence $a_{i'(1)} \notin \beta_i(\tilde{U}_i(1)(k + K))$.

Consider now the event that, at time $k + K$, exactly one player switches to a different action, i.e. $a_1 = a(k + K) = (a^*_i(1), a_{-i(1)}(0))$ for some player $i(1) \in I$ where $\tilde{U}_i(1)(a_1) > \tilde{U}_i(1)(a_0)$. This event happens with probability at least $e(1 - \bar{\epsilon})^{N-1}$. Note that if $P(\cdot)$ is a generalized ordinal potential function for the game, then $P(a_0) < P(a_1)$.

Continuing along the same lines, if the current action $a_1$ is repeated $K$ consecutive stages, i.e. $a(k + K) = \cdots = a(k + 2K - 1) = a_1$, then $\beta_i(\tilde{U}_i(k + 2K)) = \beta_i(\tilde{U}_i(1))$ for all players. The probability of such an event is at least $(1 - \bar{\epsilon})^{N(K-1)}$. If the joint action $a_1$ is an equilibrium, then we are done. Otherwise, there must be at least one player $i'' \in I$ such that $a_{i''(2)} \notin \beta_i(\tilde{U}_i(2))$ and hence $a_{i''(2)} \notin \beta_i(\tilde{U}_i(2)(k + 2K))$.

One can repeat the argument above to construct a sequence of profiles $a_0, a_1, a_2, \ldots, a_m$, ...
where \( a_l = (a^*_l, a_{-i(l)}(l-1)) \) for all \( l \geq 1 \), with property that

\[
P(a_0) < P(a_1) < \cdots < P(a_m)
\]

and \( a_m \) is an equilibrium. This means that given \( \beta(\vec{U}(k)) \), there exists constants

\[
\tilde{K} = (|A| + 1) * K > 0
\]

\[
\tilde{\epsilon} = (\epsilon(1 - \tilde{\epsilon})^{N-1})^{1/|A|} > 0
\]

both of which are independent of \( k \), such that the following event happens with probability at least \( \tilde{\epsilon} \): \( a(k + \tilde{K}) \) is an equilibrium and \( a_i(k + \tilde{K}) \in \beta_i(\vec{U}(k + \tilde{K})) \) for all players \( i \in I \). This implies that \( a(k) \) converges to a pure equilibrium almost surely. \( \Box \)

A.2 Proof of Proposition 2.2.2

Proof. It is sufficient to show that conditions of Assumption 7.1 in [6] are met. In particular, define the non-negative function

\[
V(x) = P_{\text{max}} - P(x) \geq 0, \quad x \in \Delta
\]

where \( P_{\text{max}} = \sup_{x \in \Delta} P(x) \). Note that \( \nabla_x V(x) = -\mathcal{U}_i(x) \), and

\[
\vec{U}_i(x)^T \vec{g}_i(x) = \vec{U}_i(x)^T \mathcal{U}_i(x) \vec{U}_i(x)
\]

\[
= \sum_{s=1}^{|A_i|} \sum_{j=1, j \neq s}^{|A_i|} x_{is}x_{ij}(\mathcal{U}_{is}(x) - \mathcal{U}_{ij}(x))^2
\]

\[
\geq 0
\]

Thus

\[
\nabla V(x)^T \vec{g}(x) = -\vec{U}(x)^T \vec{g}(x) = -\sum_{i \in I} \vec{U}_i(x)^T \mathcal{U}_i(x) \vec{U}_i(x) \leq 0
\]
for all $x \in \Delta$

It can also be observed that $\nabla V(x)^T \bar{g}(x) = 0$ if and only if $U_{is}(x) = U_{ij}(x)$ for any $s, j \in A_i, s \neq j$ such that $x_{is}, x_{ij} > 0$. By Proposition 5.1, these points correspond to the stationary points of $\bar{g}(x)$. Therefore, the conditions of Assumption 7.1 are satisfied. Thus, the conclusions of Theorem 7.1 hold for the class of games satisfying Property 2.2.1. □
Appendix B

Simulation code for learning algorithms

B.1 Simulation Code for Modified Q-Learning Algorithm

```matlab
function [potential_history, expected_potential_history, ...
    utility_integral, ...
    expected_action_history, expected_utility_history, ...
    action_history, utility_action_history] ...
    = MQLP(alpha,mu0,route_coefficients,x0)

%----------------------- Constants Definition -------- ----------------
zeta = 0.9999;
xi = 0.01;
number_of_routes = 10;
number_of_vehicles = 100;
action_matrix = eye(number_of_routes);
not_action_matrix = 1 - eye(number_of_routes);
max_iter = 20000;
u = ones(1,number_of_vehicles);```

68
%----------------------- Random Input Generation ------------------

if nargin<1
    mu0 = 1-0.03;
    alpha = 0.5;
    route_coefficients = -rand(number_of_routes,3);
    x0 = rand(number_of_routes,1);
    x0 = x0/sum(x0);
elseif nargin<4
    route_coefficients = -rand(number_of_routes,3);
    x0 = rand(number_of_routes,1);
    x0 = x0/sum(x0);
elseif nargin<5
    x0 = rand(number_of_routes,1);
    x0 = x0/sum(x0);
end

%----------------------- Time 1 Initialization ----------------

time = 1;
x = x0*ones(1,number_of_vehicles);
action = randsrc(1,number_of_vehicles,[1:1:number_of_routes;x0']);
bins_histo = 1:number_of_routes;
action_histo = hist(action,bins_histo);
utility_action = route_coefficients(:,1).*(action_histo').^2 + route_coefficients(:,2).*action_histo' + route_coefficients(:,3);
Q_update = utility_action*ones(1,number_of_routes);
Q = Q_update(:,action);
[Q_max,beta_Q] = max(Q,[],1);
x = alpha*x + (1-alpha)*action_matrix(:,beta_Q);
chi = x;
if all(max(x)≥zeta)
    i = ...
    randsrc(1,1,[1:1:number_of_vehicles;ones(1,number_of_vehicles)/number_of_vehicles])
    rho = xi/(1-zeta)^2*(min(max(x))-zeta)^2;
    chi(:,i) = (1-rho)*x(:,i) ...
+ rho*ones(number_of_routes,1)/number_of_routes;
end

expected_action_histo = sum(x,2);

expected_utility_vector ... = route_coefficients(:,1).*expected_action_histo.^2 ...
+ route_coefficients(:,2).*expected_action_histo ...
+ route_coefficients(:,3);

action_histo_history(time,:) = action_histo;

utility_action_history(time,:) = -utility_action;

utility_integral(time) = -sum(utility_action);

expected_action_history(time,:) = expected_action_histo;

expected_utility_history(time,:) = -expected_utility_vector;

expected_potential_history(time) = sum(-expected_utility_vector);

potential_history(time) = Potential(route_coefficients, action_histo);

potential_integral(time) = Potential(route_coefficients, action_histo);

rho = 0;

%----------------------- Time k Evolution -------------------------------
while (time < max_iter)
    time = time + 1;
    for i=1:number_of_vehicles
        action(i)=randsrc(1,1,[1:1:number_of_routes;chi(:,i)'
        if (all(max(x)≥zeta))
            mu(i) = chi(action(i),i);
        else
            mu(i) = mu0;
        end
    end

    bins_histo = 1:number_of_routes;
    action_histo = hist(action,bins_histo);
    utility_action ... = route_coefficients(:,1).*action_histo' .^2 ...
+ route_coefficients(:,2).*action_histo' ...
+ route_coefficients(:,3);
    Q_update = ones(number_of_routes,1)*mu.*Q + utility_action*(1-mu);
    if rho≠0
mu_new = sum(chi.*action_matrix(:,action));
Q_update = ones(number_of_routes,1)*mu_new.*Q + ...
    utility_action*(1-mu_new);
end
Q = Q.*not_action_matrix(:,action) + Q_update.*action_matrix(:,action);
[Q_max, beta_Q]=max(Q,[],1);
x = alpha*x + (1-alpha)*action_matrix(:,beta_Q);
chi = x;
if all(max(x)≥zeta)
i = ...
    randsrc(1,1,[1:1:number_of_vehicles;ones(1,number_of_vehicles)/number_of_vehicles])
    rho = xi/(1-zeta)^2*(min(max(x))-zeta)^2;
    chi(:,i) = (1-rho)*x(:,i) ...
    + rho*ones(number_of_routes,1)/number_of_routes;
end
expected_action_histo = sum(x,2);
expected_utility_vector ...
    = route_coefficients(:,1).*{expected_action_histo}.^2 ...
    + route_coefficients(:,2).*expected_action_histo ...
    + route_coefficients(:,3);
expected_action_history(time,:) = expected_action_histo;
expected_utility_history(time,:)=-expected_utility_vector;
action_history(time,:) = action_histo;
utility_action_history(time,:) = -utility_action;
utility_integral(time) = -sum(utility_action) + ...
    utility_integral(time-1);
expected_potential_history(time) = sum(-expected_utility_vector);
potential_history(time) = Potential(route_coefficients, action_histo);
potential_integral(time) ...
    = Potential(route_coefficients, action_histo) ...
    + potential_integral(time-1);
end
B.2 Simulation Code for Joint Strategy Fictitious Play Algorithm

function [potential_history, utility_integral, ...
    action_history, utility_action_history] ...
    = JSFP(alpha, rho, route_coefficients, x0)

%----------------------- Constants Definition ---------------------
number_of_routes = 10;
number_of_vehicles = 100;
action_matrix = eye(number_of_routes);
not_action_matrix = 1 - eye(number_of_routes);
max_iter = 20000;

%----------------------- Random Input Generation ---------------------
if nargin < 1 % No input
    alpha = 0.5;
rho = 0.03;
    route_coefficients = -rand(number_of_routes, 3);
x0 = rand(number_of_routes, 1);
x0 = x0/sum(x0);
elseif nargin < 3 % Input less than 3 arguments
    route_coefficients = rand(number_of_routes, 3);
x0 = rand(number_of_routes, 1);
x0 = x0/sum(x0);
elseif nargin < 4 % Input less than 4 arguments
    x0 = rand(number_of_routes, 1);
x0 = x0/sum(x0);
end

%----------------------- Time 1 Initialization ---------------------
time = 1;
action = randsrc(1, number_of_vehicles, [1:1:number_of_routes;x0']);
bins_histo = 1:number_of_routes;
Appendix B. Simulation code for learning algorithms

73

action_histo = hist(action, bins_histo);
utility_action ...

= route_coefficients(:,1).* (action_histo').^2 ...
+ route_coefficients(:,2).* action_histo' ...
+ route_coefficients(:,3);
not_action_histo = action_histo + 1;
utility_not_action ...

= route_coefficients(:,1).* (not_action_histo').^2 ...
+ route_coefficients(:,2).* not_action_histo' ...
+ route_coefficients(:,3);
expected_utility_matrix ...

= utility_action*ones(1,number_of_routes).*action_matrix ...
+ utility_not_action*ones(1,number_of_routes).*not_action_matrix;
expected_utility_vector = expected_utility_matrix(:,action);
[expected_utility_maximum, a_star] = max(expected_utility_vector,[],1);
x(1,:) = action;
x(2,:) = a_star;
x_freq = [alpha 1 - alpha];
action_histo_history(time,:) = action_histo;
utility_action_history(time,:) = -utility_action;
utility_integral(time) = -sum(utility_action);
potential_history(time) = Potential(route_coefficients, action_histo);
potential_integral(time) = Potential(route_coefficients, action_histo);

%----------------------- Time k Evolution ---------------------------

while (time < max_iter)

    time = time + 1;
    for i = 1:number_of_vehicles
        temp(i) = randsrc(1,1,[1:1:2;x_freq]);
        action(i) = x(temp(i),i);
    end
    bins_histo = 1:number_of_routes;
    action_histo = hist(action, bins_histo);
    utility_action ...
= route_coefficients(:,1).*(action_histo').^2 ... 
+ route_coefficients(:,2).*action_histo' ... 
+ route_coefficients(:,3);
not_action_histo = action_histo + 1;
utility_not_action ... 
= route_coefficients(:,1).*(not_action_histo').^2 ... 
+ route_coefficients(:,2).*not_action_histo' ... 
+ route_coefficients(:,3);
expected_utility_matrix ... 
= utility_action*ones(1,number_of_routes).*action_matrix ... 
+ utility_not_action*ones(1,number_of_routes).*not_action_matrix;
expected_utility_vector ... 
= rho*expected_utility_matrix(:,action) ... 
+ (1-rho)*expected_utility_vector;
[expected_utility_maximum, a_star] = max(expected_utility_vector,[],1);
x(1,:) = action;
x(2,:) = a_star;
x_freq = [alpha 1-alpha];
action_histo_history(time,:) = action_histo;
utility_action_history(time,:) = -utility_action;
utility_integral(time) ... 
= -sum(utility_action) ... 
+ utility_integral(time-1);
potential_history(time) = Potential(route_coefficients, action_histo);
potential_integral(time) ... 
= Potential(route_coefficients, action_histo) ... 
+ potential_integral(time-1);
B.3 Simulation Code for Reinforcement Learning Algorithm

```matlab
function [potential_history, expected_potential_history, ...
    utility_integral, ... ]
    expected_action_history, expected_utility_history, ...
    action_history, utility_action_history] ...
    = RLfunction(niu, route_coefficients, x0)

%----------------------- Constants Definition ----------------
zetaxi = 0.9999; 0.01;
number_of_routes = 10;
number_of_vehicles = 100;
action_matrix = eye(number_of_routes);
min_utility = -10101;
max_iter = 20000;
lower_bound = zeros(1, number_of_routes);
upper_bound = ones(1, number_of_routes);
opts = optimset('Display', 'off', ...
    'Algorithm', 'interior-point-convex');
%----------------------- Random Input Generation ----------------
if nargin<1
    niu = 0.78;
    route_coefficients = -rand(number_of_routes, 3);
    x0 = rand(number_of_routes, 1);  = x0/sum(x0);
elseif nargin<2
    route_coefficients = -rand(number_of_routes, 3);
    x0 = rand(number_of_routes, 1);
    x0 = x0/sum(x0);
elseif nargin<3
    x0 = rand(number_of_routes, 1);
    x0 = x0/sum(x0);
```
end

%----------------------- Time 1 Initialization -----------------

time = 1;
mu = 1/(time^niu+1);
x = x0*ones(1,number_of_vehicles);
action = randsrc(1,number_of_vehicles, [1:1:number_of_routes;x0']);
bins_histo = 1:number_of_routes;
action_histo = hist(action, bins_histo);
utility_action ...
= route_coefficients(:,1).*(action_histo').^2 ...
+ route_coefficients(:,2).*action_histo' ...
+ route_coefficients(:,3);
x = x + mu.(*(action_matrix(:,action) - x) .*(ones(number_of_routes,1) ...
* (((utility_action(action)- min_utility)' / abs(min_utility)));
if (min(min(x))<0 || max(max(x))>1 )
for i=1:number_of_vehicles
  if (min(x(:,i))<0 || max(x(:,i)>1))
    x(:,i) ...
    = quadprog(eye(number_of_routes), x(:,i), ...
                [], [], ones(1,number_of_routes), 1, ...
                lower_bound, upper_bound, [], opts);
  end
end
chi = x;
if max(max(x))>zeta
  for i = 1:size(x,2)
    if max(x(:,i))>zeta
      rho = x(:,i)/(1-zeta)^2*(max(x(:,i))-zeta)^2;
      chi(:,i) = (1-rho)*x(:,i) ...
                 + rho*ones(number_of_routes,1)/number_of_routes;
    end
  end
end
expected_action_histo = sum(x,2);
expected_utility_vector ... = route_coefficients(:,1).*expected_action_histo.^2 ...
+ route_coefficients(:,2).*expected_action_histo ...
+ route_coefficients(:,3);
action_histo_history(time,:) = action_histo;
utility_action_history(time,:) = -utility_action;
utility_integral(time) = -sum(utility_action);
expected_action_history(time,:) = expected_action_histo;
expected_utility_history(time,:) = -expected_utility_vector;
expected_potential_history(time) = Potential(route_coefficients, ...
    expected_action_histo');
potential_history(time) = Potential(route_coefficients, action_histo);
potential_integral(time) = Potential(route_coefficients, action_histo);

%----------------------- Time k Evolution ------------- -----------
while (time < max_iter)
    time = time + 1;
    mu = 1/(time^niu+1);
    for i=1:number_of_vehicles
        action(i)=randsrc(1,1,[1:1:number_of_routes;chi(:,i)']);
    end
    bins_histo = 1:number_of_routes;
    action_histo = hist(action,bins_histo);
    utility_action ...
        = route_coefficients(:,1).*action_histo'.^2 ...
        + route_coefficients(:,2).*action_histo' ...
        + route_coefficients(:,3);
    x = x ...
        + mu.*((action_matrix(:,action) - x) ...
            .*ones(number_of_routes,1) ... *
            *((utility_action(action)- min_utility')/abs(min_utility)));
    if (min(min(x))<0 || max(max(x))>1)
        for i=1:number_of_vehicles
            if (min(x(:,i))<0 || max(x(:,i))>1)
                x(:,i) ...
            = quadprog(eye(number_of_routes), x(:,i), ...
Appendix B. Simulation code for learning algorithms

```matlab
[], [], ones(1, number_of_routes), 1, ...
lower_bound, upper_bound, [], opts);
end
end
end
chi = x;
if max(max(x)) ≥ zeta
    for i = 1:size(x, 2)
        if max(x(:, i)) ≥ zeta
            rho = x(i)/ (1-zeta)^2 * (max(x(:, i)) - zeta)^2;
            chi(:, i) = (1-rho) * x(:, i) ...
                + rho * ones(number_of_routes, 1)/number_of_routes;
        end
    end
end
expected_action_histo = sum(x, 2);
expected.utility_vector ...
    = route.coefficients(:, 1) .* (expected_action_histo).^2 ...
        + route.coefficients(:, 2) .* expected_action_histo ...
        + route.coefficients(:, 3);
expected.action.history(time,:) = expected_action_histo;
expected.utility.history(time,:) = -expected.utility.vector;
action.history(time,:) = action_histo;
utility.action.history(time,:) = -utility_action;
utility_integral(time) = -sum(utility_action) + ...
    utility_integral(time-1);
expected.potential.history(time) = Potential(route.coefficients, ...
    expected.action_histo');
potential.history(time) = Potential(route.coefficients, action_histo);
potential_integral(time) ...
    = Potential(route.coefficients, action_histo) ...
        + potential_integral(time-1);
end
end
```
B.4 Simulation Code for Individual Q-Learning Algorithm with Multiple-Timescales

```matlab
function [potential_history, expected_potential_history, ...
    utility_integral, ... 
    expected_action_history, expected_utility_history, ... 
    action_history, utility_action_history] ...
    = IQL(mu0,tau,route_coefficients,x0)

%----------------------- Constants Definition -------------------------
number_of_routes = 10;
number_of_vehicles = 100;
action_matrix = eye(number_of_routes);
not_action_matrix = 1 - eye(number_of_routes);
max_iter = 20000;

%----------------------- Random Input Generation -----------------------
if nargin<1
    tau = 0.1;
    mu0 = ((1:number_of_vehicles) + ... 
           number_of_vehicles)/(2*number_of_vehicles);
    route_coefficients = -rand(number_of_routes,3);
    x0 = rand(number_of_routes,1);
    x0 = x0/sum(x0);
elseif nargin<4
    route_coefficients = -rand(number_of_routes,3);
    x0 = rand(number_of_routes,1);
    x0 = x0/sum(x0);
elseif nargin<5
    x0 = rand(number_of_routes,1);
    x0 = x0/sum(x0);
end

%----------------------- Time 1 Initialization -----------------------
```

Appendix B. Simulation code for learning algorithms

79


```matlab
30 time = 1;
31 mu = (time+100).^(-mu0);
32 x = x0*ones(1,number_of_vehicles);
33 action = randsrc(1,number_of_vehicles,[1:1:number_of_routes;x0']);
34 bins_histo = 1:number_of_routes;
35 action_histo = hist(action,bins_histo);
36 utility_action ...
37   = route_coefficients(:,1).*action_histo' .^2 ...
38       + route_coefficients(:,2).*action_histo' ...
39       + route_coefficients(:,3);
40 Q_update = utility_action*ones(1,number_of_routes);
41 Q = Q_update(:,action);
42 x = (exp(Q/tau))./(ones(number_of_routes,1)*sum(exp(Q/tau),1));
43 expected_action_histo = sum(x,2);
44 expected_utility_vector ...
45   = route_coefficients(:,1).*expected_action_histo' .^2 ...
46       + route_coefficients(:,2).*expected_action_histo ...
47       + route_coefficients(:,3);
48 action_histo_history(time,:) = action_histo;
49 utility_action_history(time,:) = -utility_action;
50 utility_integral(time) = -sum(utility_action);
51 expected_action_history(time,:) = expected_action_histo;
52 expected_utility_history(time,:) = -expected_utility_vector;
53 expected_potential_history(time) = sum(-expected_utility_vector);
54 potential_history(time) = Potential(route_coefficients, action_histo);
55 potential_integral(time) = Potential(route_coefficients, action_histo);
56
%----------------------- Time k Evolution -----------------------
57 while (time < max_iter)
58   time = time + 1;
59   mu = (time+100).^(-mu0);
60   for i=1:number_of_vehicles
61     action(i) = randsrc(1,1,[1:1:number_of_routes;x(:,i)']);
62     beta_q(i) = (exp(Q(action(i),i)/tau))/(sum(exp(Q(:,i)/tau)));
63     action_q(i) = Q(action(i),i);
64   end
```
Appendix B. Simulation code for learning algorithms

```plaintext
66  bins_histo = 1:number_of_routes;
67  action_histo = hist(action,bins_histo);
68  utility_action ...
69         = route_coefficients(:,1).*(action_histo')^2 ...
70         + route_coefficients(:,2).*action_histo' ... 
71         + route_coefficients(:,3);
72  Q_update = action_q + mu.*(utility_action(action)' - action_q)/beta_q;
73  Q = Q.*not_action_matrix(:,action) + ...
74      ones(number_of_routes,1)*Q_update.*action_matrix(:,action);
75  x = (exp(Q/tau))./(ones(number_of_routes,1)*sum(exp(Q/tau),1));
76  expected_action_histo = sum(x,2);
77  expected_utility_vector ...
78         = route_coefficients(:,1).*(expected_action_histo)^2 ... 
79         + route_coefficients(:,2).*expected_action_histo ... 
80         + route_coefficients(:,3);
81  expected_action_history(time,:) = expected_action_histo;
82  expected_utility_history(time,:)=-expected_utility_vector;
83  action_history(time,:) = action_histo;
84  utility_action_history(time,:) = -utility_action;
85  utility_integral(time) = -sum(utility_action) + ...
86           utility_integral(time-1);
87  expected_potential_history(time) = sum(-expected_utility_vector);
88  potential_history(time) = Potential(route_coefficients, action_histo);
89  potential_integral(time) ...
90    = Potential(route_coefficients, action_histo) ... 
91      + potential_integral(time-1);
92  end
93  end
```
B.5 Simulation Code for Plot Generation

```matlab
clear
clc

number_of_routes = 10;
number_of_vehicles = 100;
max_iter = 20000;

alpha = 0.5;
rho = 0.03;
tau = 0.1;
mu = 1 - 0.03;
niu = 0.78;
mu0 = ((1:number_of_vehicles) + number_of_vehicles)/(2*number_of_vehicles);
route_coefficients = -rand(number_of_routes,3);
x0 = rand(number_of_routes,1);
x0 = x0/sum(x0);

[PotentialIQL, Expected_PotentialIQL, USumIQL, ... 
  Expected_ActionIQL, Expected_UtilityIQL, ... 
  ActionIQL, UtilityIQL] ... 
  = IQL(mu0,tau,route_coefficients,x0);

[PotentialJSFP, USumJSFP, ActionJSFP, UtilityJSFP] ... 
  = JSFP(alpha,rho,route_coefficients,x0);

[PotentialQLNP, Expected_PotentialQLNP, USumQLNP, ... 
  Expected_ActionQLNP, Expected_UtilityQLNP, ActionQLNP, UtilityQLNP] ... 
  = MQL_NP(alpha,mu,route_coefficients,x0);

[PotentialQLP, Expected_PotentialQLP, USumQLP, ... 
  Expected_ActionQLP, Expected_UtilityQLP, ActionQLP, UtilityQLP] ... 
  = MQL_P(alpha,mu,route_coefficients,x0);
```
Appendix B. Simulation code for learning algorithms

```matlab
[PotentialRL, Expected_PotentialRL, USumRL, ... ]
Expected_ActionRL, Expected_UtilityRL, ...
ActionRL, UtilityRL] ...
= RL_P(niu, route_coefficients, x0);

figure
semilogy(PotentialQLNP, 'b')
hold on
semilogy(PotentialQLP, 'm')
semilogy(PotentialIQL, 'g')
semilogy(PotentialJSFP, 'r')
semilogy(Expected_PotentialRL, 'c')
legend('QL', 'QLp', 'IQL', 'JSFP', 'RL(EP)')
ylim([700, 10000])
title('(Expected) Value of Potential Function Vs. Time')
xlabel('Time')
ylabel('(Expected) Potential')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% JSFP
figure
plot(ActionJSFP)
xlim([0 200])
title('Number of vehicles on each route Vs. Time')
xlabel('Time')
ylabel('Number of vehicles')

figure
plot(UtilityJSFP)
xlim([0 200])
title('Utility of each route Vs. Time')
xlabel('Time')
ylabel('Utility')
```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% MQL_NP
figure
plot(ActionQLNP)
xlim([0 200])
title('Number of vehicles on each route Vs. Time')
xlabel('Time')
ylabel('Number of vehicles')

figure
plot(UtilityQLNP)
xlim([0 200])
title('Utility of each route Vs. Time')
xlabel('Time')
ylabel('Utility')

% MQL_P
figure
plot(ActionQLP)
title('Number of vehicles on each route Vs. Time')
xlabel('Time')
ylabel('Number of vehicles')

figure
plot(UtilityQLP)
ylim([0,500])
title('Utility of each route Vs. Time')
xlabel('Time')
ylabel('Utility')

% RL
figure
plot(Expected_ActionRL)
title('Expected number of vehicles on each route Vs. Time')
xlabel('Time')
Appendix B. Simulation code for learning algorithms

85

ylabel('Expected number of vehicles')

figure

plot(Expected_UtilityRL)

ylim([0, 500])

title('Expected utility of each route Vs. Time')

xlabel('Time')
ylabel('Expected utility')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% IQL

figure

plot(ActionIQL)

title('Number of vehicles on each route Vs. Time')

xlabel('Time')
ylabel('Number of vehicles')

figure

plot(UtilityIQL)

ylim([0, 500])

title('Utility of each route Vs. Time')

xlabel('Time')
ylabel('Utility')