Learning to Cooperate

by

Shenghuo Zhu

Submitted in Partial Fulfillment

of the

Requirements for the Degree

Doctor of Philosophy

Supervised by

Professor Dana H. Ballard

Department of Computer Science
The College
Arts and Sciences

University of Rochester
Rochester, New York

2003

Curriculum Vitae

Shenghuo Zhu was born August 17, 1974 in Wenzhou, China. He graduated with a Bachelor of Engineering degree at Zhejiang University in 1994, and with a Master of Engineering degree at Tsinghua University in 1997. He entered the University of Rochester in 1997, from which he received a Master of Science degree in Computer Science in 1998. His current work in machine learning has been pursued since 1999 with professor Dana Ballard.

Acknowledgments

I would like to express my gratitude to all those who gave me the possibility to complete this thesis. I want to thank the University of Rochester Computer Science department for giving me permission to commence this thesis in the first instance, to do the necessary research work and to use departmental resources. This thesis is the result of several years of work whereby I have been accompanied and supported by many people. It is a pleasant aspect that I have now the opportunity to express my gratitude for all of them.

The first person I would like to thank is my adviser, Dana Ballard, for stimulating suggestions and encouragement helped me in all the time of research for, giving me the freedom to pursue my research goals, and writing of this thesis. I would also like to thank the other members of my PhD committee who monitored my work and took effort in reading and providing me with valuable comments on earlier versions of this thesis: Mitsunori Ogihara, Chris Brown and Mark Fey. I thank you all.

The faculty, students and staff all gave me the feeling of being at home at work. I want to thank them for all their help, support, interest and valuable hints. Especially I am obliged to Tao Li, Chen Yu, Qi Li for collaboration in several projects. Also I would like to thank Deqing Chen, Chunqiang Tang and Xipeng Shen for discussing research topics and playing squash.

I feel a deep sense of gratitude for my parents always gave me boundless love and support to follow the dream I choose. I am grateful for my sister, Chunshi,

who encouraged me during the years. Especially, I would like to give my special thanks to my wife, Sang, whose patient love enabled me to complete this work.

This thesis is based upon work supported by NIH/PHS Grant 5-P41-RR09283 and NSF Grant EIA-0080124.

Abstract

Game theory is not only useful to understand the performance of human and autonomous game players, but it is also widely employed to solve resource allocation problems in distributed decision making systems. These distributed systems are mostly referred to as multi-agent systems. Reinforcement learning is a promising technique for learning agents to adapt their own strategies in such systems. Most existing reinforcement learning algorithms are designed from a single-agent's perspective and for simplicity assume the environment is stationary, *i.e.*, the distribution of the utility of each state-action pair does not change. The predominant approaches to game playing in those settings assume that opponents' behaviors are stationary. However, in a more realistic model of multi-agent systems, the agents are continually adapting their own strategies owing to different utilities at different times. Because of non-stationarity, multi-agent systems are more sensitive to the trade-off between exploitation, which uses the best strategy so far, and exploration, which tries to find better strategies. Exploration is especially important in changing circumstances.

Cooperation usually enables agents to receive a higher payoff than non-cooperative ones. This research is to explore the cooperative opportunities in unknown games. A hill-climbing exploration approach is proposed for agents to take their opponents' responses into consideration, and maximize the payoffs by gradually adapting their strategy to their opponents' behaviors in iterated games. Simulations show that the agents can efficiently learn to cooperate with or compete against

each other as the situation demands. Also, the agents are able to tolerate noise in environments and exploit weak opponents.

Assuming that the utility of each state-action pair is a stochastic process allows us to describe the trade-off dilemma as a *Brownian bandit problem* to formalize recency-based exploration bonus in non-stationary environments. To demonstrate the performance of exploration bonus, we build agents using Q-learning algorithm with a smoothed best response dynamics. The simulations show that the agents can efficiently adapt to changes in their opponents' behaviors whereas the same algorithm, using Boltzmann exploration, can not adapt. This work focuses on typical simultaneous games that represent phenomena of competition or cooperation in multi-agent environments, such as work-and-shirk game.

Table of Contents

List of Tables

List of Figures

1 Introduction

The enormous complexity of software environments has spawned interest in autonomous intelligent agents that can navigate through them. Since such agents learn and encounter other agents, there is a great demand for learning algorithms that work in multi-agent environments. This dissertation is about exploring some of theories of building distributed cooperative learning agents in multi-agent environments.

1.1 Multiple Agents v.s. Games

An agent is an entity that exerts some actions in a certain environment. An adaptive/learning agent is an agent able to learn strategies from its environment to improve its own reward. In most cases, the environment of an adaptive agent contains some other adaptive agents. Such an environment is referred to as a multiple adaptive agent environment, in short, a *multi-agent environment*.

Generally, learning problems in multi-agent environments can be modeled as simultaneous games, where players execute actions simultaneously, and the payoffs depend on the combinations of actions taken by the players. *Rock-paper-scissors* is, for example, such a simultaneous game. Another kind of games is alternative

games, like chess. In the rest of this dissertation, we refer to simultaneous games as games unless specified. This research focuses on two-player iterated games, *i.e.*, two autonomous agents simultaneously learn to improve their own strategies in order to achieve better individual rewards in games. These games can be used to study the central issues in multi-agent systems, such as competition and cooperation. Chapter 2 gives a rudimentary introduction to game theory.

1.2 Reinforcement Learning

The AI research community has increased the interest in the learning capability of systems. An adaptive system is generally more flexible and robust than a preprogrammed one. Supervised learning, where a *teacher* provides exact answers of training examples, is a useful technique in solving problems involving pattern recognition and classification. However, exact answers may not exist in some cases. Reinforcement learning is a technique only use suggestive answers, known as *rewards*, to improve the performance. Therefore, reinforcement learning is widely used in control systems, in which the mathematical models are not well defined.

Markov decision processes (MDPs) originate in the study of stochastic optimal control (Bellman, 1957) and have been widely used as a popular method for modeling and solving theoretic decision/planning problems ever since (Howard, 1960; Puterman, 1994). It is based on the Markovian assumption: the rewards and the dynamics only depend on the current state. *Reinforcement Learning* (RL) (Barto et al., 1983; Sutton, 1984; Watkins, 1989) is a formal framework for learning MDP agents' knowledge is expected optimal policy, how to choose actions to maximize reward. Chapter 2 gives a preliminary introduction to reinforcement learning.

1.3 Non-stationary and Adaptive Adversaries

Unlike systems only containing single adaptive agent, multi-agent systems are non-stationary or, even, inconsistent from the perspective of an individual agent, because learning opponents may continually adapt their individual strategies. Therefore, to explore the responses of learning opponents, it is necessary for agents to be able to keep changing their estimates of payoffs.

In some games, such as matching pennies, no pure equilibrium exists, therefore mixed strategies should be included in the set of support of a general-purposed learning agent, with the result that learning agents are nondeterministic. One interpretation of mixed strategies is to probabilistically choose actions from pure strategies. Unaware of the payoff matrix to the opponent, a learning agent can not compute the exact Nash equilibrium. A fictitious player (Fudenberg & Levine, 1998) assumes that the opponent is consistent with its mixed strategy. However, it is well known that the mixed strategies do not converge if two fictitious players play against each other in some games, such as matching pennies.

In games like prisoners' dilemma, the unique equilibrium is for both players to play d (*defect*), resulting in less payoffs $(1, 1)$ than $(3, 3)$ if both players play c (*cooperate*). If a player believes that its opponent does not change its strategy, then it will always play d. To overcome this dilemma, we also have to take the opponents' responses into consideration. To predict opponents' actions in prisoners' dilemma, Carmel and Markovitch (1996) proposed a method to model opponents' strategies as finite automata. Players are able to predict the deterministic strategy of their opponents. However, if the adversaries are also learning agents, they are unlikely to be deterministic. Moreover, deterministic players may play poorly in noisy environments, where agents may probabilistically realize an action different from the one they intend to. Noisy environments are very common in most real life problems. Therefore, we propose an approach for learning

agents to analyze their opponents' responses via statistical data without modeling entire opponents' strategies. Considering the adversaries' responses, agents can maximize the expected payoff with a better estimation.

1.4 Summary of Contributions

In this dissertation, I propose an approach of cooperative exploration for solving the cooperative problems in multiagent environments. Learning agents adapt their strategies to their opponents' strategies through statistical analysis of their opponents' responses in iterated games. Also, agents explore possible cooperative strategy and verify whether the opponents' response improve the payoff. The simulations show that learning agents can efficiently learn to compete against, cooperate with each other in respective settings and adapt to changes in their opponents' strategies. Also, learning agents are able to tolerate noise in environments and exploit weak opponents.

Another study focused on the strategy of playing against another intelligent agent in a non-stationary environment. The study formalizes recency-based exploration bonus with the Brownian bandit problem. To illustrate the efficiency of the recency-based exploration, we build Q-learning agents with the smoothed best response dynamics. The simulations demonstrate that systems with the recency-based exploration can reach expected equilibria more efficiently than those with Boltzmann exploration. The simulations also show that the learning agents can efficiently adapt to the new context owing to changes of their peers' behaviors or drift of their environment.

Last, I present my studies on some applications related to learning in multiagent environments, a neuroscience research on playing with monkey, learning machine challenge, and multiagent clustering algorithm.

1.5 Organization

Chapter 2 introduces some preliminaries used in this dissertation, including basic knowledge of game theory and reinforcement learning. Chapter 3 presents an algorithm of cooperative exploration for solving the cooperative problems in multiagent environments. Chapter 4 formulates the exploration strategy in non-stationary environments. Chapter 5 describes the player that participated Learning Machine Challenge. Chapter 6 presents the game theory point view of clustering and some other distributed learning problems. Finally, Chapter 7 summarizes the contributions of this dissertation to the theory and practice of building learning algorithm for agents in a distributed environment.

2 Preliminary

This chapter introduces the background knowledge of game theory, and the basic knowledge of Markov decision processes and reinforcement learning.

2.1 Game Theory

Game theory (von Neumann & Morgenstern, 1947) is designed for reasoning about simple multi-player systems. In a two player game, each player takes an action (in a stage), then receives a payoff that depends on its action and the opponent's. A simple example is the rock-paper-scissors game, the payoff matrix of which is as Figure 2.1.

The formal description is as follows:

Definition 1 *An n-player game, $\mathcal{G}$, contains a set of n players. The i-th player can choose strategies from a set of pure strategies, denoted by $\mathcal{A}_i$. A pure profile is an n-tuple of pure strategies of players, which is an item of $\prod_{j=1}^{n} \mathcal{A}_j$. Also for the i-th player, there is a payoff function, $r_i : \prod_{j=1}^{n} \mathcal{A}_j \rightarrow R$, which maps a pure profile into a real number.*

The rock-paper-scissors game is a two-player 3-strategy game. $\mathcal{A}_i = \{\text{rock}, \text{paper}, \text{scissors}\}$ for each i. $\mathcal{A}_i$ could also be a real number in some games, such as price-war.

Alice \ Bob	Rock	Paper	Scissors
Rock	0 0	1 -1	-1 1
Paper	-1 1	0 0	1 -1
Scissors	1 -1	-1 1	0 0

Figure 2.1: The payoff matrix of the rock-paper-scissors game. Each of the players, Alice and Bob, takes one of the actions, "rock", "paper" or "scissors". The contents of the matrix are the payoffs to both players for the different joint profiles. For example, in row "rock" and column "paper", the outcome is $(-1, 1)$, which means that, when Alice plays "rock" and Bob plays "paper", Alice loses the game (receives a payoff of -1) and Bob wins the game (receives a payoff of 1).

In the rock-paper-scissors game, the rational strategy is to choose actions randomly. This is the idea behind mixed strategies.

Definition 2 *A mixed strategy for the i-th player is a probability distribution on the set of the pure strategies, $\mathcal{A}_i$.*

To study the mixed strategy of a game, $\mathcal{G}$, we can form another game, $\mathcal{G}'$, whose strategy sets, $\mathcal{A}'_i$, are the set of probability distribution on $\mathcal{A}_i$. The payoff function of $\mathcal{G}'$ takes the expectation of the payoff function of $\mathcal{G}$ over the distributions.

2.1.1 Nash Equilibrium

An equilibrium, named after Nash (1951), is widely studied.

Definition 3 *A Nash equilibrium is such a (mixed) profile, where any player who changes her (mixed) strategy would be rewarded less. In the other word, a Nash*

equilibrium is a profile such that each player's strategy maximizes her payoff if the strategies of the others are fixed. It is formalized as Equation 2.1.

$$r_i(\mathsf{s}) = \max_{all\ t_i\text{'}s} r_i(\mathsf{s}; t_i) \tag{2.1}$$

where s *is an equilibrium,* t_i *is a (mixed) strategy of the* i*-th player, and* $\mathsf{s}; t_i$ *denotes the same profile as* s *except the* i*-th player plays* t_i*.*

No players leave a Nash equilibrium once they have reached it. However, Nash equilibria may or may not exist in some games, and some games have more than one Nash equilibrium. It can be proven that at least one mixed Nash equilibrium exists in a game (Owen, 1995).

2.1.2 The Prisoner's Dilemma

Besides Nash equilibrium, we are also interested in another type of profiles, where every player may receive a higher payoff than that at a Nash equilibrium.

Definition 4 *A profile* p *is more Pareto efficient than profile* q*, if and only if* $r_i(p) > r_i(q)$ *for each* i*.*

The game of the *Prisoner's Dilemma*, invented by Merrill Flood and Melvin Dresher in the 1950s, two individuals are arrested for committing a crime together. Each of them has two options, confessing and testifying against another or not. If both of them confess and testify against the other, they would be put in jail for 4 years; if only one of them confesses and testifies against the other but the other does not, the former would be released while the latter would get 5 years in jail; if neither of them confesses and testifies against another, both of them would receive 2 years in jail. The payoff matrix of *Prisoner's Dilemma* is shown as Figure 2.2. Each number represents as 5 minus the number of years in jail.

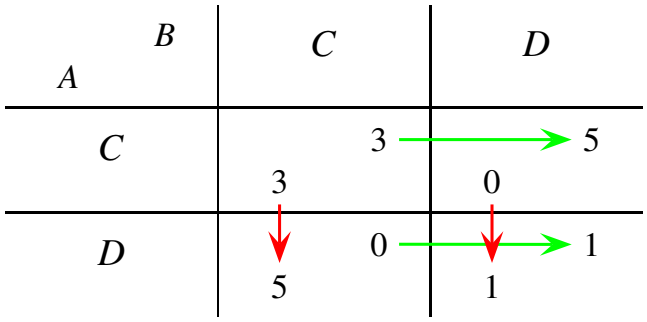


Figure 2.2: Payoff matrix of Prisoner's Dilemma. Each of the prisoners, Alice(A) and Bob(B), either defects (finks against the other) or cooperates (maintains silence). Each number represents 5 minus the number of years in jail. Each row represents the action A takes. The lower-left item of each box represents the payoff A receives. Columns and the upper-right items are for B. The arrows represent the direction of the best responses at situations, vertical ones for A and horizontal ones for B.

In the game, Alice believes that to defect is better than to cooperate, if she knows the action Bob takes. So does Bob. Therefore, the profile, (defect, defect), is the outcome of the game, *i.e.* the Nash equilibrium of the game. However, the payoffs for (defect, defect) are both less than those for (cooperate, cooperate). Therefore (cooperate, cooperate) is more Pareto-efficient than (defect, defect).

A variant of the game is called *Iterated Prisoner's Dilemma* (IPD), which the game is played sequentially an infinite number of times (Rubinstein, 1986). At each step, each player chooses a strategy, the decision of which may depend on profiles in the previous events called *history*. A policy is how a player chooses a strategy in each step. At the end of each step, the players receive payoffs.

A player is said to be restricted to *regular policy*, if and only if the strategies played by the player can be represented by deterministic finite automata (DFA) (Rubinstein, 1986). The input of the DFA is the perception of the player,

i.e. the opponent's most recent strategy. *A policy function* π is a map from the state of DFA to strategy of next stage. We write a strategy a as $\pi(s)$, where s is the state of DFA.

Tit-for-tat is a policy for IPD, with which an agent simply commits the strategy that the opponent commits in the most recent step. The policy is represented as Figure 2.3, where c means cooperate and d means defect. The policy function is $\pi(c) = c$ and $\pi(d) = d$.

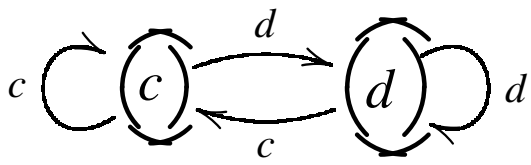


Figure 2.3: DFA representation of Tit-for-Tat. In this figure, c means cooperate and d means defect. The edge represents the opponent's most recent strategy. The state label represents the player's next action.

The best-response to tit-for-tat is given in (Axelrod, 1984) as Equation 2.2 with respect to the payoff discount factor γ.

$$\pi^{opt} = \begin{cases} \text{All c} & \frac{2}{3} \leq \gamma < 1 \\ \text{Alternate between c and d} & \frac{1}{4} \leq \gamma < \frac{2}{3} \\ \text{All d} & 0 \leq \gamma < \frac{1}{4} \end{cases} \tag{2.2}$$

When γ is greater than $\frac{2}{3}$, the player tends to cooperate, because the future reward contributes a large portion of the discounted reward. When γ is less than $\frac{1}{4}$, the current reward contributes greater than the future ones, therefore, in the first step, the player defects and keeps defecting because the future rewards can not compensate the reward to cooperate with a tit-for-tat player who defects. When $\frac{1}{4} \leq \gamma < \frac{2}{3}$, the future rewards compensate the reward of cooperating

with a tit-for-tat player who defects, therefore, the player cooperates and defects alternately.

However, TFT is not the optimal strategy against a weak player, because a TFT player can not exploit an all-cooperate player.

In a noisy environment, TFT is not a good strategy against another TFT player. The players will play each combination with equal probability.

2.1.3 The Tragedy of the Commons

Another dilemma is called *the tragedy of the commons*, in which, if players receive worse payoffs to take the same actions than to take the different actions. The payoff matrix of a simple example is shown in Figure 2.4. In the game, there are two Nash equilibria, (apple, orange) and (orange,apple).

Alice \ Bob	Apple	Orange
Apple	3, 3	4, 6
Orange	6, 4	2, 2

Figure 2.4: The payoff matrix of a simple example of the tragedy of the commons. Alice and Bob can choose one of the fruits, apple or orange. If they choose the same fruit, they have to share the fruit.

Two players, Alice and Bob, play a non-cooperative game to choose one of the fruits, apple and orange. The apple values 6, and the orange 4. If they choose the same fruit, they have to share it. If they choose differently, they can have the whole value respectively. From the payoff matrix, the apple values higher than the orange. If both of them choose the apple, the result is worse than one chooses apple and the other chooses orange. The solution for this game is to find a mechanism to coordinate the players.

2.1.4 Work and Shirk

In some games, such as work-n-shirk(Fig.2.1.4) and matching pennies, pure equilibria may not exist, therefore mixed strategies (Nash, 1951) should be included in the set of support of a general-purposed learning agent. The interpretation of mixed strategies that we use is probabilistically choosing actions from pure strategies.

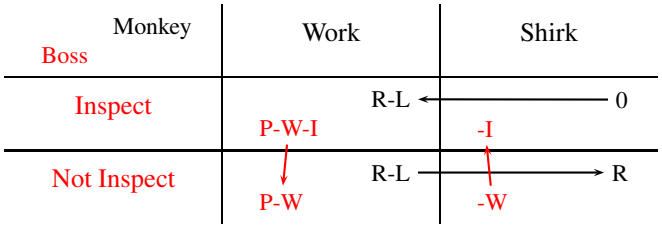


Figure 2.5: The payoff matrices of work-n-shirk. Each row represents the action A takes. The lower-left item of each box represents the payoff A receives. Columns and the upper-right items are for B. The arrows represent the direction of the best responses at situations, vertical ones for A and horizontal ones for B.

In work-n-shirk game, Boss expects Monkey to work so that Boss can have product, payoff P. To encourage Monkey to work, Boss pays a wage, W, to Monkey, unless Boss find that Monkey does not work. The wage is a reward, R, to Monkey. However, it costs Boss I to inspect and Monkey L to work. If Monkey works, Boss will reduce the probability of inspection to reduce the cost of inspection. If Monkey does not work, Boss will increase the probability of inspection to reduce the cost of wage. If Boss inspects, Monkey will work to get more reward. If Boss does not inspect, Monkey will shirk to reduce the cost of work.

2.2 Reinforcement Learning

2.2.1 Markov Decision Processes

Markov decision processes (MDPs) originate in the study of stochastic optimal control (Bellman, 1957) and have been widely used as a popular method for modeling and solving decision theoretic planning problems ever since (Howard, 1960; Puterman, 1994). It is based on the Markovian assumption: the rewards and the dynamics only depend on the current state.

This section provides a formal notation to describe the model of Markov decision processes. The model consists of two parts, the environment and one agent.

The environment is approximated by a discrete *state space*, a discrete *action space*, and discrete time. At each *time step*, the agent executes an action and the environment transforms from the current state to a new state that results from the action.

Let $\mathcal{X}$ be the finite set of states, $\mathcal{A}$ be the set of actions and $\mathcal{T}$ be the discrete set of time (Z). Let $X_t \in \mathcal{X}$ [1] be the random variable of the state at time t, $A_t \in \mathcal{A}$ be the random variable of the agent's action at time t.

In the model of an MDP, the environment is assumed *time-invariant* or *stationary*, that is, the transition of the environment does not depend on the time. The random variables X_t are Markovian. We write it as

$$T(x_k, x_i, a_j) = P(X_t = x_k | X_{t-1} = x_i, A_t = a_j),$$

where $x_k, x_i \in \mathcal{X}$ and $a_j \in \mathcal{A}$.

Now we formalize the agent. At each time step, the agent observes the environment, makes an action decision, and receives a reward and a new observation.

[1] Precisely, it should be written as $X_t : \Omega \to \mathcal{X}$ in a probability space $(\Omega, \mathcal{F}, \mathcal{P})$.

Let $\mathcal{R}$ be the set of possible rewards, $R_t \in \mathcal{R}$ be the random variable of the agent's reward at time t. In the model of MDP, R_t only depends on X_{t-1} and A_t [2]. In an iterated game, the *discounted reward* is a criterion of evaluation, which derived from discounted sum of payoff function as Equation 2.3.

$$R_{it}(s) = \sum_{k=0}^{\infty} \gamma^k r_i(s_{t+1+k}) \tag{2.3}$$

where γ is a discount factor for the future payoffs, and $0 \leq \gamma < 1$[3].

Let $\mathcal{O}$ be the finite set of the agent's observations, $O_t \in \mathcal{O}$ be the random variable of agent's observation at time t. In the model of MDP, O_t only depends on X_{t-1} and A_{t-1}.

Let $\mathcal{S}$ be the finite set of the agent's internal states. A *stationary Markovian policy* is a mapping $\pi : S \rightarrow A$, where $\pi(s)$ denotes the action an agent should perform whenever it is in state s. We adopt the discount factor γ over an infinite horizon as our optimality criterion.

2.2.2 Learning Algorithms

Reinforcement Learning (RL) (Barto et al., 1983; Sutton, 1984; Watkins, 1989) is a formal framework for learning MDP agents' knowledge is expected optimal policy, how to choose actions to maximize reward. We can rewrite the problem of optimal stationary Markovian policy as Equation 2.4 and 2.5.

$$Q(s,a) = R(s,a) + \gamma \sum_{t \in S} Pr(s,a,t)V(t) \tag{2.4}$$

$$V(s) = \max_{a \in A_s} Q(s,a) \tag{2.5}$$

[2] In (Howard, 1960), the reward does not depend on the performed action.

[3] Though cases with $\gamma = 1$ are possible, they are less common (Mahadevan, 1996). A similar model, *infinite-horizon discounted model*, could be found in (Kaelbling et al., 1996)

Then, we can define the optimal policy as Equation 2.6.

$$\pi(s) = \arg\max_{a \in A_s} Q(s, a) \tag{2.6}$$

Q-learning (Watkins, 1989; Watkins & Dayan, 1992) has attracted much attention as an implementation of reinforcement learning. The learning processing is updating Q-values and optimal value function as Equation 2.7.

$$Q(s_{t-1}, a_{t-1}) \leftarrow (1 - \alpha)Q(s_{t-1}, a_{t-1}) + \alpha(R(s_{t-1}, a_{t-1}) + \gamma V(s_t)) \tag{2.7}$$

where,

$$\begin{aligned} U^{\pi}(s) &= E\{R_t | s_t = s, \pi\} \\ &= r(s, \pi(s)) + \gamma \sum_{t \in S} Pr(t|s, \pi(s)) U^{\pi}(t) \end{aligned}$$

The best that can be done in a state is its optimal value:

$$U^*(s) = \max_{\pi} U^{\pi}(s) \tag{2.8}$$

There is always at least one *optimal policy*, π^*, that achieves this maximum at all states $s \in S$.

Other reinforcement learning methods can be found as *Monte Carlo* methods (see (Barto & Duff, 1994)), *Temporal-Difference Learning* (Sutton, 1988), *Dyna architecture* (Sutton, 1990; Sutton, 1991).

2.2.3 Perceptual Aliasing

A Markov decision process assumes that there exists a one-to-one correspondence between the agent's observations and states of the environment. However, in the real world, agents have only *partial observability* of its environment, which is

described by a *partial observable Markov decision process* (POMDP) (Astrom, 1965).

In games, an agents has no *a priori* information of the opponent model, that is, the states of the opponent model are hidden and a perception does not uniquely reflect a state of the world. This problem is called *perceptual aliasing* (Whitehead, 1992).

One approach, named *sensor-based approach*, uses additional perceptual actions to disambiguate the state, such as *Lion algorithm* (Whitehead, 1992) and *Cost Sensitive Reinforcement Learning* (Tan, 1991). However, one disadvantage of the algorithms is the assumption of a deterministic environment. A limitation of this approach is incapable to *incomplete perception* (Chrisman et al., 1991).

Another approach, named *memory-based approach*, distinguishes certain states by remembering the past perceptions and actions. The *Window-Q* algorithm (Lin & Mitchell, 1992) is an attempt to extend the Q-learning algorithm with a sliding window over the past N observations and actions. Unfortunately, N is fixed. In the real world, we can hardly estimate N precisely *a priori*. The *Recurrent-Q* algorithm (Lin, 1993) trains a recurrent neural network containing a fixed number of hidden units. This algorithm also faces the similar problem since the number of hidden units is fixed.

The *hierarchical Q-learning* (Wiering & Schmidhuber, 1997) is based on an ordered sequence of subagents, each of which learns and solves a Markovian subtask. When a agent meet its subgoal, the transfer control unit passes control to its successor. The hierarchical structure enlarges the state space. *W-learning* (Humphrys, 1996) and nested Q-learning (Digney, 1996) are similar to this approach. The problem is that the number of agents is fixed before learning begins.

Ring (1993) has developed a non-recurrent hierarchical neural network to learn the sequential tasks with hidden states. High-level units are added online when

the unit activations are unreliably predicated. When the time delay is large, the method suffers the problem of high-level units over-added. Pyeatt and Howe (1999) describe a similar method.

Perceptual Distinctions Approach (Chrisman, 1992) and *Utile Distinction Memory* (UDM) (McCallum, 1993) learn POMDP via state-splitting strategies. The difference is that in the former states are split in order to increase the ability of perceptions while in the latter in order to increase the ability of predict rewards. This method can not solve problem when predicting reward depends on the conjunction of several perceptions while irrelevant to any individual perception.

By adopting of *Prediction Suffix Tree Learning* (Ron et al., 1994; Ron, 1995), (McCallum, 1995) presents *Utile Suffix Memory* (USM), which combines the advantages of instance-based learning and utile distinctions and results in a pretty good performance. The key idea behind the utile suffix memory is to construct an *utile suffix tree* with the most recently observations and actions, then to seek a closest state (instance set) on the tree, and to choose an action based on the instances. The skeleton of the algorithm is as follows:

1. Make a tree with only a root node and an empty history.

2. Add the last observation and action to history.

3. Seek the closest node corresponding to the last state and update the Q-values of the node as Equation 2.9 [4].

$$Q(s,a) \leftarrow R(s,a) + \gamma \sum_{s'} Pr(s'|s,a)U(s') \tag{2.9}$$

   where $R(s,a)$ is average reward on pairs of state s and action a, $Pr(s'|s,a)$ is the frequency of s' given state s and action a, and $U(s) = \max_{a \in A} Q(s,a)$.

[4] McCallum uses U instead of V in previous equations.

4. Compare fringe nodes using Kolmogorov-Smirnov test. If different, promote the fringe nodes.

5. Choose next action and repeat step 2 to 4.

2.2.4 Exploration vs Exploitation

When performing a learning algorithm, an agent expects to take the action with the best estimated reward. However, the knowledge held by the agent can not guarantee the observed best action is actually the best one. Therefore, the agent must make a tradeoff between exploiting the current knowledge and exploring for the future knowledge.

Boltzmann exploration (Sutton, 1990) assigns a probability to any action according to its estimated utility U_i and a parameter T called temperature. The probability of action r_i is assigned as

$$Pr(r_i) = \frac{e^{U_i/T}}{\sum_{j=1}^{n} e^{U_j/T}} \tag{2.10}$$

When $U_i > U_j$, we have

$$\lim_{T \to \infty} \frac{e^{U_i/T}}{e^{U_j/T}} = 1 \tag{2.11}$$

$$\lim_{T \to 0} \frac{e^{U_i/T}}{e^{U_j/T}} = \infty \tag{2.12}$$

Therefore, when the temperature is large enough, the Boltzmann exploration acts exactly like random exploration; when the temperature goes to zero, the Boltzmann exploration acts exactly like best selection. When temperature is decreasing, the algorithm converges empirically, but there are not theoretical guarantees of the optimal results. Simply decreasing the temperature in Boltzmann exploration does not work with a multiagent environment. For example, suppose Bob can choose two routes, A and B. After daily driving, he learns that Route A has less traffic. Using Boltzmann exploration with decreasing temperature, he chooses

the optimal solution. However, the traffic control center changes some rules, the traffic on two routes changes as other drivers changes their strategies. Bob may notice the change of traffic, and explore two routes. The environment is not well explored due to the instability of Q-values.

Exploration problem is similar to *two-armed bandit problem*, which appears to have originated with the paper by Thompson (1933). A gambler is given N coins with which to play a slot machine having two arms. Two arms may have different payoff rates and variances. However, the gambler has no *a priori* information about two arms. Her goal is to maximize her total payoff during N trials. Bellman (1961) and Hellman and Cover (1970) give interesting discussions of the problem.

Let the average payoffs to the arms be μ_1 and μ_2 ($\mu_1 > \mu_2$) respectively, and the variances be σ_1^2 and σ_2^2. The gambler allocates n trials to the observed second best of the two bandits. Then the problem is find out the optimal allocation of trials n^*. Holland (1975) provide a solution as Equation 2.13.

$$n^* \approx b^2 \ln \frac{N^2}{8\pi b^4 \ln N^2} \tag{2.13}$$

where $b = \frac{\sigma_1}{\mu_1 - \mu_2}$. The optimal allocation of trials $N - n^*$ to the observed better arm is e^{cn^*}, where c is a constant. This looks similar to Boltzmann exploration. Holland's solution is on the finite-horizon case. In Markov decision processes, we are more interested in infinite-horizon discounted case. Bellman and Kalaba adopted a Bayesian formulation to solve the problem with dynamic programming algorithm (Bellman, 1956; Bellman & Kalaba, 1959).

The Bayesian solution to this problem was proposed by Gittins and Jones (1974). Consider a reward process, let $\mathcal{S}$ be its state space, $\mathcal{B}$ be the power set of $\mathcal{S}$, $s_k \in \mathcal{S}$ be the state of the process at time k. For $B \in \mathcal{B}$, let $\tau(B)$ be the number of transitions before it enters B, $V(s, B)$ be the expected discounted reward:

$$V(s, B) = \frac{E(\sum_{k=0}^{\tau(B)-1} \gamma^k R(s_k) | s_0 = s)}{E(\sum_{k=0}^{\tau(B)-1} \gamma^k | s_0 = s)} \tag{2.14}$$

Then the Gittins index of state s is defined as

$$V(i) = \max_{B \in \mathcal{B}} V(i, B) \tag{2.15}$$

The optimal policy just consists of action to state with largest index in each stage.

A non-Bayesian solution to two-armed bandit problem was proposed by Kaelbling (1993). The algorithm, called 'interval estimation', consists in choosing the arm with the greater upper-bound ub of a $1 - \theta$ confidence interval of random reward ρ. Suppose ρ follows a normal distribution which unknown mean and unknown variance, then we have

$$ub = \bar{\rho} + s\frac{t_{\theta/2}^{n-1}}{\sqrt{n}} \tag{2.16}$$

where $\bar{\rho}$ is the sample mean of n observations of ρ, s is the sample standard deviation of n observations of ρ and $t_{\theta/2}^{n-1}$ is Student's t-function.

However, these solutions are to systems with single state, which leads to a local-scale reasoning about information that is insufficient in many multi-state environments. Based on the local-scale bonuses, a empirical solution to multi-state environments is to back-propagate the exploration bonuses to simulate global-scale reasoning about the uncertainty (Meuleau & Bourgine, 1999).

2.2.5 Multi-layer Models

Markey (1994) applies parallel Q-learning to the problem with multi degrees of freedom. Each agent controls one degree of freedom. Tham and Prager (1994) mention the similar idea of a separate Q-network. Dayan and Hinton (1993) propose a two-level learning scheme, named *Feudal RL*. The higher-level managers set tasks for lower level managers. However, the open question is whether the structure can learned online.

2.3 Multiagent Reinforcement Learning

Many AI researchers have applied reinforcement learning techniques to solve problems in games, such as checkers (Samuel, 1959), tic-tac-toe (Boyan, 1992), backgammon (Tesauro, 1992), and Go (Schraudolph et al., 1994). These games are alternative Markov games. Another type of games called simultaneous Markov games. Some problems have been studied in (Yanco & Stein, 1993; Tan, 1993). Simultaneous Markov games are called stochastic games, which are models of sequential decision making based on the Markovian assumption (Shapley, 1953; van der Wal, 1981; Owen, 1982; Littman, 1994).

Littman (1994) proposed a frame work for learning in zero-sum games, called minimax-Q learning. The frame work is to to find optimal policies for a zero-sum game of "rock, paper, scissors". The method is a variation of Q-learning, named *minimax-Q learning algorithm*. The modification lies in using $Q(s, a, o)$ instead of $Q(s, a)$, where o is the action chosen by the opponent and replacing Equation 2.5 with Equation 2.17 and Equation 2.4 with Equation 2.18.

$$V(s) = \max_{\pi \in PD(A)} \min_{o \in O} \sum_{a \in A} Q(s, a, o)\pi_a \tag{2.17}$$

$$Q(s, a, o) = R(s, a, o) + \gamma \sum_{s'} T(s, a, o, s')V(s') \tag{2.18}$$

However, the minimax-Q learning algorithm is limited in the setting of totally negatively correlated payoffs. The minimum of Q values over opponent action is the optimal action of opponent coincidently. Therefore, agent can estimate the action a rational opponent would take. The minimax-Q learning algorithm is unlikely to be extends to non-zero-sum Markov game or other multiagent systems. Some other leaning algorithms (Hu & Wellman, 1998; Brafman & Tennenholtz, 2001; Singh et al., 2000) are based on a similar framework.

In the frame work it is assumed that both players can observe their own payoffs as well as monitor those of the opponents (In zero-sum games, the payoff matrices of their opponents are the negatives of their own ones), but for many real life problems agents are unable to monitor the opponents' payoff matrices. For example, monkeys play against machines in some neuroscientific experiments, where the monkeys' payoffs as well as their mix of strategies of monkeys are not observable to machines. Moreover, the monkeys' payoff matrices may change from time to time because the value of reward (*i.e.* juice) depends on whether they are thirsty.

For reinforcement learning in non-zero-sum games, Carmel and Markovitch (1996) describe a *model-based* approach, in which the learning process splits into two separate stages. In the first stage, the agent infers a model of the other agent based on history. In the second stage, the agent utilized the learned model to predicate strategies for the future. Unfortunately, Carmel and Markovitch's work is focus on learning agent against non-learning agent.

Sandholm and Crites (1996) build Q-learning agents for IPD. They show the optimal strategy learned against the opponent with fixed policy, tit-for-tat, and the behaviors when two Q-learning agents face each other. They address the hidden state problem in two different ways, fixed windows Q-learning and recurrent neural network. The convergence of the algorithm is not proved in non-stationary setting.

Many other approaches try to build opponents' models while playing. Bui et al. (1996) did research on learning probabilistic models of the preference of other agents in the meeting scheduling domain. Sen and Arora (1997) used a maximum expected utility principle approach to exploiting learned opponent's models. In their approach, conditional probabilities for different opponent's actions corresponding to all actions from the current state are used to compute expected utilities of each of the possible actions. The action with the maximum

expected utility is then played. A probabilistic model of the opponents' strategies is developed by observing actions played by the adversaries in different discrepancy ranges as measured by the evaluation function of the player. Model-based learning method can efficiently find some deterministic rules if the assumption of deterministic opponent is valid. However, as we stated before, in general situations, the adversaries are learning agents too, and unlikely to be deterministic. Our approach, though, can also be extended to discover deterministic rules at the cost of increasing the state space.

Some other approaches focus on a single class of games. Billings (2000) proposed an approach to play competition games like rock-paper-scissors. Claus and Boutillier (1998) proposed an algorithm that focuses on coordination games, where the individual payoff to a single agent is proportional to the global utility. Sugawara and Lesser (1998) have presented an explanation based learning approach to improving problem solving behavior. Williams (2001) proposed methods to choose equilibrium in a game with multiple stable equilibria. Sandholm and Crites (1996) did some experiments on the iterated prisoners' dilemma by using Q-learning with Boltzmann exploration. The best case is that 25 trials out of 100 converge to the optimal cooperate-cooperate result. Their experiments demonstrate that playing against another learner was more difficult than playing against a fixed strategy agent because of the peer learner's non-stationary behavior and the average payoff in each single stage game increases monotonically with longer Boltzmann exploration schedules. Some genetic algorithms were proposed to simulate the iterated prisoners' dilemma, *e.g.* (Axelrod, 1984). In contrast to these approaches, our algorithm is able to learn strategies for arbitrary games without extensive modeling. Local prisoners' dilemma (Nowak & May, 1993) and local Hawk-Dove game (Ahmed & Elgazzar, 2000).

It is worth to mention that some team learning (Stone & Veloso, 1999; Jain & Sharma, 1995; Smith, 1982). It focuses on teams of independent machines that

learn to identify functions or languages and on the theoretical characterization. There are also a number of approaches of distributed reinforcement learning such as (Weiss, 1998; Schaerf et al., 1995; Littman & Boyan, 1993; Crites & Barto, 1996).

The limitation in the mathematical framework of Markov decision processes is assuming that a single adaptive agent interacts with a stationary environment. In this view, secondary autonomous agents can only be part of environment and are therefore fixed in their behavior not adaptive. I am going to study the mechanism of coordination of agents' behaviors and stableness of multiagent systems. The research is going to be focused on general cases (not totally positive or negative), and systems with complicated setting, such as varying payoffs, communications, negotiation, etc.

Because the payoffs to their opponents are unobservable to individual agents, a selfish agent criterion is adopted in this research, with which agents attempt to maximize their own rewards instead of global rewards. The minimax payoff strategy is to use the strategy that maximizes the minimum payoffs of all strategies. This is considered to be a safe strategy, which means the player taking the strategy will get a payoff not worse than minimax payoff, whatever the opponent plays. It, however, fails to meet the selfish criterion, because it will neither cooperate with the opponent in a general sum game, nor exploit weak opponents. In this paper, we will present an approach which enable agents to cooperate with their opponents in a general sum game, and even exploit weak opponents if possible.

3 Learning to Cooperate

Many researches in multi-agent learning literature have been done in learning Nash equilibria in known or unknown games. The solution at a Nash equilibrium is safe for an individual agent because the agent would not receive less payoff if other agents do not receive less payoffs. However, the goal of learning to converge to a Nash equilibrium rules out possible cooperation in some games. If agents cooperate in some games, such as prisoners' dilemma, the payoff for each agent may be better than the payoff at a Nash equilibrium, *i.e.*, some solutions are more Pareto efficient than the Nash equilibrium. Therefore, in this chapter I am going to explore possible cooperative solutions in multi-agent learning systems. The multi-agent systems we are interested to have the following characteristics:

- Two or more autonomous agents (or decision makers) simultaneously learn to improve their own decisions in order to achieve better individual rewards. Since the goal is to maximize the individual rewards, a Nash equilibrium is not an optimal solution if another situation is more Pareto efficient than the equilibrium, and all agents are willing to accept that situation. For example, in Prisoners' dilemma, if both players satisfy the *cooperate-cooperate* situation, then the Nash equilibrium *defect-defect* is not the solution we desire.

- The underlying systems are distributed or decentralized. We are seeking a solution that autonomous agents will cooperate without the help from a external teacher or mediator.

- Each agent has no private communication with other agents. Each agent is unable to access to the internal state of the agents. To explore normal cases of multi-agent learning, we avoid the approaches relying on sharing knowledge.

- The information of rewards is partial. Each agent only knows the reward to itself, and does not share it with other agents. Therefore, it is not necessary to assume that other agents are rational to their rewards.

In the game of matching pennies, no pure equilibrium exists, therefore mixed strategies should be included in the set of support of a general-purposed learning agent, with the result that learning agents are nondeterministic. One interpretation of mixed strategies probabilistically chooses actions from pure strategies. Unaware of the payoff matrix to the opponent, a learning agent can not compute the exact Nash equilibrium. A fictitious player (Fudenberg & Levine, 1998) assumes that the opponent is consistent with its mixed strategy. However, it is well known that the mixed strategies do not converge if two fictitious players play against each other in matching pennies. To find a convergent approach, the opponents' responses should be taken into consideration.

In games like prisoners' dilemma (Figure 2.2), the unique equilibrium is for both players to play d (*defect*), resulting in less payoffs $(1, 1)$ than $(3, 3)$ if both players play c (*cooperate*). If a player believes that its opponent does not change its strategy, then it will always play d. To overcome this dilemma, we also have to take the opponents' responses into consideration.

To predict opponents' actions in prisoners' dilemma, Carmel and Markovitch (1996) proposed a method to model opponents' strategies as finite automata.

Players are able to predict the deterministic strategy of their opponents. However, if the opponents are also learning agents, they are unlikely to be deterministic. Moreover, deterministic players may play worse in noisy environments, where agents may probabilistically realize an action that is different from the one they intend to. Noisy environments are very common in some real life problems. Therefore, we propose an approach for learning agents to analyze their opponents' responses via statistical data without modeling entire opponents' strategies. Considering the opponents' responses, agents can maximize the expected payoff with a better estimation.

In a multi-agent learning systems, we expect learning agents to cooperate with each other instead of fictitiously assuming that the opponent's strategy is independent. The responses of the opponent depends on the behavior of the agent. Therefore, to achieve cooperative results, we can not simply use the fictitious gradient ascent dynamics. This chapter aims to explore possible cooperative results.

In this chapter, I first define the problem interested in Section 3.1, then describe a probabilistic Tit-for-tat (PTFT) strategy for cooperative games and analyze it Section 3.2. Using PTFT as a higher level controller, a *Hill-climbing exploration* (HCE) algorithm is presented for the case that the mixed strategy of the opponent is observable. At last, I extend the algorithm to HCE algorithm for stochastic partial information general-sum games.

3.1 Definitions

In a two-player game, let r be the payoff matrix to player A. If A plays action c and its opponent B plays d, it receives a payoff of r_{cd}, *etc.* Suppose that it randomly plays action d at a rate x and its opponent randomly plays action d at a rate y. Then, its expected reward is $R_A(x, y)$ in Eq. (3.1).

$$R_A(x,y) = r_{cc}(1-x)(1-y) + r_{dd}xy + r_{cd}(1-x)y + r_{dc}x(1-y) \quad (3.1)$$

Because R_B has a similar form, therefore we use R to denote both R_A and R_B in general.

For a general game, R could be a function without the explicit form. To make the algorithm works, we need an assumption on R.

Assumption 1 *The payoff (utility) function R is derivable along each direction and the derivatives are uniformly continuous.*

Considering players change their strategies, we have Eq. (3.2).

$$\frac{\partial R}{\partial x} = uy - (r_{dc} - r_{cc}) \qquad \text{and} \qquad \frac{\partial R}{\partial y} = ux - (r_{cd} - r_{cc}), \tag{3.2}$$

where, $u = r_{dd} + r_{cc} - r_{cd} - r_{dc}$.

If the system is treated as a dynamic system, we have

$$\frac{dR}{dt} = \frac{\partial R}{\partial x}\frac{dx}{dt} + \frac{\partial R}{\partial y}\frac{dy}{dt} \tag{3.3}$$

By assuming that the opponent's strategy is independent, the second term in the right-hand side of Eq. (3.3) can be ignored. As the result, the fictitious gradient ascent dynamics can be written as Eq. (3.4). It gradually improves the payoff given the assumption holds.

$$\frac{dx}{dt} = \frac{\partial R}{\partial x} \tag{3.4}$$

However, this assumption may not hold in multi-agent system. Simply following Eq. (3.4) may not result a increasing in the payoff. For example, in prisoners' dilemma, the strategies converge to the Nash equilibrium, (d, d), which has lower payoffs than the Pareto-optimal (c, c) in Figure 2.2.

Now we only consider the two-player continuous games or games with mixed strategies. Due to agents can not observe the payoff function of other agents, it is difficult to find out a global "optimal" strategy. Therefore, we focus on gradual

improving method, like gradient ascent dynamics. Hopefully, we can find a locally "optimal" strategy instead.

If strategies are constrained within a closed concave neighbor of (a_1, a_2), it is also a game. Now we use the closed concave neighbors.

Definition 5 *If the strategies of a two-player game $\mathcal{G}$ are constrained within $([a_1 - \delta, a_1 + \delta] \cap \mathcal{A}_1) \times ([a_2 - \delta, a_2 + \delta] \cap \mathcal{A}_2)$, the constrained game is called δ-perturbed game at (a_1, a_2).*

Now we explicitly define the criteria of the locally "optimal".

Definition 6 *A strategy combination (x, y) is locally Pareto-optimal, if and only if, there exists a $\delta_0 > 0$; for any $\delta < \delta_0$, either (x, y) is more Pareto efficient than the Nash equilibrium in δ-perturbed game at (x, y), or (x, y) is the Nash equilibrium and no other combination is more Pareto efficient than (x, y).*

Since the derivatives of $R(x, y)$ are uniformly continuous, $\partial R/\partial x$ and $\partial R/\partial y$ are the only factors affecting the characteristic of the game.

Definition 7 *In a δ-perturbed game at (x, y), the opponent's strategy y is dominant in the payoff, if and only if $|\partial R/\partial x| > |\partial R/\partial y|$. The agent's own strategy, x, is dominant in the payoff, if and only if $|\partial R/\partial x| < |\partial R/\partial y|$.*

In an unknown game, agents need to explore some unknown region or combination of strategies to achieve higher payoffs. Exploration tries to find better strategies, while exploitation uses the best strategy so far. There is a tradeoff between exploration and exploitation. Since exploration does not take the best strategy so far, it may result a less optimal payoff than the best strategy. The difference is called *exploration cost*. In this research, we want to limit the exploration cost within an arbitrary small value. Therefore, we use ϵ-optimal.

Definition 8 *A strategy $\mathcal{A}$ is an ϵ-optimal strategy against strategy $\mathcal{B}$, if and only if the difference between the expected payoff of $\mathcal{A}$ against $\mathcal{B}$ and the expected payoff of optimal strategy against $\mathcal{B}$ is less than ϵ.*

Therefore, our goal is to find a ϵ-optimal solution for any infinitesimal ϵ. As ϵ goes to 0, the solution approximates the optimal. The goal is to find a ϵ-optimal solution of a local Pareto optimal.

3.2 The probabilistic tit-for-tat strategy

The tit-for-tat strategy is not optimal when the noise exists in the system. Some literatures has mentioned probabilistic generous tit-for-tat strategy. For future use, I formally describe a version of probabilistic tit-for-tat (PTFT) strategy in this section. This strategy solves the noise problem of the original tit-for-tat at the cost of an arbitrary small.

The strategy is to play C with a probability of δ if the opponent played D in the previous step, to play C with a probability of $1-\lambda\delta$ if the opponent played C in the previous step, and to play D otherwise. Here, δ is a small positive number and λ is a non-negative number less than 1.

\ to play	C	D
opponent played C	$1-\lambda\delta$	$\lambda\delta$
opponent played D	δ	$1-\delta$

Table 3.1: The probabilistic tit-for-tat strategy, where δ is a small positive number and λ is a positive number less than 1.

Now, we analyze the cost of this strategy against all-defect strategy. Since the opponent always play defect, the expected payoff to the PTFT agent is

$$\delta S + (1-\delta)P = P - \delta(P-S).$$

Because $P > S$, PTFT is ϵ-optimal strategy against all-defect strategy if

$$\delta < \frac{\epsilon}{P - S}.$$

Suppose that a PTFT agent plays against another PTFT. The output can one of four combinations, CC, CD, DC, and DD. We can have transition matrix T of these combinations as Table 3.2.

current \ next	CC	CD	DC	DD
CC	$(1-\lambda\delta)^2$	$\lambda\delta(1-\lambda\delta)$	$\lambda\delta(1-\lambda\delta)$	$(\lambda\delta)^2$
CD	$\delta(1-\lambda\delta)$	$\lambda\delta^2$	$(1-\lambda\delta)(1-\delta)$	$\lambda\delta(1-\delta)$
DC	$\delta(1-\lambda\delta)$	$(1-\lambda\delta)(1-\delta)$	$\lambda\delta^2$	$\lambda\delta(1-\delta)$
DD	δ^2	$\delta(1-\delta)$	$\delta(1-\delta)$	$(1-\delta)^2$

Table 3.2: Transit matrix T of PTFT against PTFT

Let row vector π denote the probabilities of combinations. In long run, π satisfies

$$\pi = \pi T \quad . \tag{3.5}$$

By solving Eq. 3.5, we have

$$\pi = \left[\frac{1}{(1+\lambda)^2}, \frac{\lambda}{(1+\lambda)^2}, \frac{\lambda}{(1+\lambda)^2}, \frac{\lambda^2}{(1+\lambda)^2}\right]. \tag{3.6}$$

Thus π is independent to δ. Let π_{CC} denote the first column of π, *i.e.*, the probability that agents play CC. The relation between π_{CC} and λ is shown in Figure 3.1, which shows that π_{CC} goes to 1 as λ goes to 0.

The expected payoff of a PTFT agent against another is

$$\frac{R + \lambda S + \lambda T + \lambda^2 P}{(1+\lambda)^2}.$$

Because $2R > S + T$ and $R > P$, PTFT is ϵ-optimal strategy against itself if

$$\lambda < \min\left(\frac{\epsilon}{4R - 2S - 2T}, \sqrt{\frac{\epsilon}{2R - 2P}}\right).$$

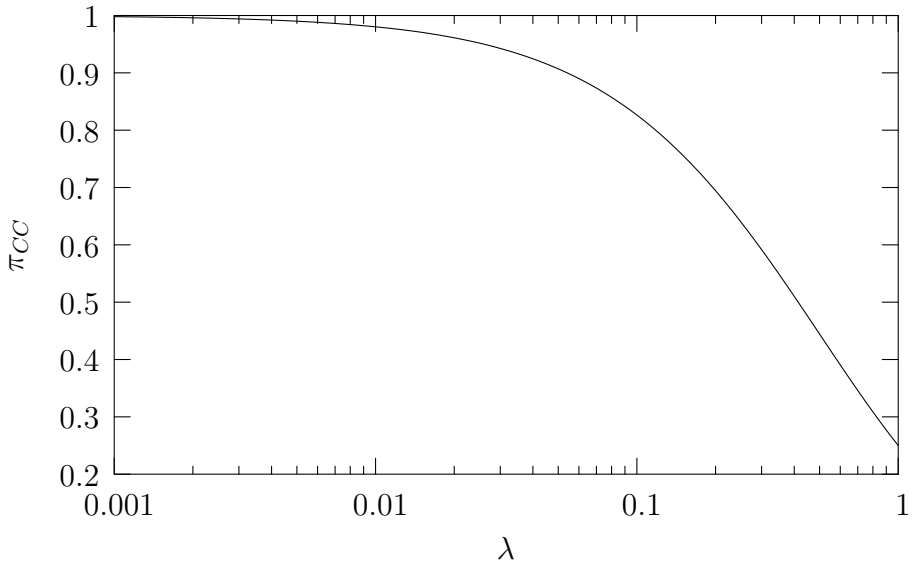


Figure 3.1: $\pi_{CC} = \frac{1}{(1+\lambda)^2}$

Though π is independent to δ, δ is related to the transition time from state DD to state CC.

Though the algorithm can not exploit a weak player, this strategy works well for two identical agents play the prisoners' dilemma. However, this strategy requires that agents are aware that they are playing the prisoners' dilemma.

3.3 The Hill-Climbing Exploration (HCE) Algorithm for Game of Continuous Strategies

Without knowing the payoff of the opponent, it makes the agent difficult to play cooperatively without risk the chance of being exploited. Even if agents are willing to cooperate, it is difficult to know what strategy is cooperative. In this section, an algorithm is proposed, with which an agent is to explore possible strategies that makes the opponent to perform certain strategies that increase its payoff. If payoffs of both agents are increased due to the opponent's strategy, it is cooperative. Also,

the risk of being exploited is limited.

The idea behind the *hill-climbing exploration* (HCE) algorithm is that agents explore strategies and hope the opponents' respond with strategies that benefit it and compensate the exploration cost. If agents are constantly benefited from such strategies, these strategies are cooperative ones.

In this section we use PTFT as an abstract model to indicate cooperation states, though PTFT presented in the previous section does not directly work for general-sum games.

The algorithm has two levels, namely hyper-level and hypo-level. In hyper-level, the hyper-controller makes the decision of abstract strategies, which are the states of the hypo-controller. There are two abstract strategies, *foresight* (F), like *cooperation* in PD, and *myopia* (M), like *defection* in PD. In the hypo level, the hypo-controller acts differently according to the states and also report abstract observations to the hyper-controller.

First, the agent evaluate the behavior of the opponent. the agent can the current perturbed-game, the current reward, the current action trend of the opponent and the payoff reference. The decisions can be made by the following criteria,

- If the agent's own action dominates in its payoff in the current perturbed-game, *i.e.*, $|\frac{\partial R}{\partial x}| > |\frac{\partial R}{\partial y}|$, it reports *malicious* (M);
- If the current change of the opponent, Δy, is unknown, *i.e.*, before the stableness of the opponent's strategy is observable, it reports *malicious* (M);
- If the current change of the opponent, Δy, is destructive, *i.e.*, $\frac{\partial R}{\partial y}\Delta y < 0$, it reports *malicious* (M);
- If the agent's current expected payoff is less than the payoff reference, *i.e.*, $R < \mu$, it reports *malicious* (M);
- Otherwise, it reports *generous* (G);

Besides, agents also monitor a payoff reference, μ, for evaluating the performance. The payoff reference can be estimated by the payoff average in a relative long period. Initially, it could be a relative large number. Agents update the reference as the game proceeds.

The hyper-controller plays a hyper-game with two strategies, *foresight* and *myopia*. The hyper-game is like a prisoners' dilemma. The strategy is the same as PTFT, here we let $\lambda = 0$. The hyper-controller instructs the hypo-controller to be *foreseeing* with a probability of δ if the hypo-controller reported that the opponent is *malicious*, to be *foreseeing* if the hypo-controller reported that the opponent is *generous*, and to be *myopic* otherwise. Initially, the state is *myopic*. The probabilistic decision is shown in Table 3.3.

	foresight	myopia
generous	$1 - \lambda\delta$	$\lambda\delta$
malicious	δ	$1 - \delta$

Table 3.3: The probabilistic decision of the hyper controller, where δ is a small positive number and λ is a positive number less than 1.

In the myopic state, the agents, acting as selfish agents, try to maximized their own reward without considering the behavior of the opponent. Following the gradient ascent method of reinforcement learning, the agents change their strategy x by Δx, where

$$\Delta x = \beta \frac{\partial R}{\partial x}$$

and β is the learning rate. Here, we apply the win-or-learn-fast (WoLF) method (Bowling & Veloso, 2001) to adjust β according to the reference μ, *i.e.*,

$$\beta = \begin{cases} \beta_{won} & \text{if } R > \mu, \\ \beta_{lose} & \text{otherwise} \end{cases} \tag{3.7}$$

where $0 < \beta_{won} < \beta_{lose}$.

The foreseeing state enables the agents to explore the neighborhood of the current joint strategy and hope to find a more Pareto efficient joint strategy than the current one. However, because the agents are unaware of the payoff of their opponents, they just perform a change Δx opposite to the changes in the myopic state, namely,

$$\Delta x = -\beta_{foresee} \frac{\partial R}{\partial x},$$

where $\beta_{foresee}$ is a positive number. Since $\beta_{foresee}$ is small, therefore the algorithm of agent searching for local Pareto efficient point is hill-climbing.

The pseudo algorithm is shown in Algorithm 1.

Initially, the hypo-controller always reports *malicious* to the hyper-controller, therefore the agent acts as a fictitious player. If the system converges, it converges to a Nash equilibrium. Actually, we only expect that the system finally converges to a small region within a diameter less than ϵ. To determine whether the goal of this phase is achieved, each agent can test a hypothesis on the other agent that the expected strategy of the opponent is out of the small region and try to reject the hypothesis with a confidence level α. Also, each agent should ensure that the other agent observe that the agent has stabilized its strategy. Once the agent is confident that the other agent has stabilized it strategy and that the other agent observed also the stableness, it switch to the exploration phase. The agents can use the expected payoff as a reference for the further exploration.

The experiments show that HCE algorithm performs as expected in playing against another HCE (Figure 3.2), All Cooperate (Figure 3.3), All Defect (Figure 3.4) and TFT (Figure 3.5).

HCE algorithm also performs as expected in matching pennies (Figure 3.6) and (Figure 3.7).

Algorithm 1 HCE algorithm

procedure *Initialization* ()

$s = N$;

Randomize x, y, μ;

end

procedure *Execute* ()

Perform strategy x.

end

procedure *Update* (x_t, y_t, R_t)

Update payoff function R according to reward R_t

if $s == M$ **then** update μ with R_t **end if**

if $|\frac{\partial R}{\partial y}| > |\frac{\partial R}{\partial x}|$ **and** $\frac{\partial R}{\partial y}(y_t - y) > 0$ **and** $R_t > \mu$ **then**

$s = C$

else

Set s to C with probability δ, N otherwise;

end if

if $s == M$ **and** $R_t > \mu$ **then**

$\beta = \beta_{win}$

else if $s == M$ **and** $R_t \leq \mu$ **then**

$\beta = \beta_{lose}$

else

$\beta = -\beta_{explore}$

end if

$x = x + \beta\frac{\partial R}{\partial x}$

$y = y + \min(1, |\beta\frac{\partial R/\partial x}{\partial R/\partial y}|)(y_t - y)$

end

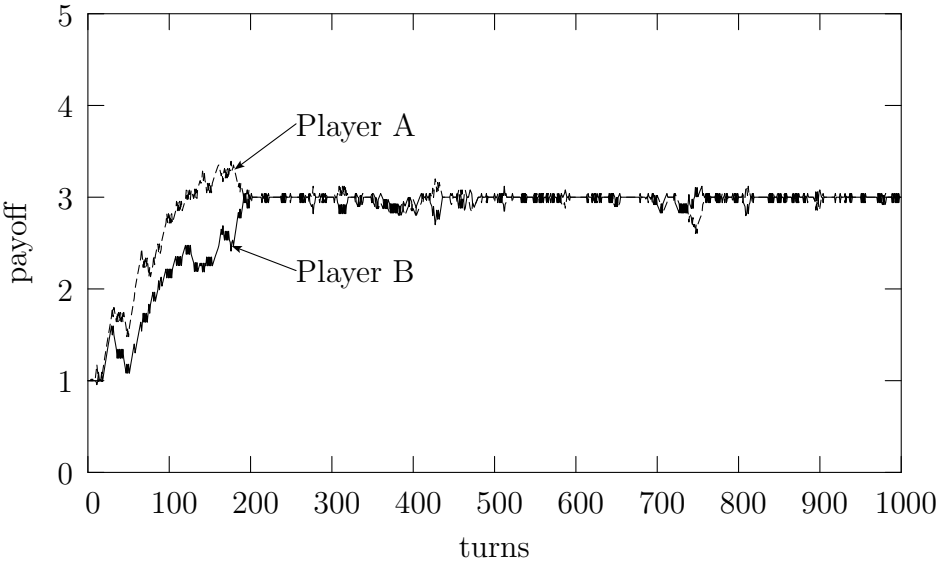


Figure 3.2: An experimental result of IPD, HCE vs HCE

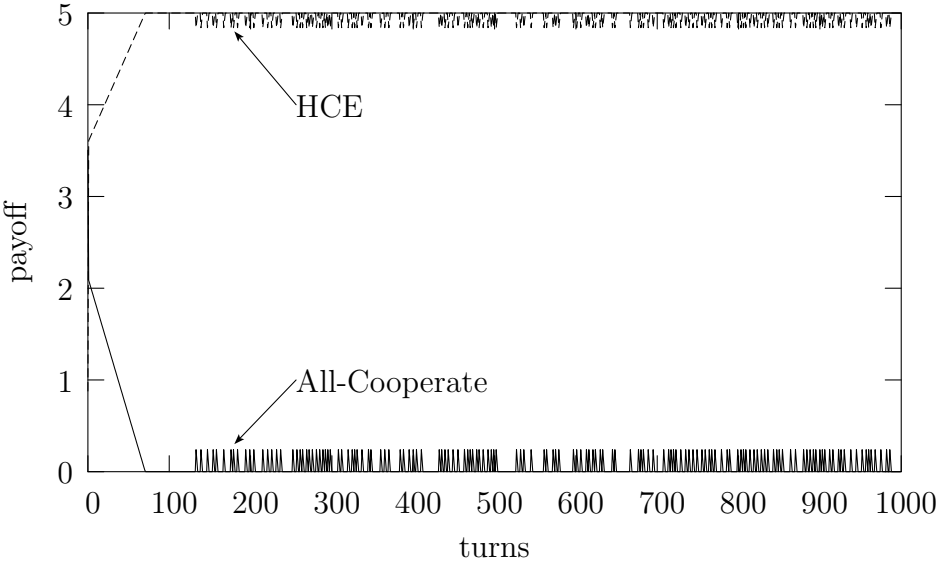


Figure 3.3: An experimental result of IPD, HCE vs All Cooperate

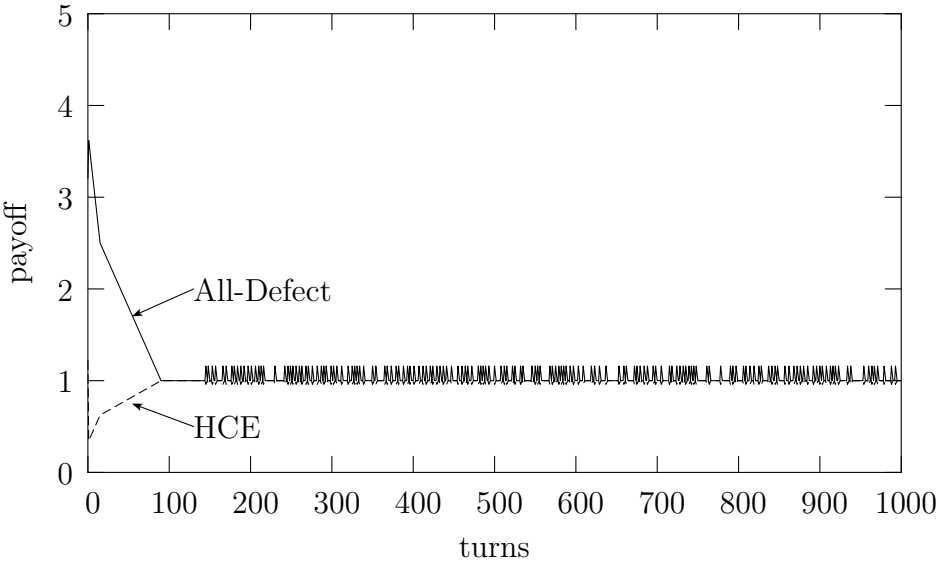


Figure 3.4: An experimental result of IPD, HCE vs All Defect

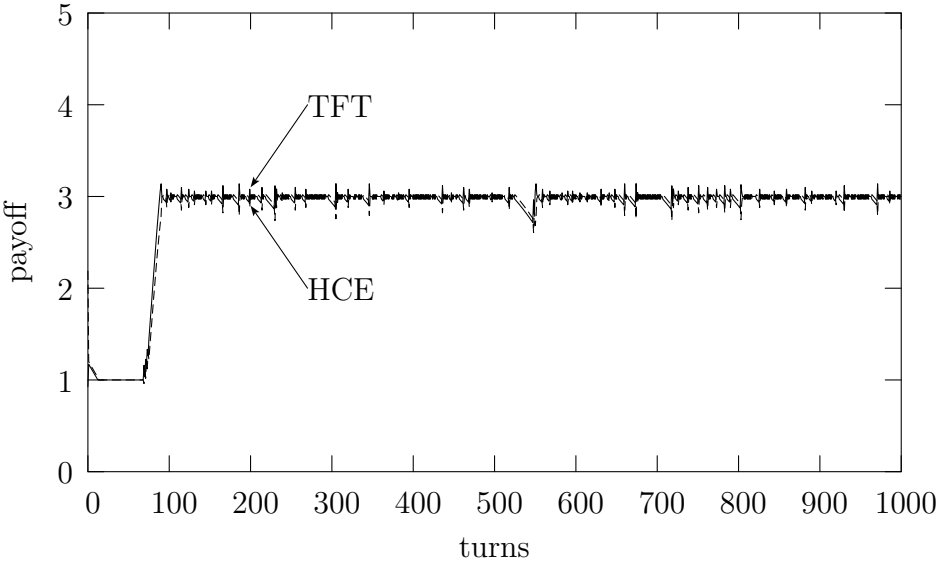


Figure 3.5: An experimental result of IPD, HCE vs TFT

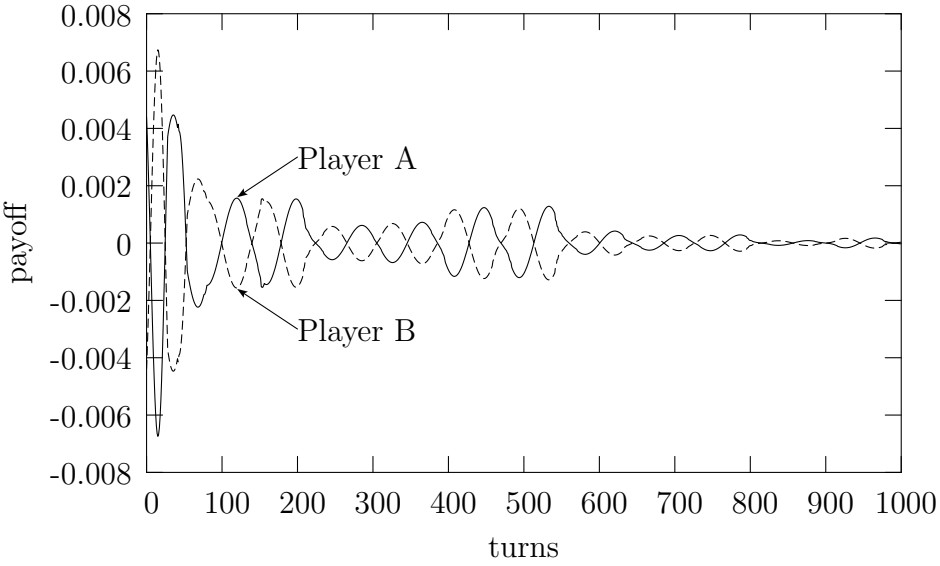


Figure 3.6: An experimental result of matching pennies

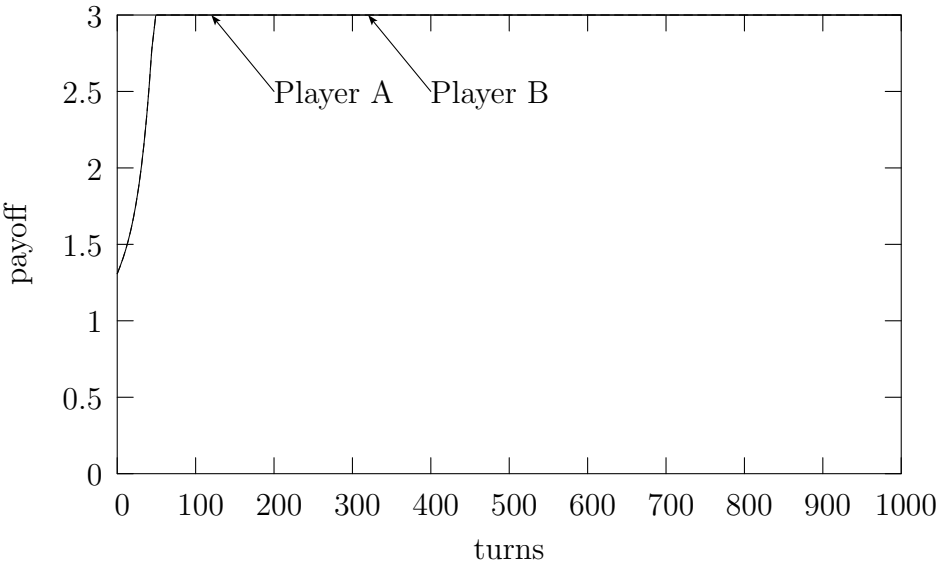


Figure 3.7: An experimental result of coordination

3.4 The HCE Algorithm for Games with Discrete Strategies

In the foresight state, if agents can only observe the pure strategies of their opponents, it has to estimate the mixed strategies for pure ones. Thus, it takes a longer time to collect information. If an agent makes a generous move, it expects its opponent is able to observe the move and to be generous too, otherwise makes a malicious decision. It is the different part from the algorithm for continuous strategy. Meanwhile, the algorithm is the same in the myopia state because the agent believe that its opponent's decision will not change if its changes.

If an agent is unable to observe its opponent's generous mixed strategy, and thus makes an error, the result of the algorithm may be different. However, if the error is constrained by $\lambda\epsilon$ for some λ. We can still treat the error as the percentage that hyper-controller performs N when observing C. If λ is small enough, the PTFT algorithm still guarantee a high percentage of the agents in a foresight state as indicated in Eq 3.6.

In a foresight state, to ensure that the probability of making an error observation is less than $\lambda\epsilon$, the agent can use the following steps:

1. Suppose $\partial R/\partial y > \partial R/\partial x > 0$. Make a move by $\Delta x = -\beta_{explore}\partial R/\partial x = -\nu$. Let $x' = x - \nu$. Then the agent expect to observe that the opponent change its strategy to be some $y' = y + \nu$.

2. Perform pure strategies with the probability of x' until it observes the average of its own observed strategies is less than $x' + \eta$ with a confidence level $1 - \alpha$.

3. Continue to perform pure strategies with the probability of x' and observe the average of its opponents strategies. To ensure that the observed $\hat{y}$ is greater or less than $y' - \eta$ with a confidence level of $1 - \alpha$.

When $\alpha < \lambda\epsilon$, it guarantees that the error is limited by $\lambda\epsilon$, therefore the PTFT algorithm still guarantee a high percentage of the agents in a cooperative state. The detailed algorithm is shown in Algorithm 2.

Algorithm 2 HCE for discrete strategies

procedure *Initialization* ()

$s = N$;

Randomize x, y, μ;

end

procedure *Execute* ()

Perform strategy x.

end

procedure *Update* (x_t, y_t, R_t)

Update payoff function R according to reward R_t

if $s == N$ **then** update μ with R_t **end if**

Update $\hat{x}$ by x_t

Update $\hat{y}$ by y_t

if $\hat{x} == x$ **and** $\hat{y} <> y$ with confidence $1 - \alpha$

then

Update as Algorithm 1. **end if**

end

3.5 Experiments

This section presents the results of two sets of simulations to illustrate the adaptiveness, the noise tolerance and the efficiency of the HCE algorithm.

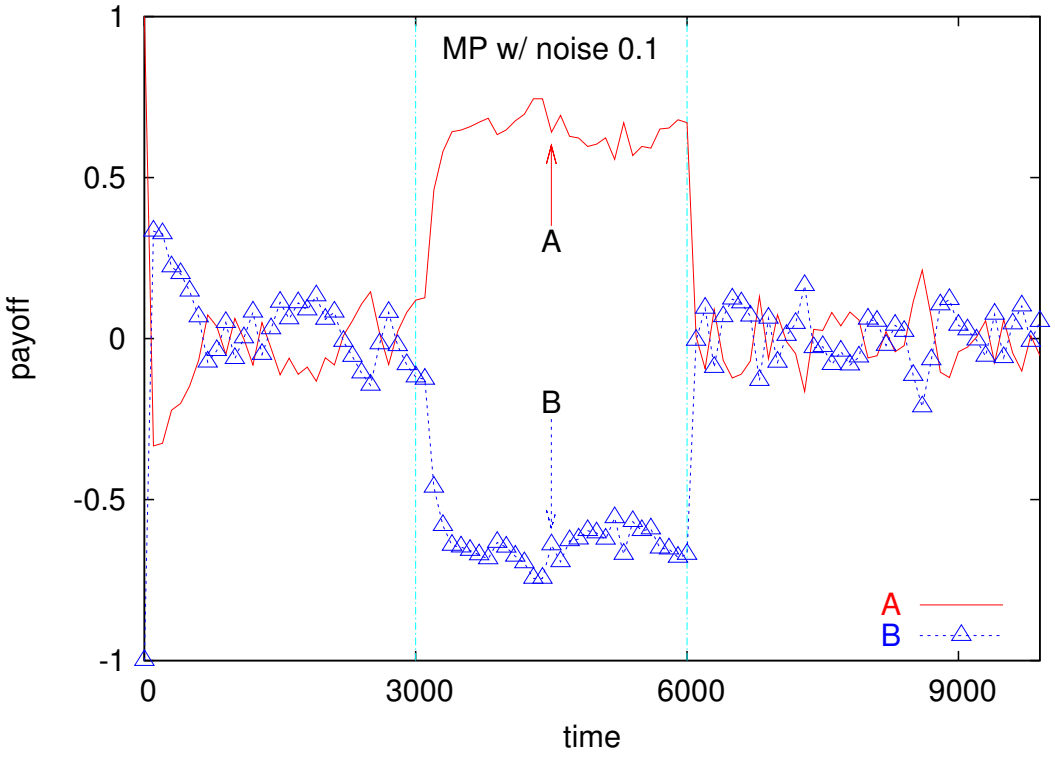


Figure 3.8: Discounted average of payoffs in matching pennies.

To illustrate the adaptiveness of the HCE algorithm, first, two learning agents play against each other for 3000 steps. Next, we exaggerate the non-stationarity of the system by manually changing the behavior of one agent in the following way. We force one player, say B, to play a programmed strategy for another 3000 steps. Finally, we restore player B's learning ability in the remaining steps, and check whether player A still can adapt to the new environment without explicitly knowing the change. Also, to demonstrate the tolerance of noise, we use a noise level η at 0.1 in the experiments, which means that with probability 0.1 a player plays an action other than it intends to.

In the game of matching pennies, the Nash equilibrium is $(0.5, 0.5)$. Figure 3.8 shows the discounted average of payoffs with $\epsilon_t = 0.01$. In the first phase, players soon learn to play the mixed strategy near the Nash equilibrium. The strategy pair oscillate around the equilibrium because the rates of playing action d have a large variance around 0.5. Also because the tile size of the CMAC approximation using in the experiment is 0.25×0.25, the estimation of opponent's response rate is fixed at the equilibrium. In the second phase, we force player B to play action c (*i.e.* heads), and player A learns to take the advantage of that and exploit the weakness of player B. Because of the noise, it achieves a score a little lower than 1. In the third phase, they restore to play the mixed strategy near the Nash equilibrium again. This experiment illustrates that the learning player can learn to play the Nash equilibrium strategy when it plays against an adaptive player, but also can exploit a weak player (in the second phase).

In the game of coordination, there are two Nash equilibria, (c, c) and (d, d). The result of the experiment is shown in Figure 3.9. In the first phase, players soon learn to play an equilibrium (d, d) with a better payoff than the other equilibrium. In the second phase we for player B play action c, then player A has to play action c to coordinate with B. In the third phase, they learn to play the better equilibrium again.

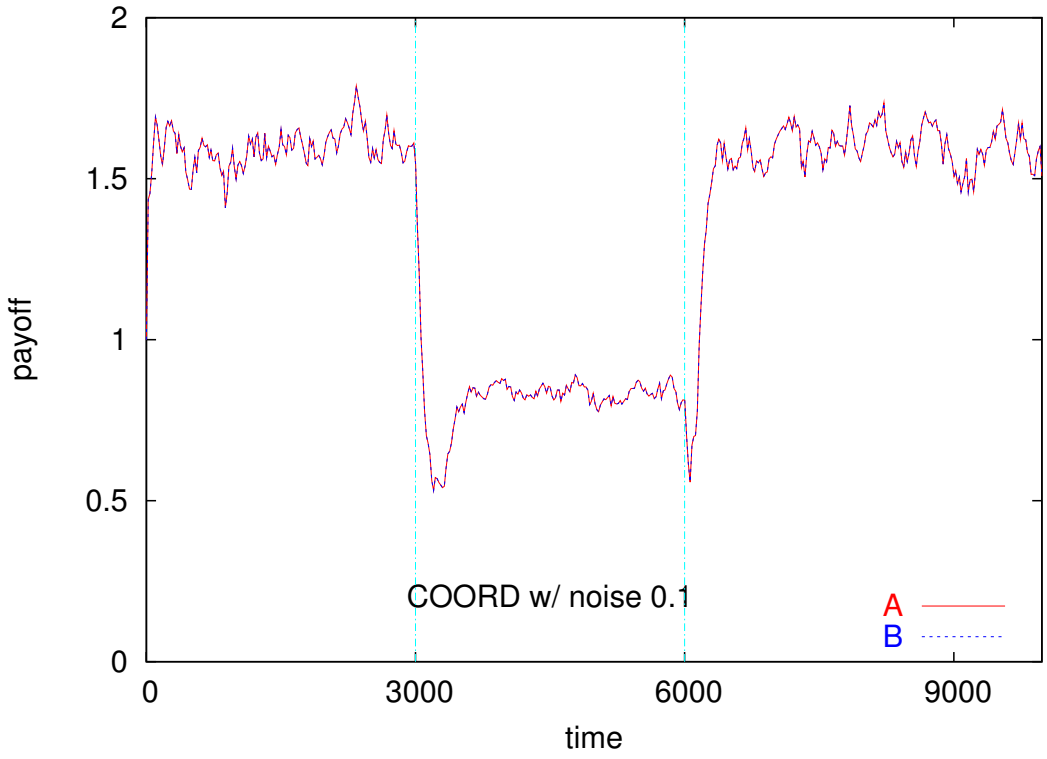


Figure 3.9: Discounted average of payoffs in coordination. The horizontal axis represents time, t. The vertical axis represents payoffs.

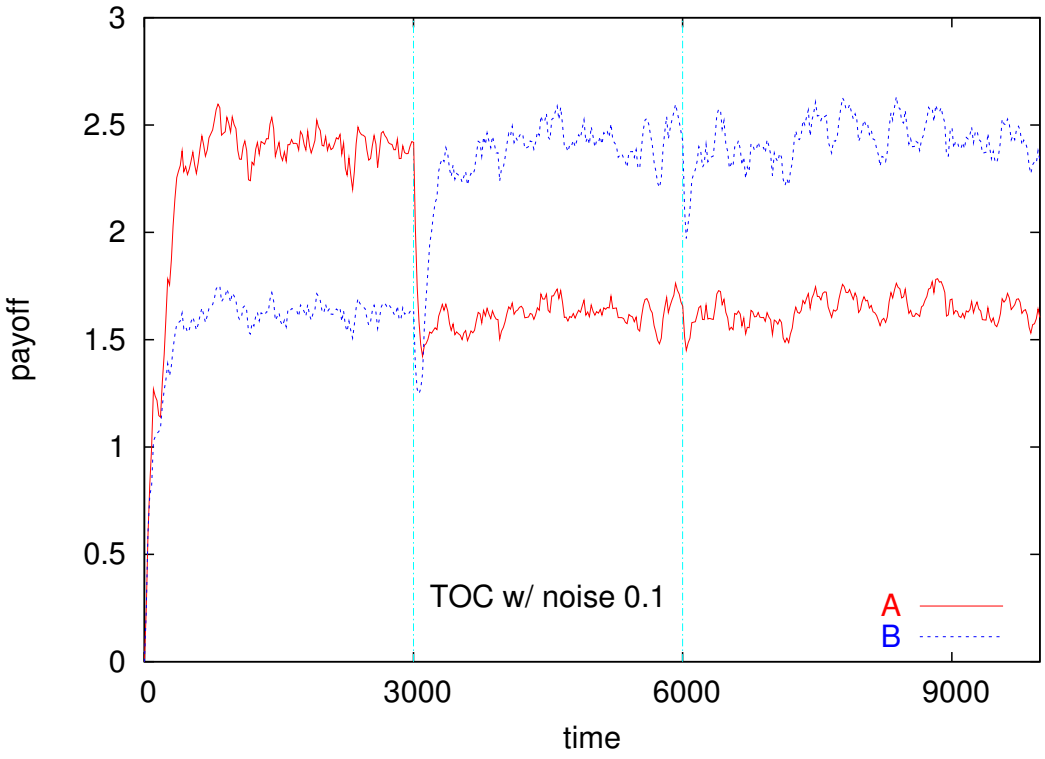


Figure 3.10: Discounted average of payoffs in tragedy of commons. The horizontal axis represents time, t. The vertical axis represents payoffs.

In the game of tragedy of commons, similarly there are two Nash equilibria, (c, d) and (d, c). The result of the experiment is shown in Figure 3.10. In the first phase, players soon learn to play an equilibrium (d, d) with a better payoff than the other equilibrium. In the second phase we force player B to play action d. Then player A has to play action c to achieve the highest reward given B playing action d. In the third phase, their behaviors do not change much, because they are playing strategies at a Pareto-optimal equilibrium.

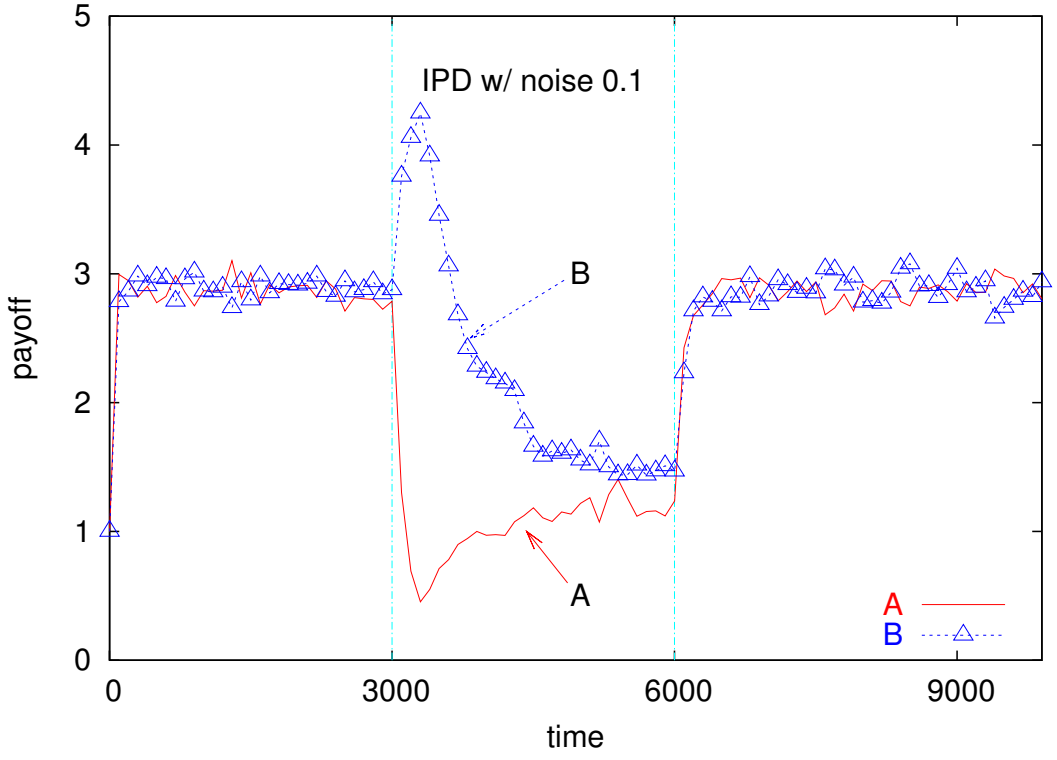


Figure 3.11: Discounted average of payoffs in prisoners' dilemma.

To illustrate the generality of the learning algorithm, we switch the game to prisoners' dilemma. In prisoners' dilemma game, there is one Nash equilibrium, (d, d), but it is not Pareto-optimal. The result of the experiment is shown in Figure 3.11. Players soon learn to play (c, c), because only by playing (c, c) can they receive the maximum average payoff. Then, after 3000 steps, we force player

B to play action d. Though player B exploits player A for a short time, player A soon learns to play action d to achieve the best given B playing action d. After 6000 steps, B is forced to optimize reward using our algorithm, and both players learn to play (c, c) again. This experiment illustrates that the learning agent is able to achieve the optimal solution when playing against another (cooperative) learning agent, and able to adapt its strategy in case that the opponent exploits its cooperative behavior.

A set of experiments compares this algorithm with several well-known strategies in the game of prisoners' dilemma at different noise levels. The result is shown in Table 3.4. The strategies are tit-for-tat, denoted by TFT, with which one plays the last action its opponent played; all defection, denoted by AllD, with which one plays action d; alternative cooperation and defection, denoted by AltCD, with which one plays c and d alternatively. The player using our algorithm is denoted by *HCE* in the experiments. AllD and AltCD receive more overall payoff as the noise level increases, because they are weak players and benefit from the uncertainty of noise. Other players receive less overall payoff as the noise level increases. The overall payoff shows that our approach is the best player in the tournament.

Tit-for-tat (TFT) is known as good strategy in prisoners' dilemma. In a noise-free environment, TFT is good against other players. However, if both players play tit-for-tat, they play every combination of actions with equal probability in a noisy environment. On the other hand, a TFT player will not exploit weak players, such as a player that always cooperates. In contrast, our algorithm is very robust. It receives a good payoff in both noise-free and noisy environments against the same or other strategies and can exploit weak players.

noise level η	0.0	0.1	0.2
AltCD vs AltCD	2.00	2.25	2.25
AltCD vs TFT	2.50	2.41	2.34
AltCD vs AllD	0.50	0.85	1.20
AltCD vs HCE	0.58	0.93	1.42
AltCD overall	1.39	1.61	1.80
AllD vs AltCD	3.00	2.85	2.70
AllD vs TFT	1.00	1.59	2.01
AllD vs AllD	1.00	1.29	1.57
AllD vs HCE	1.17	1.42	1.65
AllD overall	1.54	1.73	1.98
TFT vs AltCD	2.50	2.41	2.34
TFT vs TFT	3.00	2.25	2.25
TFT vs AllD	1.00	1.20	1.42
TFT vs HCE	2.99	3.03	2.96
TFT overall	2.37	2.22	2.24
HCE vs AltCD	2.97	2.82	2.60
HCE vs TFT	2.99	2.63	2.37
HCE vs AllD	0.96	1.26	1.54
HCE vs HCE	2.99	2.74	2.73
HCE overall	2.48	2.43	2.31

Table 3.4: Tournament results. The first column represents different competition between different players. The rest represents the average payoff to the first player in different noise levels, 0.0, 0.1, and 0.2 respectively.

3.6 Conclusions

In this chapter, I proposed the HCE algorithm that learning agents adapt their strategies to their opponents' strategies through statistical analysis of their opponents' responses in iterated games. The simulations show that learning agents can efficiently learn to compete against, cooperate with each other in respective settings and adapt to changes in their opponents' strategies. Also, learning agents are able to tolerate noise in environments and exploit weak opponents.

4 Non-Stationary Environments

To build multi-agent learning algorithm, we resort to reinforcement learning, which is an elegant mathematical framework for studying such tasks because it requires few assumptions (Sutton & Barto, 1998). The only crucial one is associating a utility for each state that reflects the possibility of getting reward by taking actions from that state. After each iteration the agent receives a utility which is used to adjust the decision-making parameters. Most reinforcement learning algorithms are designed for stationary single agent contexts. For simplicity, it is assumed that the environment is stationary, *i.e.*, the utility distribution of each state-action pair does not change. In multi-agent systems, however, the assumption is inapplicable by virtue of the fact that agents are adapting their strategies to maximize the individual achievement or owing to different layouts or utilities at different times.

Sandholm and Crites (1996) did some experiments on the iterated prisoners' dilemma by using Q-learning with Boltzmann exploration . Their experiments demonstrate that playing against another learner was more difficult than playing against a fixed strategy agent because of the peer learner's non-stationary behavior and the average payoff in each single stage game increases monotonically with longer Boltzmann exploration schedules. One reason is that the Boltzmann exploration suffers from inadequate exploration (Sutton, 1990). Although agents

with Boltzmann exploration experiment with different actions on each state, the exploration result is likely to be out-of-date if their peer agents change their own strategies. As a result, such a multi-agent system may be stuck in a local maximum. Another reason is that learning agents will never explore the other strategies to achieve potential benefits after the learning algorithm has converged, which results that each learning agent in a non-stationary system stops adapting itself to the new strategies of its peers (Kaelbling, 1993). Therefore, Boltzmann exploration is not appropriate for multi-agent systems, neither are those exploration methods based stationary environment assumption, such as counter-based exploration (Sato et al., 1990), Gittins' allocation indices (Gittins & Jones, 1974), and, interval estimation (Kaelbling, 1993). Carmel and Markovitch (1999) proposed a model-based exploration for learning agents. However, their learning agent only plays against a random generated opponent, *i.e.*, there is only one learning agent in the circumstance. Another exploration bonus model is proposed by Dayan and Sejnowski (1996). Their exploration model is based on the uncertainty of maze domains.

To deal effectively with the non-stationarity of multi-agent systems, we introduce a *Brownian bandit problem* in Sect.4.1. The Brownian bandit problem formalizes a recency-based exploration bonus (Sutton, 1990). In Sect.4.2, we describe the learning algorithm built on Q-learning algorithm (Watkins & Dayan, 1992) with a smoothed best-response dynamics (Fudenberg & Kreps, 1993). In Sect.4.3, a few simulations demonstrate the efficiency of the recency-based exploration, comparing with Boltzmann exploration.

4.1 The Brownian Bandit Problem

Most exploration models assume that the utility of each state-action pair is stationary, *i.e.*, its distribution does not change. However, in a multi-agent world,

the assumption is inapplicable and that the utility of each state-action pair of the given agent is best modeled as a stochastic process $\{X_t\}$. In this paper, we simplify the model to a Brownian motion, *i.e.*, $X_t - X_{t-1} = Z_t$, where the process $\{Z_t\}$ is white noise with mean 0 and variance σ^2. Emulating the one-armed bandit problem (Thompson, 1933), we introduce the *Brownian bandit problem*, which assumes that the rewards for a slot machine are described by a Brownian motion process X_t, with variance σ^2. Pulling the arm at the iteration t, a agent achieves a reward $R_t = X_t$, otherwise, $R_t = 0$. The problem is how to maximize the discounted sum of rewards, *i.e.*, $\sum \gamma^t R_t$, where the discount factor γ is a real number between 0 and 1.

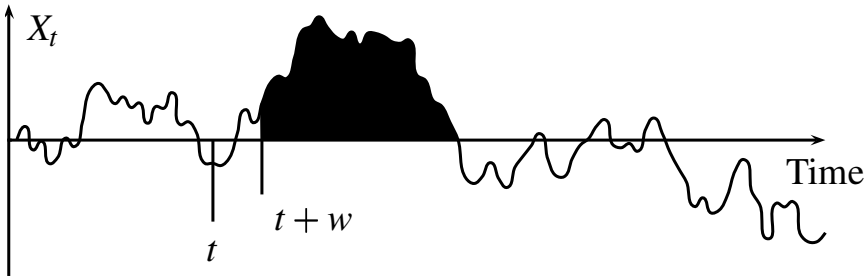


Figure 4.1: Brownian bandit problem. When $X_t = x < 0$, the agent pulls the arm at iteration $t + w$. The rewards in the shadow area pay off the expected value $E(X_{t+w}|X_t) = x < 0$.

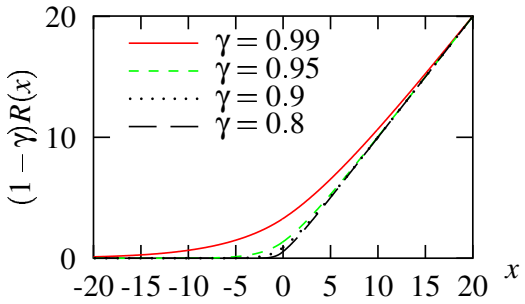
Assume that an agent pulls the arm at the iteration t. Intuitively, the agent should also pull the arm at the next iteration, $t + 1$, if $X_t = x > 0$, because the expected reward of the next iteration is greater than 0, *i.e.*, $E(X_{t+1}|X_t) = x > 0$. If $x < 0$, the next pulling X_{t+w} after waiting for $w > 0$ iterations is also expected to result a negative reward. However, in the case that $X_{t+w} > 0$, the agent expects several positive rewards in subsequent iterations. Therefore, the problem is whether the future positive rewards pay off the expected negative reward of the next pulling(Fig.4.1).

The agent receives a reward $R_t = X_t = x$ at the iteration t. $R(x)$ be the

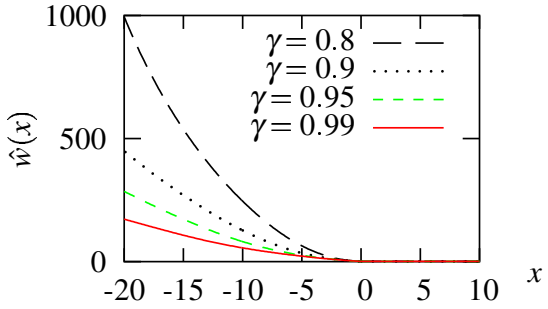
expected discounted sum of the future rewards for the iteration $t' > t$. We have

$$\begin{aligned} R(x) &= \max_w \gamma^w E(X_{t+w} + R(X_{t+w})|X_t) \\ &= \max_w \gamma^w (x + E(R(N(x, w\sigma^2)))) \end{aligned} \tag{4.1}$$

where $N(\mu, w\sigma^2)$ is a Gaussian distribution with mean μ and variance $w\sigma^2$. Let $\hat{w}(x)$ be the maximum point. The decision process can be stated as: pull the arm when $w \geq \hat{w}(x)$. Figure 4.2 shows the numerical solution for $\sigma = 1$ and $\gamma = 0.8, 0.9, 0.95, 0.99$.



(a) $R(x)$



(b) $\hat{w}(x)$

Figure 4.2: $R(x)$ and $\hat{w}(x)$.

Since the decision process is pull the arm when $w \geq \hat{w}(x)$, the bonus of waiting time $\rho = \hat{w}(x)$ is $-x$, *i.e.*, the exploration bonus $b(\rho) = -x = \hat{w}^{-1}(\rho)$. To

compare with Sutton's recency-based exploration bonus $\epsilon\sqrt{\rho}$, let $\lambda(\rho) = \frac{b(\rho)}{\sigma\sqrt{\rho}}$, so that $b(\rho) = \sigma\lambda(\rho)\sqrt{\rho}$. Figure 4.3 reveals that $\lambda(\rho)$ goes to a constant as ρ goes to infinity. This result is consistent with Sutton's recency-based exploration bonus, when ρ is large. But when ρ is small, $\lambda(\rho)$ is less than the constant, which means that the experimentation is discouraged. Also, the discount rate γ is another important factor in encouraging experimentation. When γ is large, the agent is likely to concern about the future rewards, therefore the experimentation is more encouraged than when γ is small.

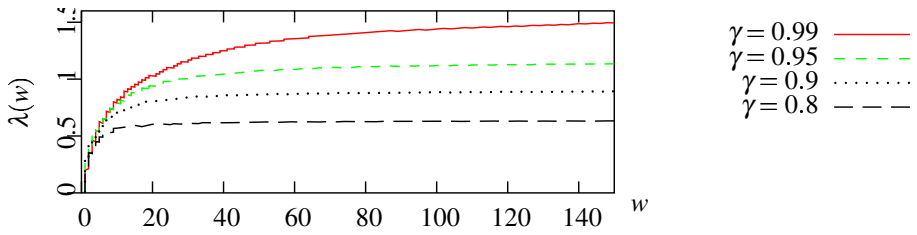


Figure 4.3: $\lambda(\rho) = \frac{b(\rho)}{\sigma\sqrt{\rho}}$

Although the recency-based exploration looks similar to interval estimation (Kaelbling, 1993) and Gittins' allocation indices (Gittins & Jones, 1974), there is a radical difference between them. Because interval estimation and Gittins' allocation indices are based on the assumption of stationary environments, the exploration bonuses decrease as the numbers of observations of state-action pairs increase. When the behavior of the peer agents changes after the learning algorithm has converged, the learning agent will never explore the changed situations. With the recency-based exploration, learning agents are always exploring those recently unvisited situations and ready to adapt to any change of their peer agents.

4.2 Learning in Repeated Games

In a multi-agent environment, the agents make decisions independently, and their joint actions determine the individual utilities and the stochastic state transitions. As each agent may not perceive the internal state of its peers, a partially observable Markov decision process (POMDP) is more applicable for multi-agent systems than a Markov decision process (MDP). However, because of the complexity of POMDP algorithms, in this paper we only use Q-learning algorithm to implement the simulation.

The action with the larger Q value implies the better response. But in some games one-shot simultaneous-move by taking the best response, the action with the largest Q value, may lead to an oscillation. For example, in the game of coordination [1], if two agents play C and D respectively, the best responses are D and C respectively. The one-shot simultaneous-move leads to a *CD-DC* loop. It is known as the Cournot adjustment in game theory. Fudenberg and Kreps (1993) suggests a smoothed best response dynamics, which can be interpreted as probabilistically taking actions according the soft-max values of Q values.

Because of the difference between Q values and the real payoffs, exploration plays an important role in learning process. A learning agent improves the utilities prediction by exploring the environment, and sensing possible changes of its peers' behavior or drift in its environment. On the other hand, exploration increases the cost of learning and also the instability of a multi-agent system. This is known as the dilemma of exploration and exploitation. Most exploration methods assume a single-agent circumstance. Owing to the fact that the environment is stationary, adequate exploration ensures that the agent accurately predicts the expected utilities of each state-action pair. However, in a multi-agent system, the history of experiments of any agent may be out-of-date if its peers change their

[1] In the game of coordination, agents get rewards only if both play the same strategy.

own strategies, which makes these exploration methods inefficient. Assuming that the utilities are stochastic processes, each decision problem raises the same issue as the Brownian bandit problem, therefore we use the recency-based exploration for multi-agent learning.

Unlike the uncertainty in simple maze domains, the non-stationarity of multi-agent systems is mainly owing to the change of the peer agents' behaviors, which are based on the information they perceive. It is reasonable to assume that agents do not update their strategies on the situations that they do not perceive. Therefore, the exploration bonus, $b(s,a)$, only increases on the number of time steps that the action, a, is not tried, given the state, s, is visited, unlike the elapsed time in (Sutton, 1990). This assumption isolates the bonus propagation, and reduces the calculation expense.

Based on the above analysis, the learning algorithm is simply described as running update steps and execute steps alternatively. In an update step, the agent changes the Q values and the mixed strategies of the best responses as follows:

$$\begin{aligned} Q_t(s,a) &= Q_{t-1}(s,a) + \alpha(r_t + \gamma \max_{a' \in A} Q_{t-1}(s',a') - Q_{t-1}(s,a)) \\ Q'_t(s,a) &= Q_t(s,a) + \sigma\lambda(\rho_t(s,a))\sqrt{\rho_t(s,a)} \\ \overline{BR}_t(s,a) &= \frac{e^{Q'_t(s,a)/\theta}}{\sum_{a' \in A} e^{Q'_t(s,a')/\theta}} \end{aligned} \tag{4.2}$$

The second equation shows how the recency-based exploration bonus is incorporated into the Q-value. In the third equation, $\overline{BR}(s,a)$ is the smoothed best response, *i.e.*, the probability that the agent takes action a at state s. α and θ are the learning rate and the smoothing rate, respectively. In an execute step, the agent takes an action based on the probabilities of $\overline{BR}(s,a)$.

4.3 Simulation Results

In the experiment of work-n-shirk, two learning agents plays against each other. The Nash equilibrium of this game is $(\frac{L}{R}, \frac{W-I}{W})$ when $0 < L < R$ and $0 < I < W$. The parameters of the game are $P = 4$,$W = 2$,$I = 1$,$R = 0$,$L = 1$ in the first 500 iterations. In the second 500 iterations, R is changed to 2. The parameters of learning agents are $\alpha = 0.1$, $\sigma = 1$, $\theta = 0.2$.

Figure 4.4(a) and 4.4(b) shows the simulation of two learning agents with the recency-based exploration. Before the change, the Nash equilibrium is at $(1, 0)$. In this simulation, the frequency of *inspect* and *work* between iteration 400 and iteration 500 is $(0.99, 0.01)$. After the change, the new equilibrium is $(0.5, 0.5)$. In this simulation, the frequency of *inspect* and *work* between iteration 900 and iteration 1000 is $(0.50, 0.50)$. It takes the system about 120 iterations to find the new equilibrium after the change. In contrast, agents with Boltzmann exploration fail to adapt to the change of the environment (Fig. 4.4(c) and 4.3).

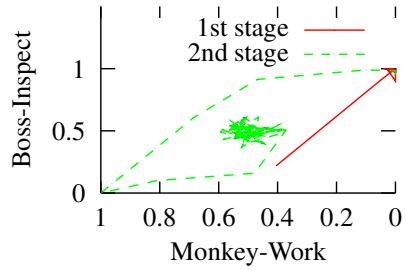
4.4 Conclusions

Multi-agent learning plays an important role in distributed intelligent systems, especially where agents can not be preprogrammed to operate as desired. Reinforcement learning is addressing the problem by dynamically learning knowledge and improving utilities. However, most reinforcement learning algorithms are designed for stationary single agent contexts. In a multi-agent system, learning agents should adapt to gradual improvement of their peers or drift in its environment. For this reason, those explorations assuming a stationary circumstance are unrealistic. To be adaptive, the agents should keep exploring the system in their lifetime to seek possibly better rewards owing to the changes.

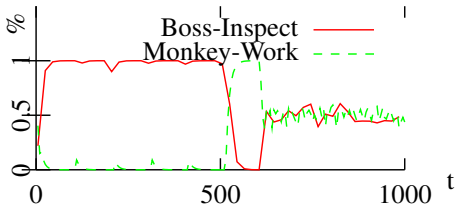
Besides the dilemma of exploration and exploitation in a multi-agent system,

another dilemma is that the exploration of the agents may result the instability of the system. Therefore, the learning agents should understand possible irregular behaviors of their peers and be able to recover from the tragic situation owing to the exploration.

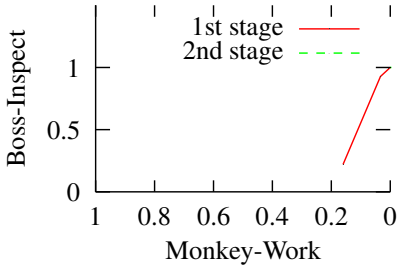
We formalize Sutton's recency-based exploration bonus with the Brownian bandit problem. To illustrate the efficiency of the recency-based exploration, we build Q-learning agents with the smoothed best response dynamics. The simulations demonstrate that systems with the recency-based exploration can reach expected equilibria more efficiently than those with Boltzmann exploration. The simulations also show that the learning agents can efficiently adapt to the new context owing to changes of their peers' behaviors or drift of their environment.



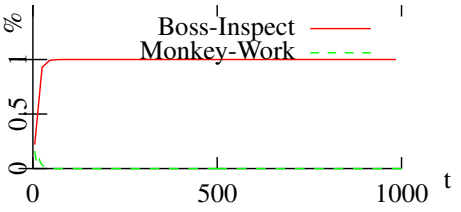
(a) Trace, recency-based expl.



(b) Strategies, recency-based expl.



(c) Trace, Boltzmann expl.



(d) Strategies, Boltzmann expl.

Figure 4.4: Two learning agents play work-n-shirk. The parameters are $P = 4$,$W = 2$,$I = 1$,$R = 0$,$L = 1$ in the first 500 iterations. In the second 500 iterations, R is changed to 2. In 4.4(a) and 4.4(c), the lines represent the trace of the mixed strategies. The solid lines represent the trace before the change, and the dash lines represent the trace after the change. The vertical axes represent the percentage of *inspect* the boss exerts and the horizontal axes represent the percentage of *work* the monkey exerts. The lines represent the mixed strategy. The values represent the discounted (0.95) average strategies over the past iterations. The average strategies for the boss are represented by the solid lines. The average strategies for the monkey are represented by the dash lines.

5 Learning Machine Challenge

The *Learning Machine Challenge*[1] is a round-robin tournament, in which all agents will play each other in a number of different games. The Challenge was announced at The Seventeenth International Joint Conference on Artificial Intelligence.

In the Challenge, a player agent plays an unknown game against an unknown opponent. An agent only knows for sure that the various moves, and they are represented by meaningless symbols. It makes a move by sending one of the symbols it knows about to the outside world. Then it observes one of the symbols it knows about coming in from the outside world and also receives a score from above. Basically, the Challenge shares the same assumption as ours.

Besides the games similar to those mentioned above, the Challenge also consists a few languages games. For example, Predict game, in which agents learn to predict the next character from a corpus; Context game, in which agents learn to emit the right symbols for the current context; *etc.*

[1] http://lmw.a-i.com/

5.1 Design

In this section, we briefly describe our design for learning agents in unknown games.

To build learning agents for multi-player games, we resort to reinforcement learning, which is an elegant mathematical framework for studying such tasks because it requires few assumptions. The only crucial one is associating a utility for each state that reflects the possibility of getting reward by taking actions from that state. After each iteration the agent receives a utility which is used to adjust the decision-making parameters. Most reinforcement learning algorithms are designed for stationary single agent contexts. For simplicity, it is assumed that the environment is stationary, *i.e.*, the utility distribution of each state-action pair does not change. In multi-player games, however, the assumption is inapplicable by virtue of the fact that agents are adapting their strategies to maximize the individual achievement or owing to different layouts or utilities at different times.

Some simple two-player games have no complicated situations such as the prisoners' dilemma. Single state solution works more efficient for such games. However, it is important for a learning algorithm to forecast the distribution of opponents' actions and estimate the expected reward of each action and take the best action accordingly.

Suppose that the opponent plays an action, $X_t = a$, in set A at each time t. A discrete data stream $X_t \in A$ is independently generated from a binomial distribution with mean of $p_t(a)$. In this paragraph, we abbreviate it and denote it as p_t. The subtask for a learning agent is to forecast p_t. The estimation can simply be the moving average of last n events, which can be formalized as $\hat{p}_{t+1} = \frac{1}{n}\sum_{i=0}^{n} X_{t-i}$. The estimation is unbiased.However, this moving average estimation requires saving a queue of rewards of last n events. A discounted estimation can avoid this as $\hat{p}_{t+1} = (1-\alpha)\sum_{i=0}^{\infty} \alpha^i X_{t-i}$, where α is discount

factor. This estimation is also unbiased. A Bayesian forecasting can also be used for such forecasting.

The agent estimates the reward of each situation according to Eq. 5.1.

$$R_{t+1}(a_t, a'_t) = R_t(a_t, a'_t) + \beta(r_t - R_t(a_t, a'_t)) \quad (5.1)$$

$R(a, a')$ is the expected reward to the player if he plays a and the opponent plays a'; r_t is the reward to the player at time t; a_t is what the player plays at time t; a'_t is what the opponent plays at time t;

Based on $p_t(a')$ and $R_t(a, a')$, the expected reward of playing a, $Q(a)$, is estimated as Equation 5.2.

$$Q_t(a) = \sum_{a'} p_t(a') R_t(a, a') \quad (5.2)$$

To make a better prediction of rewards, agents need to know the exact situation they are in. In most cases, using a large state space helps eventually, because a larger state space is used. Each state indicates the more specific situation, which helps predict rewards more accurately. However, in a same number of trials, on average, each state has fewer samples in a larger space, which results less accurate in early stages. It is known as the data sparse problem. Therefore, we have to make tradeoff between accuracy and learning speed. In continuous state/action spaces, such a problem also exists. Many approaches, such as CMACs (Sutton, 1996), case-based RL (Santamaría et al., 1998), kernel-cased RL (Ormoneit & Sen, 2002), variable resolution (Munos & Moore, 1999), have been studied. They are all based on an assumption of the similarity between a state and its neighbors owing to the continuous value function. In CMACs, the reward of a certain point is approximated by the average of the rewards of states (tiles) that cover the point. Inspired by (Jacobs et al., 1991; Jordan & Jacobs, 1992), we estimate the reward of a certain state by a weighted average of the rewards of meta-states that include the state.

A state can be represented by an observation history. If the history length is, for example, 4, then a state is represented as $CDCC$ in the prisoners' dilemma, which means in the last round, the agent played C and observed D, and played C and observed C in the one before. A *meta-state* is a set of states, that share some same features (history). In this paper, we use a wildcarded string to represent it, for instance, $*D*C$, which means all states that the agent observed D and C in the last two rounds, therefore $*D*C$ includes $CDCC$. The Q-values of meta-states are updated in the same manner as those of the regular states.

We consider $s \rightarrow \pi(s)$ as a rule. There are no conflicts between rules before meta-states are introduced. Now the (meta-)states $*D*C$, $C**$, $CDCC$, *etc.*, all match observations $CDCC$ and the respective rules may conflict. A regular state rule, $CDCC \rightarrow \pi(CDCC)$, is more specific than a meta-state rule, $*D*C \rightarrow \pi(*D*C)$. In a long-run stationary environment, it eventually predicts the expected rewards more precisely than the meta-state one. However, in a non-stationary environment, because other players change their policies, or in a short-run game, respect to the size of state space, some general meta-state rules may be more efficient. In this paper, we use the mixture of Q-values of states applied, instead of picking one using some hard rules.

The method of averaging all Q-values without weights is not suitable, because some rules are more applicable than others. In the on-line algorithm of (Jordan & Jacobs, 1992), the inverses of Hessian matrices are used to mix Q-values. However, the computation of Hessian matrices of all states is costly. In this approach, we approximate them with the inverses of diagonal matrices of variances, that is, the reward of a state is approximated by the average of Q-values of all applicable (meta-)states with weights of inverses of their variances.

5.2 Results

Ninety-four users entered the Challenge. Total 196 players were submitted. The player received the third place in the Challenge. The detailed results as the follows.

The contest games are Lang-FCC, Lang-MCC, Lang-SCG, Lang-MCG, Lang-MCGe, Lang-MCCc and Lang-MCCcev.

Lang-FCC is Fixed Context Classification. It is a simple context extension to the already published classification game. The purpose of the game is to classify a space of strings (words) into 3 classes, and respond to every particular input string with a member of the appropriate class: $A- > D$, $B- > E$, and $C- > F$. A, B and C are members of respective classes, each containing 1 to 5 symbols. D, E, and F are members of respective classes, each containing 5 symbols.

Lang-MCC is Multiple Context Classification. Like FCC, this game also involves classifying a space of strings (randomly generated "words") into 2 classes, and respond to every particular input string with a member of the appropriate output class which has not been used recently. Only this time, the context alternates between 2 contexts, 'X' and 'Y': If preceded by a symbol of class 'X', $A- > B$ and $C- > D$. But if preceded by a symbol of class 'Y', then $A- > D$ and $C- > B$. As before, The strings A, B, C, D, X and Y are members of respective classes, each containing 1 to 5 members.

Lang-SCG is Single Class Generalization. This game tests the ability of the player to classify with a single context variation. Like in the previous game, some symbols are class-members.

An input stream may be one of to forms:

```
[p-quantifier] [noun] [plural] [s-quantifier] -->
        correct response: [noun] [singular].
Or:
```

```
[s-quantifier] [noun] [p-quantifier] -->
        correct response: [noun] [plural].
```

For example:

```
[p-quantifier] = {"two", "three" "four"...}.
[s-quantifier] = {"one"}.
[noun] = {"dog", "cat", "mouse"}.
[plural] ={"s"}.
[singular] = {" "}.
```

The input sequence "one"+"dog"+"two" expects the response "dog"+"s". The input sequence "three"+"cat"+"s"+"one" expects the response "cat"+" ". After the first 1000 rounds (with examples) the [noun] set is augmented (new nouns introduced) and the game continues in the same manner, except that when the new nouns are used, no examples are provided.

Lang-MCG is Multiple Class Generalization. The game is an extension of the Lang-SCG game. It differs from the base game in the following aspects:

- The noun set alternates between six different classes,
- p-quantifier and s-quantifier, plural and singular are all classes,
- No examples are given - ever.

Lang-MCGe Multiple Class Generalization, with Examples. The game is an extension of the Lang-SCG game. It differs from the base game in the following aspects:

- The noun set alternates between six different classes,
- p-quantifier and s-quantifier, plural and singular are all classes,

- Examples are always given.

Lang-MCCc is MCC in Character Mode. This game is identical to the already published MCC, with the following complications:

- The game is in character mode. The basic symbol-set is ASCII characters, concatenated to form the "higher-level" symbol set of the MCC game. Each MCC symbol is therefore "taken apart" when emitted, and reconstructed back when retrieved from the player. This mechanism allows the players to construct "chunks", which may be close (but not identical) to the correct response. A similarity measure is applied, with a score proportional to the similarity of the response to the correct one.
- The word classes are switched (alternately) between two sets every 100 moves.

Lang-MCCcev is Character Mode w/ Examples and Variations. This game is identical to the already published MCC, with the following complications:

- The game is in character mode. The basic symbol-set is ASCII characters, concatenated to form the "higher-level" symbol set of the MCC game. Each MCC symbol is therefore "taken apart" when emitted, and reconstructed back when retrieved from the player. This mechanism allows the players to construct "chunks", which may be close (but not identical) to the correct response. A similarity measure is applied, with a score proportional to the similarity of the response to the correct one.
- The word classes are switched (alternately) between two sets every 100 moves.
- Scores include positive consideration for broad usage of the response class (variations)

The results are shown on http://lmw.a-i.com. My player achieved pretty good results, 5th in Lang-FCC, 6th in Lang-MCC, 1st in Lang-SCG, 4th in Lang-MCG and 4th Lang-MCGe. In Lang-MCCc and Lang-MCCcev, it worked fine at the beginning. Unfortunately, due to a certain bug, it failed after a thousand steps. Overall, The player received the third place in the Challenge.

6 Clustering Game

In this chapter, we discuss the clustering problem from the game point of view. Basically, clustering problem can be abstracted as a *the tragedy of the commons* problem. Based on this view, I extend our previous work on clustering.

Clustering problems arise in many disciplines and have a wide range of applications. The problem of clustering has been studied extensively in the database (Zhang et al., 1997; Guha et al., 1998), statistics (Berger & Rigoutsos, 1991; Brito et al., 1997) and machine learning communities (Cheeseman et al., 1988; Fisher, 1995) with different approaches and different focuses.

The clustering problem can be described as follows: let W be a set of n multi-dimensional data points, we want to find a partition of W into clusters such that the points within each cluster are "similar" to each other. Various distance functions have been widely used to define the measure of similarity.

Most clustering algorithms do not work efficiently in high dimensional spaces due to the *curse of dimensionality*. It has been shown that in a high dimensional space, the distance between every pair of points is almost the same for a wide variety of data distributions and distance functions (Beyer et al., 1999). Many feature selection techniques have been applied to reduce the dimensionality of the space (Kohavi & Sommerfield, 1995). However, as demonstrated in (Aggarwal

et al., 1999), the correlations in the dimensions are often specific to data locality; in other words, some data points are correlated with a given set of features and others are correlated with respect to different features.

As pointed out in (Hastie et al., 2001), all methods that overcome the dimensionality problems have an associated and often implicit or adaptive-metric for measuring neighborhoods. *CoFD*, as you will see in the following sections, is based on an adaptive metric.

The clustering problem aims at identifying the distribution of patterns and intrinsic correlations in large data sets by partitioning the data points into similarity clusters. If we use agents to represent clusters, each agent aims to collect similarity data points. Therefore the character of an agent can be depicted by the data points it collected. The problem can be restated as that each agent aims to collect data points similar to its character. The clustering problem is, thus, restated as how agents choose their characters, which are the strategies in the term of games. In *the tragedy of the commons* problem, agents are trying to make different strategies as possible to allocate more resource. In the restated clustering problem, agents aims to choose their characters to collect more points.

First, I present *CoFD*[1], a non-distance based algorithm for clustering in high dimensional spaces in Section 6.1. This part was originally published in (Zhu & Li, 2002). The main idea of *CoFD* is as follows: given a data set W and the feature set S, we want to cluster the data set into K clusters. Based on the maximum likelihood principle, *CoFD* is to optimize parameters to maximize the likelihood between the data set W and the model generated by the parameters. The parameters in the model are the data map D and the feature map F, which are functions from W and S to $\{0, C_1, C_2, \cdots, C_K\}$[2], respectively. Then several approximation methods are applied to iteratively optimize D and F. *CoFD* can

[1] *CoFD* is the abbreviation of Co-training between Feature maps and Data maps.

[2] The number 0 represents the outlier set. $C_1, \cdots, C_K$ represent K different clusters.

also be easily adapted to estimate the number of clusters instead of using K as an input parameter. In addition, interpretable descriptions of the resulting clusters can be generated by the algorithm since it produces an explicit feature map.

Section 6.2, a distributed version of the algorithm based on a game problem is presented.

Section 6.3 shows our experimental results on both the synthetic data sets and a real data set; section 6.4 surveys the related work; finally our conclusions and directions for future research are presented in section 6.5.

6.1 A Clustering Algorithm: *CoFD*

In this section, we introduce the core idea and present the details of *CoFD*. We first introduce *CoFD* for binary data sets.

Consider the zoo data set, where most animals are in the cluster of "chicken", "crow" and "dove", have "two legs". Intuitively, when given an animal with "two legs", we would say it has a large chance of being in the cluster. Therefore, we regard the feature "two legs" as a *positive* (characteristic) feature of the cluster.

Based on the above observation, we want to find out whether each feature is a positive feature of some cluster or not. *Feature map* F is defined as a function from feature set S to $\{0, C_1, \cdots, C_K\}$. $F(j) = k(k > 0)$ if feature j is a positive feature of cluster k. $F(j) = 0$ if feature j is not a positive feature of any cluster, *i.e.*, an outlier feature. Similarly, *data map* D is a function from data set W to $\{0, C_1, \cdots, C_K\}$. $D(i) = C_k(k > 0)$ if data point i is an instance of cluster k. $D(i) = 0$ if data point i is an outlier.

Let N be the total number of data points, and d be the number of features. W can be represented as a data-feature matrix. By assuming all features are binary, if W_{ij} is 1, feature j is said *active* in data point i. Also, data point i is said to be an *active* data point of feature j if W_{ij} is 1.

The motivating idea behind *CoFD* is maximum likelihood principle, that is, to find data map $\hat{D}$ and its correspondent feature map $\hat{F}$ from which the data-feature matrix W is most likely generated. Let $\hat{W}(D, F)$ be the model generated from the data map D and the feature map F. The values of $\hat{W}_{ij}$ is interpreted as the consistence of $D(i)$ and $F(j)$. We only consider the cases that $D(i)$ is consistent with $F(j)$, inconsistent with $F(j)$, or an outlier. $P(W_{ij}|\hat{W}_{i,j}(D, F))$ is the probability whether the feature j of data point i is active in the real data, given the feature j of data point i is active (or not) in the ideal model of maps D and F. $P(W_{ij}|\hat{W}_{i,j}(D, F))$ has the same value for different i's and j's if W_{ij} and $\hat{W}_{i,j}(D, F)$ have the same values, denoted by w and $\hat{w}(D, F)$ respectively. Therefore, we count the number of instances of w and $\hat{w}(D, F)$, which is $dNP(w, \hat{w}(D, F))$. Now we have Eq. (6.1).

$$\begin{aligned}
\log L(D, F) &= \log \prod_{i,j} P(W_{ij}|\hat{W}_{i,j}(D, F)) \\
&= \log \prod_{w,\hat{w}} P(w|\hat{w}(D, F))^{dNP(w,\hat{w}(D,F))} \\
&= dN \sum_{w,\hat{w}} P(w, \hat{w}(D, F)) \log P(w|\hat{w}(D, F)) \\
&\equiv -dNH(W|\hat{W}(D, F)) \qquad (6.1) \\
\hat{D}, \hat{F} &= \arg\max_{D,F} \log L(D, F) \qquad (6.2)
\end{aligned}$$

We apply the hill-climbing method to maximize $\log L(D, F)$, *i.e.*, alternatively optimizing one of D and F by fixing the other. First, we try to optimize F by fixing D. The problem of optimizing F over all data-feature pairs can be approximately decomposed into subproblems of optimizing each $F(j)$ over all data-feature pairs of feature j, *i.e.*, minimizing the conditional entropy $H(W_{\cdot j}|\hat{W}_{\cdot j}(D, F))$. If D and $F(j)$ are given, the entropies can be directly estimated. Therefore, $F(j)$ is assigned to the cluster, given which the conditional entropy of $W_{\cdot j}$ is minimum. To optimize D by fixing F is a dual problem of the problem above. Hence, we only

need minimize $H(W_{i\cdot}|\hat{W}_{i\cdot}(D,F))$ for each i. A straightforward approximation method to minimize $H(W_{\cdot j}|\hat{W}_{\cdot j}(D,F))$ is to assign $F(j)$ to $\arg\max_k |\{i|W_{ij} = k\}|$. This method also applies to minimize $H(W_{i\cdot}|\hat{W}_{i\cdot}(D,F))$ by assigning $D(i)$ to $\arg\max_k |\{j|W_{ij} = k\}|$.

6.1.1 Algorithm Description

There are two auxiliary procedures in *CoFD*: EstimateFeatureMap and EstimateDataMap. EstimateFeatureMap is to estimate the feature map from the data map. For each feature j, the procedure is to find a cluster k which minimizes conditional entropy $H(W_{\cdot j}|\hat{W}_{\cdot j}(D,F))$. If the feature is almost equally active in more than one cluster, the feature is said to be an outlier feature. EstimateDataMap is to estimate the data map from the feature map. It is a dual procedure of EstimateDataMap. For each data point i, the procedure is to find a cluster k which minimizes conditional entropy $H(W_{i\cdot}|\hat{W}_{i\cdot}(D,F))$.

The clustering algorithm proceeds in two steps. The first step is to select seed data points. *CoFD* randomly draws nK distinct data points from the set of data points, and assigns each n of them to a cluster, where n is a small integer number, for example, 5. *CoFD* estimates the feature map from the seed points by applying procedure EstimateFeatureMap.

The second step is an iterative routine. The aim is to find a best cluster by an iterative process similar to EM algorithm. In this step, *CoFD* iteratively estimates the data and feature maps based on previous estimations, until no more change occurs in the feature map. The pseudo code of *CoFD* is shown in Figure 4.

However, computing entropies is still time consuming. Here, we propose an approximating version of EstimateFeatureMap and EstimateDataMap. In EstimateFeatureMap, we calculate the cardinality of the set $\{i|D(i) = C \wedge W_{ij} = 1\}$, *i.e.*, the number of data points in cluster C whose j-th feature is active, and the cardinal-

Algorithm 3 Clustering algorithm

Procedure EstimateFeatureMap(data points: W, data map: D)

begin

for j **in** $1 \cdots d$ **do**

if $\min_C H(W_{\cdot j}|\hat{W}_{\cdot j}(D,C))/H(W_{\cdot j}) \ll 1$ **then**

$F(j) = \arg\min_C H(W_{\cdot j}|\hat{W}_{\cdot j}(D,C))$;

else

$F(j) =$ outlier;

endif

end

return F

end

Procedure EstimateDataMap(data points: W, feature map:F)

begin

for i **in** $1 \cdots N$ **do**

if $\min_C H(W_{i\cdot}|\hat{W}_{i\cdot}(C,F))/H(W_{i\cdot}) \ll 1$ **then**

$D(i) = \arg\min_C H(W_{i\cdot}|\hat{W}_{i\cdot}(C,F))$;

else

$D(i) =$ outlier;

endif

end

return D;

end

Algorithm 4 Clustering algorithm (cont.)

Algorithm CoFD(data points: W, the number of clusters: K)
begin
  let $W1$ be the set of randomly chosen nK
    distinct data points from W;
  assign each n of them to one cluster,
    say the map be $D1$;
  assign EstimateFeatureMap($W1$,$D1$) to F;
  repeat
    assign F to $F1$;
    assign EstimateDataMap(W,F) to D;
    assign EstimateFeatureMap(W,D) to F;
  until conditional entropy $H(F|F1)$ is zeros;
  return D;
end

ity of the set $\{i|W_{ij} = 1\}$, *i.e.*, the number of data points whose j-th feature is active, to approximate the conditional entropies. A similar method is also applied to EstimateDataMap.

It can be observed from the pseudo-code description that the time complexities of both procedures EstimateFeatureMap and EstimateDataMap are $O(K \times N \times d)$. The number of iterations in algorithm CoFD is not related to N or d.

6.1.2 An Example

To illustrate *CoFD*, an example is given below. Suppose we have a data set as Fig. 6.1. Initially, data points 2 and 5 are chosen as seed points. Say, data point 2 is in cluster A, data point 5 in cluster B, *i.e.*, $D(2) = A$, $D(5) = B$. EstimateFeatureMap returns that features a, b and c are positive in cluster A; features e and f are positive in cluster B; features d and g are outliers. That is, $F(a) = F(b) = F(c) = A$, $F(e) = F(f) = B$, $F(d) = F(g) = 0$. After applying EstimateDataMap, data points 1, 2 and 3 are assigned to cluster A; data points 4, 5 and 6 are assigned to cluster B. After applying EstimateFeatureMap, features a, b and c are positive in cluster A; features d, e and f are positive in cluster B; feature g is an outlier. After the next iteration, the result does not change. Therefore, we has a clustering result that cluster A contains data points 1, 2, and 3 and cluster B contains data points 4, 5, and 6.

6.1.3 Informal Description

CoFD presents an effective method for finding clusters in high dimensional spaces without explicit distance functions. The clusters are defined as the group of points that have many features in common. *CoFD* iteratively selects features with high accuracy and assigns data points into clusters based on the selected features. A feature f with high accuracy means that there is a "large" subset V of data set W

feature / data point	a	b	c	d	e	f	g
1	1	1	0	0	1	0	0
2	1	1	1	1	0	0	1
3	1	0	1	0	0	0	0
4	0	1	0	0	1	1	0
5	0	0	0	1	1	1	1
6	0	0	0	1	0	1	0

Table 6.1: An example of data set

such that f is present in most data points of set V. In other words, feature f has a small variance on set V. A feature f with low accuracy means that there is no such a "large" subset V of data set W on which feature f has a small variance. That is, f spreads largely within data set W. Hence our algorithm repeatedly projects the data points to the subspaces defined by the selected features of each cluster, assigns them to the clusters based on the projection, recalculate the accuracies of the features, then selects the features. As the process moves on, the selected features tend to converge to the set of features which have small variances among all the features. For a rigorous proof of convergence in EM-type algorithms, please refer to (Ambroise & Govaert, 1998; Selim & Ismail, 1984).

Input / Output	A	B	C	D	E
1	0	0	0	83	0
2	0	0	0	0	81
3	0	0	75	0	0
4	86	0	0	0	0
5	0	75	0	0	0

Table 6.2: Confusion matrix for Experiment 1

6.2 Agents for Distributed *CoFD*

In the agent's algorithm for distributed *CoFD*, we use agents to represent clusters, and each agent aims to collect similarity data points. Therefore the character of an agent can be depicted by the data points it collected.

Initially, each agent possesses a portion of points and features, and also a small amount of currency. The procedure is that agents trade goods (features and points alternatively) with the currency. The price is set at the buyers' estimation. Of course, the seller would sell at the highest price, which should be higher than the seller's estimation. An agent measure a point with the number of the active features that the agent holds, and measure a feature with the number of the points that the agent holds and the feature is active. After a period of trading, the currency inflates. The system will be stable as no agent can afford any goods.

The detailed algorithm is shown in Algorithm 5.

6.3 Experimental Results

There are many ways to measure how accurately the algorithm performs. One is the *confusion matrix* which is described in (Aggarwal et al., 1999). Entry (o, i) of a confusion matrix is the number of data points assigned to output cluster o and generated from input cluster i.

For input map I, which maps data points to input clusters, the entropy $H(I)$ measures the information of the input map. The task of clustering is to find out an output map O which recover the information. Therefore, the condition entropy $H(I|O)$ is interpreted as the information of the input map given the output map O, *i.e.*, the portion of information which is not recovered by the clustering algorithm. Therefore, the *recovering rate* of a clustering algorithm

Algorithm 5 Agent's algorithm for Distributed *CoFD*

Procedure Init()

begin

$w = w_0$

Randomly assign some features and points to the agent

end

Procedure FeatureTransaction()

begin

Set the price of feature j at $p_j = ||\{i|W_ij = 1\}||$;

Sell feature j at price p'_j if $p'_j > p_j$

Buy feature j at price p_j if $p_j < w$

Update w

end

Procedure PointTransaction()

begin

Set the price of point i at $p_i = ||\{j|W_ij = 1\}||$;

Sell point i at price p'_i if $p'_i > p_i$;

Buy point i at price p_i if $p_i < w$;

Update w

end

Procedure Clustering()

begin

repeat

FeatureTransation

PointTransaction

discount w by β

until no transaction happens

end

is defined as $1 - H(I|O)/H(I) = MI(I,O)/H(I)$, where $MI(I,O)$ is mutual information between I and O.

To test the performance of our algorithm, we did three experiments. Two experiments ran on synthetic data sets, and one ran on a real data set. The simulations were performed on a 700MHz Pentium-III IBM Thinkpad T20 computer with 128M of memory, running on octave 2.1.34 [3] on Linux 2.4.10.

6.3.1 Extending to Non-binary Data Sets

In order to handle non-binary data sets, we first translate raw attribute values of data points into binary feature spaces.

If an attribute is categorical, our translation scheme is similar to the method described in (Srikant & Agrawal, 1996). We use as many features as the number of attribute values. The value of a binary feature corresponding to $\langle attribute_1, value_1 \rangle$ would be 1 if $attribute_1$ had $value_1$ in the raw attribute space, and 0 otherwise.

If an attribute is continuous, the method presented in (Srikant & Agrawal, 1996) can also be applied as the translation scheme. However, in the experiments we use Gaussian mixture models to fit each attribute of the original data sets, since most of them are generated from Gaussian mixture models, *i.e.*, each value is generated by one of many Gaussian distributions. The number of Gaussian distributions n can be obtained via maximizing Bayesian information criterion of the mixture model (Schwarz, 1978). Then the value is translated into n feature values in the binary feature space. The j-th feature value is 1 if the probability that the value of the data point is generated from the j-th Gaussian distribution is the largest. Considering that some value is generated from a uniform distribution because of outliers, the attribute value is not mapped into any binary feature.

[3] GNU Octave is a high-level language, primarily intended for numerical computations. The software can be obtained from `http://www.octave.org/`.

6.3.2 A Binary Synthetic Data Set

First we generate a binary synthetic data set to evaluate our algorithm. $N = 400$ points are from $K = 5$ clusters. Each point has $d = 200$ binary features. Each cluster has $l = 35$ positive features, 140 negative features. The positive features of a point have a probability of 0.8 to be 1 and 0.2 to be 0; the negative features of a point have a probability of 0.8 to be 0 and 0.2 to be 1; the rest features have a probability of 0.5 to be 0, and 0.5 to be 1.

Fig. 6.2 shows the confusion matrix of this experiment.

In this experiment, $p(D|1) = 83/83 = 1$, $p(E|2) = 1$, $p(C|3) = 1$, $p(A|4) = 1$, $p(B|5) = 1$, and the rest are zeros. So we have $H(I|O) = \sum p(i|o) \log p(i|o) = 0$, *i.e.* the recovering rate of the algorithm is 1, *i.e.*, all data points are correctly recovered.

6.3.3 A Continuous Synthetic Data Set

The second experiment is to cluster a continuous data set. We use the method described in (Aggarwal et al., 1999) to generate a data set. The data set has $N = 100,000$ data points in a 20-dimensional space, with $K = 5$. All input clusters were generated in some 7-dimensional subspace. And 5% data points were chosen to be outliers, which were distributed uniformly at random throughout the entire space. Using the translation scheme described in section 6.3.1, we map all the data point into a binary space with 41 features.

Then, 1000 data points are randomly chosen as the bootstrap data set. Running the algorithm on the bootstrap data set, we have a clustering result of the bootstrap data set. Using the bootstrap data set as the seed points, we run the algorithm on the entire data set. Fig. 6.3 shows the confusion matrix of this experiment. About 99.65% of data points are recovered. The conditional entropy

$H(I|O)$ is 0.0226 as respect to the input entropy $H(I) = 1.72$. The recovering rate of this algorithm is $1 - H(I|O)/H(I) = 0.987$.

Input / Output	A	B	C	D	E	O.
1	1	0	1	17310	0	12
2	0	15496	0	2	22	139
3	0	0	0	0	24004	1
4	10	0	17425	0	5	43
5	20520	0	0	0	0	6
Outliers	16	17	15	10	48	4897

Table 6.3: Confusion matrix for Experiment 2

Input / Output	1	2	3	4	5	6	7
A	0	0	0	0	0	0	1
B	0	20	0	0	0	0	0
C	39	0	0	0	0	0	0
D	0	0	2	0	0	0	0
E	2	0	1	13	0	0	4
F	0	0	0	0	0	8	5
G	0	0	2	0	3	0	0

Table 6.4: Confusion matrix of Zoo

We made a rough comparison with the result reported in (Aggarwal et al., 1999). Computing from the confusion matrix reported in their paper, their recovering rate is 0.927.

6.3.4 Zoo Database

We also evaluate our algorithm on the zoo database from UCI machine learning repository. The database contains 100 animals, each of which having 15 Boolean attributes and 1 categorical attribute[4].

In our experiment, all Boolean attributes are translated into two features, which are "true" and "false" features of the attributes. The numeric attribute, "legs", is translated into six features, which represents 0, 2, 4, 5, 6,and 8 legs respectively.

Fig. 6.3 shows the confusion matrix of this experiment. The conditional entropy $H(I|O)$ is 0.317 while the input entropy $H(I)$ is 1.64. The recovering rate of this algorithm is $1 - H(I|O)/H(I) = 0.807$.

In the confusion matrix, we found that the clusters with a large number of animals are likely correctly clustered, for example, cluster 1, which contains "aardvark", "antelope", *etc.*, is mapped into cluster C; cluster 2, which contains "chicken", "crow", *etc.*, is mapped into cluster B; cluster 4, which contains "bass", "carp", *etc.*, is mapped into cluster E; cluster 5 ,which contains "frog", "gnat", *etc.*, is mapped into cluster F; and cluster 6 , which contains "flea", "gnat", *etc.*, is mapped into cluster F.

The algorithm comes with an important by-product that the resulting clusters can be easily described in terms of features, since the algorithm produces an explicit feature map. For example, the positive features of cluster A are "no eggs", "no backbone", "venomous" and "8 legs"; the positive features of cluster B are "feather", "airborne", and "2 legs"; the positive features of cluster C are "hair", "milk", "domestic" and "catsize"; the positive feature of cluster D is "0 legs";

[4]In the raw data, there are 101 instances and 18 attributes. But there are two instances of "frog", which are only counted once in our experiment. Also, the attributes "animal name" and "type" are not counted.

the positive feature of cluster E are "aquatic", "no breathes", "fins" and "5 legs"; the positive features of cluster F are "6 legs" and "no tail"; the positive feature of cluster G is "4 legs". Hence, cluster B can be described as animals having feather and 2 legs, and being airborne, which are the representative features of the animals called *birds*. Cluster E can be described as animals being aquatic, having no breathes, having fins an 5 legs.

Animals "dolphin" and "porpoise" are in cluster 1, but were clustered into cluster E, because their attributes "aquatic" and "fins" make them more like animals in cluster E than their attribute "milk" does for cluster C.

6.3.5 Scalability Results

In this subsection we present the computational scalability results for our algorithm. The results were averaged over five runs on each case to eliminate the randomness effect. We report the scalability results in terms of the number of points and the dimensionality of the space.

Number of points: The data sets we tested have 200 features (dimensions). They all contained 5 clusters and each cluster has an average 35 features. Figure 6.1 shows the scalability result of the algorithm in terms of the number of data points. The value of the y coordinate in figure 6.1 is the running time in seconds. We implemented the algorithm using octave 2.1.34 on Linux 2.4.10. If it were implemented in C, the running time could be reduced considerably. Figure 6.1 contains two curves: one is the scalability result with bootstrap and one without. It can be seen from figure 6.1 that with bootstrap, our algorithm scales sublinearly with the number of points and without bootstrap, it scales linearly with the number of points. The value of the y coordinate in figure 6.1 is the ratio of the running time to the number of points. The linear and sublinear scalability properties can also be observed in figure 6.1.

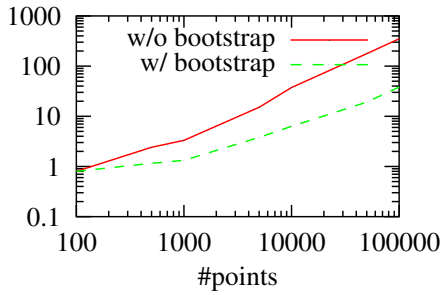


Figure 6.1: Scalability with number of points

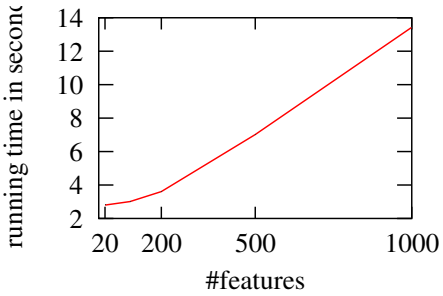


Figure 6.2: Scalability with number of dimensions

Dimensionality of the space: The data sets we tested have $10,000$ data points. They all contained 5 clusters and the average features (dimensions) for each cluster is about $\frac{1}{6}$ of the total number of features (dimensions). Figure 6.2 shows the scalability result of the algorithm in terms of dimensionality of the space. The value of the y coordinate in figure 6.2 is the running time in seconds. It can be seen from figure 6.2 that our algorithm scales linearly with the dimensionality of the space.

6.4 Relative Work

Traditional clustering techniques can be broadly classified into partitional clustering, hierarchical clustering, density-based clustering and grid-based clustering (Han et al., 2000). Partitional clustering attempts to directly decompose the data set into K disjoint clusters such that the data points in a cluster are nearer to one another than the data points in other clusters. Hierarchical clustering proceeds successively by building a tree of clusters. It can be viewed as a nested sequence of partitioning. The tree of clusters, often called *dendrogram*, shows the relationship of the clusters and a clustering of the data set can be obtained by cutting the dendrogram at a desired level. Density-based clustering is to group the neighboring points of a data set into clusters based on density conditions. Grid-based clustering quantizes the object space into a finite number of cells that form a grid-structure and then performs clustering on the grid structure. Most of the traditional clustering methods use the distance functions as objective criteria and are not effective in high dimensional spaces.

Next we review some recent clustering algorithms which have been proposed for high dimensional spaces or without distance functions and are largely related to our work.

CLIQUE (Agrawal et al., 1998) is an automatic subspace clustering algorithm

for high dimensional spaces. It uses equal-size cells and cell density to find dense regions in each subspace of a high dimensional space. Cell size and the density threshold need to be provided by the user as inputs. CLIQUE does not produce disjoint clusters and the highest dimensionality of subspace clusters reported is about 10. *CoFD* and the derivative produce disjoint clusters and does not require the additional parameters such as the cell size and density. Also it can find clusters with higher dimensionality.

In (Aggarwal & Yu, 2000; Aggarwal et al., 1999), the authors introduced the concept of projected clustering and developed algorithms for discovering interesting patterns in subspaces of high dimensional spaces. The core idea of their algorithm is a generalization of feature selection which allows the selection of different sets of dimensions for different subsets of the data sets. However, their algorithms are based on the Euclidean distance or Manhattan distance and their feature selection method is a variant of singular value decomposition (SVD). Also their algorithms assume that the number of projected dimensions are given beforehand. *CoFD* and its derivative do not need the distance measures and the number of dimensions for each cluster. Also it does not require all projected clusters to have the same number of dimensions.

Ramkumar and Swami (1998) proposed a new method for clustering without distance function. Their method is based on the principles of co-occurrence maximization and label minimization which are normally implemented using the association rules and classification techniques. Their method does not consider the correlations of dimensions with respect to data locality and critically depend on the input parameters like parameter w and thresholds for association. *CoFD* and its derivative are quite different. We consider the fact that the correlations of dimensions are often specific to data distribution.

Liu et al. (1998) proposed a method to find sparse and data regions using decision tree. The method is a grid-based method and it does not work well for

high dimensional space. It is also hard to decide the size of the grids and the density threshold. In (Liu et al., 2000), a clustering technique based on decision tree construction is presented. The technique first introduces non-existing points to the data space and then performs the decision tree algorithm to partitioning the data space into dense and sparse regions.

Cheng et al. (1999) proposed an entropy-based subspace clustering for mining numerical data. The method is motivated by the fact that a subspace with clusters typically has lower entropy than a subspace without clusters.

Strehl and Ghosh (2000) proposed OPOSSUM, a similarity-based clustering approach based on constrained, weighted graph-partitioning. OPOSSUM is based on *Jaccard Similarity* and is particularly attuned to real-life market baskets, characterized by high-dimensional sparse customer-product matrices.

Fasulo (1999) also gave a detailed survey on clustering approaches based on mixture models (Banfield & Raftery, 1993), dynamical systems (Kleinberg, 1997) and clique graphs (Ben-Dor et al., 1999).

6.5 Conclusions

In this paper, we proposed a novel clustering method which does not require the distance function for high dimensional spaces. The algorithm performs clustering by iteratively optimize the data map and the feature map. We have adopted several approximation methods to maximize the likelihood between the given data set and the generated model. Extensive experiments have been conducted and the results show that *CoFD* and its derivative are both efficient and effective.

Our contributions are:

- We propose *CoFD* and its derivative, a non-distance based clustering algorithm in high dimensional spaces based on maximum likelihood principle.

- We come up with a new perspective of viewing the clustering problem by interpreting it as the dual problem of optimizing the feature map.
- As a by-product, *CoFD* and its derivative also produce the feature map which can provide interpretable descriptions of the resulting clusters.
- *CoFD* and its derivative also provide a method to select the number of clusters based on the conditional entropy.
- We introduce the recovering rate, an accuracy measure of clustering, to measure the performance of clustering algorithms and to compare clustering results of different algorithms.

Our future work includes developing more direct methods to optimize the data map and the feature map, designing the parallel and distributed versions of our algorithm and the incremental clustering algorithm based on our algorithm.

7 Summary

Multi-agent learning plays an important role in distributed intelligent systems, especially where agents can not be preprogrammed to operate as desired. Reinforcement learning is addressing the problem by dynamically learning knowledge and improving utilities. However, most reinforcement learning algorithms are designed for stationary single agent contexts. In a multi-agent system, learning agents should adapt to gradual improvement of their peers or drift in its environment. For this reason, those explorations assuming a stationary circumstance are unrealistic. To be adaptive, the agents should keep exploring the system in their lifetime to seek possibly better rewards owing to the changes.

Besides the dilemma of exploration and exploitation in a multi-agent system, another dilemma is that the exploration of the agents may result the instability of the system. Therefore, the learning agents should understand possible irregular behaviors of their peers and be able to recover from the tragic situation owing to the exploration. It is one reason why the tit-for-tat strategy of prisoners' dilemma fails to be learned as the optimal strategy in a multi-agent system.

In this work, I propose an approach that learning agents adapt their strategies to their opponents' strategies through statistical analysis of their opponents' responses in iterated games. The simulations show that learning agents can effi-

ciently learn to compete against, cooperate with each other in respective settings and adapt to changes in their opponents' strategies. Also, learning agents are able to tolerate noise in environments and exploit weak opponents.

I also formalize Sutton's recency-based exploration bonus with the Brownian bandit problem. To illustrate the efficiency of the recency-based exploration, we build Q-learning agents with the smoothed best response dynamics. The simulations demonstrate that systems with the recency-based exploration can reach expected equilibria more efficiently than those with Boltzmann exploration. The simulations also show that the learning agents can efficiently adapt to the new context owing to changes of their peers' behaviors or drift of their environment.

At last, I propose a novel clustering method which does not require the distance function for high dimensional spaces. The algorithm performs clustering by iteratively optimize the data map and the feature map. We have adopted several approximation methods to maximize the likelihood between the given data set and the generated model. Extensive experiments have been conducted and the results show that *CoFD* and its derivative are both efficient and effective.

Bibliography

Aggarwal, C., & Yu, P. S. (2000). Finiding generalized projected clusters in high dimensional spaces. *SIGMOOD-00*.

Aggarwal, C. C., Wolf, J. L., Yu, P. S., Procopiuc, C., & Park, J. S. (1999). Fast algorithms for projected clustering. *ACM SIGMOD Conference*.

Agrawal, R., Gehrke, J., Gunopulos, D., & Raghavan, P. (1998). Automatic subspace clustering for high dimensional data for data mining applications. *SIGMOD-98*.

Ahmed, E., & Elgazzar, A. S. (2000). On the dynamics of local hawk-dove game. *International Journal of Modern Physics C*, *11*, 607–614.

Ambroise, C., & Govaert, G. (1998). Convergence of an EM-type algorithm for spatial clustering. *Pattern Recognition Letters*, *19*, 919–927.

Astrom, K. J. (1965). Optimal control of markov decision processes with incomplete state estimation. *J. Math. Anal. Appl.*, *10*, 174–205.

Axelrod, R. (1984). *The evolution of cooperation*. New York: Basic Books.

Banfield, J., & Raftery, A. (1993). Model-based gaussian and non-gaussian clustering. *Biometrics*, *49*, 803–821.

Barto, A. G., & Duff, M. O. (1994). Monte Carlo matrix inversion and reinforcement learning. In *Nips '94*, vol. 6. San Mateo, CA: Morgan Kaufman.

Barto, A. G., Sutton, R., & Anderson, C. (1983). Neuron-like elements that can solve difficult learning control problems. *IEEE Trans. on systems, Man, and Cybernetics*, *13*, 834–846.

Bellman, R. (1956). A problem in the sequential design of experiments. *Sankhya*, *16*, 221–9.

Bellman, R., & Kalaba, R. (1959). On adaptive control processes. *IRE Trans.*, *4*, 1–9.

Bellman, R. E. (1957). A markov decision process. *Journal of Mathematical Mech.*, *6*, 679–84.

Bellman, R. E. (1961). *Adaptive control processes: A guided tour*. Princeton University Publisher.

Ben-Dor, A., Shamir, R., & Yakhini, Z. (1999). Clustering gene expression patterns. *J. of Comp. Biology*, *6*, 281–297.

Berger, M., & Rigoutsos, I. (1991). An algorithm for point clustering and grid generation. *IEEE Trans. on Systems, Man and Cybernetics*, *21*, 1278–1286.

Beyer, K., Goldstein, J., Ramakrishnan, R., & Shaft, U. (1999). When is nearest neighbor meaningful? *ICDT Conference*.

Billings, D. (2000). The first international RoShamBo programming competition. *International Computer Games Association Journal*, *23*, 42–50.

Bowling, M., & Veloso, M. (2001). Convergence of gradient dynamics with a variable learning rate. *Proc. 18th International Conf. on Machine Learning* (pp. 27–34). Morgan Kaufmann, San Francisco, CA.

Boyan, J. A. (1992). Modular neural netowrks for learning context-dependent game strategies. Master's thesis, University of Cambridge, Cambridge, England.

Brafman, R. I., & Tennenholtz, M. (2001). R-MAX - a general polynomial time algorithm for near-optimal reinforcement learning. *IJCAI* (pp. 953–958).

Brito, M., Chavez, E., Quiroz, A., & Yukich, J. (1997). Connectivity of the mutual K-Nearest-Neighbor graph for clustering and outlier detection. *Statistics and Probability Letters*, *35*, 33–42.

Bui, H. H., Kieronska, D., & Venkatesh, S. (1996). Learning other agents' preferences in multiagent negotiation. *IAAI* (pp. 114–119).

Carmel, D., & Markovitch, S. (1996). Learning models of intellignet agents. *AAAI-96*.

Carmel, D., & Markovitch, S. (1999). Exploration strategies for model-based learning in multi-agent systems. *Journal of Autonomous Agents and Multi-Agent System*, *2*.

Cheeseman, P., Kelly, J., & Self, M. (1988). AutoClass: A bayesian classification system. *ICML'88*.

Cheng, C.-H., Fu, A. W.-C., & Zhang, Y. (1999). Entropy-based subspace clustering for mining numerical data. *KDD-99*.

Chrisman, L. (1992). Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. *Proceedings of the Tenth National Conference on Artificial Intelligence* (pp. 183–8).

Chrisman, L., Caruana, R., & Carriker, W. (1991). Intelligent agent design issues: Internal agent state and incomplete perception. *AAAI Fall Symposium Series: Sensory Aspects of Robotic Intelligence*.

Claus, C., & Boutillier, C. (1998). The dynamics of reinforcement learning in cooperative multiagent systems. *AAAI '98*.

Crites, R., & Barto, A. (1996). Improving elevator performances using reinforcement learning. In D. Touretzky, M. Mozer and M. Hasselmo (Eds.), *Advances in neural information processing systems*. Cambridge, MA: MIT Press.

Dayan, P., & Hinton, G. E. (1993). Feudal reinforcement learning. *NIPS '93* (pp. 271–278). Morgan Kaufmann.

Dayan, P., & Sejnowski, T. (1996). Exploration bonuses and dual control. *Machine Learning*, *25*, 5–22.

Digney, B. (1996). Emergent hierarchical control structures: Learning reactive/hierarchical relationships in reinforcement environment. *Proceedings of the Fourth International Conference on the Simulation of Adaptive Behavior: From Animals to Animats* (pp. 363–72). Cambridge, MA: MIT Press/Bradford Books.

Fasulo, D. (1999). *An analysis of recent work on clustering algorithms* (Technical Report 01-03-02). U. of Washington, Dept. of Comp. Sci. & Eng.

Fisher, D. H. (1995). *Iterative optimization and simplification of hierarchical clusterings* (Technical Report CS-95-01). Vanderbilt U., Dept. of Comp. Sci.

Fudenberg, D., & Kreps, D. (1993). Learning mixed equilibria. *Games and Economic Behavior*, *5*, 320–67.

Fudenberg, D., & Levine, D. K. (1998). *The theory of learning in games*. Cambridge, MA: The MIT Press.

Gittins, J. C., & Jones, D. M. (1974). A dynamic allocation index for the sequential design of experiments. In *Progress in statistics*, vol. I, 241–66. North-Holland.

Guha, S., Rastogi, R., & Shim, K. (1998). CURE: An efficient clustering algorithm for large database. *Proceedings of the 1998 ACM SIGMOD Conference*.

Han, J., Pei, J., & Yin, Y. (2000). Mining frequent patterns without candidate generation (pp. 1–12.).

Hastie, T., Tibshirani, R., & Friedman, J. (2001). *The elemetns of statistical learning: Data mining, inference, prediction*. Springer.

Hellman, M. E., & Cover, T. M. (1970). Learning with finite memory. *Ann. Math. Stat.*, *41*, 765–82.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. University of Michigan Press. (Second edition: MIT Press, 1992).

Howard, R. (1960). *Dynamic programming and markov processes*. Cambridge, MA: MIT Press.

Hu, J., & Wellman, M. P. (1998). Multiagent reinforcement learning: Theoretical framework and an algorithm. *ICML '98* (pp. 242–250). Madison, WI.

Humphrys, M. (1996). Action selection methods using reinforcement learning. *Proceedings of the Fourth International Conference on the Simulation of Adaptive Behavior: From Animals to Animats* (pp. 135–44). Cambridge, MA: MIT Press/Bradford Books.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., & Hinton, G. E. (1991). Adaptive mixtures of local experts. *Neural Computation*, *3*, 79–87.

Jain, S., & Sharma, A. (1995). On aggregating teams of learning machines. *Theoretical Computer Science*, *137*, 85–108.

Jordan, M. I., & Jacobs, R. A. (1992). Hierarchies of adaptive experts. *Advances in Neural Information Processing Systems 4. Proceedings of the 1991 Conference* (pp. 985–992). San Mateo, CA: Morgan Kaufmann.

Kaelbling, L. P. (1993). *Learning in embedded systems.* Cambridge, MA: MIT Press.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, *4*, 237–85.

Kleinberg, J. M. (1997). Two algorithms for nearest-neighbor search in high dimensions. *ACM Symp. on Theory of Computing.*

Kohavi, R., & Sommerfield, D. (1995). Feature subset selection using the wrapper method: overfitting and dynamic search space technology. *KDD 1995.*

Lin, L.-J. (1993). *Reinforcement learning for robots using neural networks.* Doctoral dissertation, Carnegie Mellon University.

Lin, L.-J., & Mitchell, T. M. (1992). Reinforcement learning with hidden states. *Proceedings of the Second International Conference on Simulation of Adaptive Behavior: From Animals to Animats* (pp. 271–80).

Littman, M., & Boyan, J. (1993). *A distributed reinforcement learning scheme for network routing* (Technical Report CMU-CS-93-165). School of Computer Science, Carnegie Mellon University.

Littman, M. L. (1994). Markov games as a frame work for multi-agent reinforcement learning. *ICML '94.*

Liu, B., Wang, K., Mun, L., & Qi, X. (1998). Using decision tree for discovering. *PRICAI-98.*

Liu, B., Xia, Y., & Yu, P. S. (2000). Clustering through decision tree construction. *SIGMOD-00.*

Mahadevan, S. (1996). Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine Learning*, *22*, 159–96.

Markey, K. L. (1994). Efficient learning of multiple degree-of-freedom control problems with quasi-independent q-agents. *Proceedings of the 1993 Connectionist Models Summer School*. Hillsdale, NJ: Erlbaum Assoiciates.

McCallum, A. K. (1995). *Reinforcement learning with selective perception and hidden state*. Doctoral dissertation, University of Rochester.

McCallum, R. A. (1993). Overcoming incomplete perception with utile distinction memory. *Proceedings of the Tenth International Conference on Machine Learning*.

Meuleau, N., & Bourgine, P. (1999). Exploration of multi-state environments: Local measures and back-propagation of uncertainty. *Machine Learning*, 1–43.

Munos, R., & Moore, A. (1999). Variable resolution discretization for high-accuracy solutions of optimal control problems. *IJCAI '99*.

Nash, J. (1951). Non-cooperative games. *Annals of Mathematics*, *54*, 286–295.

Nowak, M. A., & May, R. M. (1993). The spatial dilemma of evolution. *Int. J. Bifurc. & Chaos*, *3*.

Ormoneit, D., & Sen, Ś. (2002). Kernel-based reinforcement learning. *Machine Learning*, *49*, 161–178.

Owen, G. (1982). *Game theory*. New York: Academic Press. second edition.

Owen, G. (1995). *Game theory*. New York: Academic Press. third edition.

Puterman, M. (1994). *Markov decision processes: Discrete stochastic dynamic programming*. New York: Wiley.

Pyeatt, L. D., & Howe, A. E. (1999). Integrating pomdp and reinforcement learning for a two layer simulated robot architecture. *Proceedings of the Third Annual Conference on Autonomous Agents* (pp. 168–74). Seattle, WA.

Ramkumar, G., & Swami, A. (1998). Clustering data without distance function. *IEEE Data(base) Engineering Bulletin*, *21*, 9–14.

Ring, M. (1993). Learning sequential tasks by incrementally adding higher. In S. J. Hanson, J. D. Cowan and C. L. Giles (Eds.), *Nips '93*, vol. 5. San Mateo, CA: Morgan Kaufmann.

Ron, D. (1995). *Automata learning and its applications*. Doctoral dissertation, Hebrew University.

Ron, D., Singer, Y., & Tishby, N. (1994). Learning probabilistic automata with variable memory length. *Proceedings of Computational Learning Theory*. ACM Press.

Rubinstein, A. (1986). Finite automata play the repeated prisoner's delemma. *Journal of Economic Theory*, *39*, 83–96.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, *3*, 211–29. Reprinted in E. A. Feigenbaum and J. Feldman, editors, *Computers and Thought*, McGraw-Hill, New York 1963.

Sandholm, T. W., & Crites, R. H. (1996). Multiagent reinforcement learning in the iterated prisoner's dilemma. *Biosystems*, *37*, 147–66.

Santamaría, J. C., Sutton, R. S., & Ram, A. (1998). Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive behavior*, *6*, 163–218.

Sato, M., Abe, K., & Takeda, Y. (1990). Learning control of finite markov chains with explicit trade-off between estimation and control. *Connectionist Models Proceedings of the 1990 Summer School* (pp. 287–300). San Mateo, CA: Morgan Kaufmann.

Schaerf, A., Shoham, Y., & Tennenholtz, M. (1995). Adaptive load balancing: A study in multi-agent learning. *Journal of Artificial Intelligence Research*, *2*, 475–500.

Schraudolph, N. N., Dayan, P., & Sejnowski, T. J. (1994). Using the TD(λ) algorithm to learn an evaluation function for the game of Go. In *Nips '94*, vol. 6. San Mateo, CA: Morgan Kaufman.

Schwarz, G. (1978). Estimating the dimension of a model. *Annals of Statistics*, *6*, 461–464.

Selim, S. Z., & Ismail, M. A. (1984). K-Means-Type algorithms: A generalized convergence theorem and characterization of local optimality. *IEEE Trans.*, *PAMI-6*.

Sen, S., & Arora, N. (1997). Learning to take risks. *Collected papers from AAAI-97 workshop on Multiagent Learning* (pp. 59–64). AAAI.

Shapley, L. S. (1953). Stochastic games. *Proceedings of National Academy of Sciences of the United States of America* (pp. 1095–100).

Singh, S., Kearns, M., & Mansour, Y. (2000). Nash convergence of gradient dynamics in general sum games. *UAI* (pp. 541–548).

Smith, C. (1982). The power of pluralism for automatic program synthesis. *Journal of the ACM*, *29*, 1144–1165.

Srikant, R., & Agrawal, R. (1996). Mining quantitative association rules in large relational tables. *SIGMOD-96* (pp. 1–12). Montreal, Quebec, Canada.

Stone, P., & Veloso, M. (1999). Team-partitioned, opaque-transition reinforcement learning. *Proceedings of the 3rd International Conference on Autonomous Agents*.

Strehl, A., & Ghosh, J. (2000). A scalable approach to balanced, high-dimensional clustering of market-baskets. *HiPC-2000*.

Sugawara, T., & Lesser, V. (1998). Learning to improve coordinated actions in cooperative distributed problem-solving environments. *Machine Learning*, *33*, 129.

Sutton, R. S. (1984). *Temporal credit assignment in reinforcement learning*. Doctoral dissertation, University of Massachusetts at Amherst.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, *3*, 9–44.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. *Proceedings of the Seventh International Conference on Machine Learning* (pp. 216–224). San Mateo CA: Morgan Kaufman.

Sutton, R. S. (1991). Planning by incremental dynamic programming. *Proceedings of the Eighth International Workshop on Machine Learning* (pp. 353–7). Morgan Kaufmann.

Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in Neural Information Processing Systems 8* (pp. 1038–1044). MIT Press.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning*. MIT Press.

Tan, M. (1991). Learning a cost-sensitive internal representation for reinforcement learning. *Proceedings of the Eighth International Workshop on Machine Learning (ML'91)*.

Tan, M. (1993). Multi-agent reinforcement learning: independent vs. cooperative agents. *Proceedings of the Tenth International Conference on Machine Learning*. Amherst, Massachusetts: Morgan Kaufmann.

Tesauro, G. J. (1992). Practical issues in temporal difference. In J. E. Moody, D. S. Lippman and S. J. Hanson (Eds.), *Nips '92*, vol. 4. San Mateo, CA: Morgan Kaufman.

Tham, C., & Prager, R. (1994). A modular q-learning architecture for manipulator task decomposition. *Proceedings of the Eleventh International Conference on Machine Learning*. Morgan Kaufmann.

Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, *25*, 275–94.

van der Wal, J. (1981). Stochastic dynamic programming. In *Mathematical centre tracts 139*. Amsterdam: Morgan Kaufmann.

von Neumann, J., & Morgenstern, O. (1947). *Theory of games and economic behavior*. Princeton, New Jersey: Princeton University Press.

Watkins, C. (1989). *Learning from delayed rewards*. Doctoral dissertation, Cambridge University.

Watkins, C., & Dayan, P. (1992). Q-learning. *Machine Learning*, *8*, 279–292.

Weiss, G. (1998). A multiagent perspective of parallel and distributed machine learning. *Proceedings of the 2nd International Conference on Autonomous Agents* (pp. 226–230).

Whitehead, S. D. (1992). *Reinforcement learning for the adaptive control of perception and action*. Doctoral dissertation, University of Rochester.

Wiering, M., & Schmidhuber, J. (1997). HQ-learing. *Adaptive Behavior*, *6*, 219–46.

Williams, N. (2001). Stability and long run equilibrium in stochastic fictitious play. working paper.

Yanco, H., & Stein, L. A. (1993). An adaptive communication protocol for cooperating mobile robots. *Proceedings of the Second International Conference on the Simulation of Adaptive Behavior: From Animals to Animats* (pp. 478–85). MIT Press/Bradford Books.

Zhang, T., Ramakrishnan, R., & Livny, M. (1997). BIRCH: A new data clustering algorithm and its applications. *Data Mining and Knowledge Discovery*, *1*, 141–182.

Zhu, S., & Li, T. (2002). An algorithm for non-distance based clustering in high dimensional spaces. *IJCNN'02*.