Efficient Reinforcement Learning in Multiple-Agent Systems and its Application in Cognitive Radio Networks

Jing Zhang
Western Michigan University, zhnjn@yahoo.com

Follow this and additional works at: https://scholarworks.wmich.edu/dissertations

Part of the OS and Networks Commons, and the Theory and Algorithms Commons

Recommended Citation
Zhang, Jing, "Efficient Reinforcement Learning in Multiple-Agent Systems and its Application in Cognitive Radio Networks" (2012). Dissertations. 11. https://scholarworks.wmich.edu/dissertations/11
EFFICIENT REINFORCEMENT LEARNING IN MULTIPLE-AGENT SYSTEMS AND ITS APPLICATION IN COGNITIVE RADIO NETWORKS

by

Jing Zhang

A Dissertation
Submitted to the
Faculty of The Graduate College
in partial fulfillment of the
requirements for the
Degree of Doctor of Philosophy
Department of Computer Science
Advisor: Dionysios Kountanis, Ph.D.

Western Michigan University
Kalamazoo, Michigan
April 2012
WE HEREBY APPROVE THE DISSERTATION SUBMITTED BY

Jing Zhang

ENTITLED EFFICIENT REINFORCEMENT LEARNING IN MULTIPLE-AGENT SYSTEMS AND ITS APPLICATION IN COGNITIVE RADIO NETWORKS

AS PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF Doctor of Philosophy

Computer Science

(Department)

Computer Science

(Program)

APPROVED

Date

April 2012

Dionysios Koufiantes, Ph.D.
Dissertation Review Committee Chair

Ala Al-Fuqaha, Ph.D.
Dissertation Review Committee Member

Leszek Lilien, Ph.D.
Dissertation Review Committee Member

Liang Dong, Ph.D.
Dissertation Review Committee Member

Dean of the Graduate College

Date
EFFICIENT REINFORCEMENT LEARNING IN MULTIPLE-AGENT SYSTEMS AND ITS APPLICATION IN COGNITIVE RADIO NETWORKS

Jing Zhang, Ph.D.
Western Michigan University, 2012

The objective of reinforcement learning in multiple-agent systems is to find an efficient learning method for the agents to behave optimally. Finding Nash equilibrium has become the common learning target for the optimality. However, finding Nash equilibrium is a PPAD (Polynomial Parity Arguments on Directed graphs)-complete problem. The conventional methods can find Nash equilibrium for some special types of Markov games.

This dissertation proposes a new reinforcement learning algorithm to improve the search efficiency and effectiveness for multiple-agent systems. This algorithm is based on the definition of Nash equilibrium and utilizes the greedy and rational features of the agents. When the agents adjust their behavior strategies following certain rules based on the feedback, their behavior strategies display special patterns. The special patterns are tightly related to the Nash equilibrium. The agents can find their Nash equilibrium strategies according to the patterns’
properties even though each of the agents doesn’t have information about the other agents.

The new reinforcement learning algorithm can be applied in many areas as long as the target problem can be mapped to a Markov game. We apply the learning algorithm to solve the spectrum sharing problem in cognitive radio networks.

There are several contributions of this research. First, the proposed reinforcement learning algorithm for multiple-agent systems doesn’t require the agents to have information about other agents. Second, our learning algorithm is more efficient than other similar learning algorithms. Third, the learning algorithm also effectively finds a Nash equilibrium. Fourth, the learning algorithm attempts to impact the learning problem in multiple-state Markov games. The algorithm is expected to be extended to the scalable Markov games. Last but not least, this reinforcement learning algorithm can be applied in many areas. We have applied the learning algorithm to find solution to the spectrum sharing problem in a cognitive radio network model demonstrated in the dissertation.
ACKNOWLEDGMENTS

I would like to thank my academic advisor and committee chair, Dr. Dionysios Kountanis for his patience, professional guidance, and continual support in my PhD journey. I would also thank for Dr. Ala Al-Fuqaha for his help with his expertise and his encouragement. Many thanks are given to Dr. Leszek Lilien for his valuable technique article-writing skills. I also want to express my appreciation to Dr. Liang Dong for his professional advice.

I want to thank my husband, Guowei He for his endless support, sacrifice, help and continual encouragement. Without him, I couldn’t have reached this point. I want to thank my parents Zufeng Zhang and Shiqing Xiang. They are always the source of my spiritual power.

Jing Zhang
# TABLE OF CONTENTS

ACKNOWLEDGMENTS .................................................................................. ii

LIST OF TABLES ..................................................................................... viii

LIST OF FIGURES ................................................................................... ix

CHAPTER

I. INTRODUCTION .................................................................................. 1

1.1. Introduction to Agent .................................................................... 1

1.2. Multiple-Agent Systems ............................................................... 2

1.3. Reinforcement Learning ............................................................... 4

1.3.1. Introduction to Reinforcement Learning .............................. 5

1.3.2. Reinforcement Learning for Single-Agent Systems .............. 7

1.3.3. Reinforcement Learning for Multiple-Agent Systems .......... 11

1.4. The Application Areas of Reinforcement Learning ............... 13

1.4.1. Application Areas .................................................................. 13

1.4.2. Cognitive Radio Networks ...................................................... 14

1.5. Motivations and Contributions .................................................... 16

1.5.1. Motivations .......................................................................... 16

1.5.2. Contributions ...................................................................... 19

1.6. Outline ......................................................................................... 20

II. LITERATURE REVIEW ..................................................................... 21
Table of Contents—continued

CHAPTER

2.1. Reinforcement Learning for Multiple-Agent Systems ................................................................. 22
  2.1.1. Strategic Form Games ................................................. 22
  2.1.2. Learning Rules .......................................................... 24
  2.1.3. Reinforcement Learning for Strategic Form Games in Game Theory ................................. 25
  2.1.4. Reinforcement Learning for Strategic Form Games in Artificial Intelligence .................... 28
  2.1.5. Reinforcement Learning for Multiple-State Systems ......................................................... 33

  2.2.1. Spectrum Sharing Methods in Cognitive Radio Networks .................................................. 34
  2.2.2. Learning Methods Applied in Cognitive Radio Networks ................................................ 35

2.3. Summary ............................................................................. 36

III. NEW EFFICIENT REINFORCEMENT LEARNING ALGORITHM FOR SINGLE-STATE MULTIPLE-AGENT SYSTEMS ........................................................................................................... 38

3.1. The Problem of Reinforcement Learning in Multiple-Agent Systems ........................................ 39
  3.1.1. Decomposing the Problem .............................................. 39
  3.1.2. Learning Target ............................................................. 41
Table of Contents—continued

CHAPTER

3.1.3. Information Needed ................................. 42

3.1.4. The Definition of Reinforcement Learning
Problem ................................................. 42

3.2. Characteristics of the Agents .......................... 43

3.2.1. Rationality ........................................ 43

3.2.2. Greed ............................................. 44

3.3. Formal Definition for Nash Equilibrium and Its
Features .................................................... 45

3.4. A New Learning Algorithm with No Information of
the Environment ......................................... 52

3.4.1. Analysis ......................................... 53

3.4.2. Finding NE with Knowing Others’
Strategies ............................................... 59

3.4.3. Finding NE without Knowing Any
Information ............................................. 66

3.5. Experimental Results ................................. 70

3.6. Summary ............................................. 87

IV. REINFORCEMENT LEARNING FOR MULTIPLE-STATE
MARKOV GAMES ........................................... 88

4.1. Reinforcement Learning Problem in Multiple-State
Markov Games ........................................... 88

4.1.1. Learning Target ................................ 89

4.1.2. The Reinforcement Learning Problem in
Multiple-State Multiple-Agent Systems .......... 89
# Table of Contents—continued

## CHAPTER

4.2. Analysis ............................................................................................................. 89

4.2.1. Computing Payoffs in Multiple-State Multiple-Agent Systems .................. 89

4.2.2. Convergence ............................................................................................... 91

4.3. The New Reinforcement Learning Algorithm .............................................. 92

4.4. The Experimental Results ............................................................................. 95

4.4.1. Grid World ................................................................................................. 95

4.4.2. Push Box ................................................................................................... 100

4.5. Summary ....................................................................................................... 105

## V. THE APPLICATION OF THE LEARNING ALGORITHM IN COGNITIVE RADIO NETWORKS ........................................... 107

5.1. The Spectrum Sharing Problem in Cognitive Radio Networks .......................... 107

5.2. Mapping Cognitive Radio Networks to Multiple-Agent Systems ...................... 108

5.3. A Cognitive Radio Network Model ................................................................. 109

5.4. Two Learning Methods Applied in Resource Sharing ...................................... 111

5.4.1. Cooperation and Competition ..................................................................... 111

5.4.2. Efficiency and Fairness .............................................................................. 112

5.4.3. A No-Regret Learning Algorithm for Single Agent ...................................... 112
Table of Contents—continued

CHAPTER

5.4.4. A No-Regret Learning Algorithm for Multiple-Agent Systems ......................................... 114
5.4.5. Experimental Results ................................................................. 115
5.5. Summary ................................................................................. 118

VI. CONCLUSIONS AND FUTURE WORK ........................................... 119

6.1. The Innovations of the Reinforcement Learning Algorithm for Multiple-Agent Systems ........... 119
6.2. Further Research Topics ............................................................ 121

REFERENCES .................................................................................. 123
LIST OF TABLES

3.1. The Strategies of 3 Agents in Case 5 ........................................... 87
4.1. The Strategies of the Two Agents in Every State ......................... 99
4.2. Strategies for Each Agents in the States ..................................... 105
5.1. Transmitting Rates of the Second Users ..................................... 116
5.2. Strategies by Applying Reinforcement Learning Algorithm .......... 117
5.3. Strategies by Applying No-Regret Learning Algorithm ................. 117
LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>A General Model of Multiple-Agent System</td>
<td>3</td>
</tr>
<tr>
<td>1.2</td>
<td>An Example of Spectrum Holes</td>
<td>14</td>
</tr>
<tr>
<td>3.1</td>
<td>The Curve of $f(x) = \theta^x$, $\theta = 1.5$</td>
<td>54</td>
</tr>
<tr>
<td>3.2</td>
<td>Learning Process: Initial Strategies-([0.27, 0.73], [0.69, 0.31], [0.84, 0.16])</td>
<td>55</td>
</tr>
<tr>
<td>3.3</td>
<td>Learning Process: Initial Strategies-([0.47, 0.53], [0.58, 0.42], [0.55, 0.45])</td>
<td>56</td>
</tr>
<tr>
<td>3.4</td>
<td>The Learning Process of Two Person Matching Pennies Game</td>
<td>65</td>
</tr>
<tr>
<td>3.5</td>
<td>The Learning Process of Two Person Matching Pennies Game with NE as the Initial Strategy Profile</td>
<td>66</td>
</tr>
<tr>
<td>3.6</td>
<td>Case 1: Learning Process</td>
<td>76</td>
</tr>
<tr>
<td>3.7</td>
<td>Case 1: Convergence Process-One Action for Each Agent</td>
<td>77</td>
</tr>
<tr>
<td>3.8</td>
<td>Case 2: Learning Processes of Round 1 and Round 12</td>
<td>79</td>
</tr>
<tr>
<td>3.9</td>
<td>Case 2: Convergence Process - One Action for Each Agent</td>
<td>79</td>
</tr>
<tr>
<td>3.10</td>
<td>Case 3: Learning Processes for Round 1 and Round 10</td>
<td>81</td>
</tr>
<tr>
<td>3.11</td>
<td>Case 3: Convergence Process - One Action for Each Agent</td>
<td>81</td>
</tr>
<tr>
<td>3.12</td>
<td>Comparing Algorithm 3.2. with WoLF Algorithm</td>
<td>82</td>
</tr>
<tr>
<td>3.13</td>
<td>Case 4: Learning Processes for Round 1 and Round 13</td>
<td>84</td>
</tr>
<tr>
<td>3.14</td>
<td>Case 4: Convergence Process</td>
<td>84</td>
</tr>
<tr>
<td>3.15</td>
<td>Case 5: Learning Process for Round 1 and Round 6</td>
<td>86</td>
</tr>
<tr>
<td>4.1</td>
<td>The Grid World</td>
<td>95</td>
</tr>
</tbody>
</table>
List of Figures – Continued

4.2. Push Box Game .................................................................................. 101

5.1. A Cognitive Radio Network Model...................................................... 110

5.2. The Curve of $f(x) = \beta^x, \beta = 1.5$............................................... 113
CHAPTER I

INTRODUCTION

Intelligent agent is the fundamental research object in the artificial intelligence area. It's a central task for the scientists to design intelligent agents with learning capability.

Reinforcement Learning is one important type of machine learning. Reinforcement learning in multiple-agent systems can be applied in many areas. However, the complicated environments in multiple-agent systems make it very difficult for the agents to learn. Reinforcement learning in multiple-agent systems is still far from being mature. Our research is inspired by this situation. This dissertation presents new reinforcement learning algorithms for multiple-agent systems. The method is more efficient and effective than other methods.

In this chapter, basic concepts about intelligent agents and machine learning are introduced first. The motivations and the contributions of the research are presented next. Finally, the structure of the dissertation is outlined.

1.1. Introduction to Agent

An intelligent agent (or “an agent” in short) is the fundamental research object in artificial intelligence area. An agent can be anything that is able to perceive its environment and act upon the observed environment (Russell and Norvig 2003). For example, a robot is an agent -
it takes actions after “watching” its environment; an automatic door is an agent - it opens when it senses objects approaching; even a software application is an agent - it gives outputs to the users after it takes inputs from the users.

According to (Russell and Norvig 2003), there are different types of agents. Simple reflex agents respond immediately to what they have observed, like the automatic door agent. Model-based reflex agents maintain internal state depending on the observation history. These agents can act more effective in a partial observable environment. Goal-based agents make decisions driven by their goals. Utility-based agents pursue higher utility. A utility function maps a state or a sequence of states onto a real number which describes the associated degree of satisfaction. Learning agents are equipped with capability which can improve their behavior based on the feedbacks reflecting the quality of the behavior.

It is a central task for the scientists to design intelligent agents which are able to smartly respond to the environment. The means of agent classification are not unique. Agents equipped with learning capability are more intelligent. Our research will focus on this type of agents.

1.2. Multiple-Agent Systems

A single agent may be useful in some circumstances. However, in many situations multiple agents co-exist in the same environments. For example, the nodes in a network, buyers and sellers in a market, and a
group of agents in a team are all the examples of multiple-agent systems.

Figure 1.1. shows a general model of a multiple-agent system. Each agent contains a performance element which can take certain actions. Actions from all the agents change the state of the environment accordingly. The agents can sense the state change and get feedback from the environment. The learning element of each agent studies the feedback and enriches its knowledge. The agents then adjust their strategies to better ones through learning. New strategies guide their next actions impacting the environment.

![Figure 1.1. A General Model of Multiple-Agent System](image)

A multiple-agent system consists of a group of agents that can interact with each other. By “interact”, we mean that each of the agents’
rewards (feedback from the environment) is affected by other agents’ behaviors. The agents in the system can be of the same type or of different types. That means that the agents may be designed to have different functions or equipments. A multiple-agent system is able to accomplish certain tasks. The agents may compete with each other to realize their individual goal optimally, or they may cooperate to achieve a common goal. Some agents might have the capability to communicate.

Agents take actions to respond to the environment. Meanwhile, these actions also impact other agents’ observation to the environment. These interactions of the agents will greatly affect the behaviors of the agents in a multiple-agent system. Therefore, the design of the agents in a multiple-agent system is quite different than that of single agent, particularly the learning capability.

1.3 Reinforcement Learning

*Reinforcement learning* means that an agent can learn to behave optimally based on the feedbacks from its actions expected upon the environment. Reinforcement learning in single-agent systems is different from that in multiple-agent systems. The interactions among the agents in a multiple-agent system impact the agents’ learning. This makes the learning for multiple-agent systems much harder than that for single-agent systems. It is a problem how do the agents learn in a multiple-agent system.

On the other hand, the applicability of reinforcement learning makes solving the learning problem meaningful. Therefore, another issue
of this research is the application of reinforcement learning in multiple-agent systems. Especially, we will concentrate the application of the learning algorithm for solving the spectrum sharing problem in cognitive radio networks.

1.3.1. Introduction to Reinforcement Learning

An agent is capable to observe and take actions upon its observations. Then, how does an agent take an action to respond to the environment? An agent must have some knowledge to help it make certain decisions. For example, an automatic door must have a knowledge database. The database should specify that the action is opening if there is an object approaching, and the action is closing if there is nothing nearby. This kind of knowledge database has already been set up when the door agent is created.

The mankind is much smarter than other creatures because man can obtain knowledge through learning. There are two ways to design an intelligent agent. One is setting up knowledge database initially. Another way is to design the learning mechanism for an intelligent agent. If an agent is able to learn, more and more knowledge will be accumulated by it. The agent will be getting more and more intelligent.

Generally speaking, learning can range from trivial memorization of experience to the creation of entire scientific theories (Russell and Norvig 2003). For agents, learning is a method that improves their performance over time based on data obtained from sensors and database. It’s typically a statistical process.
To be intelligent, the learning should be guided by rationality. This means that an agent's actions should be reasonably optimal (Osborne 2003). In (Russell and Norvig 2003 p36), a rational agent is defined as below:

"For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has."

Machine learning usually is distinguished into three categories: supervised learning, unsupervised learning and reinforcement learning. Supervised learning is a method of learning a function from the inputs and outputs of provided examples. The goal of learning is to produce correct output from a new input guided by the samples. For unsupervised learning, an agent models a set of inputs without specific output example values available. Therefore, in unsupervised learning there are neither supervised target outputs, nor rewards from the environment for guidance.

The one we are most interested in is reinforcement learning. Reinforcement learning is a learning method based on Markov Decision Process (MDP) model in which an agent learns from the feedback of their behavior acting on the environment. It means that an agent learns an optimal (or nearly optimal) policy by using observed rewards (or reinforcement) through trial-and-error interactions within the dynamic environment. Reinforcement learning is rooted in the human learning process. It is attractive because it is more intelligent and adaptive than other learning methods. Moreover, it can be applied in many complex domains where other learning methods may not be applicable.
Reinforcement learning is mature in single agent situations. But in multiple-agent systems, reinforcement learning is still very immature. The most difficult issue for reinforcement learning in multiple-agent systems is that the dynamics of concurrently learning systems can be very complicated (Vlassis 2007). This is because an agent not only needs to deal with the environment, but also needs to face the dynamic changes brought by other agents who are learning as well.

1.3.2. Reinforcement Learning for Single-Agent Systems

In reinforcement learning, the environment of the agent is formulated as a Markov decision process (MDP).

An MDP is defined by a tuple \( < S, A, T, R > \), where:

- \( S \) is a discrete set of states, \( s \in S \);
- \( A \) is a discrete set of actions, \( a \in A \);
- \( T \) is a stochastic transition function that describes that the environment transitions stochastically to state \( s' \) when the agent takes action \( a \) in state \( s \), denoted as \( p(s'|s,a) \);
- \( R \) is the reward function that reflects the feedback from the environment when the agent takes action \( a \) at state \( s \), denoted as \( R(s, a) \), and \( R: S \times A \rightarrow \mathbb{R} \).

Every MDP model is based on a planning horizon, which is a time series and can be infinite.

The planning problem is how an agent should behave in all the states in a Markov decision process. In other words, what action the agent should take in the state? Usually an agent is rational, that is, it always
seeks an optimal answer. It makes a plan to get maximum accumulative
rewards (or payoffs, utilities, they are interchangeable) in any state over
the planning horizon.

The accumulative payoff for a state sequence is computed by the
following method:

\[ V([s_t, s_{t+1}, s_{t+2}, ...]) = R(s_t, a) + \gamma R(s_{t+1}, a) + \gamma^2 R(s_{t+2}, a) + ... \]  

(1.1)

where \( V \) is the expected accumulative reward for the state sequence, \( s_t \) is
the state at time \( t \), and \( \gamma \) is the discount factor between 0 and 1.

The value of \( \gamma \) conveys two aspects of information. First, it shows
the preference of the agent for current rewards over future rewards.
Second, it ensures that the result remains finite for an infinite planning
horizon.

The solution for the planning problem is called a policy (or strategy),
and specifies the actions of the agent in each state. Policy is an optimal
policy if it yields the maximum expected accumulative reward. That is:

\[ \pi^* = \arg \max_\pi E[\sum_{t=0}^\infty \gamma^t R(s_t, a) | \pi] \]  

(1.2)

where \( \pi^* \) indicates the optimal policy, \( E \) means expected value, and \( \pi \)
stands for any policy. The starting point is the initial state \( s_0 \), where \( t = 0 \).

Since the environment will transition to different states with
different probabilities from time \( t \) to \( t + 1 \) when the agent takes different
actions, we need to take the transition model into account to compute the
accumulative rewards. To distinguish the accumulative rewards and the immediate rewards, we use \( V \) to record the expected accumulative reward for each state. To compute the value of the accumulative reward for the current state, assume we know the values of the accumulative rewards for all the next states \((V(s'))\) no matter which action is chosen. According to (1.1) and taking the transition model into account, we get the formula for computing the accumulative reward for the current state as follows:

\[
V(s) = R(s, \alpha) + \gamma \max_{\alpha'} \sum_{s'} T(s, \alpha, s') V(s')
\]

(1.3)

where \( s' \) is the new state the system transitions to when the agent takes action \( \alpha \) in state \( s \).

This equation indicates that the accumulative reward of a state is determined by the immediate reward plus the expected discount accumulative rewards of the following states. The assumption beneath is that the agent will choose the optimal action in the subsequent states.

Equation (1.3) is called the *Bellman equation* (Russell and Norvig 2003).

There are \( n \) different equations if there are \( n \) different states in the MDP. In fact, the values for the next states \((V(s'))\) are unknown usually. To solve these non-linear equations, an iterative approach is used:

- Start with arbitrary values for each state \( V(s) \);
- Compute the new \( V(s) \) values for all the states by using Bellman equation and plugging the old \( V(s) \) values into the equation’s right side;
- Regard the new $V(s)$s as old $V(s)$s and plug them into the equation's right side again to get the new iteration's $V(s)$s;
- Repeat the iteration until it is converged.

To summarize, the Bellman equation becomes the Bellman update:

$$ V_{i+1}(s) \leftarrow R(s, a) + \gamma \max_a \sum_{s'} T(s, a, s') V_i(s') $$

(1.4)

where $V_i(s)$ is the expected accumulative value at $i$ iteration. We are ensured to reach equilibrium if the update (called Bellman update) is applied infinitely often (Russell and Norvig, 2003). The arbitrary initial value for $V(s)$ can be ignored eventually in the infinite planning horizon because of the discount factor. After convergence, the actions in every state combine to be the optimal policy (or strategy, plan) for the agent to behave in this MDP.

However, the agent must know the transition model to realize the iteration process. In reality, this information most probably is unavailable. The reinforcement learning or Q-learning method is the solution for this issue. Without the transition model, an agent may compute the values of doing action $a$ in state $s$, noted as $Q(s, a)$, instead of $V(s)$. That is, the agent records the accumulative reward information indexed by state-action pair instead of a single state. The relationship between this two is:

$$ V(s) = \max_a Q(s, a) $$

(1.5)
In this situation, the Bellman update (1.4) still holds for Q-function. Plugging (1.5) into (1.4), we get the new update equation with following format:

\[ Q_{t+1}(s, a) \leftarrow R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q_{t}(s', a') \] (1.6)

An iteration process can get rid of the transition model by using statistical method (Watkin 1989). The update becomes as below:

\[ Q_{t+1}(s, a) \leftarrow Q_t(s, a) + \alpha (R(s, a) + \gamma \max_{a'} Q_{t}(s', a') - Q_t(s, a)) \]

or

\[ Q_{t+1}(s, a) \leftarrow (1 - \alpha) Q_t(s, a) + \alpha (R(s, a) + \gamma \max_{a'} Q_{t}(s', a')) \] (1.7)

Within the update process, the transition model is actually working implicitly from statistical viewpoint. By Q-learning, an agent can behave optimally even it doesn't know the transition model. This is a revolution step since it realizes the agent’s autonomous learning only from its experiences.

1.3.3. Reinforcement Learning for Multiple-Agent Systems

When there are multiple agents in a system, things are getting much more complicated. A Markov game (MG), or stochastic game, is used to model a multiple-agent system. It can be defined as a tuple \( < n, S, A, T, R > \), where

- \( n \) is the number of the agents;
• $\mathcal{S}$ is the set of states, $s \in \mathcal{S}$;

• $A$ is the space of joint actions. Each agent $i$ has a discrete action set $A_i$, $a_i^{k_i} \in A_i$ and $k_i = 1, 2, ..., m_i$. $m_i$ is the number of actions in $A_i$. A joint action $\vec{a}$ is defined as $\vec{a} = a_1^{k_1} \times a_2^{k_2} \times ... \times a_n^{k_n}$, that is a joint action is a vector composed by anyone action of every agent.

• The stochastic transition function $T(s'|s, \vec{a})$ is conditioned on joint action $\vec{a}$, i.e. the probability that the environment transitions to state $s'$ when the agents takes joint action $\vec{a}$ at state $s$;

• For agent $i$, a reward function $R_i: \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ specifies the reward $R_i(s, \vec{a})$ received by the agent when joint action $\vec{a}$ is taken at state $s$, which is the feedbacks to each agent when the joint action acts on the environment.

Similarly, the planning horizon may be infinite. Since there are multiple agents, each agent receives its own reward when it takes an action and also impacts the environment. The agents don't know other agents' situation. The environment becomes more unpredictable. If a Markov game contains only one agent, it is reduced to a Markov decision process. On the other hand, it is reduced to a strategic form game if a Markov game contains only one state. Strategic form games are introduced in Chapter 2. Therefore, Markov game is a two dimensional model.

Reinforcement learning in multiple-agent systems is still far to be mature. However, it is important research issue because it can be
extensively applied to abroad areas. Generally, there are two types of learning in multiple-agent systems: independent learning or coupled learning. Learning independently means that the agents learn to behave individually as if it were the only agent in the system. In this case, the learning problem becomes the learning in a Markov decision process. Learning coupled is an approach that takes the interactions among the agents into account.

1.4 The Application Areas of Reinforcement Learning

Although reinforcement learning in single-agent systems is used widely in many areas, the focus of this section is mainly about the application areas of reinforcement learning in multiple-agent systems. Especially, cognitive radio networks in which the reinforcement learning is applicable will be introduced.

1.4.1. Application Areas

Reinforcement learning in multiple agent systems can be applied in abroad areas that the agents are in a dynamically changing environment, unknown dangerous environment or unreachable by human being environment.

For example, it can be applied in wireless network engineering, especially the cognitive radio networks in which the environments are dynamically changeable. It also can be applied in the situation of multiple robots searching victims in a dangerous accident. In a robot team, the agents may have the task to rescue some victims in dangerous
environments, such as a fire or deep water. The agents don’t have knowledge in advance and need to learn to behave in these situations. Also learning can be applied in economics, i.e. simulating the stock markets.

The most interesting area to me is cognitive radio networks.

1.4.2 Cognitive Radio Networks

In wireless networks, the frequency spectrum is governed by a fixed spectrum assignment policy. It is used in a sporadic fashion resulting in a significant amount of the spectrum unutilized. When the licensed users (or primary users) to whom the spectrum is assigned don't use their bands, other unlicensed users (or secondary users) might be able to utilize these bands to improve the utilization.

![Diagram of Spectrum Holes]

Note: Indicates the portion of time the frequency is occupied by licensed users

Figure 1.2. An Example of Spectrum Holes

Figure 1.2. shows a situation of the spectrum holes left by the
licensed users in three different spectrum frequencies. If there are some other users can switch their spectrum frequencies back and forth, the users may utilize the frequency holes whenever they know there is a hole. For example, a user other than a licensed users may access f3 first then f1 and back to f3. In this way, the spectrum resource is not wasted. To improve the utility efficiency, researchers conceive of a dynamic spectrum access network. The proposed network, which is known as cognitive radio network, can provide high bandwidth to unlicensed users by offering opportunistic access to the bands without interfering with the existing licensed users via heterogeneous wireless architectures and dynamic spectrum access techniques. Cognitive radio is the key technique on which the implementation of the dynamic spectrum access network relies. The phrases “cognitive radio network” and “dynamic spectrum access network” are inter-exchangeable thereafter.

A “Cognitive Radio” is a radio that can change its transmitter parameters based on interaction with the environment in which it operates. (FCC 2003) The two main characteristics of cognitive radio are: cognitive capability and re-configurability. The first one indicates the ability that a user can sense unused portions of the spectrum that are at a specific time or location. Re-configurability enables the radio to be programmed dynamically according to the radio environment (Akyildiz et al. 2006).

The main functions of cognitive radio include spectrum sensing, spectrum management, spectrum mobility and spectrum sharing. Among them, spectrum sharing is the function of providing a fair spectrum
scheduling method among users in the networks. (Akyildiz et al. 2006) In fact, resource sharing is a long term existing problem in wireless networks. The solutions for resource allocation may apply in spectrum sharing in cognitive radio networks directly. However, the old methods are for general networks. Because of the spectrum mobility function, users in cognitive radio networks may not stay on one channel regularly. The system environments in cognitive radio networks are dynamically changeable. Therefore, the spectrum sharing problem is more complicated than general resource allocation problem.

Many works have been done in the literature to realize an optimal sharing in cognitive radio networks. Variety categories include game theory, graphical theory, and algebra method, etc... (Akyildiz et al. 2006). In chapter 2, the approaches by using game theory and learning algorithms will be reviewed.

1.5 Motivations and Contributions

1.5.1. Motivations

I was impressed by the sensors when I was watching movie "Tornado". The sensors sent up into the sky in the middle of tornado make up a wireless sensor network. Each sensor sends the sensed information to a base station. I don't know if the sensors can learn new knowledge or not. By any means, that wireless sensor network is also a multiple-agent system from the logical viewpoint. Similarly, cognitive radio networks are multiple-agent systems as well.

If agents are equipped with learning capability in multiple-agent
systems, agents will be more intelligent. Meanwhile, the application area of multiple-agent systems will be broadly expanded. The application area will not be limited to the cases with dangerous environment and human unreachable environment. The agents with learning function can also substitute for human being to work with less cost.

The essential work for artificial intelligence scientists is to design intelligent agent. Learning capability reflects the inherent aspect of an agent's intelligence. Therefore, improving the agent's learning skill is the most important aspect in the research. Learning skill of an agent in a multiple-agent system is also very important. Reinforcement learning is the key for an agent to autonomously improve its behavior and knowledge. If efficient reinforcement learning can be realized, it will become a huge step toward the agent system design and it will benefit various practical applications.

Although many reinforcement learning researches for multiple-agent systems have been conducted (see details in Chapter 2), how to learn efficiently and effectively in an unknown environment is still a challenge in the literature. The agents in a multiple-agent system may be cooperative to achieve a unique goal or they may be competitive that they act against each other. In a general situation, the agents in a multiple-agent system may have their individual goals and they may also have a common task at the same time. The agents should cooperate but also they should compete. This make learning much more complicated. Coordination becomes an issue in the multiple-agent system learning. Some scientists tend to separate the coordination issue from the learning
problem. In my point of view, reinforcement learning problem implicitly contains many other issues, including the coordination issue.

No doubt it is a difficult task to develop an efficient reinforcement learning algorithm in multiple-agent systems. However, we believe there must be inherent disciplines of learning. Once we grasp those disciplines, we can use them in the developing of the learning algorithm.

On the other hand, how to efficiently and effectively share the bands in cognitive radio networks is another very important problem. It is also the goal of cognitive radio networks to improve the efficiency of the bandwidth usage. Secondary users in cognitive radio networks should know the environment very well so that they could take the best decisions to achieve optimal usage of the vacant bands. The environment is very complicated since the primary users can release and request the frequencies at any time. There is considerable learning that the secondary users should do in order to efficiently use the available spectrum holes.

Cognitive radio networks are multiple-agent systems from logical viewpoint. The learning problem in cognitive radio networks right coincides with the learning in multiple-agent systems. It is very meaningful if the efficiency of the frequency utilization problem in cognitive radio networks is solved by the learning algorithm. This is because the final goal of cognitive radio network is to efficiently use the spectrum frequencies and to improve the utilization in the network over time.
1.5.2. Contributions

There are several contributions from this research.

First, a new reinforcement learning algorithm for multiple-agent systems is proposed to work in unknown environment. This feature makes the learning algorithm more practical in the real world. In reality, the agents in a system most probably don’t have any information about their peer agents and have no environmental information in advance. To behave reasonably in this situation, with learning capability in unknown environment is an essential technique to the agents. The learning algorithm also helps lower the cost of the agent design. With the new reinforcement learning algorithm, agents don't need to equip with communication component or any other components for fetching the environment information.

Second, the proposed reinforcement learning algorithm is more efficient than other algorithms. The learning iteration time is much less than other algorithms. The comparison will be shown in Chapter Three. Generally speaking, learning usually is a statistical procedure. It always time consuming. Achieving a good enough result within a shorter time is very important.

Third, the proposed reinforcement learning algorithm is also effective. Effectiveness is another key factor beside efficiency. It guarantees the correct direction. Almost all the researches in this area regard Nash equilibrium as the goal of learning. But few have properly proved their results are Nash equilibriums. The algorithm in this research is based on the definition of Nash equilibrium. This ensures the
effectiveness of the new learning algorithm. Meanwhile, the algorithm works not only for games with mixed Nash equilibrium but also for games with pure Nash equilibrium.

Fourth, the proposed reinforcement learning algorithm attempts to impact the learning problem in multiple-state Markov games. There is almost no known reinforcement learning algorithm that works for multiple-state Markov games in the literature. There are many issues (Buşoniu et al. 2005) in solving reinforcement learning problem in multiple-state multiple-agent Markov games. Many of the obstacles are surmounted in this research.

Last but not least, this reinforcement learning algorithm can be applied in other areas as long as a case can be mapped to a Markov game. In this research, the spectrum resource sharing problem in a modeled cognitive radio network is solved by applying the proposed learning algorithm. The new reinforcement learning excels in the comparison with a no-regret learning algorithm.

1.6. Outline

In the following chapters, the related works are reviewed first. Then, I introduce the new reinforcement learning algorithm developed for Markov games with single state. In the fourth chapter, the extended reinforcement learning method working for Markov games with multiple states is introduced. In Chapter 5, we show the application of the reinforcement learning algorithm in cognitive radio networks. Finally, the conclusion of the dissertation and the future work are described.
CHAPTER II

LITERATURE REVIEW

Reinforcement learning for multiple-agent systems is still a very immature research area. Reinforcement learning becomes much more complicated in a multiple-agent system, especially a system with multiple states. The reason is that each one of the agents must deal with both the environment and the peer agents.

A Markov game is a system model containing multiple agents and multiple states. Most of the known researches are concentrating on the learning problem for single state Markov games. Very few works have proposed to tackle the learning problem in multiple-state Markov games. This is because of the complexity of the problem.

Usually, the learning target is to find Nash equilibrium of a game. Finding a Nash equilibrium has been proved to be a PPAD (Polynomial Parity Arguments on Directed graphs)-complete problem which belongs to NP-total functions (TFNP) (Daskalakis, Goldberg, and Papadimitriou 2006). TFNP contains all search problems in NP for which every instance is guaranteed to have a solution.

In this chapter, the reinforcement learning methods for one-state Markov games are reviewed first. Then, the related works on multiple-state Markov games are introduced. The researches of applying learning on cognitive radio networks will be summarized next. The final section concludes this chapter.
2.1. Reinforcement Learning for Multiple-Agent Systems

If there is only one state in a Markov game, the Markov game is a strategic form game. Learning in strategic form games is studied by both game theory literature and artificial intelligence literature. For multiple-state Markov games, the research on reinforcement learning is still far to be mature. This section will review the research status of reinforcement learning in both single-state and multiple-state Markov games step by step.

2.1.1. Strategic Form Games

Along the state-dimension, a Markov game is reduced to a strategic form game if it contains only one state. A strategic form game is also known as a normal form game. It is defined as a tuple $< m, A, R >$, where

- $m$ is the number of the agents;
- $A$ is the space of joint actions. For any agent $i$, it has a discrete action set $A_i = \{ a_{i}^{k_i} \in A_i \mid k_i = 1, 2, ..., m_i \}$, where $m_i$ is the number of actions in $A_i$. A joint action $\hat{a}$ is defined as $\hat{a} = a_1^{k_1} \times a_2^{k_2} \times ... \times a_m^{k_m}$, that is an action composed by one action from each agent for all agents;
- For agent $i$, a reward function $R_i : S \times A \rightarrow \mathbb{R}$ specifies the reward $R_i(s, \hat{a})$ received by the agent when joint action $\hat{a}$ is taken at state $s$.

In game theory literature, strategic form games are usually expressed by matrices. The matrices specify the rewards of the agents. For example, a three-agent Matching Pennies game is expressed by two
matrices:

\[
\begin{pmatrix}
head_3 & head_1 \\
tail_3 & tail_1
\end{pmatrix}
\begin{pmatrix}
1,1,-1 & -1,-1,-1 \\
-1,1,1 & 1,-1,1
\end{pmatrix}
\]

The first element of the first matrix means that Agent 1 receive Reward 1 when Agent 1 takes action head (make the head face up), Agent 2 takes action head and Agent 3 takes action head. Meanwhile, Agent 2 and Agent 3 receive Reward 1 and -1 respectively. The other elements in the matrices can be deduced in the same way. When the rewards of the agents always get zero by addition in a game, that game is called a zero-sum game. Otherwise, the game is called a general-sum game.

Learning in strategic form games is a research topic in both game theory area and artificial intelligence area. Since learning usually requires the agents play the same game repeatedly, the played game is called a repeated game in game theory literature. There is no essential gap between the researches of the two areas. The main difference is that more accurate results are pursuit in game theory literature while the scientists in AI are more inclined to approximate results. In fact, learning usually is a statistical process. Pursuing accurate results is not a practical target.
2.1.2. Learning Rules

Learning rule is also known as learning target. When agents learn to behave optimally, they should comply with certain rules which guide the learning action. Learning rules are also known as solution concepts in game theory literature. The most general solution concept is Nash equilibrium. Others include minimax rule and no-regret rule (Shoham and Leyton-Brown 2009), etc...

The concept of Nash equilibrium is developed by John Nash in 1951 (Nash 1951). When Nash equilibrium is achieved in a game, no agent in this game can gain any benefit if one unilaterally changes its action. Then, the system is in a steady state (This state is different from the system states defined in Markov games). Since it was born, Nash equilibrium has deeply affected many areas, such as politics, economy, and engineering etc. It can be applied almost every aspects of human activities.

Minimax is another learning rule introduced by Littman (Littman 1994). The main idea is that the agents should always maximize the expected value in the face of the worst-possible joint action choice of the other agents. The worst-possible joint action means that the joint action brings the worst-possible reward. Therefore, the corresponding learning method only guarantees the minimum good solution.

No-regret rule specifies a learning guide that the learned strategy of an agent should yield an expected payoff no less than the payoff brought by some other strategies. It describes a relatively better strategy. However, there is no standard for the strategies which are compared with.
Besides the rules mentioned above, the agents are assumed to be rational and selfish. If an agent always seeks the best response to the strategies whenever the other agents settle on, the agent is called rational. Rationality has become a fundamental property of an agent in the reinforcement learning researches.

2.1.3. Reinforcement Learning for Strategic Form Games in Game Theory

There are several directions for learning in repeated games, i.e. fictitious play, Bayesian learning and no-regret learning. Strictly speaking, fictitious play and Bayesian learning are not reinforcement learning methods. Therefore, only simple introduction for both will be described here.

Fictitious play is a learning method introduced by G. W. Brown in 1951 (Brown 1951). This method is for two agent zero-sum games. When the rewards of the two agents always get zero by addition in a game, that game is called a zero-sum game. Otherwise, the game is called a general-sum game. In fictitious play, each agent presumes that the other agent is playing stationary strategies. The assumptions in a fictitious play are that the agents know their own reward functions and they may observe the actions taken by the other agent, the agents believe that the other agent plays the action strategy given by the empirical distribution of its previous actions. At beginning of the play, each agent guesses the other agent’s strategy and takes corresponding action. Then, in the following each learning round each agent takes the best action which maximizes the reward to respond the guessed strategies of other agents. The guessed
strategies of other agents are from the empirical distribution of the others' behaviors up to the last round.

Some work (Robinson 1951) shows that the fictitious play may lead to Nash equilibrium. However, it is proved (Shapley 1964, Shapley’s Game) that it cannot achieve Nash equilibrium in non-zero-sum games such as Shapley’s game. The shortcomings of fictitious play are obvious. First, the guessed initial beliefs of the other agent’s strategy radically affect the learning process. Since one has to pick a non-zero probability distribution for the belief. Second, the assumption that all the agents play stationary strategies is unrealistic. Overall, fictitious play is an old, simple learning method which may lead to Nash equilibrium in limited games.

Fudenberg et al. extend the fictitious method to Bayesian learning. In Bayesian learning, agents believe that the other agent has not only stationary strategies. Moreover, the agents update their beliefs on the strategy of the other agent by using Bayesian updating that the distribution is conditioned on the past history. Under certain conditions, Bayesian learning method leads to the Nash equilibrium (Fudenberg and Levine 1998).

Both methods mentioned above require the agents to try to model the other agents’ strategy patterns. Another category of learning methods based on regret testing has also been extensively investigated. The works of (Foster and Vohra 1997; Foster and Young 2006; Freund and Schapire 1995; Hart and Mas Colell 2000; Jafari 2003; Robinson 1951) are the examples of researches on no-regret learning. But many of them have
different learning targets instead of Nash equilibrium. General speaking, “regret” means the difference between the rewards brought by one strategy and the rewards that the agent would have received if it had played a reference strategy. If the result is negative, the agent has regret of the absolute value of the differences. As mentioned before, the main problem of no-regret learning method is that the reference strategy is not standardized.

A variety of no-regret learning techniques have been developed. The work worthy of more attention is the radically uncoupled no-regret learning method proposed by Foster and Young (Foster and Young 2006). The words "radically uncoupled" mean that an agent does not condition its strategy on either the actions or the rewards of the other agents. In other words, the agents improve their strategies without knowing any information of other agents. The method tries to search Nash equilibrium by a regret testing process. In the process, the agent keeps trying new actions during intermittently experimental periods. If the average rewards received during a period is significantly larger than the average rewards got in the non-experimental periods, the agent will adjust its strategy to a new one. Eventually, the system will converge to Nash equilibrium. However, similar to fictitious play, the no-regret learning doesn't always converge to a Nash equilibrium in a game, e.g. in a Shapley game (Jafari 2001).

Both fictitious play and Bayesian learning method are not radically uncoupled approaches. In game theory literature, researches usually tend to comply with the Nash equilibrium definition more strictly, yet it’s not
the case in artificial intelligence area.

2.1.4. Reinforcement Learning for Strategic Form Games in Artificial Intelligence

Plentiful researches on learning in single state Markov games have conducted in AI. Scientists in AI usually turn to find approximate results of Nash equilibriums. However, the notable shortcoming in these works is that almost every work claims the result is Nash equilibrium yet without justification.

At the beginning research stage, scientists try to extend the reinforcement learning for one agent system (i.e. MDP) to a multiple-agent system by assuming that each agent ignores other agent’s existence in the system. However, this idea dooms to be failure because it does not reflect the real world. Since an agent can’t determine the optimal strategy only according to its own behaviors.

The first multiple-agent reinforcement learning method is proposed by Littman in 1994 (Littman 1994). The developed learning approach is called minimax-Q learning algorithm which does not pursue Nash equilibrium. The essence is to maximize the reward in the worst case. This algorithm only works for two-agent non-zero games. The value of one agent in a state is computed by using the following equation:

\[ V(s) = \max_{\pi \in \Pi_D(A)} \min_{\omega \in \Omega} \sum_{\omega' \in \Omega} Q(s, a, \omega') \pi(a), \]

and

\[ Q(s, a, \omega) = R(s, a, \omega) + \gamma \sum_{s'} T(s, a, \omega, s') V(s') \] (2.1)
where $\pi$ specifies the agent’s policy (or strategy) which is a probability distribution over its actions, $PD(A)$; $a$ is the other agent’s action, $a \in O$. The assumption is that the agents must know other agent’s actions. Therefore, this is not a radically uncoupled method. Moreover, the author uses a simplified method to substitute the transition function in the equation (2.1) that can not reflect the real transition model (Littman 1994).

The strength of this algorithm is that it makes the agents converge to a strategy that is guaranteed to be “safe”. Although, this is an approach extended directly from one-agent system of reinforcement learning method, it’s the only one which works for multiple-state Markov games.

JALs (Claus and Boutilier 1998) and R-max (Brafman and Tennenholtz 2002) are both model-based reinforcement learning algorithms for repeated games. Both maintain beliefs about the strategies of other agents by applying fictitious play method and Bayesian method respectively. These two algorithms may be simpler in some degree of the computation, but they can be applied in very limited games. The authors of JALs claim to find Nash equilibrium. However, there is no justification. Hu and Wellman (Hu and Wellman 1998) and Littman (Littman 2001) generalize the JALs to general-sum games.

Another direction of learning algorithms is based on gradient ascent. Since Singh et al. (Singh, Kearns, and Mansour 2000) published the first gradient learning method in 2000, several extended versions have been developed, including the WoLF learning algorithm (Bowling and Veloso 2002) and WPL learning algorithm (Abdallah and Lesser 2008).
These algorithms are radically uncoupled. The common of all the methods is that the agents keep updating the strategies relying on the changes of the Q-value until it converges and the system reaches Nash equilibrium. The algorithm by Singh et al. is very limited that only works in the base case of two-person, two-action general-sum games. However, these works are good attempts to solve the problem of reinforcement learning in Markov games with limit environment information. I have drawn my inspiration from their works to develop the new learning algorithm.

“WoLF” stands for “Win or Learn Fast”, which is a principle for varying the learning rate so that the computation will finally converge. WoLF claims that it can be applied for multiple-state, general-sum and multiple-agent stochastic games. Unfortunately, their analysis and their empirical result only show the case of the initial state of a multiple-state game.

“WPL” is the initiate for “Weighted Policy Learner”. The authors of WPL assert the method is radically uncoupled that can be applied for 2-agent-2-action games with minimum knowledge. They also claim that the method works for Shapley’s game while other gradient ascent based algorithms do not.

All the gradient ascent based algorithms have the same foundation about the adjustment of the agent strategies in the learning process. The purpose for an agent is to adjust its strategy at the end of iteration every time so as to increase the expected accumulative payoffs. For example, in a two-agent, two-action game with the reward matrices as below:
\[ R_r = \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix}, \quad R_c = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}. \]

Agent \( r \) has reward matrix \( R_r \) and agent \( c \) has reward matrix \( R_c \). Agent \( r \) uses strategy \( (\alpha, 1 - \alpha) \), i.e. agent \( r \) takes Action 1 with probability \( \alpha \) and takes Action 2 with probability \( 1 - \alpha \). Agent \( c \) uses strategy \( (\beta, 1 - \beta) \). Each reward component contributes to the overall expected payoff of an agent with certain portion. For example, to compute the contribution of reward \( r_{11} \) to the overall expected payoff of agent \( r \), the probability of joint action \( (1, 1) \) (agent \( r \) takes Action 1 and agent \( c \) takes Action 1) is \( \alpha \beta \). Therefore, the contribution will be \( \alpha \beta r_{11} \). Then, the expected payoffs for agent \( r \) and \( c \) are respectively as below:

\[
V_r(\alpha, \beta) = \alpha \beta r_{11} + \alpha (1 - \beta) r_{12} + (1 - \alpha) \beta r_{21} + (1 - \alpha)(1 - \beta) r_{22} \\
= u \alpha \beta + \alpha (r_{12} - r_{21}) + (1 - \beta) (r_{22} - r_{11}) + r_{22} \tag{2.2}
\]

\[
V_c(\alpha, \beta) = \alpha \beta c_{11} + \alpha (1 - \beta) c_{12} + (1 - \alpha) \beta c_{21} + (1 - \alpha)(1 - \beta) c_{22} \\
= v \alpha \beta + \alpha (c_{12} - c_{21}) + (1 - \beta) (c_{22} - c_{11}) + c_{22} \tag{2.3}
\]

where

\[
u = r_{11} - r_{12} - r_{21} + r_{22} \]

\[
v = c_{11} - c_{12} - c_{21} + c_{22} .
\]

The partial derivatives of the expected payoffs with respect to the strategies for both agents are:
\[
\frac{\partial v_r(\alpha, \beta)}{\partial \alpha} = \beta u + (r_{12} - r_{22}) \quad (2.4)
\]

\[
\frac{\partial v_r(\alpha, \beta)}{\partial \beta} = \alpha v + (c_{21} - c_{22}) \quad (2.5)
\]

If \((\alpha_t, \beta_t)\) are the current strategies for the agents, the both agents are using gradient ascent to get the updated new strategies for next iteration as:

\[
\alpha_{t+1} = \alpha_t + \eta \frac{\partial v_r(\alpha, \beta_t)}{\partial \alpha_t} \quad (2.6)
\]

\[
\beta_{t+1} = \beta_t + \eta \frac{\partial v_r(\alpha_t, \beta)}{\partial \beta_t} \quad (2.7)
\]

The fatal defect of gradient ascent method is that the authors try to use the gradients of expected rewards to control the updates of the strategies. In fact, there is no directly linear relationship between an agent’s strategy and its real reward. This can be observed from Equation (2.4) and (2.5): the gradient of the expected reward is actually determined by the other agent’s strategy.

Other works include the approaches developed by Kalai et al. (Kalai and Lehrer 1993), Wang et al. (Wang and Sandholm 2002), Verbeeck et al. (Verbeeck et al. 2002), Nanduri et al. (Nanduri and Das 2009), and the works (Conitzer and Sandholm 2003; Bowling 2004; Powers and Shoham 2004; Greenwald and Hall 2003; Sen et al. 1994; Weinberg and Rosenschein 2004; Suematsu and Hayashi 2002; Schmidhuber 1996).
Most of the learning algorithms assume that the agents have certain degree of knowledge about the environment. They may be able to observe other agents’ payoff functions, strategies or actions; or they may communicate to exchange information. Or, the learning target is other than Nash equilibrium.

2.1.5. Reinforcement Learning for Multiple-State Systems

The only algorithms known that claim they can work for multiple-state Markov games are Minimax-Q learning and WoLF algorithm. However, this algorithm is extended directly from the Q-learning algorithm for single agent systems. Many issues existing in multiple-state Markov games have been left out of the picture. For example, is the minimax-goal of learning is optimal for multiple-agent systems? Instead, currently the most researches of the learning methods make the Nash equilibrium as the learning target. Another crucial problem of minimax-Q learning is that the transition model in a Markov game is essential and cannot be ignored. Another learning method, WoLF also claims that it can work for multiple state games. However, it only shows the result of the first state of an experimental case with multiple states (Bowling and Veloso 2002).

The reinforcement learning in multiple-state Markov games is still a fierce research problem. To tackle this problem, some issues cannot be avoided. These issues will be discussed in Chapter 4.

Researching in cognitive radio networks is almost the hottest topic in computer area. Many researches try to tackle the resource allocation problem in cognitive radio networks. However, the issue is that everyone has his own network model. It's hard to tell which method is better than others.

2.2.1 Spectrum Sharing Methods in Cognitive Radio Networks

Spectrum sharing is one of the fundamental functions in cognitive radio networks. Different methods have been used in solving the spectrum sharing problem.

In (Xin et al. 2010), the authors study a random dynamic spectrum access scheme, where each node randomly selects its operating band based on the detected accessible spectrum bands. Optimization will be an issue in this kind of method without a doubt. Lagrangian duality theory is used to optimize the resource assignment in cognitive radio networks (Ngo and Le-Ngoc 2010). Some methods do consider the balance or fairness in the spectrum sharing, like in (Wang, Ren and Li 2010). Their method not only takes the overall data rate maximizing account but also the greedy feature of individual note. Integer programming problem is another method used in channel allocation in cognitive radio networks (Xie et al. 2010, Hou and Huang 2010). In (Rao et al. 2010), the authors use graph theoretic tool to resolve channel allocation problem.

Artificial intelligent methods and game theory are also utilized to
optimize the performance in cognitive radio networks. Auction is used in (Taleghani and Hamdaoui 2010), bargain is used in (Brahma and Chatterjee 2010) and pricing game is used in (Tan, Sengupta and Subbalakshmi 2010).

However, all these works don’t have a united cognitive radio network model. No one can tell which method is better than others.

2.2.2. Learning Methods for Spectrum Sharing in Cognitive Radio Networks

Learning algorithms have been extensively applied in cognitive radio networks, particularly in solving the resource sharing problem. The paper (Meshkati, Poor and Schwartz 2007) indicates that game theory can be used as a unifying framework for study of radio resource sharing in wireless networks. Some learning algorithms are proposed and applied in resource allocation problem in cognitive radio networks. For example, the work by Haddad et al. (Haddad, Altman and Elayoubi 2010) develops Nash-Stackelberg fuzzy Q-learning algorithm to improve the global network performance of the utilization. A no-regret learning algorithm named regret matching is used to solve the spectrum resource sharing problem (Su and van der Schaar 2010).

However, usually it’s hard to compare these applications of learning algorithms in cognitive radio networks. The model of cognitive radio networks is not unified. There are many different scenarios. To compare different learning algorithms applying in different network environments is meaningless. It is more important that more diversified network models and learning algorithms will greatly improve the progress of the research
in the literature.

In fact, reinforcement learning method has its own advances in applying to solve the spectrum sharing problem in cognitive radio networks. Cognitive radio networks are quite different from the traditional wireless networks. The users equipped with cognitive radio should be able to adjust their working bands according to the primary users' behavior patterns. Without knowing the primary users' behavior patterns, the secondary users can hardly work efficiently and effectively in the networks. The reinforcement learning algorithms, especially the radical uncoupled ones are just right for this kind of situations.

Learning algorithms are also used in solving power control problem in cognitive radio networks. Examples can be found in (Meshkati, Poor and Schwartz 2007, Haddad, Altman and Elayoubi 2010, Su and van der Schaar 2010).

2.3. Summary

From the literature review, we found that reinforcement learning method can help agents learn with only reinforcement information. It is more realistic for the agents to learn without knowing other agents' existence.

Most of the learning methods regard Nash equilibrium as the learning target. However, many researches cannot justify their results are Nash equilibriums.

The research on learning problem in multiple-state Markov games is far from being mature. Most current researches are concentrating on the learning problem in single-state Markov games. The reason is the
complexity of the learning problem.

Learning applied in spectrum sharing in cognitive radio networks is common. But the numerous different learning methods are proposed for different special cognitive radio networks. It is very hard to compare the learning methods to each other.

Hence, we can get our research ideas from the literature review.
CHAPTER III

NEW EFFICIENT REINFORCEMENT LEARNING ALGORITHM FOR SINGLE-STATE MULTIPLE-AGENT SYSTEMS

Because of the complexity of the learning problem, it is still an immature research topic how to efficiently learn to behave optimally based on the reinforcement for the agents in a Markov game. Many algorithms have been proposed in the literature. However, neither of the algorithms can correctly solve all the benchmark games. Moreover, there is no serious solution for multiple state Markov games.

There are two directions to solve the reinforcement learning problem. Since a Markov game may contain multiple agents and multiple states, we may extend the reinforcement learning method in MDPs to that in Markov games. Or we may choose the other way around: solve the learning problem in single-state Markov games, and then extend the method to solve the learning problem in multiple-state Markov games. Our research chooses the second direction.

First, we propose a reinforcement learning algorithm for the agents who know the other agents’ strategy. Then, we improve this algorithm to a reinforcement learning algorithm that the agents do not know other agents’ information. Step by step, a reinforcement learning algorithm for strategic form games is developed. The learning algorithm is based on the definition of Nash equilibrium.

This chapter will review the important issues in the reinforcement
learning research. Our research ideas are given. Then, a radically uncoupled reinforcement learning algorithm for single-state Markov games is proposed. The last part of this chapter will show that this algorithm is more efficient than other algorithms and works effectively in general multiple-agent games with more than 2 agents. This research result has been published in 2011 (Zhang and Kountanis 2011).

3.1. The Problem of Reinforcement Learning in Multiple-Agent Systems

From the description of the learning problem, we know our goal is to make the agents behave optimally through the reinforcement learning. The problem is that the Markov game is a complex model. We may break down the complex learning problem into parts, resolve the sub-problems and eventually reach our goal. Moreover, we must accurately define "behave optimally" so that it will be concrete. The analysis of the reinforcement learning problem and the formal definition of the problem will be presented in this section.

3.1.1. Decomposing the Problem

It is introduced in Chapter 1 that there are two learning methods in Markov games: individual learning and coupled learning. Individual learning totally ignores other agents’ existence and the interactions among the agents. It can not reflect the dynamical situation with the changes brought by multiple agents together in the system. Especially, the transition function and the reward function in a Markov game are determined by joint actions. The deviation could be huge if only
considering one individual agent. Therefore, individual approach is not an option.

On the other hand, coupled method is better since it considers the existences of other agents. However, coupling is very hard.

To make things easier, we may decompose the learning problem. Since there are agent-dimension and state-dimension in a Markov game, we may extend the reinforcement learning method in MDPs to that in Markov games. Or we may choose the other way around.

Extending the reinforcement learning in MDPs to that in Markov games seems to be easy since the reinforcement learning in MDPs is mature. However, it is not. This direction is similar to extending the individual learning method to coupled learning method. It is almost impossible because of the same reason: the transition function and the reward function are determined by the joint actions in the Markov games. The action choosing for an agent is highly depending on the other agents' choices. The learning update equation (1.7) does not work anymore, let alone to find a method to extend the individual learning to coupled learning.

The only direction left is along the state dimension. Extending learning method in one-state games to learning for MGs is more practicable. Once the reinforcement learning in one-state games is mature, it can be extended to multiple state systems. In each state of a Markov game, the game is reduced to a one-state strategic form game. These strategic form games are relatively independent. After we resolve the learning problem in strategic form games, we add the factor of
transition function. Eventually, we can reach the learning goal for Markov games.

From now on, all the Markov games mentioned are one-state games in this chapter.

3.1.2. Learning Target

It is the ultimate goal that the agents behave optimally through learning. To make “behave optimally” concrete, we must make clear the learning target. For rational agents, they always settle a best response to the strategy whenever the other agents settle on. However, how to evaluate optimality is not fully deterministic in the literature. Minimax-Q learning (Littman 1994) seeks maximum value of each state in worst cases. It’s a safe option but far to be optimal. Most other researches regard Nash equilibrium as the learning target. In this research, we make Nash equilibrium as our learning target as well.

Nash equilibrium is a strategy profile when every agent cannot get any benefit (more payoffs) if a single agent unilateral deviates (chooses another strategy). There may be multiple Nash equilibriums in a game (Nash 1951). Sometimes, some of the agents may get more payoffs if they coalitional deviate from one Nash equilibrium to another. Such preferable Nash equilibriums are called Pareto optimal Nash equilibriums or strong Nash equilibriums. Strictly speaking, strong Nash equilibrium seems to be the best choice for us. Unfortunately, strong Nash equilibrium doesn’t always exist (Kockesen) yet there must be at least one Nash equilibrium in a game (Nash 1951). The way to find the strong Nash equilibrium for a
game is also a very tough question. Moreover, that Nash equilibrium leads steady state in the system implies the guaranteed convergence of the system. This makes Nash equilibrium a tangible learning target which has real value. For one agent, it will get maximum reward value if the other agents keep the current strategies. In this sense, finding Nash equilibrium is a good enough option.

3.1.3. Information Needed

How much information on the basis of which the agents can learn effectively? The more information about the system an agent can get the easier for the agent to choose an action or its strategy. However, in most occasions, agents are in a system with very few information. For this very reason, the agents need learning capability. Learning plays a significant role in the agent design. If more information is required, an agent must be equipped with more costly functions. For example, if the agents need to get information through communication, they must have communication equipments. We should take more general situations into account. The research in this dissertation shows that agents in a system may learn effectively even with no information about other agents while the agents interact in the system. The only information the agents need is the rewards of its own actions.

3.1.4. The Definition of Reinforcement Learning Problem

Since a Markov game is decomposed to a series of strategic form games, the original problem of reinforcement learning for Markov games
is also changed.

In this chapter, we will resolve the problem in a one-state strategic form game. According to the analysis of the learning target, we may formally describe the problem for the current stage: how to find Nash equilibrium in a strategic form game with little information about the environment.

3.2. Characteristics of the Agents

The purpose of solving learning problem is to design intelligent agents. Therefore, the characteristics we want to set up for the agents will determine the features of learning.

3.2.1. Rationality

Rationality is one of the basic characteristics of an intelligent agent. If the agents always take the best response to the beliefs about the other agents’ action, we say the agents are rational.

In a random 2-agent-2-action game with the reward matrix as below:

\[
\begin{pmatrix}
1, 5 & 6, 2 \\
4, 3 & 2, 1
\end{pmatrix}
\]

which means that Agent 1 gets reward 1 and Agent 2 gets reward 5 when Agent 1 takes Action 1 and Agent 2 takes Action 1, or \( R_1(1,1) = 1, R_2(1,1) = 5 \). The rest are similar deduced.

In this game, Agent 1 knows that it can earn 6 (the maximum reward) if it takes Action 1 under the condition that Agent 2 takes Action
2. However, Agent 1 knows that Agent 2 is rational that under the situation it must take Action 1 since it can earn 5 which are more than 2. Thus, Agent 1 will get only 1. The second highest reward for Agent 1 is 4 by taking Action 2. Agent 1 believes that Agent 2 will take Action 1 to get Reward 3 which are greater than 1 if it takes Action 2. Therefore, Agent 1 decides to take Action 2 to respond. Similar analysis happens to Agent 2. Finally, the agents will reach the strategy profile $((0,1.0),(1.0,0))$ (This indicates that Agent 1 will take Action 2 with probability 1.0 and won’t take Action 1. Agent 2 will take Action 1 with probability 1.0 and won’t take Action 2.) Then, the system reaches a steady status. Everyone gets the best solution. This is not coincident. The solution is also a Nash equilibrium of this game. Agent rationality and Nash equilibrium are consistency (Osborne 2003).

3.2.2. Greed

Greed is another property of agents. To get optimal result, an agent needs to be greedy. For any selfish individual, greed is one of the build-in attributes.

However, the greed feature must obey the feature of rationality. Otherwise, the system will not reach a balance. Every agent keeps searching the highest reward without compromising. But the interactions among the agents may make the agents get degraded results (refer the last game example). This process will continue forever and make the system not converge. In this situation, the agents cannot learn to behave optimally at all. Therefore, greed and rationality are indispensable.
3.3. Formal Definition of Nash Equilibrium and Its Features

In game theory, Nash equilibrium is the most important solution concept in a game. The strategies of all the agents (or players in game theory) form a strategy profile. A strategy profile is a Nash equilibrium when no agent can get more reward if an agent unilaterally changes its action. Nash equilibrium may not be unique. There is at least one Nash equilibrium in every finite strategic form game (Nash 1951). However, finding a Nash equilibrium is very hard. It has been proved to be a PPAD-complete problem which belongs to NP total functions (TFNP) (Daskalakis, Goldberg, and Papadimitriou 2006). TFNP contains all search problems in NP for which every instance is guaranteed to have a solution. If an agent only knows its own reward function but not that of the rest of the agents, the method to find Nash equilibrium is exhaustive search (Cesa-Bianchi and Lugosi 2006).

A worse situation is that the strategy profile could be very complicated. A probability distribution over an agent’s actions is called the agent’s strategy. A strategy of an agent may be a pure one or a mixed. Pure strategy is a special case of mixed strategy with probability 1.0 on one single action and 0.0 on all the other actions. All the strategies of the agents in the system make up a strategy profile. If all the strategies in a Nash equilibrium strategy profile are pure, the Nash equilibrium is pure. The random game listed before has a pure Nash equilibrium. The three agent Matching Pennies game has a mixed Nash equilibrium which will be shown later.

Let $n_i$ be a strategy of agent $i$, $S_i$ be the strategy space of agent $i$. 
and $S$ be the space of the strategy profile; Let $\pi_{-i}$ denote the strategy profile without the strategy of agent $i$. Nash equilibrium can be formally defined as below:

**Definition 3.1.** Nash equilibrium.

A Nash equilibrium (NE) in a strategic form game $G = \langle n, A, R \rangle$ is defined as a profile $\pi^*$, such that: $R_i(\pi_i^*, \pi_{-i}^*) \geq R_i(\pi_i, \pi_{-i}^*)$, for all $\pi_i \in S_i$, and $\pi_i^* \in S_i$.

If $\pi_i^*$ is a pure strategy for any Agent $i$, $\pi^*$ is a pure Nash equilibrium. There is at least one mixed Nash equilibrium in a finite strategic form game (Nash 1951).

To make the computation clearer, we use a new format to express the reward functions. Assume that a strategic form game $G = \langle n, A, R \rangle$ has $n$ reward matrices. Each one has the format as below:

$$
\begin{align*}
\alpha_{i1} & \quad \bar{a}_{-i1} \quad \cdots \quad \bar{a}_{-iK} \quad \bar{a}_{-i(k+1)} \quad \cdots \quad \bar{a}_{-iM} \\
\alpha_{i2} & \quad R_{i1}^{(1)}(\bar{a}_{-i1}) \quad \cdots \quad R_{i1}^{(1)}(\bar{a}_{-iK}) \quad R_{i1}^{(1)}(\bar{a}_{-i(k+1)}) \quad \cdots \quad R_{i1}^{(1)}(\bar{a}_{-iM}) \\
& \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\
\alpha_{im} & \quad R_{im}^{(1)}(\bar{a}_{-i1}) \quad \cdots \quad R_{im}^{(1)}(\bar{a}_{-iK}) \quad R_{im}^{(1)}(\bar{a}_{-i(k+1)}) \quad \cdots \quad R_{im}^{(1)}(\bar{a}_{-iM}) \\
\end{align*}
$$

(3.1)

The column indices are the different actions of agent $i$. The row indices are the different joint actions from the agents except $i$. Among the items, $\bar{a}_{-i1} = \alpha_{i1} \times \alpha_{21} \times \ldots \times \alpha_{(i-1)1} \times \alpha_{(i+1)1} \times \ldots \times \alpha_{(n-1)1} \times \alpha_{n1}$, means the joint action containing the first action of every agent without action of Agent $i$;
\[ \tilde{d}_{-it} = a_{1i} \times a_{2i} \times \ldots \times a_{(i-1)i} \times a_{(i+1)i} \times \ldots \times a_{(n-1)i} \times a_{ni}, \] means the joint action containing the first action of every agent except Agent \( n \). Agent \( n \) takes action \( m_n \) in this joint action. There is no action of Agent \( i \) in this joint action; 
\[ \tilde{d}_{-i(n+1)} = a_{1i} \times a_{2i} \times \ldots \times a_{(i-1)i} \times a_{(i+1)i} \times \ldots \times a_{(n-1)i} \times a_{ni}, \] the joint action containing the actions from all agent except Agent \( i \). Besides, Agent \( n - 1 \) takes Action 2 and all the others take Action 1; 
\[ \tilde{d}_{-1n} = a_{2m_1} \times a_{2m_2} \times \ldots \times a_{(n-1)m_{n-1}} \times a_{(n+1)m_{n+1}} \times \ldots \times a_{(n-1)m_{n-1}} \times a_{n,m_n}, \] the last joint action containing the last action of all the agents without action from Agent \( i \). \( R_{1n}(a_{-i,j}) \) specifies the reward from the action profile \( (a_{2i},a_{-i,j}) \). Each joint action maps to a reward.

To compute the total expected payoffs, each agent needs to know how much payoffs every action can produce. From Matrix (3.1) we know the contribution to the overall expected payoffs from an action \( j \) of agent \( l \) is given by:

\[ R_{lj} = \text{prob}(a_{lj}) \sum_{i=1}^{M} \text{prob}(\tilde{d}_{-il})R_{il}(\tilde{d}_{-il}), \]

where \( \text{prob}(a_{lj}) \) is the probability of actions \( j \) in Agent \( l \)'s current strategy, and \( \text{prob}(\tilde{d}_{-il}) \) is one of the probability products of the other agents' strategies, or say the probability of the joint action \( l \) except Agent \( l \)'s action. \( l \leq M, M = m_2 \times m_2 \times \ldots \times m_{l-1} \times m_{l+1} \times \ldots \times m_n \). That means the expected payoff brought by an action is the probability of this action in the Agent's (\( l \)) strategy times the summation of the payoffs brought by all the joint actions without the agent's own action. Every payoff produced by a
joint action is the product of all the probabilities of the joint action without the current Agent \( t \) and the corresponding reward value in the reward matrix.

\[
\text{prob}(\tilde{a}_{-it}) = \prod_{h=1}^{t-1} \text{prob}(a_h) \times \prod_{h=t+1}^{n} \text{prob}(a_h)
\]

(3.2)

where \( a_h \) is any action of agent \( h, h \neq t \).

We define \( \sum_{i=1}^{M} \text{prob}(\tilde{a}_{-it})R_{ij}(\tilde{a}_{-it}) \) as \( \bar{R}_{ij} \) hereafter and call it net payoff of action \( j \) of Agent \( i \). Therefore,

\[
\bar{R}_{ij} = \sum_{i=1}^{M} \text{prob}(\tilde{a}_{-it})R_{ij}(\tilde{a}_{-it})
\]

(3.3)

\[
R_{ij} = \text{prob}(a_{ij})\bar{R}_{ij}
\]

Then, sum up the expected payoffs from all the actions to get the overall expected payoffs of Agent \( t \):

\[
R_t = \sum_j R_{ij}
\]

(3.4)

Sometimes, there might be negative rewards in a game. For convenience we only consider positive rewards in the games. If there are negative values, we add a common constant to the reward functions to get all positive numbers. This step won't affect the Nash equilibrium of the game.
Theorem 3.1.

If a common constant is added to the reward functions of a strategic form game, the Nash equilibrium will remain the same.

Proof: Assume profile $\pi^*$ is the Nash equilibrium of a strategic form game, then $R_i(\pi_i^*, \pi_{-i}^*) \geq R_i(\pi_i, \pi_{-i}^*)$ for any agent $i$. If a constant $c$ is added to the reward matrix of agent $i$, we have $R_i^{\text{new}}(\pi_i^*, \pi_{-i}^*) = R_i(\pi_i^*, \pi_{-i}^*) + c$, since the sum of the strategy probabilities is equal to one and the sum of the probability products of the other agents' strategies is equal to one as well.

In a similar way, we can get $R_i^{\text{new}}(\pi_i, \pi_{-i}^*) = R_i(\pi_i, \pi_{-i}^*) + c$. The relation $R_i^{\text{new}}(\pi_i^*, \pi_{-i}) \geq R_i^{\text{new}}(\pi_i, \pi_{-i})$ still holds. The strategy profile $\pi^*$ is still the Nash equilibrium.

Q.E.D.

Theorem 1 indicates that the relative quantities among the reward values instead of the numbers themselves determine the Nash equilibrium.

According to NE's definition, Agent $i$ won't get any benefit if it deviates from its own strategy when all the other agents' strategies are fixed. In other words, no matter how the Agent $i$ switches its strategy quota $\text{prob}(a_{ij})$ for any action $j$, $R_{ij}$ won't get bigger when $R_{ij}$ is fixed at NE. Recall from Theorem 1 that all $R_{ij} \geq 0$, in this case there is only one possibility: the values of all nonzero $R_{ij}$ are equal. If there is a single nonzero $R_{ij}$, it's a pure strategy.
**Theorem 3.2.**

When it’s converged to a mixed Nash equilibrium, the net payoff of each action of an agent is the same and vice versa.

**Proof:** In this context, the expected accumulative payoff of an agent is defined as \( R_t = \sum_i R_{ij} \). That is:

\[
R_t = \sum_i R_{ij} = \sum_i \text{prob}(a_{ij})R_{ij} \\
= \sum_i \text{prob}(a_{ij})(U - (U - R_{ij})) \\
= U - \sum_i \text{prob}(a_{ij})(U - R_{ij})
\]  

(3.5)

All the items \( \text{prob}(a_{ij}) \) in (3.5) are greater than or equal 0. The item \( U \) is a random number which is greater than or equal 0. If \( U < R_{ij} \) is chosen, the value of \( R_t \) will be much degrader because of \( \text{prob}(a_{ij}) \leq 1 \). Thus, it must be \( U \geq R_{ij} \). Therefore, the expected accumulative payoff \( R_t \) is maximum if and only if \( \sum_i \text{prob}(a_{ij})(U - R_{ij}) = 0 \). Thus, \( R_{ij} = U \). This implies the claimed theorem.

Q.E.D.

Theorem 3.2 is the common foundation for computing mixed Nash equilibriums (Nisan et al. 2007, Datta 2003). Stengel gives a proof in Chapter 2 of (Nisan et al. 2007).

We also need to know, when a mixed Nash equilibrium is achieved, the total expected reward of an agent is the same value of its net payoff. This is because the sum of the probabilities of all the actions is 1. Therefore, we call the net payoff at Nash equilibrium as *Nash equilibrium payoff*.

For example, the three agent Matching Pennies game is a tuple
\( G = (3, A, R) \), where \( A = \{ A_1 \times A_2 \times A_3 \} \), with the action sets of agents \( A_1 = \{ \text{head, tail} \}, A_2 = \{ \text{head, tail} \}, A_3 = \{ \text{head, tail} \} \). The reward functions are expressed as below:

Agent 1:

\[
\begin{align*}
\text{head} & : (1 & 1 & -1 & -1) \\
\text{tail} & : (-1 & -1 & 1 & 1)
\end{align*}
\]

Agent 2:

\[
\begin{align*}
\text{head} & : (1 & -1 & 1 & -1) \\
\text{tail} & : (-1 & 1 & -1 & 1)
\end{align*}
\]

Agent 3:

\[
\begin{align*}
\text{head} & : (-1 & -1 & 1 & 1) \\
\text{tail} & : (1 & 1 & -1 & -1)
\end{align*}
\]

There are negative rewards in this game. After the negative rewards are get rid of, the rewards matrices become as below:

Agent 1:

\[
\begin{align*}
\text{head} & : (2 & 2 & 0 & 0) \\
\text{tail} & : (0 & 0 & 2 & 2)
\end{align*}
\]

Agent 2:

\[
\begin{align*}
\text{head} & : (2 & 0 & 2 & 0) \\
\text{tail} & : (0 & 2 & 0 & 2)
\end{align*}
\]
Agent 3: \[
\begin{pmatrix}
\ h_1 h_2 & h_1 t_2 & t_1 h_2 & t_1 t_2 \\
head & (0 & 0 & 2 & 2) \\
tail & (2 & 2 & 0 & 0)
\end{pmatrix}
\]

The original game and the updated game have the same Nash equilibrium: \((0.5, 0.5), (0.5, 0.5), (0.5, 0.5)\).

To compute the Nash equilibrium, assume the Agent 1 has strategy \((x_1, x_2)\), and \(x_1 + x_2 = 1\); Agent 2 has strategy \((y_1, y_2)\), and \(y_1 + y_2 = 1\); Agent 3 has strategy \((z_1, z_2)\), and \(z_1 + z_2 = 1\). Thus, for Agent 1, we have:

\[
y_1 z_1 + y_1 z_2 - y_2 z_1 - y_2 z_2 = -y_1 z_1 - y_1 z_2 + y_2 z_1 + y_2 z_2.
\]

Since \(y_1 + y_2 = 1, z_1 + z_2 = 1\), we get \(y_1 = y_2 = 0.5\). Similarly, we can get \(z_1 = z_2 = 0.5\) and \(x_1 = x_2 = 0.5\). Therefore, the strategy profile \(((0.5, 0.5), (0.5, 0.5), (0.5, 0.5))\) is the mixed Nash equilibrium for the three person Matching Pennies game. This method reveals the fact that the Nash equilibrium strategy for an agent is essentially determined by the other agents’ strategies.

### 3.4. A New Learning Algorithm with No Information of the Environment

According to the definition of Nash equilibrium and its inherent properties, the method for computing Nash equilibrium can be found. Moreover, the computation with the agent knowing very limit about the environment can be realized by utilizing statistical tool.
3.4.1. Analysis

Theorem 3.2. offers us a method to compute Nash equilibrium for strategic form games. However, this method requires the agents know others' rewards and strategies. Like in the three person Matching Pennies game, the second agent must know the first agent's rewards and the third agents' strategy although in this example \( z_1, z_2 \) can be canceled in the computing. As a matter of fact, the computation can become very complicated as the number of agents and the number of actions of the agents grows larger.

All the agents in a strategic form game are greedy. Everyone always seeks the maximum payoff. The agents keep adjusting their strategies to favor whichever action brings a higher payoff. It is a reinforcement learning process. However, this adjustment won’t bring the agents to equilibrium. Intrinsically speaking, the continuous inter-impact of the strategies and rewards of the agents to each other forms a pattern: one's reward \( \Rightarrow \) one's strategy \( \Rightarrow \) others' rewards \( \Rightarrow \) others' strategies \( \Rightarrow \) one's reward.

The adjustment of strategies may be achieved by using an exponential function of the net payoffs \( f(R_{ij}) = \theta^{R_{ij} - \bar{R}} \), where \( \theta > 1 \) (growth factor) and \( \bar{R} \) is the average of the net payoffs of all actions. Then,

\[
prob(a_{ij}) = prob(a_{ij}) \times f(R_{ij}),
\]

or

\[
prob(a_{ij}) = prob(a_{ij}) \times \theta^{R_{ij} - \bar{R}}. \tag{3.6}
\]

This method is inspired by gradient ascend algorithms. However,
their algorithms fail to connect the strategy updating with Nash equilibrium. Also, from the analysis followed we can predict that their strategy updating method will never make the learning converge. But our algorithm is different from gradient ascent method in essence.

Figure 3.1. The Curve of \( f(x) = \theta^x, \theta = 1.5 \)

The value of \( R \) is the average net payoff. It is used to simulate the Nash equilibrium payoff. It reflects the target: the same value of payoff for each action. When a mixed Nash equilibrium is reached, every \( R_{ij} \) equals \( R \) and \( prob(a_{ij}) \) needs no more change. Figure 3.1. shows the curve of the function \( f(x) = \theta^x \), where \( \theta = 1.5 \) (greater than 1). The base \( \theta > 1 \) makes \( \theta \) a growth factor. This feature reflects the greedy characteristic of the agents. If \( R_{ij} < R \), \( prob(a_{ij}) \) is getting smaller. If \( R_{ij} > R \), \( prob(a_{ij}) \) is getting bigger and there is no change for \( prob(a_{ij}) \) if \( R_{ij} = R \).
When the adjustment Equation (3.6) is applied to the learning processes of the agents in a game, strategy oscillations are generated. Figure 3.2 shows a learning process for the three person Matching Pennies game. Three colors show the three agents respectively. The agents adjust their strategies according to their payoffs respectively. Figure 3.3 shows another process with different initial strategy profile of the same game. The strategy profile in figure 3.3 is closer to the Nash equilibrium of the game than the one in figure 3.2. In both figures, the adjustment of strategy makes the payoff change as well. Both strategy and payoff are oscillating in the learning process.

![Utility Values](image1)

![Strategies](image2)

Figure 3.2. Learning Process: Initial Strategies-([0.27, 0.73], [0.69, 0.31], [0.84, 0.16])
Observing the two figures, we notice that the strategies of the agents wave around the middle points of the oscillations which is their Nash equilibrium strategies of this game. In addition, the waves at the beginning of the process are much smaller if the initial strategy profile is much closer to the Nash equilibrium. This is not coincidental. The shape of the waves is determined by the exponential function $f(\pi_{ij})$. Since there is no pure Nash equilibrium strategy, no action is dominating.

Each action may get chances to occupy the largest or smallest portion alternatively in the strategy distribution. For the strategy adjustment, an agent expects that the payoff of each action can reach the Nash equilibrium payoff. However, the interactions among the strategies and payoffs of the agents make the adjusted strategies approach the Nash
equilibrium and then leave away. Thus, the strategies wave around the Nash equilibrium forever.

**Proposition 3.1.**

The strategies will oscillate around the Nash equilibrium if the agents keep switching their strategies to pursue higher payoff values when there is no pure Nash equilibrium and the initial strategy profile of the learning process is not a Nash equilibrium.

**Proof:** In a learning round, if the initial strategy profile is not a Nash equilibrium, the system is not in steady status. Therefore, the agents will keep changing their strategies.

If there is no pure Nash equilibrium in a game, there is no single action \( j \) that dominates other actions. That means no \( \bar{R}_{ij} \) is obviously bigger than others. Therefore, the probability of any action in a strategy won’t be approaching 1.0 when the adjustment of equation (3.6) is applied.

The probability of an action in a strategy is adjusted according to equation (3.6) where \( f(\bar{R}_{ij}) = a^{\bar{R}_{ij} - \bar{R}} \). Since the initial strategy profile is not a Nash equilibrium, the reward of action \( j \), \( \bar{R}_{ij} \) won’t equal \( \bar{R} \). Therefore, the probability of an action is decided by the distance between \( \bar{R}_{ij} \) and \( \bar{R} \). Then, new strategy profile affects each \( \bar{R}_{ij} \) for every agent and makes it oscillate around \( \bar{R} \). On the other hand, the strategies will oscillate around the Nash equilibrium.

Q.E.D.

If there is a pure Nash equilibrium, the oscillations will disappear
after certain period of time because one action will dominate others. If the strategy profile is closer to the mixed Nash equilibrium, the payoffs brought by the actions will be closer to the Nash equilibrium payoffs for every agent. It means that the agents should make smaller changes on their strategies (from Equation (3.6)). Therefore, the oscillation amplitudes will be smaller. Unfortunately, the interactions among the strategies and payoffs of the agents make it impossible after a certain time. Namely, smaller strategy oscillations can only appear at the beginning of a learning process. If the initial strategy profile is the Nash equilibrium, there would be straight lines in the learning process.

**Proposition 3.2.**

The closer to the Nash equilibrium the initial strategy profile is chosen by the agents, the smaller strategy oscillation amplitudes at the beginning of the learning process will be.

**Proof:** The values of all \( R_{i,j} - R \) in equation (3.6) are much smaller if the initial strategy profile chosen by the agents is much closer to the Nash equilibrium of a game. Because the distance to the Nash equilibrium of the probabilities in a mixed strategy is decided by function \( f(R_{i,j}) = \theta^{R_{i,j} - R} \), the adjustments for the strategies of the agents are smaller. Therefore, the strategy oscillation amplitudes are smaller at the beginning of the learning process.

Q.E.D.

According to this, we develop an algorithm for the agents learning
to find a Nash equilibrium with the strategy information of the others. Then, we propose a new radically uncoupled learning algorithm in which the agents don't know anything about the environment.

3.4.2. Finding NE with Knowing Others' Strategies

We assume that the agents in a game can obtain information about the other agents' strategies. Also, we assume the agents are rational and greedy. Strategies can be randomly initialized to the values in $[0, 1]$. An agent keeps the adjustment procedure of its strategy in a learning process. After one round of learning process, an agent re-initializes its strategy with the one closer to NE strategy and start a new round of learning process. The middle point of the first oscillation in the previous learning round is picked as the initial strategy since it is closer to the Nash equilibrium strategy.

We propose Algorithm 3.1. which can find a Nash equilibrium when the agents know the others' strategies in a strategic form game.
Algorithm 3.1.
1. **Initialize** strategy randomly for each agent;
2. **Repeat** (for each agent $i$):
3.  **if** (a learning round is finished)
4.    **if** (a Nash equilibrium strategy has been found)
5.      terminate learning program;
6.  **else**
7.      initialize strategy for the next learning round;
8.      go to step 2;
9.  **endif**
10.  **else**
11.    compute all the probabilities of the joint actions:
12.      $prob(\hat{d}_u)$
13.    compute net payoffs $R_{ij}$ for all the actions:
14.      $R_{ij} = \sum_{i=1}^{M} prob(\hat{d}_{-u})R_{ij}(\hat{d}_{-u})$
15.    adjust the strategy:
16.      $prob(\alpha_{ij}) = prob(\alpha_{ij}) \times \theta^{R_{ij} - \bar{R}}$
17.  **endif**

The flow chart is shown as followed:
There might be multiple learning rounds in a whole learning process. Each learning round, the agents perform three step learning. The main purpose for the three steps is to make the agents update their strategies to close the Nash equilibrium strategies. The third step (shadowed in flow chart), adjust the strategy is the common step in almost all the learning algorithms. However, most learning algorithms fail to connect the adjustment to Nash equilibrium. Our strategy adjustment method has tight relationship to Nash equilibrium.

As an example, the algorithm is implemented in the two agent Matching Pennies game. The game is defined as tuple $G = \langle 2, A, R \rangle$, where $A = \{A_1 \times A_2\}$, with the action sets of agents
$A_1 = \{\text{head, tail}\}, A_2 = \{\text{head, tail}\}$. The reward function $R$ is defined as below:

\[
R_1; h_1 = \begin{pmatrix} h_2 & r_2 \\ -1 & 1 \end{pmatrix} \quad R_2; h_2 = \begin{pmatrix} h_1 & r_1 \\ 1 & -1 \end{pmatrix}
\]

The expression $h_1$ indicates selecting head action of Agent 1 and $r_1$ means selecting tail action of Agent 1. The learning process with non-Nash equilibrium initial strategy profile is shown in figure 3.4.

![Graph of Strategies](image)

a). The First Learning Round
b). The Second Learning Round

c). The Third Learning Round
d). The Fourth Learning Round

e). The Fifth Learning Round
f). The Sixth Learning Round

Figure 3.4. The Learning Process of Two Person Matching Pennies Game

It takes total 6 learning rounds for the agents to reach the Nash equilibrium $((0.5, 0.5), (0.5, 0.5))$. Inside each learning round, the strategies of the two agents are adjusted according to equation (3.6). We may notice the strategies oscillate around a fixed point. Meanwhile, the amplitudes of the oscillations become bigger and bigger along the iteration axis (x axis). There might be multiple ways to initialize the strategies for the next learning round according to the oscillation amplitudes. In this dissertation, we use the middle point of the first oscillation. The initial strategies are getting closer and closer to the Nash equilibrium in a learning round after another. After 6 learning rounds, the oscillation amplitudes in the first 10 iterations are very small. We know they are
close enough to the Nash equilibrium.

If the initialized strategy profile happens to be the Nash equilibrium, the learning process is shown as Figure 3.5. The process is a straight line for both agents.

![Figure 3.5. The Learning Process of Two Person Matching Pennies Game with NE as the Initial Strategy Profile](image)

3.4.3. Finding NE without Knowing Any Information

In a more realistic situation, agents may have no way to obtain any information about the other agents. They might not even know how many agents are in the system. The only thing agents may observe is their own rewards in the whole process. In this case, an agent has no idea that a reward is obtained by which joint action. It only knows its own action. For example, in the three person Matching Pennies game, the first agent can only observe a reward 1 or -1 brought by its first action without knowing
how many different joint actions bring 1 or −1 and which those joint actions are.

Let’s put the reward matrix of Agent 1 as followed:

\[
\begin{array}{cccc}
 h_2 h_3 & h_2 t_3 & t_2 h_3 & t_2 t_3 \\
\text{head} & 1 & 1 & -1 \\
\text{call} & -1 & -1 & 1 \\
\end{array}
\]

Agent 1 getting reward 1 with its first action (head) is decided by 
\(\text{prob(head)} \times \text{prob}(h_2) \times \text{prob}(h_3) + \text{prob(head)} \times \text{prob}(h_2) \times \text{prob}(t_3)\). Therefore, the expected payoff brought by the first action of Agent 1 is:

\[
1 \times (\text{prob(head)} \times \text{prob}(h_2) \times \text{prob}(h_3) + \text{prob(head)} \times \text{prob}(h_2) \times \text{prob}(t_3)) + (-1) \times (\text{prob(head)} \times \text{prob}(t_2) \times \text{prob}(h_3) + \text{prob(head)} \times \text{prob}(t_2) \times \text{prob}(t_2))
\]

However, Agent 1 doesn’t know any information of items \(\text{prob}(h_2)\), \(\text{prob}(h_3)\), \(\text{prob}(t_2)\) or \(\text{prob}(t_3)\). It even doesn’t know how many items should be in the computation. It only can observe its own reward 1 or −1 when it takes action head.

Fortunately, statistical methods can help an agent to estimate the strategy products of the other agents. The agents are rational. They don’t modify their strategies blindly. They may keep trying the same strategies with some delay. During this period of delay, the number of times that the same reward appears is determined by the summation of all the different probability products of joint actions. For the above example, we know
\[
\begin{align*}
\text{prob}(h_2)\text{prob}(h_2) + \text{prob}(h_2)\text{prob}(t_2) + \text{prob}(t_2)\text{prob}(h_3) \\
+ \text{prob}(t_2)\text{prob}(t_2) &= 1
\end{align*}
\]

Therefore, \(\{\text{num}(\text{head}_1,\text{rew} = 1) + \text{num}(\text{head}_1,\text{rew} = -1)\}\) is determined by \(\text{prob}(\text{head})\). Thus,

\[
\begin{align*}
\bar{R}_{\text{head}} &= \{\text{num}(\text{head}_1,\text{rew} = 1) \ast 1 + \text{num}(\text{head}_1,\text{rew} = -1) \ast (-1)\} \\
&/\{\text{num}(\text{head}_1,\text{rew} = 1) + \text{num}(\text{head}_1,\text{rew} = -1)\}
\end{align*}
\]

Hence, in a period of delay time, Agent 1 only needs to observe the number of times that reward 1 appears and the number of times that -1 appears. It can then estimate the value of \(\text{prob}(h_2)\text{prob}(h_2) + \text{prob}(h_2)\text{prob}(t_2)\) without knowing the specific values of \(\text{prob}(h_2), \text{prob}(h_3), \text{prob}(t_2)\) or \(\text{prob}(t_3)\). Thus, the agent can compute all the net payoffs \(\bar{R}_i\).
Algorithm 3.2.
1. **Initialize** strategy randomly for each agent;
2. **Repeat** (for each agent \(i\)):
3. \( \text{if} \) (a learning round is finished)
4. \( \text{if} \) (a Nash equilibrium strategy has been found)
5. terminate learning program;
6. \( \text{else} \)
7. initialize strategy for the next learning round;
8. go to step 2;
9. \( \text{endif} \)
10. \( \text{else} \)
11. keep the same strategy and observes the reward in a period of delay;
12. compute net payoffs \( R_i \) for all the actions;
13. adjust the strategy:
\[
\text{prob}(a_{ij}) = \text{prob}(a_{ij}) \times e^{R_i - R}
\]
14. \( \text{endif} \)

The corresponding flow chart is as below:
The bold statements in Algorithm 3.2. and the red steps in flow chart show the differences from Algorithm 3.1. They show the delay step to get the information for computing $R_{ij}$.

### 3.5. Experimental Results

The rational greedy algorithm (Algorithm 3.2.) has been applied to several games. All the experiments in this dissertation are implemented in MATLAB. The 100 iteration delays are used in all the examples.

**Case 1:** Two person's Matching Pennies game. This example has been listed in last section. Recall the reward function is the matrices as
below:

\[
R_1: \begin{pmatrix} h_1 & t_1 \\ h_2 & t_2 \end{pmatrix} \quad R_2: \begin{pmatrix} h_3 & t_1 \\ h_4 & t_2 \end{pmatrix}
\]

Figure 3.6. shows the learning process. The Nash equilibrium of the game is \((0.5, 0.5), (0.5, 0.5))\). The initial strategy profile of the learning is \(((0.6679, 0.3321), (0.8887, 0.1113))\). It finally converges to the strategy profile \(((0.4944, 0.5056), (0.4971, 0.5029))\). It takes 10 learning rounds, 10 iterations per round and 100 delays per iteration for total of 10,000 iterations. It's much faster compared with WPS and GIGA-WoLF methods with 200,000 iterations for both (Bowling and Veloso 2002, Abdallah and Lesser 2008).

\[a). \text{The First Learning Round} \]
b). The Second Learning Round

c). The Third Learning Round
d). The Fourth Learning Round

e). The Fifth Learning Round
f). The Sixth Learning Round

![Graph of strategies for the sixth learning round.]

---

g). The Seventh Learning Round

![Graph of strategies for the seventh learning round.]
h). The Eighth Learning Round

i). The Ninth Learning Round
j). The Tenth Learning Round

Figure 3.6. Case 1: Learning Process

Figure 3.7. shows the convergence process of the initial strategy profile for each learning round. Each of the two lines presents one of the two actions for an agent. The middle point of the first oscillation of the current round is used to be the initial strategy of the next round.
Figure 3.7. Case 1: Convergence Process - One Action for Each Agent

Case 2: Rock-paper-scissor game. This game contains two agents. When the "zero-sum" factor is removed from the reward functions by adding 1 to the reward matrices, the game is similar to a Shapley's game. Shapley’s games are basically non-zero sum games in form of paper-rock-scissors.

The reward matrices $R_1$ and $R_2$ are the same as below:

$$
\begin{bmatrix}
  r & p & s \\
  0 & -1 & 1 \\
  1 & 0 & -1 \\
-1 & 1 & 0
\end{bmatrix}.
$$

The Nash equilibrium of this game is $((0.33, 0.33, 0.33), (0.33, 0.33, 0.33))$. This is still a symmetric game in which
each action has the same portion in the Nash equilibrium strategy.

Figure 3.8. shows the first round and the last round of the learning process. Figure 3.9. shows the convergence process. The convergence time is 12,000 iterations which are much faster than 1,000,000 iterations required by WPL and WoLF methods for rock-paper-scissor game. However, those methods don’t work for Shapley’s games.

a). The First Learning Round
b). The Twelfth Learning Round

Figure 3.8. Case 2: Learning Process of Round 1 and Round 12

Figure 3.9. Case 2: Convergence Process - One Action for Each Agent
The 10-round learning process eventually converges to strategy profile \((0.3332, 0.3276, 0.3392), (0.3234, 0.3366, 0.3398))

Case 3: Three person's Matching Pennies game. The reward matrices are introduced in the section of Strategic Form Games. We know the Nash equilibrium of this game is: \(((0.5, 0.5), (0.5, 0.5), (0.5, 0.5))\). After 10 learning rounds, the process converges to the strategy profile \(((0.5004, 0.4996), (0.5134, 0.4866), (0.5004, 0.4996))\).

![Graph of Strategies](image)

a). The First Learning Round
b). The Tenth Learning Round

Figure 3.10. Case 3: Learning Processes for Round 1 and Round 10

Figure 3.11. Case 3: Convergence Process - One Action for Each Agent
Figure 3.10. shows the beginning and the ending of a learning process. Figure 3.11. shows the convergence process. The total number of iterations is 60,000. WoLF algorithm requires 100,000 iterations. The comparison of Algorithm 3.2. for three person Matching Pennies game with WoLF algorithm is shown as Figure 3.12.

![Three Person Matching Pennies Game](image)

**Figure 3.12.** Comparing Algorithm 3.2. with WoLF Algorithm

The x axis shows the number of iterations. Only the probability of one action of an agent is shown in the figure.

**Case 4:** An arbitrary two person's game. Unlike the other games, this is not a symmetric game. The reward matrices $R_1$ and $R_2$ are as below:
\begin{equation*}
R_1: \begin{pmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{pmatrix} = \begin{pmatrix} \frac{8}{4} & \frac{5}{11} \end{pmatrix}, \quad R_2: \begin{pmatrix} a_{11} & a_{31} \\ a_{12} & a_{32} \end{pmatrix} = \begin{pmatrix} \frac{3}{4} & 6 \end{pmatrix}.
\end{equation*}

\text{a). The First Learning Round}
b). The Thirteenth Learning Round

Figure 3.13. Case 4: Learning Processes for Round 1 and Round 13

Figure 3.14. Case 4: Convergence Process
There is a mixed Nash equilibrium for this game: 
\(((0.6,0.2),(0.6,0.4))\). Figure 3.13. shows the learning process. Figure 3.14. shows the convergence process. After 13 learning rounds, it converges to the strategy profile 
\(((0.8205,0.1795),(0.5961,0.4039))\). It takes 13,000 iterations.

This game is a general Markov game. It’s neither a zero-sum game nor a symmetric game. The probability of each action is different in Nash equilibrium.

**Case 5:** arbitrary three person's game:

\[
R_1: \begin{array}{cccc}
\alpha_{11} & \alpha_{21} & \alpha_{31} & \alpha_{12} \\
\alpha_{12} & \alpha_{22} & \alpha_{32} & \alpha_{13} \\
\end{array}
= \begin{pmatrix}
1 & 13 & 3 & 19 \\
5 & 2 & 16 & 11 \\
\end{pmatrix},
\]

\[
R_2: \begin{array}{cccc}
\alpha_{11} & \alpha_{21} & \alpha_{31} & \alpha_{12} \\
\alpha_{12} & \alpha_{22} & \alpha_{32} & \alpha_{13} \\
\end{array}
= \begin{pmatrix}
13 & 6 & 12 & 3 \\
7 & 5 & 12 & 4 \\
\end{pmatrix},
\]

\[
R_3: \begin{array}{cccc}
\alpha_{11} & \alpha_{21} & \alpha_{31} & \alpha_{12} \\
\alpha_{12} & \alpha_{22} & \alpha_{32} & \alpha_{13} \\
\end{array}
= \begin{pmatrix}
7 & 15 & 13 & 5 \\
9 & 4 & 8 & 14 \\
\end{pmatrix}
\]

This is a general Markov game with pure Nash equilibriums. The learning algorithm finds a pure Nash equilibrium.
a). The First Learning Round

b). The Sixth Learning Round

Figure 3.15. Case 5: Learning Process for Round 1 and Round 6
There are two pure Nash equilibriums for this game: 

\(((1, 0), (1, 0), (0, 1))\) and \(((0, 1), (1, 0), (1, 0))\). The learning algorithm may converge to either of them. Figure 3.15. shows a learning process. For this learning, the final strategies are shown in the table below.

<table>
<thead>
<tr>
<th></th>
<th>Action 1</th>
<th>Action 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent 1</td>
<td>0.9797</td>
<td>0.0203</td>
</tr>
<tr>
<td>Agent 2</td>
<td>0.9718</td>
<td>0.0282</td>
</tr>
<tr>
<td>Agent 3</td>
<td>0.0328</td>
<td>0.9672</td>
</tr>
</tbody>
</table>

Table 3.1. The Strategies of 3 Agents in Case 5

3.6. Summary

The proposed reinforcement learning algorithm is different from other learning algorithms in several aspects.

First, the new learning algorithm works for Markov games no matter the games have pure Nash equilibrium or mixed Nash equilibrium. The other learning algorithms do not mention if they can find pure Nash equilibrium for a Markov game.

Second, the experimental results show that the proposed learning algorithm runs with much fewer iteration numbers than other algorithms. This means the new learning algorithm is more efficient.

Third, the new learning algorithm can find Nash equilibrium for general Markov games. Most other learning algorithms only work for special games, such as zero-sum games, symmetric games.
CHAPTER IV

REINFORCEMENT LEARNING FOR MULTIPLE-STATE MARKOV GAMES

In a more complicated Markov game, there could be multiple states. Designing the learning for intelligent agents in this situation becomes more difficult. There are very few algorithms have been developed to resolve the reinforcement learning problem in multiple-state Markov games.

We have learnt that each state of a multiple-state Markov game is a strategic form game. Our method is to extend the reinforcement learning algorithm proposed in Chapter 3 to a learning algorithm for multiple-state Markov games (Zhang and Kountanis 2012).

However, the extension is not straight forward. Some issues can’t be avoided before going further. These problems will be reviewed step by step first in this chapter. The new reinforcement learning algorithm for multiple-state Markov games will be present then. The last part is the experimental results for the learning algorithm applied in two Markov games with multiple states.

4.1. Reinforcement Learning Problem in Multiple-State Markov Games

In a multiple-state Markov game, the environment becomes more complicated than the one in a strategic form game. The transition functions must be taken into account in the learning process. Similar to
one state Markov games, we need to clarify what is the learning target.

4.1.1. Learning Target

In a strategic form game, Nash equilibrium is the learning target. Since the stage game is a strategic form game in each state, searching for Nash equilibrium should be also the learning target in Markov games. If a Nash equilibrium is found in every state, the solution for the overall Markov game is also optimal.

4.1.2. The Reinforcement Learning Problem in Multiple-State Multiple-Agent Systems

From the above analysis, we can precisely express our problem: how to learn to find Nash equilibrium for the agents in each of the state in a Markov game with little information about the environment.

Since we need to take transition function into account, we must deduce a way to compute the expected accumulative payoffs for the agents.

4.2. Analysis

4.2.1. Computing Payoffs in Multiple-State Multiple-Agent Systems

From one-state strategic form games to Markov games, the biggest difference is that the transition functions in a Markov game can make the system change from one state to another. The purpose is to find an optimal policy for each agent in the Markov game to get maximum
expected overall payoff. The formal definition of Markov game is in
Chapter 1. Each state of the multiple-state Markov game can be regarded
as a one-state strategic form game. Once the agents have observed their
rewards in a state, the Nash equilibrium can be found by applying the
learning algorithm proposed in Chapter 3. However, since there are
multiple states in the Markov game, transition functions must be taken
into account. Assume \( Q_i(s, \tilde{a}_i) \) records the expected accumulative payoff for
Agent \( i \) with Joint Action \( \tilde{a}_i \) in State \( s \). There are total \( m_1 \times m_2 \times \ldots \times m_n \)
items of \( Q_i(s, \tilde{a}_i) \) in state \( s \). Refer to Matrix (3.1), the reward matrix now
becomes as below:

\[
\begin{pmatrix}
  a_{i1} & \ldots & a_{ik} & \ldots & a_{iM} \\
  q_{i11} & \ldots & q_{i1k} & \ldots & q_{i1M} \\
  q_{i21} & \ldots & q_{i2k} & \ldots & q_{i2M} \\
  \vdots & \ddots & \vdots & \ddots & \vdots \\
  q_{iN1} & \ldots & q_{iNk} & \ldots & q_{iNM}
\end{pmatrix}
\]

Then, based on Bellman equation (1.3), the Q-values for each joint
action can be computed by an equation similar to equation (1.6). Thus, the
update equation is as below:

\[
Q_i(s, \tilde{a}_i) \leftarrow R_i(s, \tilde{a}_i) + \gamma \sum_{s'} T(s, \tilde{a}_i, s') \max_{a'_{i}} Q_i(s', a'_i)
\]  

(4.1)

where \( Q_i(s', a'_i) = \sum_{k=1}^{K} \text{prob}(\tilde{a}_i) Q_i(s', (a_k, \tilde{a}_i)) \). This indicates that the
payoff is the value of current reward plus the discount payoffs of the next
states with the expectation that the agent will rationally choose optimal
actions bringing maximum payoffs in the next states. However, this
method requires the model of transition which may not accessible in most of realistic situation. To get rid of the transition model, the statistical method used for MDP Q-learning can be applied. Thus, we get the update equation:

\[ Q_t(s, a_t) \leftarrow (1 - \alpha)Q_t(s, a_t) + \alpha(R_t(s, a_t) + \gamma \max_{a'_t} Q_t(s', a'_t)) \] (4.2)

The following problem is how to compute \( \max_{a'_t} Q_t(s', a'_t) \). If the agent has a pure strategy when NE is reached in state \( s' \), the corresponding Q-value of that action is the maximum \( Q_t(s', a'_t) \). If the agent has a mixed strategy when NE is reached in \( s' \), the Q-values of the actions are all the same according to the theorem 3.2. Therefore, \( \max_{a'_t} Q_t(s', a'_t) \) is the Nash equilibrium payoff \( \bar{Q}_t \) in state \( s' \). Otherwise, the system is not steady. Since the reinforcement learning algorithm proposed in Chapter 3 solves the problem of finding NE in one state system, the equation (4.2) can be solved.

The strategy update follows the same rule as equation (3.6).

\[ \text{prob}(a_{tj}) = \text{prob}(a_{tj}) \times \theta \bar{Q}_{tj}^{-\lambda} \] (4.3)

4.2.2. Convergence

Convergence is another very important issue which cannot be avoided. If we only consider the final state, the stage game is a pure strategic form game. Since there is no transition any more, the Q-values are converged to almost the original R-values according to equation (4.2).
Moreover, this strategic form game can be solved directly according to chapter 3. Then, back tracking the state right before the final state, it’s still a strategic form game. Once the game of the final state is converge to Nash equilibrium, the Q-value of the state before the final state will also converge. The strategic game can be solved, too. Similarly situation can be derived in all the other states. Eventually, all the Q-values are converged to fixed values in infinite time horizon process. Therefore, the system is guaranteed to be convergent.

4.3. The New Reinforcement Learning Algorithm

Taking the transition model into account, new reinforcement learning is derived based on the algorithm in chapter 3. The algorithm is presented as below:
Algorithm 4.1.

1. **Initialize** strategy randomly for each agent;
2. **Repeat** (for each agent i):
3. \[ \text{if} \ (\text{a learning round for current state is finished}) \]
4. \[ \text{if} \ (\text{a Nash equilibrium strategy has been found}) \]
5. mark it;
6. \[ \text{if} \ (\text{all the states have been processed}) \]
7. terminate learning program;
8. \[ \text{else} \]
9. go to step 2 to process the next state;
10. \[ \text{endif} \]
11. \[ \text{else} \]
12. initialize strategy for the next learning round;
13. go to step 2;
14. \[ \text{endif} \]
15. \[ \text{else} \]
16. keep the same strategy and observes the reward in a period of delay;
17. compute Q values \( Q_i(s, \bar{a}_i) \) for all the joint actions;
18. compute net payoffs \( q_y \) for all the actions;
19. adjust the strategy:
\[
prob(a_y) = prob(a_y) \times \theta q_y - q
\]
20. \[ \text{endif} \]
The flow chart of the algorithm is as followed:

The bold part shows the differences between the algorithm for multiple-state Markov games and the one in Chapter 3. The test for end of loop is different since there are multiple states. The program can stop only if a Nash equilibrium for each state is found. Another difference is that the expected accumulative payoffs are computed by using equation (4.2). Because the accumulative payoffs are different, the computation for strategy adjustment is also changed accordingly.
4.4. The Experimental Results

Two multiple-state Markov games are investigated in this section. In the first Markov game, the agents don’t cooperate. In the second Markov game, the agents must be cooperative. But in both cases, the agents need to coordinate to achieve the targets. The experiments show how the agents coordinate implicitly in the learning processes.

4.4.1. Grid World

This example is a simplified version of the grid world in (Hu 1998). Two agents are escaping from the grid world. The only exit is Cell 6. Figure 4.1 shows the environment.

![Figure 4.1. The Grid World](image)

Agent 1 (black) starts from Cell 1 and Agent 2 (white) starts from Cell 3. Their action sets are the same: \{Up, Down, Left, Right, Stay\}. There are total 25 different joint actions. When the agents happen to go in the same cell, they will be bounced back to the original cells. That means they cannot be together at the same cell except the exit cell. The escaping
process finishes only when they both reach Cell 6. Thus, there are total 11
different states:

\[ s_1: \text{Agent 1 is in Cell 1 and Agent 2 is in Cell 3. It is the start state;} \]
\[ s_2: \text{Agent 1 is in Cell 1 and Agent 2 is in Cell 2;} \]
\[ s_3: \text{Agent 1 is in Cell 1 and Agent 2 is in Cell 4;} \]
\[ s_4: \text{Agent 1 is in Cell 2 and Agent 2 is in Cell 3;} \]
\[ s_5: \text{Agent 1 is in Cell 2 and Agent 2 is in Cell 4;} \]
\[ s_6: \text{Agent 1 is in Cell 3 and Agent 2 is in Cell 4;} \]
\[ s_7: \text{Agent 1 is in Cell 5 and Agent 2 is in Cell 1;} \]
\[ s_8: \text{Agent 1 is in Cell 5 and Agent 2 is in Cell 2;} \]
\[ s_9: \text{Agent 1 is in Cell 5 and Agent 2 is in Cell 3;} \]
\[ s_{10}: \text{Agent 1 is in Cell 5 and Agent 2 is in Cell 4;} \]
\[ s_{11}: \text{Agent 1 is in Cell 6 and Agent 2 is in Cell 6;} \]

Assume the action going up is not that easy, every agent can go up
successfully with 0.5 probabilities. For example, if the joint action is (up, up), the transition function \( T(s_{11}, (up, up), s_{10}) \) is 0.25; if the joint actions is (up, down), the transition function \( T(s_{11}, (up, down), s_{3}) \) is 0.5. All the other
actions of an agent will be successful with 1.0 probability.

The edges of the grid world are walls. If an agent decides to take an
action which makes it hit the wall, it gets rewards -1. If the two agents
take actions make them bounce back, they get -1 reward respectively. If
an agent reaches the cell next to the exit cell and takes the right action, it
gets 100 rewards. All the other situations, the agents just get 0 rewards.
Below are two reward matrices for Agent 1 and Agent 2 in State 1 respectively:

\[
\begin{align*}
&\text{u} \quad u \quad d \quad l \quad r \quad s \\
&\text{d} \quad \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \end{pmatrix} \\
&\text{l} \quad \begin{pmatrix} -1 & -1 & -1 & -1 & -1 \end{pmatrix} \\
&\text{r} \quad \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \end{pmatrix} \\
&\text{s} \quad \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \end{pmatrix}
\end{align*}
\]

The index "u" indicates action going up. Similarly, "d" means going down, "l" means going left, "r" means going right and "s" means action stay. The reward matrices of other states are listed as below:

State 2:

\[
\begin{align*}
&\text{u} \quad u \quad d \quad l \quad r \quad s \\
&\text{d} \quad \begin{pmatrix} -1 & -1 & -1 & -1 & -1 \end{pmatrix} \\
&\text{l} \quad \begin{pmatrix} -1 & -1 & -1 & -1 & -1 \end{pmatrix} \\
&\text{r} \quad \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \end{pmatrix} \\
&\text{s} \quad \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \end{pmatrix}
\end{align*}
\]

State 3:

\[
\begin{align*}
&\text{u} \quad u \quad d \quad l \quad r \quad s \\
&\text{d} \quad \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \end{pmatrix} \\
&\text{l} \quad \begin{pmatrix} -1 & -1 & -1 & -1 & -1 \end{pmatrix} \\
&\text{r} \quad \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \end{pmatrix} \\
&\text{s} \quad \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \end{pmatrix}
\end{align*}
\]

State 4:
State 5:

\[
\begin{pmatrix}
    u & d & l & r & s \\
    100 & 100 & 100 & 100 & 100 \\
    -1 & -1 & -1 & -1 & -1 \\
    0 & 0 & 0 & 0 & 0 \\
    0 & 0 & 0 & 0 & 0 \\
\end{pmatrix}
\]

State 6:

\[
\begin{pmatrix}
    u & d & l & r & s \\
    -1 & -1 & -1 & -1 & -1 \\
    -1 & -1 & -1 & -1 & -1 \\
    0 & 0 & 0 & 0 & 0 \\
    -1 & -1 & -1 & -1 & -1 \\
    0 & 0 & 0 & 0 & 0 \\
\end{pmatrix}
\]

State 7:

\[
\begin{pmatrix}
    u & d & l & r & s \\
    -1 & -1 & -1 & -1 & -1 \\
    -1 & -1 & -1 & -1 & -1 \\
    -1 & -1 & -1 & -1 & -1 \\
    0 & 0 & 0 & 0 & 0 \\
    0 & 0 & 0 & 0 & 0 \\
\end{pmatrix}
\]

State 8:

\[
\begin{pmatrix}
    u & d & l & r & s \\
    -1 & -1 & -1 & -1 & -1 \\
    0 & 0 & -1 & 0 & 0 \\
    -1 & -1 & -1 & -1 & -1 \\
    100 & 100 & 100 & 100 & 100 \\
    0 & 0 & 0 & 0 & 0 \\
\end{pmatrix}
\]
State 9:

\[
\begin{align*}
    u & \begin{pmatrix} -1 & -1 & -1 & -1 & -1 \end{pmatrix} & u & \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \end{pmatrix} \\
    d & \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \end{pmatrix} & d & \begin{pmatrix} -1 & -1 & -1 & -1 & -1 \end{pmatrix} \\
    l & \begin{pmatrix} -1 & -1 & -1 & -1 & -1 \end{pmatrix} & l & \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \end{pmatrix} \\
    r & \begin{pmatrix} -1 & -1 & -1 & -1 & -1 \end{pmatrix} & r & \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \end{pmatrix} \\
    s & \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \end{pmatrix} & s & \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \end{pmatrix}
\end{align*}
\]

State 10:

\[
\begin{align*}
    u & \begin{pmatrix} -1 & -1 & -1 & -1 & -1 \end{pmatrix} & u & \begin{pmatrix} -1 & -1 & -1 & -1 & -1 \end{pmatrix} \\
    d & \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \end{pmatrix} & d & \begin{pmatrix} 100 & 100 & 100 & 100 & 100 \end{pmatrix} \\
    l & \begin{pmatrix} -1 & -1 & -1 & -1 & -1 \end{pmatrix} & l & \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \end{pmatrix} \\
    r & \begin{pmatrix} 100 & 100 & 100 & 100 & 100 \end{pmatrix} & r & \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \end{pmatrix} \\
    s & \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \end{pmatrix} & s & \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \end{pmatrix}
\end{align*}
\]

Table 4.1 shows the result strategies of the agents in each state.

<table>
<thead>
<tr>
<th>State</th>
<th>Agent1</th>
<th>Agent2</th>
</tr>
</thead>
<tbody>
<tr>
<td>State1</td>
<td>(0.45, 0.02, 0.05, 0.07, 0.42)</td>
<td>(0.36, 0.05, 0.28, 0.04, 0.28)</td>
</tr>
<tr>
<td>State2</td>
<td>(0.54, 0.05, 0.00, 0.10, 0.32)</td>
<td>(0.03, 0.01, 0.05, 0.54, 0.37)</td>
</tr>
<tr>
<td>State3</td>
<td>(0.35, 0.06, 0.07, 0.42, 0.10)</td>
<td>(0.03, 0.45, 0.06, 0.04, 0.41)</td>
</tr>
<tr>
<td>State4</td>
<td>(0.03, 0.07, 0.44, 0.04, 0.43)</td>
<td>(0.49, 0.04, 0.04, 0.04, 0.39)</td>
</tr>
<tr>
<td>State5</td>
<td>(1.00, 0.00, 0.00, 0.00, 0.00)</td>
<td>(0.00, 0.00, 1.00, 0.00, 0.00)</td>
</tr>
<tr>
<td>State6</td>
<td>(0.02, 0.02, 0.71, 0.02, 0.23)</td>
<td>(0.01, 0.05, 0.01, 0.01, 0.91)</td>
</tr>
<tr>
<td>State7</td>
<td>(0.00, 0.08, 0.01, 0.02, 0.88)</td>
<td>(0.03, 0.04, 0.02, 0.76, 0.15)</td>
</tr>
<tr>
<td>State8</td>
<td>(0.00, 0.00, 0.00, 1.00, 0.00)</td>
<td>(1.00, 0.00, 0.00, 0.00, 0.00)</td>
</tr>
<tr>
<td>State9</td>
<td>(0.02, 0.02, 0.02, 0.01, 0.93)</td>
<td>(0.10, 0.00, 0.63, 0.07, 0.21)</td>
</tr>
<tr>
<td>State10</td>
<td>(0.00, 0.00, 0.00, 1.00, 0.00)</td>
<td>(0.00, 0.00, 1.00, 0.00, 0.00)</td>
</tr>
</tbody>
</table>

Table 4.1. The Strategies of the Two Agents in Every State
The actions in the strategy profile are Go-up, Go-down, Go-left, Go-right, Stay in order for both Agent 1 and Agent 2. State 5, 8 and 10 are all the last states to the exit cell. Both Agent 1 and Agent 2 take actions with probability 1.0. They will reach the goal cell with no other option. For example, Agent 1 is in Cell 2 and Agent 2 is in Cell 4 in State 5. Agent 1 takes action going right and Agent 2 takes action going left to reach Cell 6 with probability 1.0.

In State 1, Agent 1 is in Cell 1 and Agent 2 is in Cell 2. Agent 1 will most probably take action going up or staying. Agent 2 will most probably take action going up, going left or staying. Agent 1 most probably won't take action going right to avoid a conflict with Agent 2 if Agent 2 goes left. Similar method can be used to explain the strategies for Agent 1 and Agent 2 in other states. One may evaluate the strategies by his intuition.

4.4.2. Push Box

This Markov game describes a system with 3 agents pushing the same box to get to the exit. Different from the Grid world, the agents in this game must coordinate to push the box out of the world. Figure 4.2 shows the environment.
The agents may push the box with some force or with no force at all. If the white agent (the left one) push box with a certain degree of force while the red agent (the bottom one) and the black agent (the right one) don’t push the box, the box will go to Cell 2. If white agent and black agent push the box at the same time, while the red agent doesn’t push, most probably the box will stay at the same cell. Therefore, there are 8 joint actions for total. The joint action {stay, push-up, push-left} stands for white agent doesn’t push; red and black agents push together. Similar to last game, cell 6 is the exit cell.

There are total 6 states in this game. Each state, they are all in the same cell: in $s_1$ all are in Cell 1, in $s_2$ all are in Cell 2, in $s_3$ all are in Cell 3, in $s_4$ all are in Cell 4, in $s_5$ all are in Cell 5, in $s_6$ all are in Cell 6. If there is no movement of the box, each agent gets -10 rewards. If they are in the cells next to the exit cell and they take the right joint action to make the box out, they get 100 rewards. Otherwise, they get 0. According to the rewards, if the agents are intelligent enough, they should be able to coordinate effectively and succeed. The reward matrices of the three agents in State 1 are respectively as below:
\[
\begin{bmatrix}
    s_2s_3 & s_2l_3 & u_2s_3 & u_2l_3 \\
    s_1 & (-10 & -10 & 0 & 0) \\
    r_1 & 0 & -10 & 0 & 0 \\
    u_2 & (0 & 0 & 0 & 0) \\
    s_5 & (0 & 0 & 0 & 0) \\
    r_1u_2 & (0 & 0 & 0 & 0) \\
    l_3 & (-10 & 0 & -10 & 0) \\
\end{bmatrix}
\]

The symbol \( s_1 \) indicates the action stay (no push) for Agent 1, \( r_1 \) means the action pushing right for Agent 1, \( s_2 \) is for Agent 2's action stay, \( u_2 \) means Agent 2's action pushing up, \( s_3 \) is Agent 3's action stay and \( l_3 \) indicates Agent 3's action pushing left.

The reward matrices for other states are listed below:

**State 2:**

\[
\begin{bmatrix}
    s_2s_3 & s_2l_3 & u_2s_3 & u_2l_3 \\
    s_1 & (-10 & 0 & 100 & 0) \\
    r_1 & 0 & -10 & 0 & 0 \\
    u_2 & (100 & 0 & 0 & 100) \\
    s_5 & (0 & 0 & 0 & 0) \\
    r_1u_2 & (0 & 0 & 0 & 0) \\
    l_3 & (-10 & 100 & 0 & 0) \\
\end{bmatrix}
\]

**State 3:**

\[
\begin{bmatrix}
    s_2s_3 & s_2l_3 & u_2s_3 & u_2l_3 \\
    s_1 & (-10 & 0 & 100 & 0) \\
    r_1 & 0 & -10 & 0 & 0 \\
    u_2 & (100 & 0 & 0 & 100) \\
    s_5 & (0 & 0 & 0 & 0) \\
    r_1u_2 & (0 & 0 & 0 & 0) \\
    l_3 & (-10 & 100 & 0 & 0) \\
\end{bmatrix}
\]
\[
\begin{align*}
\mathbf{s}_1 &= \begin{pmatrix} u_2 s_3 & u_2 l_3 & u_2 s_3 & u_2 l_3 \\ r_1 &= \begin{pmatrix} -10 & 0 & 0 & 0 \\
-10 & -10 & 0 & 0 
\end{pmatrix}
\end{align*}
\]

\[
\begin{align*}
\mathbf{s}_2 &= \begin{pmatrix} s_1 s_3 & s_1 l_3 & r_1 s_3 & r_1 l_3 \\ u_2 &= \begin{pmatrix} -10 & 0 & -10 & -10 \\
0 & 0 & -10 & 0 
\end{pmatrix}
\end{align*}
\]

\[
\begin{align*}
\mathbf{u}_3 &= \begin{pmatrix} s_1 s_2 & s_1 u_2 & r_1 s_2 & r_1 u_2 \\ l_3 &= \begin{pmatrix} -10 & 0 & -10 & 0 \\
0 & 0 & -10 & 0 
\end{pmatrix}
\end{align*}
\]

State 4:

\[
\begin{align*}
\mathbf{s}_4 &= \begin{pmatrix} s_2 s_5 & s_2 l_3 & u_2 s_3 & u_2 l_3 \\ r_1 &= \begin{pmatrix} -10 & 100 & -10 & 0 \\
-10 & -10 & -10 & -10 
\end{pmatrix}
\end{align*}
\]

\[
\begin{align*}
\mathbf{u}_2 &= \begin{pmatrix} s_1 s_3 & s_1 l_3 & r_1 s_3 & r_1 l_3 \\ u_2 &= \begin{pmatrix} -10 & 100 & -10 & -10 \\
-10 & 0 & -10 & -10 
\end{pmatrix}
\end{align*}
\]

\[
\begin{align*}
\mathbf{u}_3 &= \begin{pmatrix} s_1 s_2 & s_1 u_2 & r_1 s_2 & r_1 u_2 \\
\mathbf{l}_3 &= \begin{pmatrix} -10 & -10 & -10 & -10 \\
100 & 0 & -10 & -10 
\end{pmatrix}
\end{align*}
\]

State 5:

\[
\begin{align*}
\mathbf{s}_5 &= \begin{pmatrix} s_2 s_5 & s_2 l_3 & u_2 s_3 & u_2 l_3 \\ r_1 &= \begin{pmatrix} -10 & -10 & -10 & -10 \\
100 & -10 & 0 & -10 
\end{pmatrix}
\end{align*}
\]

\[
\begin{align*}
\mathbf{u}_2 &= \begin{pmatrix} u_1 s_3 & u_1 l_3 & r_1 u_3 & r_1 l_3 \\ u_2 &= \begin{pmatrix} -10 & -10 & 100 & -10 \\
-10 & -10 & 0 & -10 
\end{pmatrix}
\end{align*}
\]

\[
\begin{align*}
\mathbf{s}_5 &= \begin{pmatrix} s_1 s_2 & s_1 u_2 & r_1 s_2 & r_1 u_2 \\ l_3 &= \begin{pmatrix} -10 & -10 & 100 & 0 \\
-10 & -10 & -10 & -10 
\end{pmatrix}
\end{align*}
\]

Because the different force directions of the three agents, the state
transition may be uncertain. For example, if the joint action \{stay, push-up, push-left\} is taken in State 1, the box may stay in Cell 1 or may move to Cell 5. This happens because the black agent pushes the box toward the wall and the red agent pushes the box toward Cell 5. The competition result is half chances staying and half chances moving. Therefore, the transition probability of \(s_1\) to \(s_5\) on joint action \{stay, push-up, push-left\} is 0.5.

The other transition probabilities are as below:

\[
\begin{align*}
prob(s_1|s_1, \{stay, push-up, push-left\}) &= 0.5, \\
prob(s_2|s_2, \{stay, push-up, push-left\}) &= 0.5, \\
prob(s_3|s_3, \{stay, push-up, push-left\}) &= 0.5, \\
prob(s_1|s_2, \{push-right, push-up, stay\}) &= 0.5, \\
prob(s_2|s_2, \{push-right, push-up, stay\}) &= 0.5, \\
prob(s_3|s_3, \{push-right, push-up, stay\}) &= 0.5, \\
prob(s_1|s_4, \{push-right, push-up, stay\}) &= 0.5, \\
prob(s_2|s_4, \{push-right, push-up, push-left\}) &= 0.5, \\
prob(s_3|s_4, \{push-right, push-up, push-left\}) &= 0.5, \\
prob(s_4|s_4, \{push-right, push-up, push-left\}) &= 0.5.
\end{align*}
\]

Besides the listed transitions, all the others have probability 1.0. The experimental results that show the strategies of the agents are in table 4.2. The actions in the strategy profile are (Stay, Push-right) for Agent 1 (white agent), (Stay, Push-up) for Agent 2 (red agent) and (Stay,
Push-left) for Agent 3 (black agent).

We can notice the results in all the states are nearly pure Nash equilibriums. The strategy profile in State 1 shows that Agent 1 takes action pushing right, Agent 2 takes action pushing up and Agent 3 takes action pushing left. This combination most probably will make the box move to Cell 5 or stay in Cell 1. In State 2, the box is next to the exit cell. The strategy profile is \( ((0.9684, 0.0316), (0.0371, 0.9629), (0.9593, 0.0407)) \) which means Agent 1 and Agent 3 most probably will stay and Agent 2 will push the box in up direction.

<table>
<thead>
<tr>
<th>State</th>
<th>Agent1</th>
<th>Agent2</th>
<th>Agent3</th>
</tr>
</thead>
<tbody>
<tr>
<td>State1</td>
<td>(0.0926, 0.9074)</td>
<td>(0.0334, 0.9666)</td>
<td>(0.0330, 0.9670)</td>
</tr>
<tr>
<td>State2</td>
<td>(0.9684, 0.0316)</td>
<td>(0.0371, 0.9629)</td>
<td>(0.9593, 0.0407)</td>
</tr>
<tr>
<td>State3</td>
<td>(0.0319, 0.9681)</td>
<td>(0.0367, 0.9633)</td>
<td>(0.0312, 0.9688)</td>
</tr>
<tr>
<td>State4</td>
<td>(0.9564, 0.0436)</td>
<td>(0.9556, 0.0444)</td>
<td>(0.0428, 0.9572)</td>
</tr>
<tr>
<td>State5</td>
<td>(0.0456, 0.9544)</td>
<td>(0.9665, 0.0335)</td>
<td>(0.9555, 0.0445)</td>
</tr>
</tbody>
</table>

Table 4.2. Strategies for Each Agent in the States

### 4.5. Summary

Since very few learning algorithms have tackled the learning problem in multiple-state Markov games, we cannot compare the learning of our algorithm in the two multiple-state Markov games with the learning of other algorithms.

The agents in the first case do not know each other’s existence only when they meet in the same cell. The agents in the second case need to coordinate to fulfill their task. No matter what the scenario is, our learning algorithm can solve the problem. Those scenarios are implicitly
set up in the games.
CHAPTER V

THE APPLICATION OF THE LEARNING ALGORITHM IN COGNITIVE RADIO NETWORKS

Cognitive radio networks can efficiently improve the utility of network bandwidth resources. Sharing spectrum resource is one of the essential functions in cognitive radio networks. In a dynamical environment, the nodes of cognitive radio networks equipped with intelligent learning capability can make the nodes behave optimally. The proposed reinforcement learning algorithm for Markov game can be applied to the resource sharing problem.

In this chapter, the way to map a cognitive radio network to a Markov game is present first. Then, a cognitive radio network model will be introduced. Finally, the experimental results of using reinforcement learning to solve the problem and using no-regret learning to solve the problem will be compared.

5.1. The Spectrum Sharing Problem in Cognitive Radio Networks

The main functions of cognitive radio include spectrum sensing, spectrum management, spectrum mobility and spectrum sharing. Among them, spectrum sharing is the function of providing a fair spectrum scheduling method among users in the networks. (Akyildiz et al. 2006)

How to share spectrum resource efficiently is an essential problem in cognitive radio networks. Because of the complicated dynamical
environments, this problem is not easy to solve. A number of methods have been proposed in cognitive radio network literature. However, the different methods are proposed for different models of cognitive radio networks. To compare the different solutions to different models of cognitive radio networks is almost impossible.

The proposed reinforcement learning algorithm in this study can be applied to solve the spectrum sharing problem. The cognitive radio users may improve their strategies through reinforcement learning, thereby increase resource efficiency of the network. Meanwhile, a no-regret learning algorithm is proposed to solve the same problem for the same network model. The results from the two methods will be compared.

5.2. Mapping Cognitive Radio Networks to Multiple-Agent Systems

In a cognitive radio networks, the users can be regarded as agents in a Markov game. The users may contain secondary users only or both secondary users and primary users. The actions of a user in the cognitive radio network are the ways it chooses the communication channels or its power levels, etc. The purpose of the cognitive radio networks is to improve the efficiency of utility. If every user, especially the secondary users can efficiently utilize the bandwidth resources, the whole network is more efficient. Each user may improve its performance according to the feedbacks or rewards from the environment. Therefore, a cognitive radio network can be mapped to a multiple-agent system. If there are multiple states, a multiple-state Markov game can be mapped. Otherwise, it’s a single-state Markov game, or a strategic form game. This depends on the
complex situation of the cognitive radio network.

5.3. A Cognitive Radio Network Model

There are two types of cognitive radio networks: with primary users or without primary users. In the first type cognitive radio network, the secondary users cannot interfere with the primary users’ activities. No matter in which type cognitive radio network, a secondary user always shares the available bandwidth resource with other secondary users.

In a cognitive radio network with both primary users and secondary users, the existence of primary users can be peeled off. For example, in (Liu et al. 2010), they isolated the primary and secondary users from the original cognitive radio network respectively by employing a service-decomposition approach. The residual resources left by the primary users can be used by the secondary users. Another example is a cognitive radio network based on standard IEEE802.22 (IEEE802.22). The white spaces in the TV frequency spectrum are predictable. It’s able to easily peel off the occupied portion of the bandwidth resources by primary users.

The second type of cognitive radio network is that there only contains secondary users in the network. The main task is how to allocate the bandwidth resources to improve the efficiency of the network.

In this research, a cognitive radio network based on standard IEEE802.22 is considered. Because the behaviors of the primary users in the network are predictable and the time period of the spectrum holes is fixed, we may focus on only the behaviors of the secondary users in the network.
The system is formed by base stations (BSs) and customer-premises equipments (CPEs). Base stations are necessary only for no-regret learning algorithm. Each TV channel has a bandwidth of 6 MHz. The standard specifies spectral efficiencies in the range of 0.5 to 5 bit/(sec/Hz). The data rate in one channel for different CPEs is from 3.0 to 30.0 Mbps. When the licensed users are not using the channels, the CPEs are able to share the spectrum resources. Figure 5.1 shows an example of the architecture.

![Figure 5.1. A Cognitive Radio Network Model](image)

In this study, one cell of the network is investigated. There is one base station and multiple secondary agents, or CPEs called in
5.4. Two Learning Methods Applied in Resource Sharing

Both reinforcement learning and no-regret learning method can be applied in the resource sharing problem in the cognitive radio network (Zhang, Kountanis, and Al-Fuqaha 2012). The application of proposed no-regret learning algorithm in a cognitive radio network model has been published (Zhang and Kountanis 2010).

For the reinforcement learning algorithm, the rewards of the agents are the throughputs of a channel when the agent chooses to transmit that the agents can observe. To make the comparison fair, the reward model in reinforcement learning will be consistent with the parameters setup in the no-regret learning algorithm.

5.4.1. Cooperation and Competition

In a multiple-agent system, agents may completely cooperate to accomplish a common goal or may compete against each other for their individual assignment. Usually, they are both cooperating and competitive at the same time. The users in our cognitive radio network model are both cooperating and competing. Each agent has its own task which is to achieve the highest throughput. Meanwhile, all the agents in the system need to coordinate in order to realize overall communication efficiency and fairness.
5.4.2. Efficiency and Fairness

In a cognitive radio network, spectrum is the limit resource which we plan to improve the utility efficiency. The more bands are allocated, the higher utilization of spectrum is. The users can achieve different capacities in different channels. Therefore, in a strategy allocation, the more weight is on the strategies with bigger capacity in the strategy distribution, the higher utilization of spectrum will be.

There is no unified definition for fairness in the resource allocation problem. Some authors use max-min fairness (Sridharan and Krishnamachari 2009). In our study, fairness is realized when each agent has the same chance (time period) to access the available spectrum bands. The total access time duration over the channels for each CPE is enforced to be equal for the co-exist CPEs at the same channel.

5.4.3. A No-Regret Learning Algorithm for Single Agent

No-regret learning algorithm has been introduced in chapter 1. No-regret learning is a type of online learning methods based on the learning rule: no-regret. Regret measures how much worse an agent using the algorithm performs compared to its best static strategy. No-regret learning algorithms guarantee a non negative average regret, i.e. no-regret. Hedge (Freund and Schapire 1995) describes a no-regret learning algorithm for a single agent system. The algorithm is shown below, where \( \bar{w} \) is the strategy vector which describes the probability distribution over the actions of the agent; \( T \) stands for the number of trials; \( \beta \) is a constant controlling the learning rate; \( \bar{l} \) is the vector which describes the loss
feedback of every action in the strategy vector:

Step1: Initial strategy vector $\overline{w}$, number of trials $T$ and $\beta$

Step2: Do for $t = 1, 2, ..., T$
  - Allocate resource according to the strategy vector $\overline{w}$;
  - Receive loss vector $\overline{l}$ from the environment;
  - Improve the strategy by $\overline{w} = \overline{w}\beta^t$ and $\overline{w} = \overline{w}/\Sigma \overline{w}$;

The agent has $n$ choices, or pure strategies. If the agent decides on a probability distribution over the choices, it gets a mixed strategy. Each strategy choice acts on the environment and receives a loss as a feedback from the environment. The agent is guided by the loss vector to adjust its strategies so that it can improve its behavior. This process continues till a total number of trials are reached.

The rate $\beta$ is a real number in $(0, 1)$. Figure 5.2 shows the curve of the function $f(x) = \beta^x$, where $\beta = 0.5$ (less than 1). In the algorithm, the agents adjust their strategies according to $\overline{w} = \overline{w}\beta^t$. If loss is bigger, the weight of the strategy will be smaller.

![Graph](image)

Figure 5.2. The Curve of $f(x) = \beta^x$, $\beta = 1.5$
Based on Hedge’s idea, we expand the one-agent algorithm to an algorithm for multiple-agent systems. The cognitive radio network we consider essentially is a multiple-agent system. Before the description of the multiple-agent no-regret algorithm, some important issues are described below.

5.4.4. A No-Regret Learning Algorithm for Multiple-Agent Systems

Each CPE sends its own strategy to the BS. The BS gathers all the CPEs’ strategies, the ways they want to share the channels cooperatively. The BS sends back the losses to each CPE according to the fairness principle. The CPEs adapt their strategies according to the efficiency rules and the feedbacks distributedly. They try to give more weight on the channels where they can achieve higher data rate when they plan their strategies. Then, they send their new strategies to the BS again. This process will continue till the convergence is reached or certain trial time has elapsed.

In the algorithm, two parameters, $\alpha$ and $\beta$ are used to adjust the learning rate, where $\alpha \in \mathbb{N}, \beta \in \mathbb{R}$. The smaller $\beta$ is, the faster the convergence will be. The parameter $\alpha$ determines how much the capacity vector will affect learning. The vector $\vec{\mathcal{C}}$ for each CPE specifies the capacities on different channels the CPE can achieve. Vector $\vec{\mathcal{S}}$ is the strategy vector describing the probability distribution of the communication over $M$ channels. The parameter $\vec{\mathcal{L}}$ indicates the waste of spectrum utilization or lack of spectrum band by excessive distribution on each channel. The parameter $\vec{s}$ is a threshold to control the loop. The
following is the detail of the algorithm:

Step1: Let $N$ be the number of CPEs, $M$ be the number of channels and $\vec{c}$ be the capacity vector for each CPE; Initialize each CPE's strategy $\vec{s}$ randomly, initialize the loss $\vec{L}$, and also $\varepsilon, \alpha, \beta$;

Step2: Repeat for each CPE:
- Send the strategy $\vec{s}$ to the base station; Receive the feedback $\vec{L}$ from the base station;
- Update the strategy $\vec{s}$ by $\vec{s} = \vec{s} + \beta \vec{L} / (\alpha \vec{c})$;
- If $\sum(|L|) < \varepsilon$, exit the loop.

With suitable adjustments of $\alpha$ and $\beta$, the learning algorithm for the multiple-agent system will always converge after a certain trial time ellipses.

5.4.5. Experimental Results

In the experiment, assume there are 3 secondary users, or agents and 2 channels in the cell. Therefore, for the 3 agents, each has two actions respectively. In reality, the rewards (the real throughputs) of the agents on the channels are fluctuating. However, for the purpose of convenient comparing, the figures are assumed to be fixed.

In the experiment, assume the transmitting rates of the agents are:
<table>
<thead>
<tr>
<th></th>
<th>Agent1</th>
<th>Agent2</th>
<th>Agent3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel1 (Mbps)</td>
<td>5</td>
<td>18</td>
<td>27</td>
</tr>
<tr>
<td>Channel2 (Mbps)</td>
<td>12</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>

Table 5.1. Transmitting Rates of the Second Users

Assume there is no other interference such that the agents can get complete resources during a 100 second round of learning. Assume each agent may get equal time access to the channel. Then, the reward functions of the agents in the reinforcement learning are as below:

Agent1: $c_1^1 c_2^1 c_1^2 c_2^2 c_1^3 c_2^3$  
$c_1^4$  
$c_2^4$  

Agent2: $c_1^1 c_2^1 c_1^2 c_2^2 c_1^3 c_2^3$  
$c_1^4$  
$c_2^4$  

Agent3: $c_1^1 c_2^1 c_1^2 c_2^2 c_1^3 c_2^3$  
$c_1^4$  
$c_2^4$  

Where $c_1^1$ means that Agent 1 takes the action to communicate on Channel 1. Others can be similarly explained.

The experimental result from reinforcement learning is a pure Nash equilibrium:
<table>
<thead>
<tr>
<th>Agent1</th>
<th>Agent2</th>
<th>Agent3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, 1)</td>
<td>(1, 0)</td>
<td>(1, 0)</td>
</tr>
</tbody>
</table>

Table 5.2. Strategies by Applying Reinforcement Learning Algorithm

However, the result from the no-regret learning method is as below:

<table>
<thead>
<tr>
<th>Agent1</th>
<th>Agent2</th>
<th>Agent3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0.003, 0.997)</td>
<td>(0.497, 0.503)</td>
<td>(1, 0)</td>
</tr>
</tbody>
</table>

Table 5.3. Strategies by Applying No-Regret Learning Algorithm

The strategies of Agent 1 and Agent 2 obtained from no-regret learning method are different from those from reinforcement learning method. Obviously, Agent 1 and Agent 2 will sacrifice more in no-regret learning. This further proves that no-regret learning cannot guarantee to achieve Nash equilibrium in certain circumstances. In fact, the enforced fairness in no-regret learning sacrifices the efficiency of the spectrum usage. Some agents with lower rate share the same time slots with others. More important, the no-regret learning method converges to the steady status depending on the base station. But in reinforcement learning, the allocation is determined by the inherent features of the agents. There is no requirement for a base station.

This experiment can be expanded to more complicated cases with more agents and channels. The reason of choosing a simple example is that it’s easy to compare these two methods. In fact, it’s very easy to verify
that the strategy profile obtained by the reinforcement learning method is the Nash equilibrium. This is based on the definition of Nash equilibrium directly.

5.5. Summary

There are many learning algorithms have been proposed to solve the spectrum sharing problem for special cognitive radio networks. Since the models of cognitive radio networks are not unified, it is very hard to compare these learning algorithms.

We proposed a no-regret learning algorithm for multiple-agent system. Both the no-regret learning algorithm and our reinforcement learning algorithm are applied to solve the same spectrum sharing problem in a cognitive radio network model. The result shows that the result of reinforcement learning is better than that of no-regret learning.

However, we only show one method to map the spectrum sharing problem to a Markov game. Different mapping methods may conduct different results. How to find a best mapping method in each case is an important issue for applying the learning algorithm.
CHAPTER VI

CONCLUSIONS AND FUTURE WORK

Reinforcement learning in Markov games is a very important research topic in Artificial Intelligence area. It can be broadly applied in many areas. But the learning problem is a hard problem. Many algorithms have been proposed in the literature. This research develops an innovative radically uncoupled reinforcement learning algorithm for Markov games. The algorithm is more efficient and effective than others. Moreover, the algorithm can also solve the learning problem in multiple-state Markov games. The application of the algorithm in cognitive radio network shows its applicability. The algorithm can be applied in the systems as long as the systems can be mapped to Markov games.

Although the learning algorithm has been successfully implemented in many Markov games, some issues are worth for further investigation.

6.1. The Innovations of the Reinforcement Learning Algorithm for Multiple-Agent Systems

This dissertation presents a few new reinforcement learning algorithms for Markov games. The learning algorithms are innovative in several aspects.

First, the reinforcement learning algorithm is based on the definition of Nash equilibrium that guarantees to converge to a Nash
equilibrium. The new algorithm not only finds mixed Nash equilibrium but also pure Nash equilibrium if the game has one. Nash equilibrium has certain properties. Many other algorithms claim that their learning target is Nash equilibrium but without justification. Those algorithms may find Nash equilibrium in some benchmark Markov games. However, that doesn't mean their algorithms work for any Markov games. The fatal defect of these algorithms is that their methods fail to build the connection between their learning process and Nash equilibrium.

Usually learning is a statistical process. There is no exception in the new proposed reinforcement learning algorithm. The new algorithm utilizes the inherent discipline of the interaction among the agents in a system. Meanwhile, it exploits the characteristics of the agents: rationality and greed. Through applying the statistical method, it realizes learning with the agents knowing no information about the environment. It is a radically uncoupled reinforcement learning algorithm. This is a major breakthrough in the reinforcement learning research on Markov games. Other algorithms also apply statistical method. Many other algorithms either assume the agents know each other's actions or the agents may communicate to get the information.

Moreover, the algorithm works not only for single-state Markov games as usual reinforcement learning algorithms, but also works for multiple-state Markov games which other reinforcement learning algorithms have never achieved. The experimental results for both single-state and multiple-state Markov games show that the algorithm works efficiently and effectively. In this research, several issues in reinforcement
learning for multiple-state Markov games have been listed and explained. This is a path breaking attempt. It's a hope that this remarks may draw forth by other researches.

The new algorithm may be applied in a Markov game in which the agents may have different number of actions. Although there are few such cases, it's still a new attempt.

The new reinforcement learning algorithm may apply to actual problems as long as the real problems can be mapped to Markov games. The application of the reinforcement learning algorithm in the cognitive radio network model shows the practicability of the reinforcement learning algorithm. Comparing with no-regret learning algorithm, the reinforcement learning finds best responds for the secondary users.

6.2. Further Research Topics

There is still some problems worth to be investigated to more depth to improve the learning algorithm.

First is the scalability of the reinforcement learning algorithm. If there is pure Nash equilibrium for a Markov game, the algorithm will converge to NE very quickly no matter how many agents are there. The reason is that the adjustment of the strategies of the agents has unique direction. Based on the strategy update equation (3.6), the equilibrium will be reached without oscillation. The computation is not complex even if there are a large number of agents. However, if there is only mixed Nash equilibrium in a Markov game, learning becomes very hard. Because of the complexity of the learning problem, the computation will
be increased geometrically with more agents. The maximum number of agents in a Markov game that the reinforcement learning algorithm can achieve mixed Nash equilibrium is unknown yet. We have tested some random Markov games with more than 6 agents.

Second, the learning algorithm may be improved to be more accurate. There are several learning parameters in the algorithm, such as learning round length, the growth factor of the updating method and the learning delay. Because every Markov game has its own features, the learning round length (the number of iterations) for different Markov games should be different. But how to determine the learning round length for a Markov game is a problem. The learning round is also determined by the theta’s value (the growth factor) in the strategy adjustment step. The relationship between the theta’s value and the learning round length should be revealed. If the round length for a Markov game can be determined explicitly, the algorithm will be improved to be more efficient. Moreover, no doubt the more learning delay will improve the accuracy of the learning result.

Besides the two most important questions mentioned above, other issues such as the mapping method also needs to be studied. Mapping method affects the result when the learning algorithm is applied in solving a problem. The most challenging issues about reinforcement learning are summarized in (Vlassis 2007).
REFERENCES


FCC. 2003. ET Docket No 03-222 Notices of proposed rulemaking and order.


Foster, D. P., and Young, H. P. 2006. Regret testing: Learning to play Nash equilibrium without knowing you have an opponent. Theoretical Economics 1: pp. 341–367


Kockesen, L. *Lecture Notes on Game Theory*. [Online]
http://www.wepapers.com/Papers/7634/Kockesen:_Lecture_notes_on_game_theory


Sen, S.; Sekaran, M.; and Hale, J. Lerning to coordinate without sharing information. in Proceedings 12th National Conference on Artificial Intelligence (AAAI-94), Seattle, US. pp. 426-431.


Zhang, J. and Kountanis, D. 2009. A New Interpolation Scheme In Grid-Based Approximation For POMDPs. *International Conference on Computational Intelligence and software Engineering (CISE) 2009*. Wuhan, China. pp.1-4 (Digital Object Identifier 10.1109/CISE.2009.5362704)


